A Study on Efficient Designs of Approximate Arithmetic Circuits

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
University of Saskatchewan
Saskatoon

By
Suganthi Venkatachalam

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Electrical and Computer Engineering
57 Campus Drive
University of Saskatchewan
Saskatoon, Saskatchewan, S7N 5A9
Canada

OR

Dean of the College of Graduate and Postdoctoral Studies
116 Thorvaldson Building, 110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan, S7N 5C9
Canada

Abstract

Approximate computing is a popular field where accuracy is traded with energy. It can benefit applications such as multimedia, mobile computing and machine learning which are inherently error resilient. Error introduced in these applications to a certain degree is beyond human perception. This flexibility can be exploited to design area, delay and power efficient architectures. However, care must be taken on how approximation compromises the correctness of results. This research work aims to provide approximate hardware architectures with error metrics and design metrics analyzed and their effects in image processing applications.

Firstly, we study and propose unsigned array multipliers based on probability statistics and with approximate 4-2 compressors, full adders and half adders. This work deals with a new design approach for approximation of multipliers. The partial products of the multiplier are altered to introduce varying probability terms. Logic complexity of approximation is varied for the accumulation of altered partial products based on their probability. The proposed approximation is utilized in two variants of 16-bit multipliers. Synthesis results reveal that two proposed multipliers achieve power savings of 72% and 38% respectively compared to an exact multiplier. They have better precision when compared to existing approximate multipliers. Mean relative error distance (MRED) figures are as low as 7.6% and 0.02% for the proposed approximate multipliers, which are better than the previous state-of-the-art works. Performance of the proposed multipliers is evaluated with geometric mean filtering application, where one of the proposed models achieves the highest peak signal to noise ratio (PSNR).

Second, approximation is proposed for signed Booth multiplication. Approximation is introduced in partial product generation and partial product accumulation circuits. In this work, three multipliers (ABM-M1, ABM-M2, and ABM-M3) are proposed in which the modified Booth algorithm is approximated. In all three designs, approximate Booth partial product generators are designed with different variations of approximation. The approximations are performed by reducing the logic complexity of the Booth partial product generator, and the accumulation of partial products is slightly modified to improve circuit performance. Compared to the exact Booth multiplier, ABM-M1 achieves up to 15% reduction in power

consumption with an MRED value of 7.9×10^{-4}. ABM-M2 has power savings of up to 60% with an MRED of 1.1×10^{-1}. ABM-M3 has power savings of up to 50% with an MRED of 3.4×10^{-3}. Compared to existing approximate Booth multipliers, the proposed multipliers ABM-M1 and ABM-M3 achieve up to a 41% reduction in power consumption while exhibiting very similar error metrics. Image multiplication and matrix multiplication are used as case studies to illustrate the high performance of the proposed approximate multipliers.

Third, distributed arithmetic based sum of products units approximation is analyzed. Sum of products units are key elements in many digital signal processing applications. Three approximate sum of products models which are based on distributed arithmetic are proposed. They are designed for different levels of accuracy. First model of approximate sum of products achieves an improvement up to 64% on area and 70% on power, when compared to conventional unit. Other two models provide an improvement of 32% and 48% on area and 54% and 58% on power, respectively, with a reduced error rate compared to the first model. Third model achieves MRED and normalized mean error distance (NMED) as low as 0.05% and 0.009%. Performance of approximate units is evaluated with a noisy image smoothing application, where the proposed models are capable of achieving higher PSNR than existing state of the art techniques.

Fourth, approximation is applied in division architecture. Two approximation models are proposed for restoring divider. In the first design, approximation is performed at circuit level, where approximate divider cells are utilized in place of exact ones by simplifying the logic equations. In the second model, restoring divider is analyzed strategically and number of restoring divider cells are reduced by finding the portions of divisor and dividend with significant information. An approximation factor p is used in both designs. In model 1, the design with p = 8 has a 58% reduction in both area and power consumption compared to exact design, with a Q-MRED of 1.909×10^{-2} and Q-NMED of 0.449×10^{-2}. The second model with an approximation factor p = 4 has 54% area savings and 62% power savings compared to exact design. The proposed models are found to have better error metrics compared to existing designs, with better performance at similar error values. A change detection image processing application is used for real time assessment of proposed and existing approximate dividers and one of the models achieves a PSNR of 54.27 dB.

Acknowledgements

I am grateful to my supervisor Dr. Seok-Bum Ko for his continuous guidance, his unfailing moral and emotional support and giving me the freedom I needed. He was always there for me to overcome any obstacle I was facing during my research. I would not have reached the end of this journey if it is not for him.

Thank you, Hao Zhang, for lending me a helping hand whenever I needed one. It was a great learning and sharing experience with my lab mates Juan Yepez, Yi Wang, Elizabeth Adams, Riel Castro Zunti and Zhexin Jiang during the last four years. It was a fantastic experience in my lab 2C60 and at the University of Saskatchewan. My special thanks to my committee members for their valuable feedback.

A special gratitude to my mom, my dad and my daughter for their patience and encouraging words. They made it possible for me to reach this milestone. My special thanks to Harrison Kunkel and Anup Reddy for their unwavering emotional support throughout my program.

My gratitude to Natural Sciences and Engineering Research Council of Canada (NSERC) and the Department of Electrical and Computer Engineering, University of Saskatchewan.

Dedicated to my family and friends

Contents

List of Tables

List of Figures

List of Abbreviations

ABM-M1	Approximate Booth Multiplier-Model1
ABM-M2	Approximate Booth Multiplier-Model2
ABM-M3	Approximate Booth Multiplier-Model3
ACM	Approximate Compressors based Multipliers
AD-M1	Approximate Divider-Model1
AD-M2	Approximate Divider-Model2
APP	Area Power Product
ASOP	Approximate Sum Of Products
AVC	Advanced Video Coding
DCT	Discrete Cosine Transform
DRAM	Dynamic Random Access Memories
DSP	Digital Signal Processing
ED	Error Distance
ETA	Error Tolerant Adder
HEVC	High Efficiency Video Coding
JPEG	Joint Photographic Experts Group
LOA	Lower part OR Adder
LVA	Load Value Approximation
MAC	Multiply and Accumulate
MOSFET	Metal Oxide Semiconductor Field Effect Transistors
MPEG	Motion Pictures Expert Group
MRED	Mean Relative Error Distance
MSE	Mean Square Error
NMED	Normalized Mean Error Distance
PDP	Power Delay Product
PERF-SOP	Perforated multiplier based SOP

PIM	Processor In Memory
PPP	Partial Product Perforation
PROB-SOP	Probabilistic multiplier based SOP
PSNR	Peak Signal to Noise Ratio
R4ABM	Radix-4 Approximate Booth Multiplier
SIMD	Single Instruction Multiple Data
SNG	Stochastic Number Generator
SSM	Static Segment Multiplier
UDM	Under Designed Multiplier
VOS	Voltage Over Scaling

1 Introduction and Motivation

Approximate computing is producing imprecise results instead of perfect precise results, with varying differences in the efforts of producing them. It is about replacing guaranteed exact results with viable inexact results.

> " If I asked you to divide 500 by 21 and I asked you whether the answer is greater than one, you would say yes right away. You are doing division but not to the full accuracy. If I asked you whether it is greater than 30, you would probably take a little longer, but if I ask you if it's greater than 23, you might have to think even harder. The application context dictates different levels of effort, and humans are capable of this scalable approach, but computer software and hardware are not like that. They often compute to the same level of accuracy all the time [1]. "
>
> *Dr. A. Raghunathan,*
> *Purdue University*

Similar to scalable efforts in human thinking, modern computing devices can too choose to have varying strain in its computing resources with help of approximate or inexact computing. Some main motivational factors behind approximate computing are discussed in the following section.

1.1 Motivation

The key motivational factors to maximize the opportunity for approximate computing are listed in this section.

1.1.1 Saturation in Rate of Progress in Chip Technology

Gordon Moore, founder of Fairchild electronics and co-founder of Intel made some interesting observations and projections based on historical trends. He made a prediction in 1965 that

integrated circuits leading to home computers and personal portable, powerful computing systems with the number of components in integrated circuit double every year and this continued to be true for a decade [2]. After a decade, he revised his prediction that approximated a doubling of transistors every 24 months in 1975 [3]. In the Figure 1.1 of [3], Moore had taken chip area into consideration and the components density that can be packed in the chip area and even with the reduced slope, he expected integrated circuits to have several million components in the future years. This led to another prediction by David House, an Intel executive at the time, that the computer performance would double every 18 months. These predictions foretold the infiltration of technology in a common man's life ranging from iPods, mobile phones to play stations, and Moore's law served as a rule of thumb to reach in technological treadmill [4]. Trends in digital electronics and goals in engineering and technology are strongly associated with Moore's law for many decades. Shrinking transistors have enabled advance in computing for around five decades, as shown in Figure 1.2.

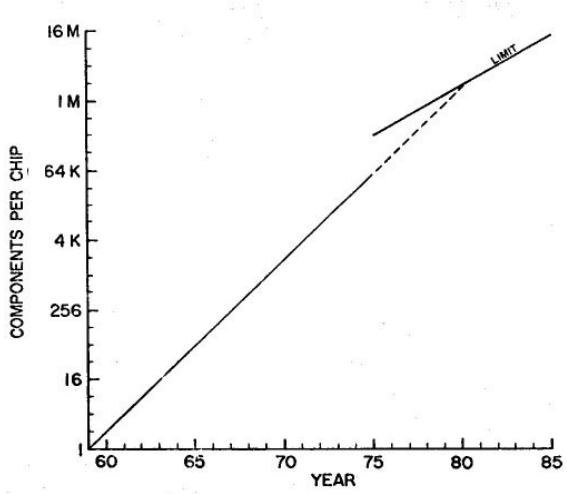


Figure 1.1: Prediction of Moore in 1975 [3]

Manufacturers were able to fit twice as many transistors roughly every 24 months, since 1970s. This leads to the possibility of smart phones, efficient data mining and data processing, and breakthroughs in powerful deep learning machines. However, shrinking of transistors has been slowing down. In the regulatory filing in 2016 [6, 7], Intel disclosed that gap between successive newer, smaller chips will widen. Transitioning to smaller feature sizes is becoming increasingly difficult, starting at 22 nm feature size in 2012, Intel's latest chips have feature

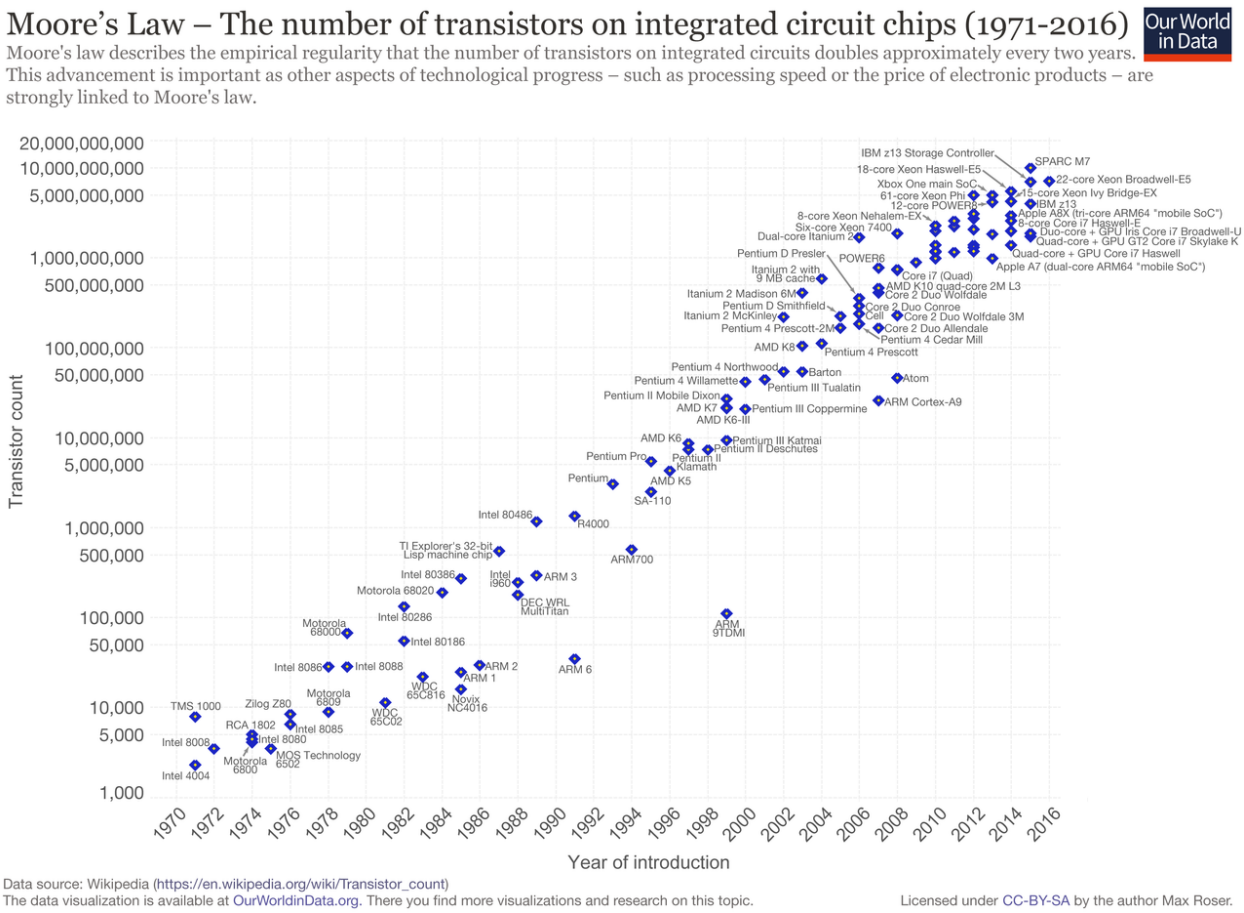


Figure 1.2: Number of tranistors on integrated chips in five decades [5]

size of 14 nm released in 2014, and in future Intel has pushed back on delivering its next transistor technology 10 nm silicon to the end of 2019. It is evident that it is becoming increasingly hard to shrink the sizes further down. The increasing gap of Moore's law is discussed in [8, 9]. In 2015, Gordon Moore predicted that the rate of increase in transistor density will reach saturation.

It should be noted that Moore's law predicts the transistor count doubling, not the raw performance. Although transistor count kept on increasing in last few decades, there has been not much improvement in performance. Manufacturers have focused more on reducing power consumption. On a parallel note, in a scaling law known as Dennard scaling coined in 1974 and originally formulated for metal oxide semiconductor field effect transistors (MOSFETs) [10], Robert H. Dennard observed that with reduction of transistors in size, their power density remains constant, i.e., power being in proportion to the area, even though size of

transistors is being reduced. This would allow the transistors to be packed denser, as the heat generated by any transistor is reduced. It can be said that performance per watt growing exponentially at the same rate as the rate of fitting more transistors per chip at cost-effective optimum. Power in a integrated circuit chip can be given as

$$Total\ power = P_{dynamic} + P_{leakage} \tag{1.1}$$

$$P_{dynamic} \propto \alpha \cdot Q \cdot f \cdot C \cdot V^2 \tag{1.2}$$

where α is the switching activity factor, Q is the number of transistors, f represents the frequency of operation, C is the capacitance, V stands for operating voltage and $P_{leakage}$ is the leakage power.

Power in the chip for a given area size remained almost same from process generation to process generation. Dynamic power consumption is directly proportional to frequency. When there is drastic increase in clock frequencies, overall power consumption remained almost same because of the transistor power reduction. Feature size continued to shrink, but with feature sizes below 65 nm, these rules could no longer remain sustained, since the leakage power exponentially increased with smaller feature sizes. In general, industry consensus is that Dennard scaling broke down around 2006 [11] as shown in Figure 1.3. The graph shows the trend from increasing clock speed to limited clock speed. Higher clock speeds became increasingly difficult mainly due to too high power consumption, heat too hard to dissipate and current leakage. At small feature sizes, leakage power causes chip to heat up and thermal runaway, thus resulting in inability to increase clock frequencies [12]. With shrinking of transistors, energy dissipation issue is becoming harder. Rather focusing on highly integrated chips to meet the demands, it is time to look for alternatives at different angles. One of the alternative to the end of Dennard scaling is to use multicore processors. Still, individual switching activities of each core will result in overall power consumption and more issues with power dissipation [13].

The way to overcome these issues is that companies need to get creative. Alternative ways include making specialized chips to fasten particular applications. The end of Moore's law and Dennard's scaling will pave way to new era of computer architecture. When the existing

strategies are no longer sufficient to meet future needs in the industry, new technologies have to be analyzed. This leads to trying new ideas rather than basing their relentless improvements on shrinking of transistors. In recent years, we are seeing more specialized architectures catered for specific applications. It is safe to say that industry shifted its focus from semiconductor scaling to meeting the demands of major computing applications. For example, Google has had their own chips TPUs- TensorFlow Units optimized for deep learning networks. The significant factor in this chip is that fixed point values are used instead of single or double precision floating point values of traditional deep learning neural networks and network is trained to required precision thereby reducing computational complexity. In a typical mobile phone, there are ARM processor cores and special purpose processors in the same silicon. The special purpose processors serve needs like managing and processing images and videos from the camera and perform face detection and audio signal processing. In similar fashion, special purpose processors with approximate computing can be designed for specific applications which are discussed in next section.

1.1.2 Power Hungry and Error Tolerant Applications

There are numerous applications that do not need precise answer. This phenomenon of a particular application being tolerant to deviation from accurate results is termed as error resilience. There exists wide spectrum of applications which fits into above mentioned category-digital signal processing for multimedia signals, data mining, data analytics, computer vision, deep neural networks and so on.

Digital signal processing involves processing information such as images, video and audio processing. To understand the error resilience of these applications, let us take few examples. A grayscale image consists of shades of gray ranging from black to white and each shade of gray can be represented using a 8-bit pixel value ranging from 0 to 255, as shown in Figure 1.4. In the Figure 1.4, from perception of human vision, shade of gray represented by pixel value 102 would be same as the shade of gray represented by pixel value 120. A color image consists typically of three pixel values of basic colors contributing to red, green and blue planes. When three values in each pixel ranges from 0 to 255, the combination (0, 0, 0) stands for a black pixel and the group (255, 255, 255) stands for white color. Any value in

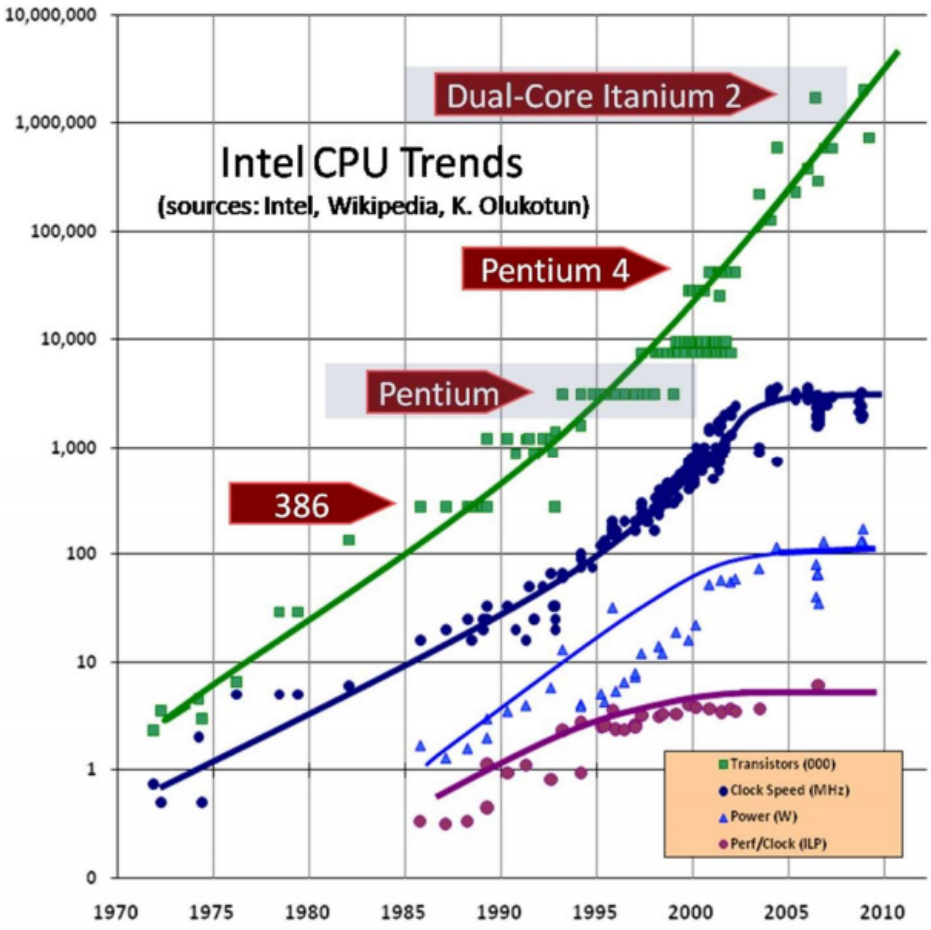


Figure 1.3: CPU scaling comparing transistor density, limit trends in maximum clock speed, power consumption and efficiency [11]

between would represent different combination of red, green and blue colors. As can be seen from Figure 1.5, shade of green with values (64, 255, 0), (0, 255, 0) and (0, 255, 64) are similar. This concept of human visual perception is taken into account while implementing our approximate arithmetic circuits in image processing application.

Analog audio signals are digitally represented after sampling in range of digital steps. Bit depth is the number of bits of information representing each sample. There are audio coding formats for storage and transmission of digital audio. Analogous to digital images and videos, number of bits can be carefully reduced or altered with minimum loss in quality imperceptible to human senses. In [14], resilience characterization is performed for 12 applications from recognition, computer vision, data mining, search and digital signal processing. In this work, resilient and sensitive computation portions are identified. Applications including

document and image search, hand written digit recognition, data classification, data clustering are analyzed for dominant units, resilient units and their contribution to percentage of runtime. Main dominant units of error resilient applications are dot product computation, distance computation and matrix vector multiplication. In [14], it is shown that the popular applications spend 83% of the runtime in error resilient computations and emphasizes the prospective of approximate computing in these areas.

In popular video coding standard called High Efficiency Video Coding (HEVC) [15], motion estimation blocks which essentially finds the best block to match with next frame while compressing the video takes about 80% of the total energy consumption of the encoder. It should be noted that motion estimation block primarily consists of adders, multipliers and dividers. On a similar note, machine learning algorithms primarily employ distance computation units which return values, and based on the maximum or minimum value, the match is found. In these cases, the result of distance computation unit does not have direct impact on the result, rather it decides the match. These applications are highly error resilient. In a machine learning algorithm such as K-means clustering application, Euclidean distance computation can be replaced with approximate units and will result in no effects or negligible effects in the results. Iterative nature of machine learning algorithms can make use of approximation.

Similarly, in image transforms such as discrete cosine transforms (DCT) in Joint Photographic Experts Group (JPEG), DCT converts time domain information into frequency domain information which is later quantized to reduce the size during storage and transmission of images. Since quantization error is going to be introduced after DCT in JPEG, DCT can be taken as a candidate for approximation.

Data mining is about extracting information in large data sets and is at the intersection of machine learning, statistics and database systems. In case of results from search engines, there are no right compilation of answers. Multiple sets of results in different combinations are permissible, the base challenge being relevance of the results to the search key terms. Approximate computing can be implemented in these scenarios. It should be noted that dominant units in most of the resilient applications are made of basic arithmetic units comprising addition, multiplication and division.

Approximate computing replaces traditional exact logic with carefully analyzed inexact logic [16]. Overall, applications such as image and video processing, deep learning, data mining and image recognition are computationally expensive and their major arithmetic blocks include addition, multiplication, sum of product units and division units which account for large extent of energy consumption, and in turn there is always an increasing interest to increase their energy efficiency. Taking advantage of this robustness, to reduce the hardware complexity of such applications, significant amount of research works was proposed. Approximate computing is one promising solution to reduce design complexity and to improve performance. Approximation offers an effective solution for energy efficient system designs [16].

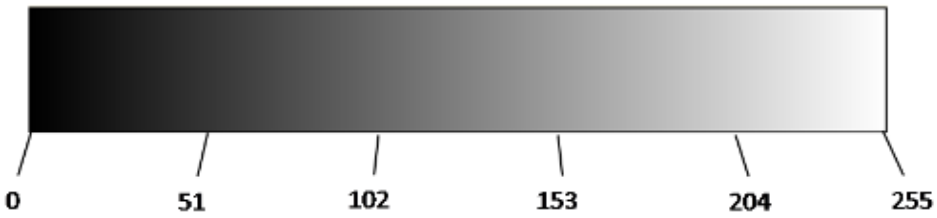


Figure 1.4: Grayscale image with its pixel values

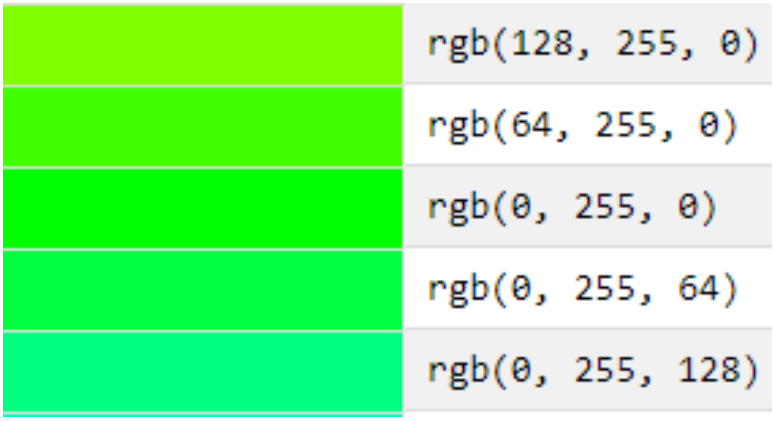


Figure 1.5: Different shades of green with its pixel values

1.1.3 Other External Factors

Defects in hardware during the manufacturing process [18] is a factor to employ approximation. Faulty hardware need not be tossed off and can still be used in applications with

inherent tolerance. In case of processing intrinsic noisy nature of the data such as information collected from global positioning system sensors [17], approximate computing can be implemented.

1.1.4 Flexible Quality as a Function of Performance

Approximate computing enables to configure quality as a trade-off to energy efficiency. Based on the required quality, consumption of energy can be dictated.

1.2 Challenges in Approximate Computing

The nature of approximate computing poses certain challenges. Some key challenges can be stated as

- *Limitations in applications:* Approximation cannot be applied to applications which involves sensitive data such as encryption and decryption in cryptography, defense applications, avionics and other hard real-time systems.

- *Threshold of approximate computing and optimal trade-off:* The degree of incorrectness in result after approximate computing plays an important role in the quality perceived by the end user. Trade-off space between quality and performance has to be optimized.

- *Finding the right approximation approach:* There is no one approximation approach for all the applications. Each application can utilize one or combination of different approximation schemes and efforts have to be taken to optimize the approximation strategies per application basis.

1.3 Other Computing Strategies for Error Resilient Applications

Some other computing models which are existing for error tolerant applications, and how they differ from approximate computing are discussed here.

1.3.1 Stochastic Computing

Stochastic computing is an unconventional computing method, first introduced in 1967 [19]. Stochastic computing offers an alternative approach for traditional binary computing. It converts binary numbers to bit-streams which are processed using basic logic units. Digitized probabilities can be derived from the bit-streams independent of their structure and length. Arithmetic operations such as addition, subtraction, multiplication and division are replaced by simplified and basic logic elements. To multiply two numbers, binary representation is converted to stochastic representation and multiplication arithmetic is replaced by AND gates. To add two numbers, addition arithmetic is replaced by a multiplexer. Multiplication and addition of two stochastic numbers using AND logic and multiplexer logic respectively are depicted in Figure 1.6 and Figure 1.7 respectively [27]. A stochastic number generator (SNG) is used to generate stochastic number from input binary number and a random number generator. Different kinds of stochastic number generators are discussed in [20], [21]. Stochastic numbers are processed by logically simple circuits rather than complex arithmetic hardware.

Applications which have fault tolerance can benefit from stochastic computing and its inherent faulty nature and simplified circuits. In [22], stochastic computing is used in implementing distance computations in vector quantization for image compression application. Distances are calculated based on L1 norm, squared L2 norm and p^{th} law in [22]. Stochastic error calculators are designed based on XOR gates and multiplexers, replacing arithmetic subtractors and adders. Stochastic computing also finds its applications in reliablility analysis [23], polynomial arithmetic [24] and neural networks [25, 26].

Advantages of stochastic computing are

- High degree of error tolerance, specifically when there are soft errors or transient errors due to process variation or cosmic radiation. For instance, when the error output is 10001100 instead of 00001100 in stochastic computing, it results in small error in arithmetic distance from 2/8 to 3/8. In binary computing, similar bit flip located at most significant part will result in large error.

- Complex arithmetic modules are replaced with simple circuits made of basic logic gates

and/or multiplexers. The bit streams can be implemented in series or parallel.

Stochastic computing has several issues such as

- For increase in precision, longer bit stream is required and an exponential increase would result in huge increase in computation time. To increase the precision from 4 bits to 8 bits, length of stochastic bit stream increases from 16 to 256.
- Identical bit streams would result in large deviation from correct result, for example, a stochastic stream 's' multiplied with stream 's' with AND logic would result in 's' instead of s^2.
- Stochastic number generators and conversion processes itself are computationally expensive.

Although stochastic computing has its merits [27], the major drawback is increased complexity in the implementation of stochastic computing itself and exponential increase in the number of bits to increase the precision. Due to these factors, stochastic computing is not an ideal candidate to apply in error intrinsic applications when we are looking for less computational complexity. The main difference between approximate computing and stochastic computing is that approximate computing has a deterministic design nature that produce imprecise results, but not non-deterministic random results. Although it utilizes the statistical properties of the data, approximate computing does not have to generate stochastic bit streams.

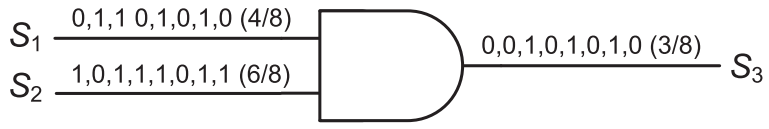


Figure 1.6: Stochastic computing of multiplication [27]

1.3.2 Data Compression

Data compression is a method of converting bit structure of the data to optimize the storage space and it is achieved by leveraging on the statistical redundancy in the data. Degree of

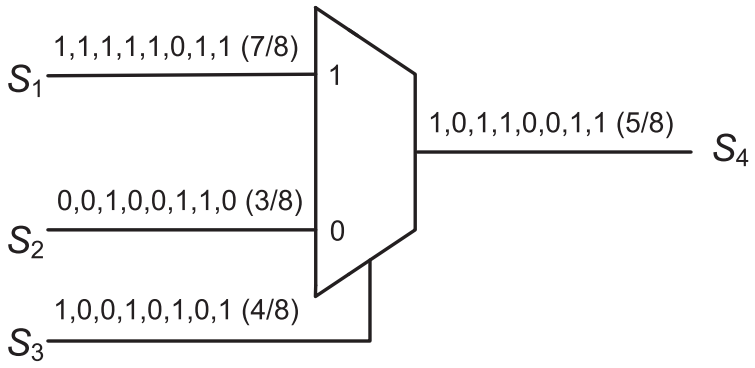


Figure 1.7: Stochastic computing of addition [27]

compression decides the amount of distortion in the results. Lossless and lossy compression techniques are two major fields in data storage and transmission. Compression algorithms based on converting frequency domain to time domain such as discrete cosine transform, discrete wavelet transform, pulse code modulation are used in popular standards such as JPEG, Motion Pictures Expert Group (MPEG)-4, Advanced Video Coding (AVC) and High Efficiency Video Coding (HEVC). Compression techniques developed especially for computer vision applications are discussed in [28]. In [31], compression is used in efficient market analysis. Choice of compression algorithm resulting in definable representation of data within a feature space and its use in machine learning are discussed in [32].

Compression and decompression are computationally intensive and require expensive hardware. They are mainly proposed for saving the storage and transmission bandwidth and more computational effort is taken while encoding and decoding the compression algorithms. The main objective of data compression schemes is to save to memory bandwidth, while approximate computing looks to reduce energy consumption, while the common theme of both data compression schemes and approximate computing is to produce results without heavily degrading the quality perceived. Considering these factors, approximate computing can be used as complementary technique for better performance with these data compression algorithms [29, 30].

1.4 Current Research on Approximate Computing

Approximate computing is currently researched across two broad domains- hardware systems and software systems. Approximate computing has attracted attention from researchers in hardware architecture and circuit designs, software engineering, programming languages and computer organization and software designers. Our thesis focuses on approximation in hardware architecture on basic arithmetic circuits widely used in intrinsic error applications. In this section, approximate computing researches going on in different domains are briefly reviewed.

1.4.1 Hardware Systems

Most of the research work in hardware is happening on arithmetic and logic units (ALU) level. It focuses on redesigning the arithmetic circuits into inexact versions to get high performance. Addition, multiplication and division are fundamental components in ALU. Since our thesis focuses on approximation at ALU level, detailed review on this topic is done in next chapter.

There are other hardware approximation strategies applied at memory level [33, 34] and hardware accelerator cores such as neural network accelerator [35, 36]. In [33], an application level technique is proposed where different refresh rates are used for critical and non-critical sections of the data reducing the energy used by Dynamic Random Access Memories (DRAM). In [34], approximation techniques are proposed for solid-state memories to address the challenges due to wear-out and slow writes. Voltage over-scaling (VOS) is discussed in [37, 38]. Lowering supply voltage creates paths failing to meet delay constraints leading to early termination of results.

1.4.2 Software Systems

In software systems, approximations include proposals of approximate programming languages, compilers, libraries and cache fetching optimizations. Modifications at the programming language level to conserve energy is introduced in [39], where approximation features are built on a new language Eon for self-adapting perpetual systems. In [40], approximate

data types are proposed and implemented on top of Java and tested with image processing, gaming and scientific computing applications. A language called Rely is presented in [41] which makes efficient use of unreliable components exhibiting soft errors.

Semantics of the programs are altered to reduce the consumption of hardware resources thereby reducing energy consumption [45, 46]. For example, non-sensitive kernels of an application can be made to execute on approximate hardware while sensitive kernels can choose to execute on exact hardware [43]. In iteration based methods, more number of iterations increases accuracy. But, in most cases after initial iterations, improvement in accuracy does not happen or negligible. Program can be modified to execute fewer iterations when requisite quality is obtained [45, 44].

Approximate library called UncertainT [17] encapsulates approximate data under object oriented programming language constructs. In [42], a framework supporting energy conscious programming is introduced. Load Value Approximation (LVA) is a technique to hide cache miss latency. When applications do not require exact data, instead of fetching the data from main memory or next level of cache, load value can be estimated, and stalling of the processor can be prevented. Spatially and temporally correlated data is the motivation behind LVA. LVA in [47] uses block fetching to train the approximator and is based on graphics applications which can tolerate errors. LVA in [48] reduces memory stalls in graphics processing units by interpolating the cache entries.

Memoization is a technique where results of a function are stored and reused for repeated task having similar functions. In [49], a spatial memoization is used in single instruction multiple data (SIMD) architecture. The result of an instruction is memoized and reused. Number of reuses is inversely proportional to the precision of the computation. In [50], precision of the value cache is monitored.

1.5 Contributions of the Thesis

Novel approximations are proposed for arithmetic circuits. Main contributions of this thesis are

- *Design of approximate arithmetic circuits:* Approximation in array based multiplier

design is presented. Partial product matrix of an array multiplier is split into low information and more relevant information regions. To accumulate the partial products, new approximate adders and compressors with fast sum and carry generations are used.

Approximation in modified Booth multipliers is analyzed. Widely used three signal encoding scheme is taken as the basis of approximation. Two signal encoding and one signal encoding approximations are presented.

Approximation for sum of products units based on distributed arithmetic is presented. Distributed arithmetic algorithm is altered to minimize the hardware resources consumption. Number of lookup tables, adders and length of the multipliers are reduced.

Approximation for restoring division is presented. Restoring division is analyzed at circuit level and strategy level and two new approximations are presented.

- *Error characterization:* Accuracy of all the presented approximations are mathematically characterized using MRED and NMED. The error characterizations when compared with area, delay and power characterizations give a better idea about the trade-off between accuracy and quality. Peak signal to noise ratio (PSNR) is used as the quality metric in applications of approximate circuits. Proposed approximate circuits generate results with acceptable values of PSNR.

- *Real-time implementations:* Proposed architectures are implemented in real time applications including machine learning algorithm - K-means clustering for color image compression, and mainly image processing applications such as image smoothing, geometric mean filtering for noise reduction, matrix multiplication, image multiplication and motion detection. The proposed architectures are compared with respective existing architectures.

- *Comparative study:* Proposed architectures are compared with state-of-the-art approximate architectures and exact architectures. Circuit level models are designed in Verilog language, implemented in 65 nm TSMC technology using Synopsys design compiler. Modelsim, Python and Matlab are used to run simulations and for finding and comparing the error characteristics.

1.6 Publications and Submissions during Ph.D. Study

1.6.1 Published/Accepted Journals

1. Suganthi Venkatachalam and Seok-Bum Ko, "Design of Power and Area Efficient Approximate Multipliers," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1782-1786, May 2017. doi: 10.1109/TVLSI.2016.2643639.

   Major portion of this paper is included in *Chapter 4: Power and Area Efficient Approximate Multipliers*

2. Suganthi Venkatachalam and Seok-Bum Ko, "Approximate Sum-of-Products Designs Based on Distributed Arithmetic," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 8, pp. 1604-1608, Aug. 2018.
   doi: 10.1109/TVLSI.2018.2818980

   Major portion of this paper is included in *Chapter 6: Approximate Sum of Products Designs based on Distributed Arithmetic*

1.6.2 Published Conference

1. Suganthi Venkatachalam, H. J. Lee and Seok-Bum Ko, "Power Efficient Approximate Booth Multiplier," *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, Italy, 2018, pp. 1-4. doi: 10.1109/ISCAS.2018.8351708.

   Some portion of this paper is included in *Chapter 5: Design and Analysis of Approximate Booth Multipliers*

1.6.3 Submitted Journal and Conferences

1. Suganthi Venkatachalam, Elizabeth Adams and Seok-Bum Ko, "Design and Analysis of Approximate Booth Multipliers," *IEEE Transactions on Computers*.

   Major portion of this paper is included in *Chapter 5: Design and Analysis of Approximate Booth Multipliers*

2. Suganthi Venkatachalam, Elizabeth Adams and Seok-Bum Ko, "Design of Approximate Restoring Dividers," *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, Japan, 2019.

   Major portion of this paper is included in *Chapter 7: Design of Approximate Restoring Dividers*

3. Elizabeth Adams, Suganthi Venkatachalam and Seok-Bum Ko, "Energy Efficient Approximate MAC Design," *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, Japan, 2019.

1.7 Organization of the Thesis

This thesis is organized as follows.

- *Chapter 1: Introduction and Motivation:* presents the introduction, motivation behind approximate computing, challenges faced in approximations, current research in approximate computing, contributions, our publication and submissions in journals and conferences and organization of this thesis.

- *Chapter 2: Review on Approximate Computing of Arithmetic Circuits:* presents the existing works on arithmetic units such as adders, compressors, counters, multipliers, dividers and sum of products units.

- *Chapter 3: Motivation Behind Our Works:* gives the motivation behind our works - array multiplier approximation in chapter 4, Booth multiplier approximation in chapter 5, approximation of sum of products based on distributed arithmetic in chapter 6 and approximate restoring division in chapter 7.

- *Chapter 4: Power and Area Efficient Approximate Multipliers:* presents an approximate multiplier based on probability of the partial product matrix and verifies the proposal with a noise reduction image processing application.

- *Chapter 5: Design and Analysis of Approximate Booth Multipliers:* presents three approximate models based on signal encoding of Booth algorithm. The architectures are used in image multiplication and matrix multiplication applications.

- *Chapter 6: Approximate Sum of Products Designs Based on Distributed Arithmetic:* presents three models on sum of products approximations by altering the distributed arithmetic, thereby saving hardware resources. Sum of product units are applied in Gaussian filtering and K-means clustering application.

- *Chapter 7: Design of Approximate Restoring Dividers:* presents two variation of approximation in restoring dividers. Real life implementation is verified using a change detection application.

- *Chapter 8: Conclusions and Future Research:* includes a summary of this thesis and thoughts on related future research.

2 Review on Approximate Computing of Arithmetic Circuits

In this chapter, previous works on major arithmetic blocks of digital signal processing algorithms- adders, array multipliers, Booth multipliers, sum of products units and dividers are discussed. Also, error metrics used to verify the inexactness of approximate circuit are discussed.

2.1 Approximate Works on Adders

There exist many variations in hardware implementation of addition- ripple carry adder works on basic addition principle, carry skip or bypass adder improves the delay of ripple carry by skipping the carry bit over a block, carry lookahead adders uses the concept of generating and propagating carries, carry speculative adders uses carry predictor circuit to reduce computational time and carry select adder has a multiplexer to select a carry out once correct carry in is known. Many approximation schemes for addition are discussed widely in literature. Approximation in carry-select adder based on speculation with error detection and recovery is proposed in [59]. In this work, a speculateve carry select adder (SCSA) is proposed, with the main idea being long carry chains are an rare event in addition of large block of inputs. Input bits of width n are segmented into same size blocks of width k with total number of blocks $m = \lceil n/k \rceil$ and carry-out of a block is speculated with k input bits of the block, resulting in reduced latency. An error tolerant adder 2 (ETA2) based on segmentation is analyzed and compared with their previous work ETA1 in [60]. ETA1 eliminates the entire carry propagation, whereas ETA2 approximates carry at block level increasing accuracy, while sum computation depends on carry in, carry out computation does not need the carry of previous block. In [58], four imprecise adders are designed at transistor level by reducing

the number of transistors and are then used in digital signal processing applications. Inexact full adders in [58] are further used in accumulation of partial products in multipliers. In [61], Lower part OR Adder (LOA) segments p bits of inputs into significant upper part using a precise adder and insignificant lower part using approximation. The proposed circuits of [61] are implemented in neural networks and fuzzy logic. In [67], multi-bit adder is divided into sub adders and conditional bound signals are used to improve accuracy. It proposes a timing starved adder model representing implementations of different adder types- ripple carry adders and tree adders. The proposed adders are demonstrated in a DCT application of image decompression and an image sharpening application.

2.2 Approximate Works on Multipliers

Multipliers are an integral part and more resources consuming operation in applications such as digital signal processing. Approximation is applied in two types of multipliers, namely AND array multipliers and Booth multipliers. AND array multipliers use AND-gates in partial product generation to produce a partial product matrix with n rows for $n \times n$ inputs. In Booth multipliers, the input combination is recoded and then used in the partial product generator to produce signed and plural values of the multiplicand. Booth multiplication reduces number of partial product rows in partial product matrix and suitable for signed multiplication. In radix-4 Booth multipliers, partial product generation produces values of 0, ± 1, and $\pm 2 \times$ multiplicand and reduces the size of the partial product matrix by nearly half. Radix-8 multipliers further reduce the number of rows in partial product matrix where the encoding signals are 0, ± 1, ± 2, ± 3 and ± 4. Partial product generation of array multipliers is made of AND gates, whereas Booth multipliers has comparatively more complicated partial product generation circuits.

Two main approximation approaches used in multipliers are approximation in partial product generation and approximation in partial product accumulation. In many cases, approximation in partial product accumulation also means reducing hardware complexity of partial product genereation. For instance, if n least significant columns are truncated during partial product accumulation stage of a multiplier, partial products need not to be generated

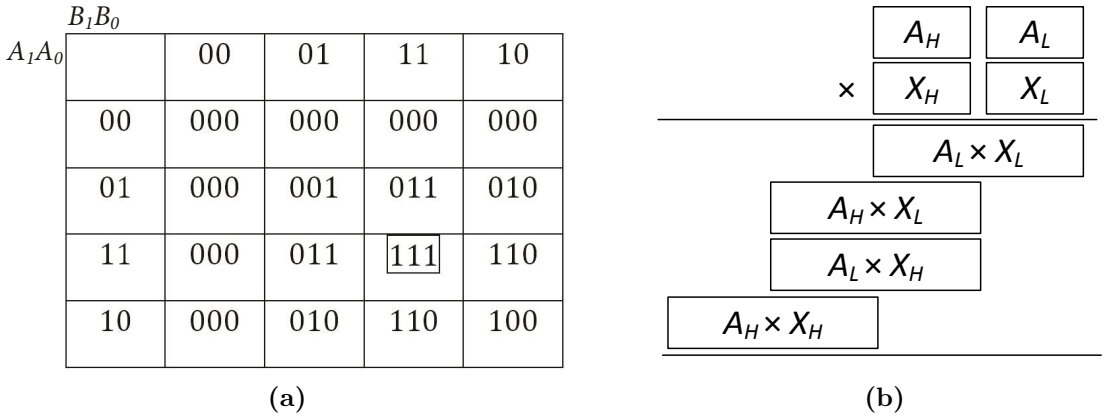
for those columns. In array multipliers, partial product accumulation is compute expensive and in Booth multipliers, both generation and accumulation are compute expensive.

2.2.1 Approximate Works in Partial Product Generation

In [51], an approximate multiplier is constructed by changing one of the values in the K-map of a 2 × 2 multiplier and the 2 × 2 approximate multiplier is used as fundamental block to construct larger approximate multipliers. This multiplier is referred as under designed multiplier (UDM). UDM multipliers achieve a mean error ranging from 1.39% to 3.35% with power savings from 30% to 50%. They are implemented in JPEG application and in image filtering application. The truth table of the UDM and the multiplier structure of UDM are given in Figure 2.1. Similar to approximation of segmented adders in ETA1 and ETA2, multiplier based on segmentations of multiplicand and multiplier is introduced in [68]. In [68], multiplier is split into most significant and least significant sections, and conventional multiplication is performed in higher order bits while approximation is applied in the remaining part by finding the first leading one of either multiplicand or multiplier. A control block is also used to utilize accurate multiplier for least significant part if most significant part does not have any information. If n bits multiplication is split into two $n/2$ parts, approximate partial product generation is reduced to one fourth of actual partial products.

Partial product perforation (PPP) multiplier in [54], omits q consecutive partial products starting from pth position, where $p \in [0, n\text{-}1]$ and $q \in [1, \text{minimum } (n\text{-}p, n\text{-}1)]$ of a n-bit multiplier. [54] targets approximation in generation by reducing the number of operands in one of the inputs and also reduces complexity in partial product matrix. Approximations are analyzed in Wallace and Dadda tree structures, with 3-2 compressors and 4-2 compressors. In case of radix-4 Booth multiplier approximation, the approximate technique used in [54] relies on approximation by eliminating the group of recoded signals and thereby reducing the accumulation hardware. They are implemented in image processing domain applications-Canny edge detection to find optimal edges, geometric mean filtering and K-mean clustering to cluster 100000 four dimensional data points into 100 clusters. Partial product perforation of accurate and approximate 8×8 Dadda multiplier is given in Figure 2.2.

In [78], approximation is applied in the generation of partial products of radix-8 Booth multipliers. Approximation is applied in the adder when adding 1 × multiplicand and 2 × multiplicand to form 3 × multiplicand. The truth table and circuit of the proposed approximate 2-bit adder is shown in Figure 2.3. The proposed multipliers are then implemented in a finite impulse response filter application.



(a) **(b)**

Figure 2.1: Truth table and structure of approximate 2 × 2 multiplier of UDM [51]

In [56], the complexity of exact radix-4 partial product generation is reduced by modifying the truth table and two approximate Booth partial product generators are proposed. In the first approximate generator, 4 out of 32 cases in the truth table are altered. In the second partial product generator, 8 out of 32 cases are altered. This multiplier is referred as Radix-4 approximate Booth multiplier (R4ABM) in this thesis. In this multiplier, the inputs are grouped as three bits at a time and each group decides one of partial product value from $\{-2A, -1A, 0, +1A, +2A\}$. The exact partial product generator of [56] uses five XOR gates, one inverter and one four-input AND gate. First approximate generator uses two XOR gates and one AND gate and second approximation uses one XOR gate. The proposed multipliers are implemented in a image multiplication application where they achieve PSNR values up to 57 dB.

2.2.2 Approximate Works in Partial Product Accumulation

Approximation in partial product accumulation can include using approximate compressors in partial product matrix or truncating and rounding the partial products based on their

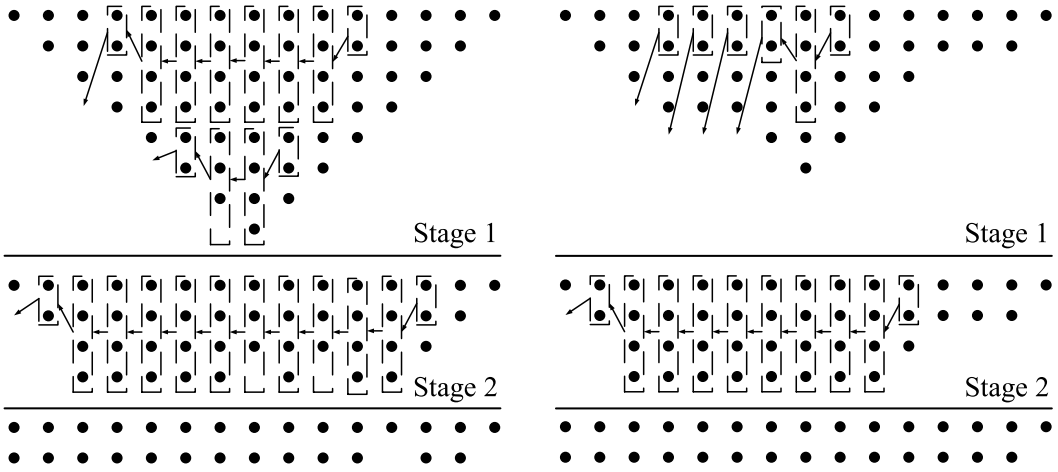


Figure 2.2: Accurate and approximate PPP with perforated third and fourth partial product rows [54]

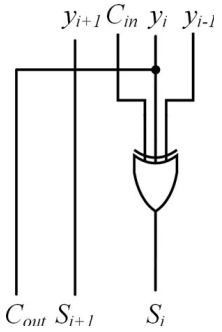
position in partial product matrix or a combination of both.

In [62], two designs of approximate 4-2 compressors are presented and used in partial product reduction tree of four variants of 8×8 Dadda multiplier. Four multipliers are further implemented in a image multiplication application and they achieve PSNR up to 54 dB. The proposed compressors are used in four designs of multipliers in [63]. Their proposed 8×8 multiplier designs perform a recursive multiplication of 4×4 multiplications. The inputs A and B are divided into two segments L and H and partial products are generated. Based on the required quality, approximate and accurate smaller multipliers are used in HH, HL, LH and LL sections. The proposed multipliers are tested with image sharpening applications. It should be noted that in [63], even though inputs are split into segments, all partial products are generated and approximation is applied only in partial product accumulation using approximate compressors.

In [52], approximate compressors for the use in high performance multipliers are introduced. In [53], inaccurate 4-2 counter design has been proposed which is utilized in power efficient 4×4 Wallace tree multiplier, and larger blocks are built recursively from smaller blocks. An error recovery module with slight overhead is introduced. The multiplier in [53] reduces power and delay consumption by 11% and 10% on average, with a high pass rate. In [65], a new approximate adder is presented for the use in partial product accumulation of the

$C_{out}S_{i+1}S_i$		y_iy_{i-1}			
		00	01	11	10
$C_{in}y_{i+1}$	00	000	001	100	**10**1
	01	010	011	110	1**1**1
	11	011	**01**0	111	110
	10	001	0**0**0	101	100

(a)



(b)

Figure 2.3: Approximate adder in radix-8 Booth multiplier of [78] (a) Truth table (b) Circuit

multiplier. The proposed multiplier of [65] shows a delay reduction of 20% and power savings up to 69%. In [66], an approximate 15-4 compressor using four designs of approximate 5-3 compressor is analyzed for 16×16 multiplier. The multipliers provide a pass rate up to 40% with no significant power improvement.

Broken array multiplier is implemented in [61], where the adder cells are omitted along the rows and columns while forming partial products to reduce hardware complexity. The proposed multiplier in [61] saves few adder circuits in partial product accumulation. In static segment multiplier (SSM) proposed in [64] as shown in Figure 2.4, the number of bits in inputs is reduced thereby m-bit segments are derived from n-bit operands based on leading one bit of the operands. Then $m \times m$ multiplication is performed instead of $n \times n$ multiplication, where $m < n$. With overhead of static circuit being small, SSM shows better energy savings. Approximation of 8-bit Wallace tree multiplier due to voltage over-scaling (VOS) is discussed in [69]. This work analyzes multiple values of over-scaling and corresponding error distribution.

To reduce hardware complexity of multipliers, truncation is widely employed in fixed-width multiplier designs. Fixed width multipliers produce n most significant bits output for $n \times n$ inputs. Truncation and rounding are performed to produce fixed word size output introducing quantization error. Various techniques are applied to reduce the quantization error after truncation in fixed point multipliers [70, 71, 72, 73, 75, 74, 76, 77]. A post-truncated fixed width Booth multiplier designed using a compensation vector is discussed

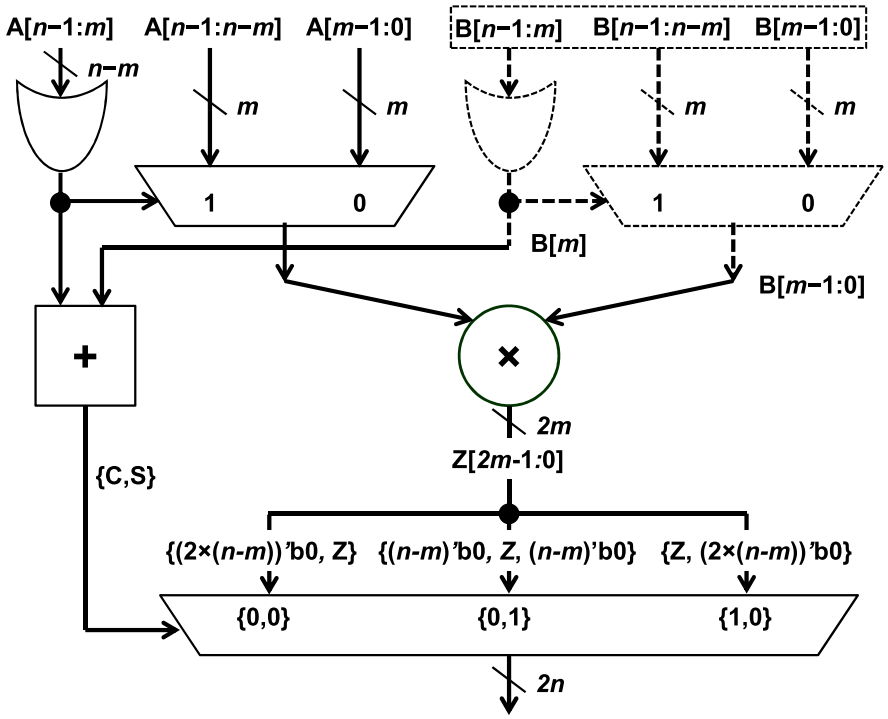


Figure 2.4: Static segment multiplier of [64]

in [70]. In [71], quantization error is compensated with approximate carry values. An error compensation circuit composed of simplified sorting networks is proposed in [72]. An adaptive estimator based on conditional probability theory is studied in [73]. In [74], a simple truncation and rounding technique for fixed point multiplier is introduced. In [76], a self-compensation fixed width multiplier with a Fast Fourier Transform application is discussed. A probabilistic estimation bias derived from probabilistic analysis of partial product array is introduced to reduce the truncation error in [77].

2.3 Approximate Works on Composite Units

Composite units such as sum of products units and multiply accumulate units are key components in dot product computations in error tolerant applications. Sum of products is a widely used arithmetic unit in signal processing and machine learning applications. Distributed arithmetic is a well-known method to save resources in sum of products structures for implementing DSP functions. Distributed arithmetic trades lookup tables with combinational

circuits especially multipliers and adders. Bit-parallel versions of distributed arithmetic are proposed in [83, 84]. Approximate sum of products model based on truncation is discussed in [85]. Their scheme involves truncation in the number of lookup tables, by eliminating the least significant part of distributed arithmetic operation. In [86], two approximations are proposed for sum of products unit. In the first approach, a single multiplication is used instead of two multiplications when one result is significantly larger than other one and two error modes are analyzed. In second approach, 16×16 multiplier is used instead of 32×32 multiplier, based on a combination of leading one prediction and 32-bit multiplexers. In [87], an approximate multiply and accumulate (MAC) is proposed. Approximate compression and column deletion are applied in partial product matrix of the proposed circuits achieving area improvement ranging from 39% to 69% and power improvement ranging from 40% to 71%. The proposed MAC units are targeted towards 2D-convolution operation with 3×3 or 5×5 kernel.

2.4 Approximate Works on Dividers

Division, one of the fundamental arithmetic operations, is computationally expensive. Different kinds of division algorithms have been discussed in literature. They might be using single or combination of techniques such as digit recurrence, functional iteration, high radix and look-up tables [102]. In digit recurrence method, the error is reduced by incremental use of operand [89]. Functional iteration based Newton-Raphson method uses multiplication as its principal operation [90]. Look-up table methods are mostly used in initial approximation and can be used in combination with other techniques [91]. Combinational array division circuits based on restoring division and non-restoring division are discussed in [92]. Depending upon the requirement of an application, a suitable division algorithm is chosen.

Division being the energy hungry unit, approximation in division is gaining attention. A dynamic divider is designed to select the most relevant bits and implemented in applications –change detection, foreground extraction and JPEG compression in [93]. Three approximate subtractor cells are proposed to design restoring and non-restoring dividers in [94]. Higher-radix division is investigated in [95]. Exact cells being replaced with approximate cells in

binary signed-digit adder, cell truncation and error compensation in higher radix division is analyzed in [95]. In [96], $2n/n$ divider is replaced with $2k/k$ dynamic divider, where $k < n$. [97] introduces a rounding based approximate divider where division is performed by rounding the value of divisor.

2.5 Accuracy Measurement

In approximate computing, design metrics such as area, delay and power are traded against error metrics like error distance (ED), mean relative ED (MRED) and normalized mean ED (NMED). Distances are based on differences in arithmetic values of exact and approximate outputs. In [98], approximate adders are evaluated and NMED is proposed as nearly invariant error measure regardless of the size of the approximate circuit. Also, traditional error analysis, MRED is found for existing and proposed multiplier designs. The errors introduced by approximate designs must be carefully analyzed because the results can be significantly affected by the approximate design. To understand the effects of the approximate unit, MRED and NMED are to be used.

NMED is an effective metric to quantify the approximation irrespective of the size of the circuit [98]. Also, traditional MRED error metric is used to evaluate the impact of approximation. Error distance is the difference between exact value and approximate value, whereas relative error is the value of error distance divided by the exact value. NMED is calculated by normalizing the error distance by maximum possible exact output. Mean relative error is calculated from mean of relative errors of all possible values.

Analysis of approximate circuits would not be complete if they are not tested with relevant applications. In multimedia applications, peak signal to noise ratio (PSNR) is used to find the effects between exact and approximate design. In machine learning applications, accuracy of classification is to be used. Thanks to the abundant redundancy in multimedia and machine learning applications, where approximate computation often does not severely deteriorate the quality while it can significantly reduce the power consumption.

In our work, we investigated novel design approaches for approximation in arithmetic circuits, studied the effect of approximation in power and accuracy, and evaluated their

performance in image processing applications. Major arithmetic circuits used in such computations include accumulators (adders and compressors), multipliers and composite units such as dividers, multiply accumulate units and sum of product units. In our work, using approximate circuits had resulted in significant power savings compared with exact designs.

3 Motivation Behind Our Works

In this chapter, motivation behind each of arithmetic circuit approximation presented in the following chapters are presented.

Previous works on logic complexity reduction of multipliers [75, 51, 53, 54] have focused on truncation, approximation in generation of partial products and straight forward application of approximate accumulators to the partial products. In our first work based on array multipliers which uses AND gates to generate partial product matrix, the partial products are altered to introduce terms with different probabilities. Probability statistics of the altered partial products are analyzed, which is followed by systematic approximation. Simplified arithmetic units (half adder, full adder and 4-2 compressor) are proposed for approximation. The arithmetic circuits are reduced in logic complexity, while error distance is kept low. Better accuracy is achieved using systemic approximation.

In our second work, novel approximation techniques are introduced in partial product generation and partial product accumulation circuits of Booth multiplier. Booth multipliers reduce number of partial products in the partial product matrix and are widely used. While the complexity of partial product matrix is reduced, there is an increase in complexity in partial product generation compared to array multiplier. When it comes to applying approximation to Booth multipliers, an efficient approximation would include approximation at partial product generation. Although truncation of fixed point Booth multipliers has received its due attention, approximation methods of Booth multipliers are being explored [56, 78]. In our work, approximation is introduced by reducing the number of signals and hardware complexity of partial product generation circuit. After approximating the generation of partial products and corresponding correction terms, Booth multipliers are implemented in image multiplication and matrix multiplication application.

In the third work, approximations are applied to sum of products units. Sum of products

units are main components in dot product computations between vectors. Distributed arithmetic is an effective means to compute sum of products [55]. Distributed arithmetic replaces multiplication with a set of look-up tables and shift-accumulate operation. Approximate sum of products units have not received much attention. Due to the flexibility of the level of parallelism in the distributed arithmetic structure, the area-speed trade-off can be adjusted. Distributed arithmetic [55] is a bit-serial operation that computes the inner product of two vectors in parallel. It requires no multiplication and it has an efficient mechanism to perform sum of products operation. Diverse set of sum of products units based on parallel distributed arithmetic are proposed and extensively analyzed with their approximation errors.

Different kinds of division algorithms have been discussed in literature. Division incurs more latency because of not being able to perform shift subtract operations in parallel. When compared with multipliers which have a latency of $O(n)$, Dividers have a latency of $O(n^2)$. Several research works can be found on division strategies. Approximation in division is gaining attention. There is much existing literature on approximation in adders and multipliers, while more works on approximation of dividers have to be analyzed.

The objectives of this thesis are twofold- to design approximate circuits and analyze its performance trade-off with accuracy and to test the approximate units in image processing applications.

4 Power and Area Efficient Approximate Multipliers [1]

In this chapter, approximate array multipliers are proposed. Two multiplier models, Multiplier1 and Multiplier2 with different approximation factors are proposed. Partial product matrix is split into low information and high information sections. While OR gates are used for compression of low information sections, high information sections are compressed using proposed approximate half adder, full adder and 4-2 compressor. Proposed approximate multipliers, state-of-the-art existing approximate multipliers and exact multiplier are analyzed in terms of area, power, delay, MRED and NMED. Further, approximate multipliers are tested with geometric mean application.

In section 4.1, proposed architectures are discussed. In section 4.2, proposed multipliers are compared with existing multipliers and found to have better area, power and error metrics. In section 4.3, the approximate multipliers are implemented in noise reduction of images and found to have reasonable quality metrics.

4.1 Proposed Architectures

Implementation of multiplier comprises three steps: generation of partial products, partial products reduction tree and finally, a vector merge addition to produce final product from the sum and carry rows generated from the reduction tree. Second step consumes more power. In this work, approximation is applied in reduction tree stage.

A 8-bit unsigned multiplier is used for illustration to describe the proposed method in

[1] Major part of this work has been published in "Design of Power and Area Efficient Approximate Multipliers," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1782-1786, May 2017. Authors: Suganthi Venkatachalam and Seok-Bum Ko.

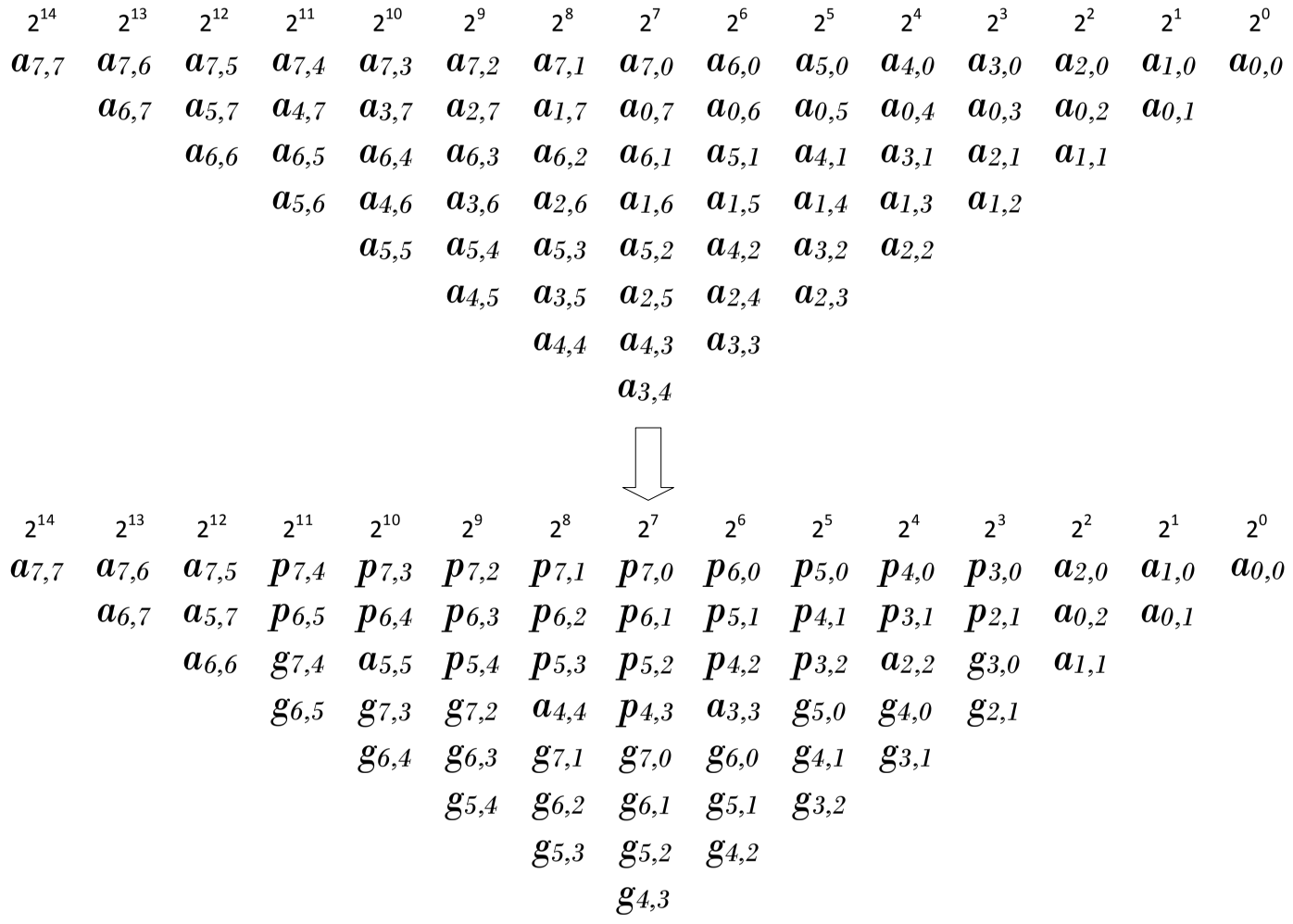


Figure 4.1: Transformation of generated partial products into altered partial products

approximation of multipliers. Consider two 8-bit unsigned input operands $\alpha = \sum_{m=0}^{7} \alpha_m 2^m$ and $\beta = \sum_{n=0}^{7} \beta_n 2^n$. The partial product $a_{m,n} = \alpha_m \cdot \beta_n$ in Figure 4.1 is the result of AND operation between the bits of α_m and β_n.

From statistical point of view, α_m and β_n has a probability of $1/2$ and the partial product $a_{m,n}$ has a probability of $1/4$ of being 1. In the columns containing more than three partial products, the partial products $a_{m,n}$ and $a_{n,m}$ are combined to form *propogate* and *generate* signals as given in equation 4.1. The resulting *propogate* and *generate* signals form altered partial products $p_{m,n}$ and $g_{m,n}$. From column 3 with weight 2^3 to column 11 with weight 2^{11}, the partial products $a_{m,n}$ and $a_{n,m}$ are replaced by altered partial products $p_{m,n}$ and $g_{m,n}$. The original and transformed partial product matrices are shown in Figure 4.1.

$$\begin{aligned} p_{m,n} &= a_{m,n} + a_{n,m} \\ g_{m,n} &= a_{m,n} \cdot a_{n,m} \end{aligned} \tag{4.1}$$

The probability of $a_{m,n}$ being 1 is $1/4$ and being 0 is $3/4$. The probability of the altered partial product $g_{m,n}$ being one is $1/16$ $(1/4 \times 1/4)$, which is significantly lower than $1/4$ of $a_{m,n}$. The probability of altered partial product $p_{m,n}$ being one is $1/16 + 3/16 + 3/16 = 7/16$ $(1/4 \times 1/4 + 1/4 \times 3/4 + 3/4 \times 1/4)$, which is higher than $g_{m,n}$. These factors are considered while applying approximation to the altered partial product matrix.

Table 4.1: Probability statistics of *generate* signals

m	Probability of the *generate* elements being				P_{err}
	all 0's	one 1	two 1's	three 1's and more	
2	0.8789	0.1172	0.0039	-	0.00390
3	0.8240	0.1648	0.0110	0.00024	0.01124
4	0.7725	0.2060	0.0206	0.00093	0.02153

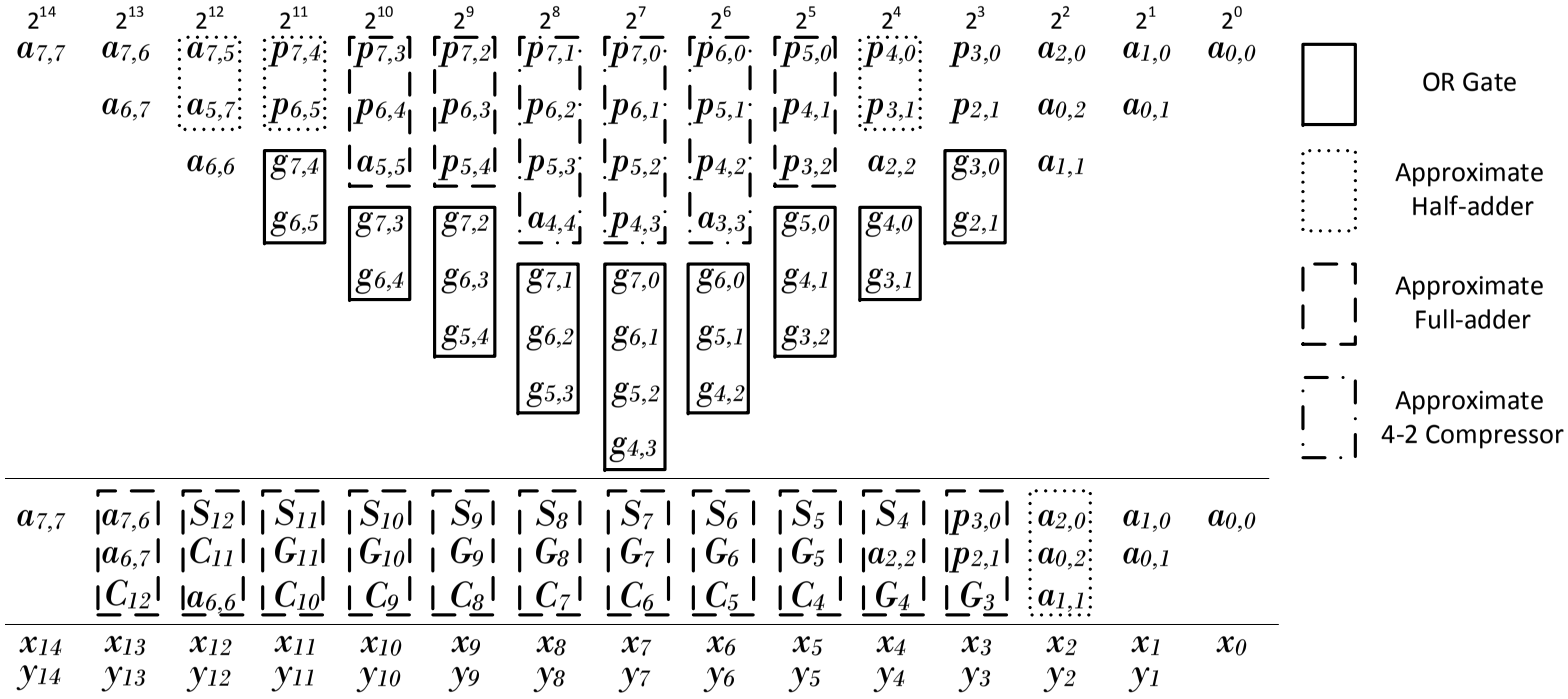


Figure 4.2: Reduction of altered partial products

4.1.1 Approximation of Altered Partial Products $g_{m,n}$

The accumulation of *generate* signals is done column wise. As each element has a probability of 1/16 of being one, two elements being 1 in the same column even decreases. Let pr be the probability of the element being 1, and $1 - pr$ be the probability of element being 0, where pr is 1/16.

In a column of 2 *generate* elements,

The probability of all elements being 0 is $P_{all0} = (1 - pr)^2$

The probability of one element being 1 is $P_{one1} = 2pr(1 - pr)$

The probability of all elements being 1 is $P_{all1} = pr^2$

In a column of 3 *generate* elements,

The probability of all elements being 0 is $P_{all0} = (1 - pr)^3$

The probability of one element being 1 is $P_{one1} = 3pr(1 - pr)^2$

The probability of two elements being 1 is $P_{two1} = 3pr^2(1 - pr)$

The probability of all elements being 1 is $P_{all1} = pr^3$

In a column with 4 *generate* signals,

The probability of all elements being 0 is $P_{all0} = (1 - pr)^4$

The probability of one element being 1 is $P_{one1} = 4pr(1 - pr)^3$

The probability of two elements being 1 is $P_{two1} = 6pr^2(1 - pr)^2$

The probability of three elements being 1 is $P_{three1} = 4pr^3(1 - pr)$

The probability of all elements being 1 is $P_{all1} = pr^4$

The probability statistics for number of *generate* elements m in each column are given in Table 4.1. The probability of error when OR gate is used for accumulation would be $P_{err} = 1 - (P_{all0} + P_{one1})$.

Based on Table 4.1, using OR gate in the accumulation of column wise *generate* elements in the altered partial product matrix provides exact result in most of the cases. The probability of error (P_{err}) while using OR gate for reduction of *generate* signals in each column

is also listed in Table 4.1. As can be seen, the probability of mis-prediction is very low. As the number of *generate* signals increases, the error probability increases linearly. However, the value of error also rises. To prevent this, the maximum number of *generate* signals to be grouped by OR gate is kept at 4. For a column having m *generate* signals, $\lceil m/4 \rceil$ OR gates are used.

4.1.2 Approximation of Other Partial Products

The accumulation of other partial products with probability $1/4$ for $a_{m,n}$ and $7/16$ for $p_{m,n}$ uses approximate circuits. Approximate half-adder, full-adder and 4-2 compressor are proposed for their accumulation. $Carry$ and Sum are two outputs of these approximate circuits. Since $Carry$ has higher weight of binary bit, error in $Carry$ bit will contribute more by producing error difference of two in the output. Approximation is handled in such a way that the absolute difference between actual output and approximate output is always maintained as one. Hence $Carry$ outputs are approximated only for the cases, where Sum is approximated.

In adders and compressors, XOR gates tend to contribute to high area and delay. For approximating half-adder, XOR gate of Sum is replaced with OR gate as given in equation 4.2. This results in one error in the Sum computation as seen in the truth table of approximate half-adder in Table 4.2. A tick mark denotes that approximate output matches with correct output and cross mark denotes mismatch.

$$\begin{aligned} Sum &= x1 + x2 \\ Carry &= x1 \cdot x2 \end{aligned} \tag{4.2}$$

In the approximation of full-adder, one of the two XOR gates is replaced with OR gate in Sum calculation. This results in error in last two cases out of 8 cases. $Carry$ is modified as in equation 4.3 introducing one error. This provides more simplification, while maintaining the difference between original and approximate value as one. An exact full adder and proposed approximate version are shown in figure 4.3.The truth table of approximate full-adder is given in Table 4.3.

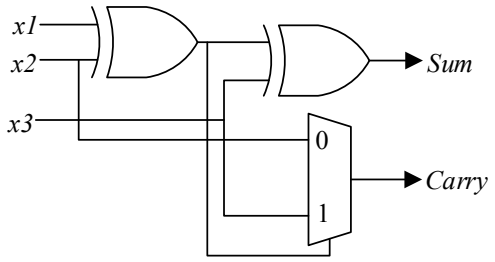
Table 4.2: Truth table of approximate half adder

Inputs		Exact Outputs		Approximate Outputs		Absolute Difference
$x1$	$x2$	$Carry$	Sum	$Carry$	Sum	
0	0	0	0	0 ✔	0 ✔	0
0	1	0	1	0 ✔	1 ✔	0
1	0	0	1	0 ✔	1 ✔	0
1	1	1	0	1 ✔	1 ✗	1

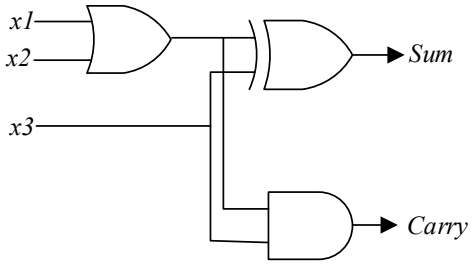
$$\begin{aligned} W &= (x1 + x2) \\ Sum &= W \oplus x3 \\ Carry &= W \cdot x3 \end{aligned} \tag{4.3}$$

Two approximate 4-2 compressors in [62] produce non-zero output even for the cases where all inputs are zero. This results in high error distance and high degree of precision loss especially in cases of zeros in all bits or in most significant parts of the reduction tree. The proposed 4-2 compressor overcomes this drawback.

In 4-2 compressor, three bits are required for the output only when all the four inputs are 1, which happens only once out of 16 cases. This property is taken to eliminate one of the three output bits in 4-2 compressor. To maintain minimal error difference as one, the output "100" (the value of 4) for four inputs being one has to be replaced with outputs "11" (the value of 3). For Sum computation, one out of three XOR gates is replaced with OR gate. Also, to make the Sum corresponding to the case where all inputs are ones as one, an additional circuit $x1 \cdot x2 \cdot x3 \cdot x4$ is added to the Sum expression. This results in error in five out of 16 cases. $Carry$ is simplified as in equation 4.4. The corresponding truth table is given in Table 4.4. An exact 4-2 compressor and proposed approximate version are shown in Figure 4.4.



(a)



(b)

Figure 4.3: Full adder (a) Exact version (b) Approximate version

$$\begin{aligned} W1 &= x1 \cdot x2 \\ W2 &= x3 \cdot x4 \\ Sum &= (x1 \oplus x2) + (x3 \oplus x4) + W1 \cdot W2 \\ Carry &= W1 + W2 \end{aligned} \tag{4.4}$$

Figure 4.2 shows the reduction of altered partial product matrix of 8×8 approximate multiplier. It requires two stages to produce sum and carry outputs for vector merge addition step. Four 2-input OR gates, four 3-input OR gates and one 4-input OR gates are required for the reduction of *generate* signals from columns 3 to 11. The resultant signals of OR gates are labeled as G_i corresponding to the column i with weight 2^i. For reducing other partial products, 3 approximate half-adders, 3 approximate full-adders and 3 approximate compressors are required in the first stage to produce Sum and $Carry$ signals, S_i and C_i

Table 4.3: Truth table of approximate full adder

Inputs			Exact Outputs		Approximate Outputs		Absolute Difference
$x1$	$x2$	$x3$	$Carry$	Sum	$Carry$	Sum	
0	0	0	0	0	0 ✔	0 ✔	0
0	0	1	0	1	0 ✔	1 ✔	0
0	1	0	0	1	0 ✔	1 ✔	0
0	1	1	1	0	1 ✔	0 ✔	0
1	0	0	0	1	0 ✔	1 ✔	0
1	0	1	1	0	1 ✔	0 ✔	0
1	1	0	1	0	0 ✗	1 ✗	1
1	1	1	1	1	1 ✔	0 ✗	1

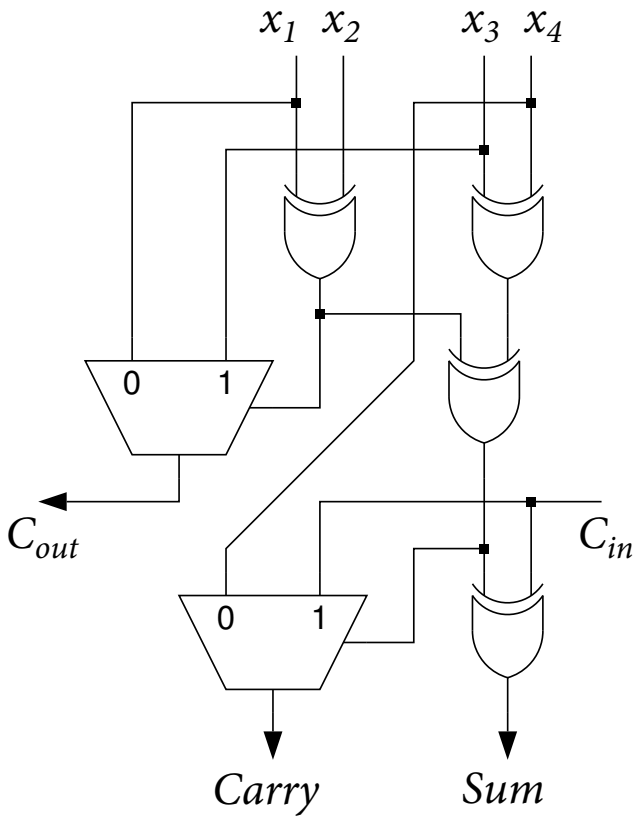
corresponding to column i. The elements in the second stage are reduced using 1 approximate half-adder and 11 approximate full-adders producing final two operands x_i and y_i to be fed to ripple carry adder for the final computation of the result.

4.2 Results and Discussion

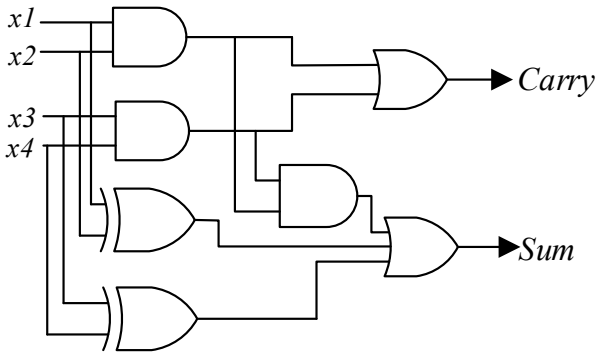
All approximate multipliers are designed for n = 16. The multipliers are described in Verilog HDL and synthesized using Synopsys compiler DC with a TSMC 65nm cell library under the typical process corner, with temperature 25°C and supply voltage 1V. From the Synopsys DC reports, we get area, delay, dynamic power and leakage power. Multiplier1 applies approximation in all columns, whereas in Multiplier2, approximation is applied in 15 least significant columns during partial product reduction. For the proposed multipliers, the altered partial

Table 4.4: Truth table of approximate 4-2 compressor

Inputs				Approximate outputs		Absolute Difference
$x1$	$x2$	$x3$	$x4$	$Carry$	Sum	
0	0	0	0	0 ✔	0 ✔	0
0	0	0	1	0 ✔	1 ✔	0
0	0	1	0	0 ✔	1 ✔	0
0	0	1	1	1 ✔	0 ✔	0
0	1	0	0	0 ✔	1 ✔	0
0	1	0	1	0 ✗	1 ✗	1
0	1	1	0	0 ✗	1 ✗	1
0	1	1	1	1 ✔	1 ✔	0
1	0	0	0	0 ✔	1 ✔	0
1	0	0	1	0 ✗	1 ✗	1
1	0	1	0	0 ✗	1 ✗	1
1	0	1	1	1 ✔	1 ✔	0
1	1	0	0	1 ✔	0 ✔	0
1	1	0	1	1 ✔	1 ✔	0
1	1	1	0	1 ✔	1 ✔	0
1	1	1	1	1 ✗	1 ✗	1



(a)



(b)

Figure 4.4: 4-2 compressor (a) Exact version (b) Approximate version

products are generated and compressed using half adder, full adder and 4-2 compressor structures to form final two rows of partial products. The efficiency of the proposed multipliers is compared with existing approximate multipliers [62, 64, 54, 51]. Inexact compressor design 2 of [62] is used to design compressor based multipliers- ACM1, where all columns are approximated and ACM2, where only 15 least significant columns are approximated. SSM [64] for m = 12 and n = 16 is designed for implementation. Partial product perforation design discussed in [54] for j = 2, k = 2 is designed and implemented under Dadda tree structure. In [51], the partial product matrix of 16-bit under designed multiplier (UDM) comprises approximate 2×2 partial products accumulated together with exact carry save adders. Exhaustive error analysis of the approximate multipliers is done using MATLAB.

Table 4.5: Synthesis results of exact, existing and proposed approximate multipliers

Multiplier Type	Area (μm^2)	Delay (ns)	Power (μW)	PDP (fJ)	APP $(\mu m^2 \cdot \mu W)(10^5)$
Exact	4859.28	0.68	1776.49	1208.01	86.32
Multiplier1	2158.56	0.47	503.15	236.48	10.86
Multiplier2	3319.20	0.66	1102.03	727.34	36.57
ACM1 [62]	2871.72	0.4	435.31	174.12	12.50
ACM2 [62]	3782.16	0.63	1250.70	787.94	47.30
SSM [64]	3953.88	0.69	1225.29	845.45	48.44
PPP [54]	4547.52	0.64	1570.79	1005.31	71.43
UDM [51]	3938.00	0.67	1318.51	883.40	51.92

Exact 16-bit multiplier is designed using Dadda tree structure. Table 6.2 compares all designs in terms of area, delay, power, power delay product (PDP) and area power product (APP). NMED and MRED of the approximate multipliers are listed in Table 7.2. If high approximation can be tolerated for saving more power, Multiplier1 and ACM1 are the candidates to be considered. It can be seen that Multiplier1 has better APP, whereas ACM1

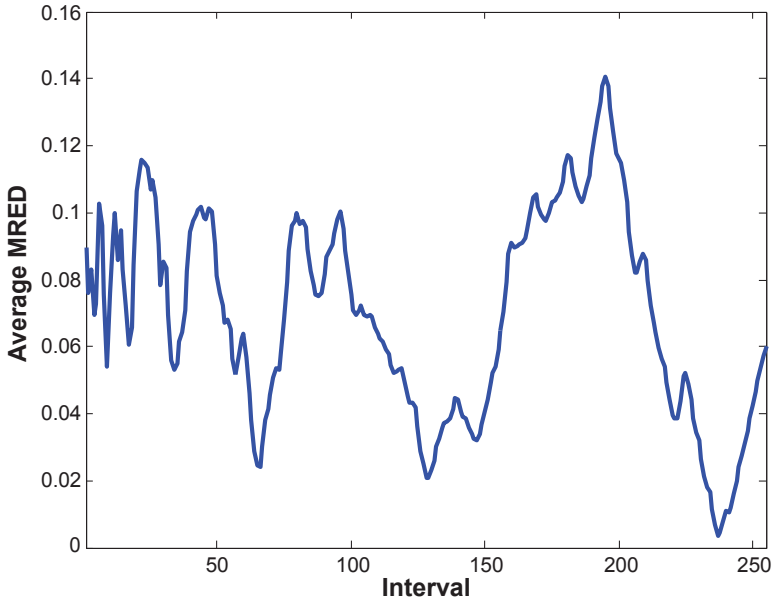
Table 4.6: Error metrics for 16-bit multipliers

Multiplier	MRED	NMED
Multiplier1	7.63×10^{-2}	1.78×10^{-2}
Multiplier2	2.44×10^{-4}	7.10×10^{-6}
ACM1 [62]	16.6	4.96×10^{-2}
ACM2 [62]	2.30×10^{-3}	6.36×10^{-6}
SSM [64]	6.34×10^{-4}	1.07×10^{-4}
PPP [54]	8.98×10^{-4}	4.58×10^{-5}
UDM [51]	3.32×10^{-2}	1.39×10^{-2}

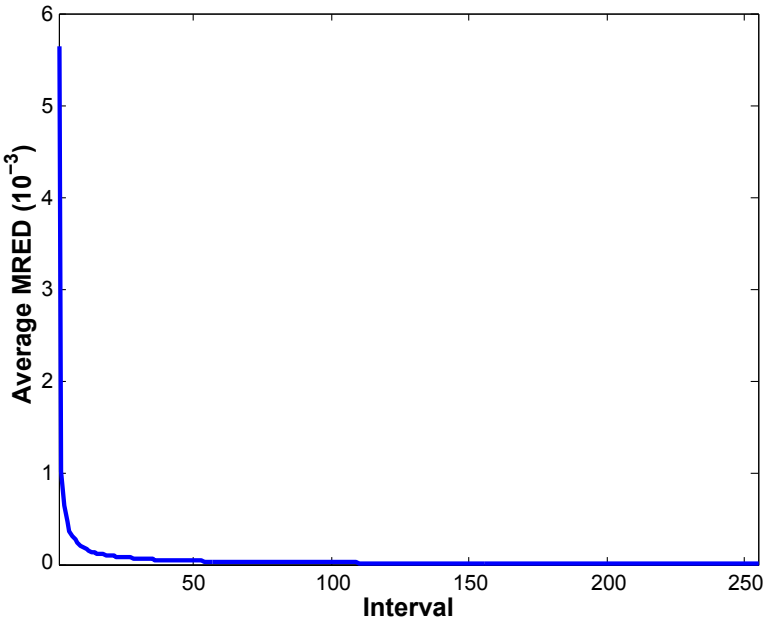
has better PDP. However, Multiplier1 has 64% lower NMED and three orders of magnitude lower MRED, compared to ACM1. It should be noted that high values of MRED for ACMs are due to non-zero output for inputs with all zeros.

Multiplier2 offers 32% area savings and 38% power savings, over the exact multiplier. ACM2 provides 22% and 30% area and power savings, respectively. SSM has 19% area and 31% power savings over accurate multiplier. Perforated multiplier has 6% and 12% area and power savings, respectively. UDM provides 19% and 26% area and power savings. Multiplier2 has one order of lower MRED than ACM2, two orders of lower MRED than UDM, 73% lower MRED than PPP, and 62% lower MRED than SSM. NMED of Multiplier2 outperforms all approximate multipliers except ACM2. ACM2 exhibit 10% lower NED than Multiplier2. Multiplier2 produces large error distance relative to ACM2. However, lower MRED indicates that Multiplier2 has smaller relative error values.

Table 4.7 gives a comprehensive comparison of approximate multipliers to get an idea of trade-off between design metrics and error metrics. Multiplier1 delivers the lowest APP, Multiplier2 delivers the lowest MRED value. Overall, Multiplier2 has better PDP, APP and MRED over ACM2, SSM, Perforated multiplier and UDM, with lower NMED in most cases as well. For applications where high power savings are desired with more error tolerance,



(a)



(b)

Figure 4.5: MRED distribution of (a) Multiplier1 (b) Multiplier2

Table 4.7: Ranking of approximate multipliers in terms of design and error metrics

Approximate Multiplier Type	APP Gain	PDP Gain	NMED	MRED
Multiplier1	1	2	6	6
Multiplier2	3	3	2	1
ACM1 [62]	2	1	7	7
ACM2 [62]	4	4	1	4
SSM [64]	5	5	4	2
PPP [54]	7	7	3	3
UDM [51]	6	6	5	5

Multiplier1 can be used. For moderate power savings with better performance, Multiplier2 is suggested.

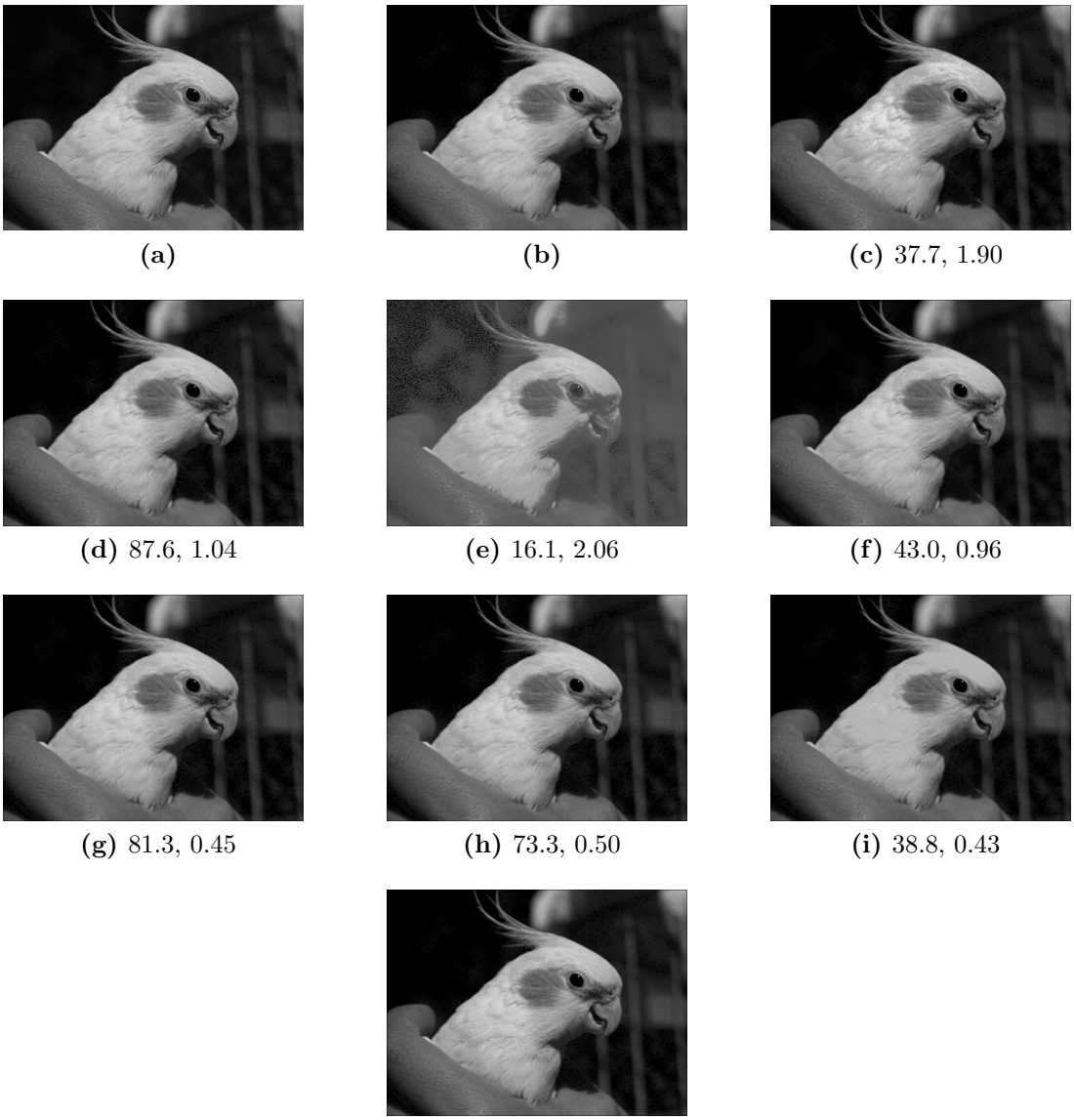
MRED distribution of 16-bit versions of Multiplier1 and Multiplier2 are shown in Figure 4.5. All possible outputs ranging from 0 to 65535^2 are categorized into 255 intervals. MRED of Multiplier2 is significantly low at higher product values, as exact units are used in most significant part of the multiplier.

4.3 Application- Noise Reduction

Geometric mean filter is widely used in image processing to reduce Gaussian noise [99]. When compared to arithmetic mean filter, geometric mean filter preserves more edge features. Two 16-bits per pixel gray scale images with Gaussian noise are considered. 3×3 mean filter is used, where each pixel of noisy image is replaced with geometric mean of 3×3 block of neighboring pixels centered around it. The algorithms are coded and implemented in MATLAB. Exact and approximate 16-bit multipliers are used to perform multiplication between 16-bit pixels.

PSNR is used as figure of merit to assess the quality of approximate multipliers. PSNR is based on mean square error (MSE) found between resulting image of exact multiplier and the images generated from approximate multipliers. Energy required by exact and approximate multiplication process while performing geometric mean filtering of the images are found using Synopsys Primetime. Further, exact multiplier is voltage scaled from 1 V to 0.85 V (VOS), and its impact in energy consumption and image quality is computed.

The noisy input image and resultant image after denoising using exact and approximate multipliers, with their respective PSNRs and energy savings in μJ are shown in Figure 4.6 and Figure 4.7, respectively. Energy required for exact multiplication process for image-1 and image-2 is $3.24\mu J$ and $2.62\mu J$ respectively. Although ACM1 has better energy savings compared to Multiplier1, Multiplier1 has significantly higher PSNR than ACM1. Multiplier2 shows the best PSNR among all the approximate designs. Multiplier2 has better energy savings, compared to ACM2, PPP, SSM, UDM and VOS. The intensity of image-1 being mostly on the lower end of the histogram causes poor performance of ACM multipliers. As the switching activity impacts most significant part of the design in VOS, PSNR values are affected.



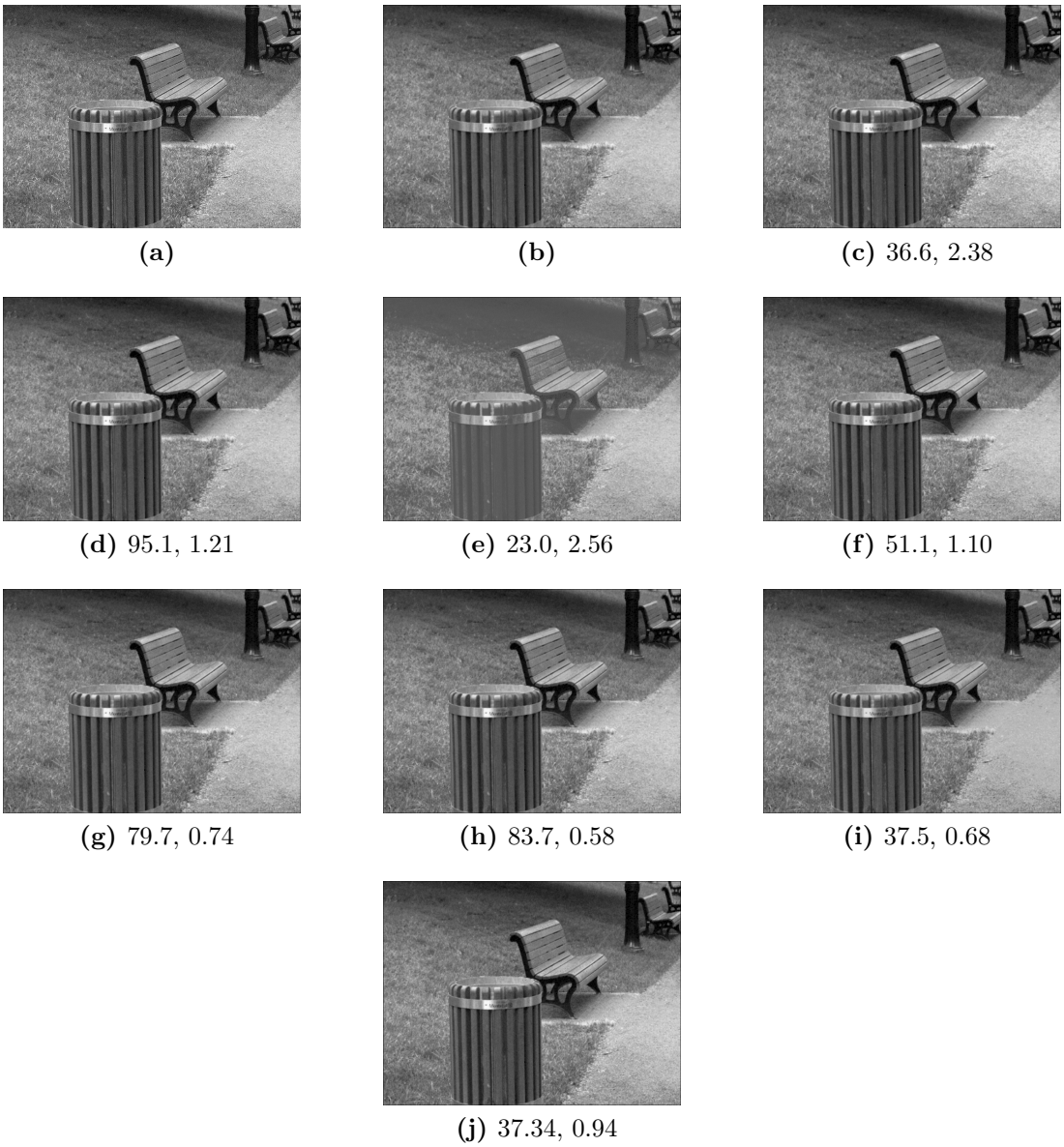
(a) **(b)** **(c)** 37.7, 1.90

(d) 87.6, 1.04 **(e)** 16.1, 2.06 **(f)** 43.0, 0.96

(g) 81.3, 0.45 **(h)** 73.3, 0.50 **(i)** 38.8, 0.43

(j) 38.05, 0.76

Figure 4.6: (a) Input image-1 with Gaussian noise. Geometric mean filtered images and corresponding PSNR and energy savings in μJ using (b) Exact multiplier (c) Multiplier1 (d) Multiplier2 (e) ACM1 (f) ACM2 (g) SSM (h) PPP (i) UDM (j) VOS



(a) **(b)** **(c)** 36.6, 2.38

(d) 95.1, 1.21 **(e)** 23.0, 2.56 **(f)** 51.1, 1.10

(g) 79.7, 0.74 **(h)** 83.7, 0.58 **(i)** 37.5, 0.68

(j) 37.34, 0.94

Figure 4.7: (a) Input image-2 with Gaussian noise. Geometric mean filtered images and corresponding PSNR and energy savings in μJ using (b) Exact multiplier (c) Multiplier1 (d) Multiplier2 (e) ACM1 (f) ACM2 (g) SSM (h) PPP (i) UDM (j) VOS

5 Design and Analysis of Approximate Booth Multipliers [1]

In this chapter, three approximate Booth multipliers models (ABM-M1, ABM-M2 and ABM-M3) are proposed. The ABM-M1 multiplier uses an approximate Booth partial product generator that produces error cases of 4 out of 32, resulting from the $\pm 2 \times$ multiplicand being replaced by $\pm 1 \times$ multiplicand. In ABM-M2, the same approximate Booth partial product generator in ABM-M1 is used, but the multiplicand input in consolidated and the set of partial products are replaced by single partial product in each row. In ABM-M3, a partial product generator based on the zero values of the encoded signal and multiplicand is proposed.

This work is an extension of our conference work [82]. The main improvements and novel contributions of this work include:

1. Error distance of the partial product generator in the ABM-M1 multipliers is discussed and analyzed using 16-bit multipliers.

2. ABM-M2 multipliers are introduced, where partial product generation and accumulation is further simplified based on a consolidated value of the multiplicand and replacing a set of partial product generators with a single partial product generator.

3. A partial product generator based on zero values of the multiplicand and encoded signal is proposed. The proposed partial product generator is used in ABM-M3 multipliers.

4. An approximation factor p is used to implement and analyze the design metrics and error metrics of all the proposed multipliers.

[1] Major part of this work has been published in "Power Efficient Approximate Booth Multiplier," *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, Italy, 2018, pp. 1-4. Authors: Suganthi Venkatachalam, H. J. Lee and Seok-Bum Ko.

The proposed techniques are compared with state-of-the-art approximate Booth multipliers. In each design, approximation factor p refers to the number of columns in the partial product matrix to which approximation is applied, in order of increasing significance. As p increases, a higher number of columns make use of the approximate partial product generator, and the inexactness of the multiplier increases. In all the proposed multipliers, the partial product accumulation is performed using a dadda tree structure composed of exact 4-2 compressors, full-adders, and half-adders. The exact, proposed, and existing approximate multipliers are evaluated with image multiplication and matrix multiplication applications.

The rest of the work is organized as follows. In section 5.1, Radix-4 Booth multipliers are explained. In section 5.2, three approximate designs ABM-M1, ABM-M2, and ABM-M3 are presented. In section 5.3, design and error metrics of proposed and approximate Booth multipliers are compared and analyzed. In section 5.4, approximate multipliers are used in image multiplication and matrix multiplication applications.

5.1 Radix-4 Booth multipliers

The output P_{out} from two signed inputs A and B, can be given as

$$\begin{aligned} A &= -a_{N-1}2^{N-1} + \sum_{n=0}^{N-2} a_n 2^n \\ B &= -b_{N-1}2^{N-1} + \sum_{n=0}^{N-2} b_n 2^n \\ P_{out} &= -P_{2N-1}2^{2N-1} + \sum_{n=0}^{2N-2} P_n 2^n \end{aligned} \tag{5.1}$$

The input B is grouped into bits $\{b_{2i+1}, b_{2i}, b_{2i-1}\}$ and the radix-4 Booth encoder encodes these three consecutive bits into three signals neg_i, two_i, and $zero_i$ as shown in Table 5.1. neg_i refers to the sign of each partial product operation, two_i indicates whether the generated partial product is to be shifted, and $zero_i$ is marked high if the partial product is zero. Based on the signals neg_i, two_i, and $zero_i$, the corresponding row-wise partial product PP_i is selected from $\{-2A, -1A, 0, +1A, +2A\}$.

Table 5.1: Recoding of multiplier bit groups and corresponding operation in exact radix-4 multiplier

b_{2i+1}	b_{2i}	b_{2i-1}	neg_i	two_i	$zero_i$	PP_i
0	0	0	0	0	1	0
0	0	1	0	0	0	$+A$
0	1	0	0	0	0	$+A$
0	1	1	0	1	0	$+2A$
1	0	0	1	1	0	$-2A$
1	0	1	1	0	0	$-1A$
1	1	0	1	0	0	$-1A$
1	1	1	0	0	1	0

5.2 Proposed Architectures

Booth multipliers are suitable candidates for approximation partial product accumulation in addition to partial product generation. An exact radix-4 partial product generator requires all three signals neg_i, two_i, and $zero_i$, to generate the partial product. In our work, a partial product generator is designed using only two of the three signals, namely neg_i and two_i. This partial product generator is used in proposed multipliers ABM-M1 and ABM-M2. In ABM-M3, a partial product generator is proposed which uses only the signal $zero_i$. In all three designs, approximation is applied in partial product accumulation in addition to partial product generation.

5.2.1 ABM-M1 Approximate Multipliers

The exact partial product generator is shown in Figure 5.1. The logic equation of the partial product p_{ij} outputted from the exact partial product generator is given by

$$\begin{aligned} m_i &= (a_j \cdot \overline{two_i}) + (a_{j-1} \cdot two_i) \\ p_{ij} &= \overline{zero_i} \cdot (neg_i \oplus m_i) \end{aligned} \tag{5.2}$$

where m_i is the output of the multiplexer and a_j is the multiplicand input. The partial product generator circuit is approximated by changing 4 of 32 entries in the corresponding k-map, as shown in Figure 5.2, where $\boxed{1}$ represents a change from '0' to '1' and $\boxed{0}$ represents a change from '1' to '0'. This results in an approximate partial product generator based on two signals, neg_i and $zero_i$, subsequently referred to as PPG-2S. The circuit schematic for this approximate partial product generator is shown in Figure 5.3 and can be expressed as

$$p_{ij} = \overline{a_j} \cdot neg_i + a_j \cdot \overline{neg_i} \cdot \overline{zero_i} \tag{5.3}$$

When compared to the exact partial product generator, the PPG-2S circuit does not require a multiplexer or an XOR-gate and the output can be expressed using only AND and OR-gates. The error distance between the exact partial product generator and PPG-2S

is given in Table 5.2. Since the two_i signal is absent in PPG-2S, the $+2A$ and $-2A$ cases are replaced with $+1A$ and $-1A$, respectively, which results in two cases with an error distance of ± 1.

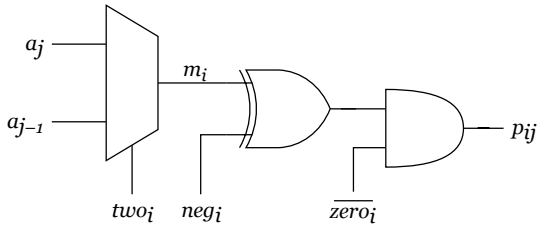


Figure 5.1: Circuit schematic for exact partial product generator

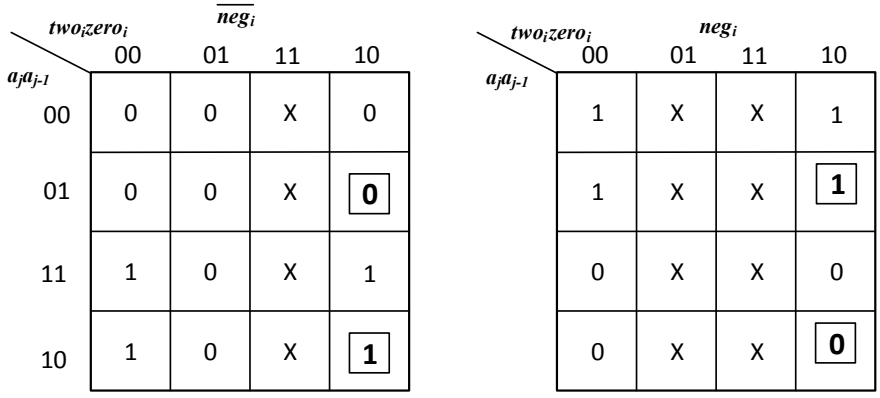


Figure 5.2: k-map of approximate partial product generator

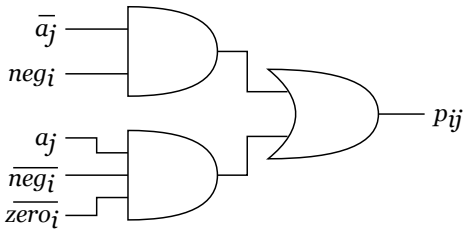


Figure 5.3: Circuit schematic for approximate two-signal partial product generator PPG-2S

The proposed approximate partial product generator PPG-2S is implemented in the ABM-M1 multipliers. For these multipliers, approximation factors $p = 12$, 14, and 16 are chosen. The partial product matrix for a 16-bit exact multiplier is shown in Figure 5.4, while the partial product matrix for a 16-bit ABM-M1 multiplier is shown in Figure 5.5. In the

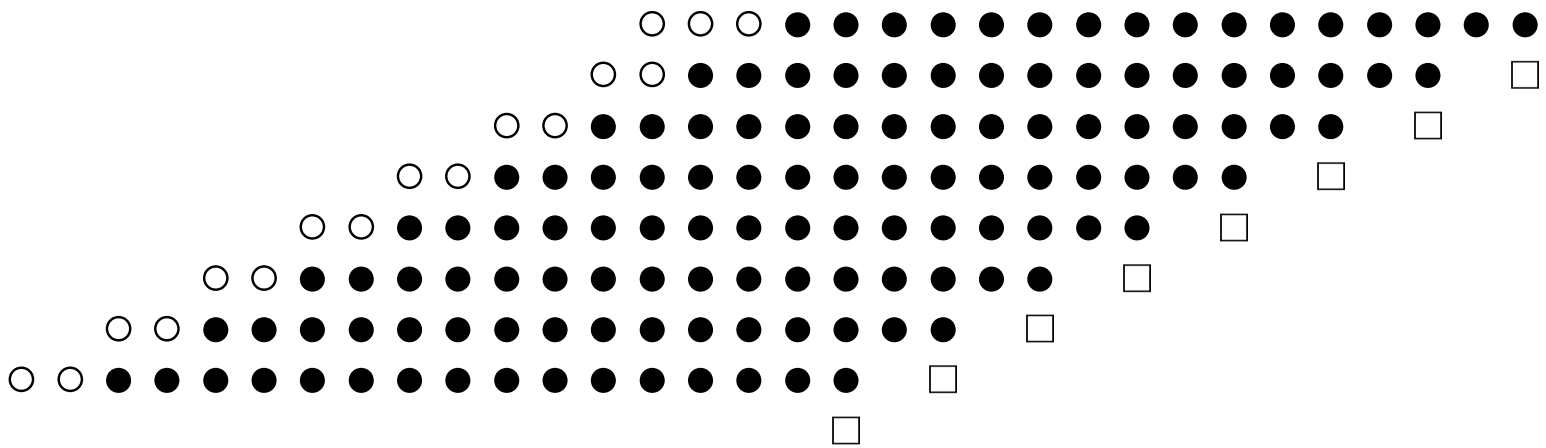


Figure 5.4: Partial product matrix of an exact radix-4 Booth multiplier (●: indicates a partial product, ○: indicates a sign-extension term, □: refers to a correction term).

Table 5.2: Error distance of proposed approximate partial product generator based on two signals

neg_i	two_i	$zero_i$	Exact pp_{ij} for $a_j a_{j-1}$				Exact OP	Approx. pp_{ij} for $a_j a_{j-1}$				Approx. OP	Error
			00	01	10	11		00	01	10	11		
0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	$+1A$	0	0	1	1	$+1A$	0
0	0	0	0	0	1	1	$+1A$	0	0	1	1	$+1A$	0
0	1	0	0	1	0	1	$+2A$	0	0	1	1	$+1A$	1
1	1	0	1	0	1	0	$-2A$	1	1	0	0	$-1A$	-1
1	0	0	1	1	0	0	$-1A$	1	1	0	0	$-1A$	0
1	0	0	1	1	0	0	$-1A$	1	1	0	0	$-1A$	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0

ABM-M1 design, all partial products with a significance less than p are generated using the approximate PPG-2S circuit and all remaining partial products are generated using the exact circuit. To reduce the height of the matrix, each correction term is combined with a partial product in its respective column using an OR-gate. Further, PPG-2S can be used in higher radix approximation. For radix-16 operation, AND operation of two $zero$ signals of two consective rows is performed to find combined $zero$ signal. It is then fed to PPG-2S and a partial product matrix as shown in Figure 5.6 is generated.

5.2.2 ABM-M2 Approximate Multipliers

The ABM-M2 multipliers differ from ABM-M1 in that the set of exact partial product generators in each row i of the partial product matrix is replaced with a single PPG-2S. Three approximation factors $p = 6$, 8, and 10 are chosen. The least-significant p bits of input A are added and a value a_{sum} is found. Based on a_{sum}, a value $a_{\forall j \in (0,p-1)}$ is found as given in equation 5.4. Partial product $p_{i,\forall j \in (0,p-1)}$ for each row is found by inputting $a_{\forall j \in (0,p-1)}$, and neg_i and $zero_i$ into PPG-2S. Small approximation factors were chosen because the approximation mechanism in ABM-M2 is more drastic, meaning that smaller values of p are required to generate reasonable error metrics.

$$a_{sum} = \sum_{j=0}^{p-1} a_j$$
$$a_{\forall j \in (0,p-1)} = \begin{cases} 1, & a_{sum} > p/2 \\ 0, & a_{sum} \leq p/2 \end{cases} \tag{5.4}$$

An example with approximation factor $p = 8$ is illustrated in Figure 5.7, which depicts the transformation of the exact partial product matrix to an approximate partial product matrix. For $p = 8$, the least 8 bits of input A are added and a value a_{sum} is found. From a_{sum}, a value from $a_{\forall j \in (0,7)}$ is found as per equation 5.4, which is then used to generate the approximate partial product $p_{i,\forall j \in (0,7)}$ using neg_i and $zero_i$ in the row i.

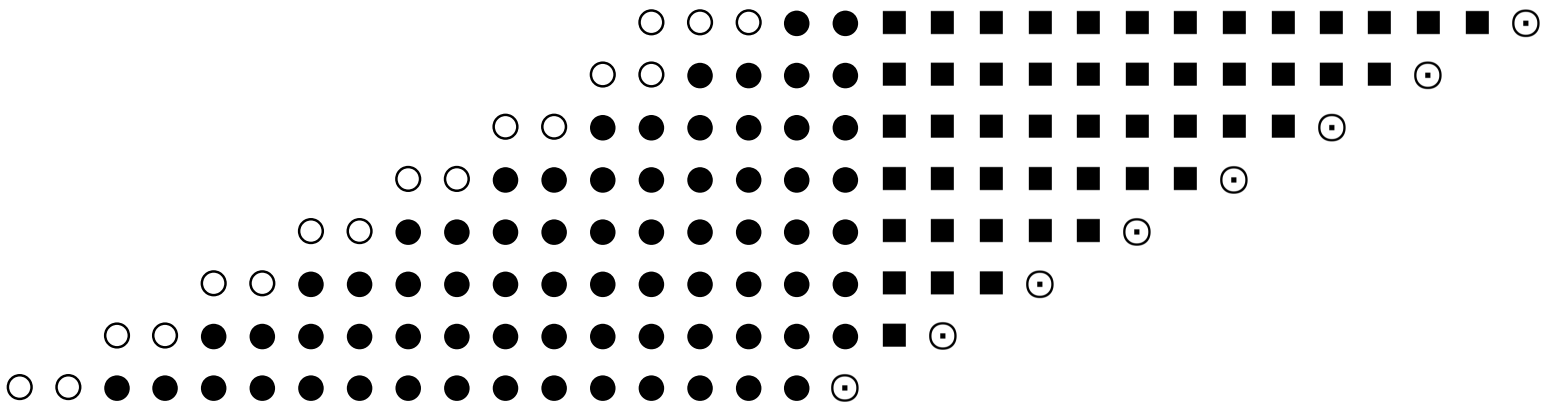


Figure 5.5: Partial product matrix of a 16-bit ABM-M1 multiplier with $p = 14$ (●: a partial product, ○: a sign-extension term, ■: an approximate partial product generated with PPG-2S, ⊙: term resulting from OR-ing the least-significant bit of a partial product with its correction term).

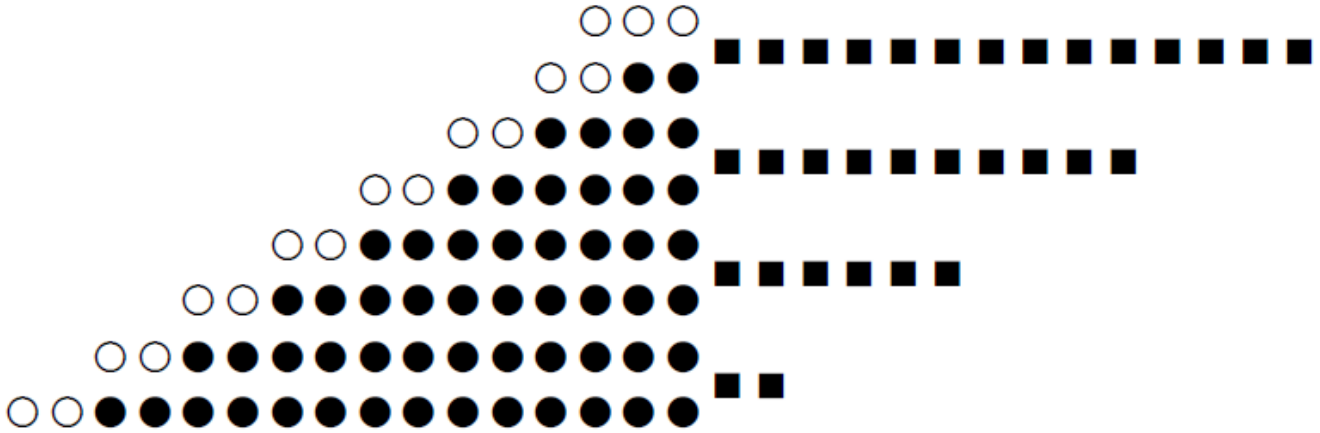


Figure 5.6: Partial product matrix of a 16-bit ABM-M1 multiplier with p = 16 for radix-16 operation (●: a partial product, ○: a sign-extension term, ■: an approximate partial product generated with PPG-2S).

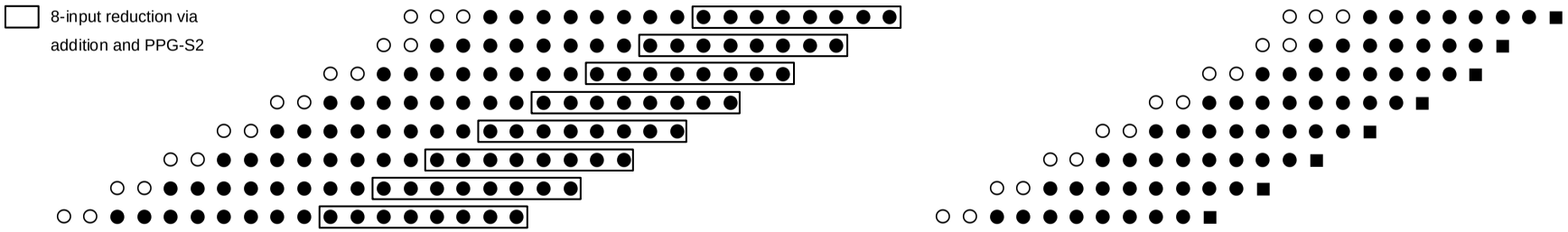


Figure 5.7: Partial product matrix of a 16-bit ABM-M2 multiplier with p = 8. The width of the matrix is reduced by adding the p least significant bits of each partial product, comparing the result to p, and then using the resulting 1-bit or 0-bit as an input to PPG-2S (●: a partial product, ○: a sign-extension term, ■: approximate partial product generated using PPG-2S).

5.2.3 ABM-M3 Approximate Multipliers

In the ABM-M3 multiplier design, an approximation is proposed based on the zero-values of input A and signal $zero_i$. Three approximation factors p = 12, 14, and 16 are chosen. For approximation factor p, all partial products with significance less than p are reduced to a single approximate partial product. Considering the exact partial product matrix in Figure 5.4, for a row i, let l be the number of bits with a significance less than p. For a row i, $a_{\forall j \in (0,p-(2i+1))}$ is generated by OR-ing the l least-significant bits of A. The approximate partial product for the row i is then generated by the use of PPG-1S as shown in Figure 5.8. PPG-1S takes in the result of the OR operation and the single signal $zero_i$ to produce the approximate partial product for that row.

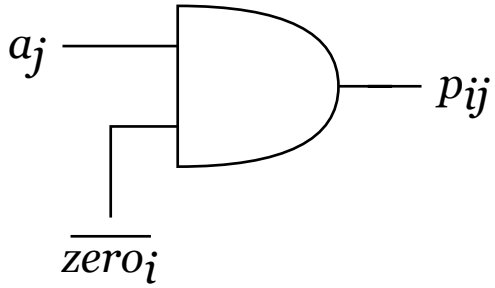


Figure 5.8: Circuit schematic for approximate single-signal partial product generator PPG-1S

An example with p = 14 is given in Figure 5.9. For the fourth row with i = 3, a ranging from 0 to $p-7$ are given to the $p-6$ input OR-gate. The output of this OR-gate $a_{\forall j \in (0,p-7)}$ and the signal $\overline{zero_3}$ are given to the AND-gate. Based on these values, the partial product $p_{i,\forall j \in (0,p-7)}$ is found. Similarly, for the first row, input a from 0 to $p-1$ and $zero_0$ are considered. In the second row, the value of the first row, input a from 0 to $p-3$, and $zero_1$ are considered. In the third row, input a from 0 to $p-5$ and $zero_2$ are considered, and so on.

5.3 Results and Discussion

The proposed approximate multipliers (i.e. ABM-M1, ABM-M2, and ABM-M3) and existing approximate multipliers are implemented in Verilog HDL and are verified using the ModelSim

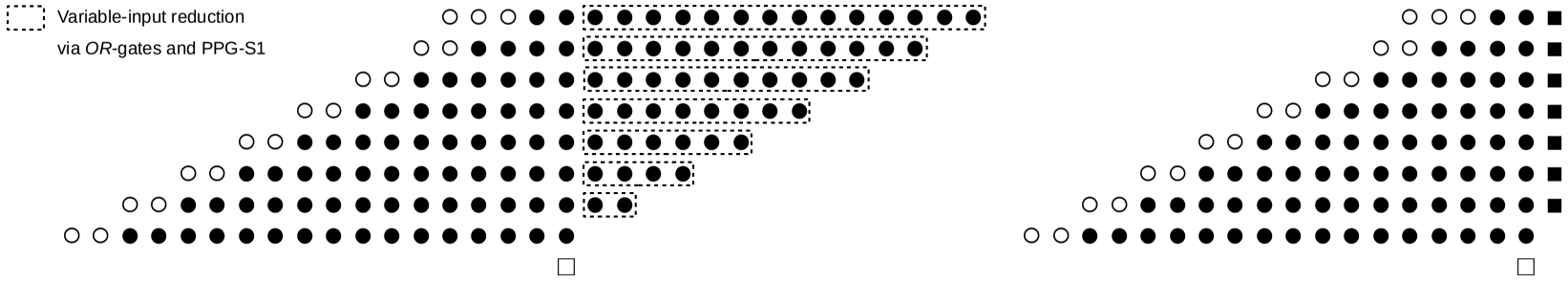


Figure 5.9: Partial product matrix of a 16-bit ABM-M3 multiplier with p = 14. The width of the matrix is reduced by OR-ing together for each partial product all bits with a significance less than p and then using the result of the OR operation as an input into PPG-1S (●: a partial product, ○: a sign-extension term, □: a correction bit, ■: approximate partial product generated using PPG-1S).

HDL simulator. The multipliers are then synthesized with the Synopsys compiler DC using the TSMC 65 nm library at typical process corner. The largest critical path delay of all designs, i.e. exact radix-8 multiplier was measured to be 1.19 ns and that delay was then used as the timing constraint for all following simulations.

The error characteristics of the proposed multipliers are reported in Table 7.2, along with the error metrics of other contemporary multipliers used as comparison designs. Area, power and area power product (APP) metrics of both the proposed multipliers and the comparison multipliers are reported in Table 7.3.

Comparison multipliers include the ABM1 and ABM2-C9 designs proposed by Jian et al. in [78], as well as the R4ABM1 multiplier proposed by Liu et al. in [56]. ABM1 and ABM2-C9 of [78] utilizes an approximate adder with error recovery and error compensation modules. In R4ABM1 design of [56], the input groups of multiplier are used in exact and approximate partial product generation and to reduce the logic complexity of exact partial product generator, $\pm 2 \times$ multiplicand is replaced by zero.

5.3.1 Error Analysis

A SystemVerilog testbench and a Python script were used to compare the output of the proposed multipliers with the result of the exact multiplication operation for one million randomly-generated pairs of inputs. MRED and NMED were calculated for all multipliers as summarized in Table 7.2.

The ABM-M1 and ABM-M3 designs exhibit the most competitive error characteristics. When comparing the proposed designs to R4ABM1 for each value of p, the NMED values of both ABM-M1 and ABM-M3 are significantly lower and the MRED values are less than half of those measured for R4ABM1. The error values of ABM-M1 for p = 14 are competitive with ABM1, in which MRED values are similar but NMED of ABM-M1 is an order of magnitude less than that of ABM1. All three designs of ABM-M1 achieve better error characteristics than ABM2-C9. ABM-M3 exhibits error metrics competitive with ABM1 for p = 12 and p = 14, and all three versions of ABM-M3 exhibit lower NMED than ABM2-C9. NMED values of ABM-M2 are comparatively large, but MRED of ABM-M2 for p = 6 is within the same order of magnitude as R4ABM1 for p = 14 and p = 16. When partial product

Table 5.3: MRED and NMED values of proposed and existing approximate multipliers

Approximate Design (16-bit)	p	MRED (10^{-2})	NMED (10^{-5})
ABM-M1	12	0.016	0.549
	14	0.030	0.766
	16	0.079	2.080
ABM-M2	6	0.663	26.611
	8	2.689	108.727
	10	10.723	441.429
ABM-M3	12	0.020	0.200
	14	0.082	0.853
	16	0.340	3.615
R4ABM1 [56]	12	0.038	0.843
	14	0.124	1.844
	16	0.473	6.419
ABM1 [78]	–	0.040	1.936
ABM2-C9 [78]	–	0.089	4.450

Table 5.4: Area, power, and area-power product values of proposed and existing approximate multipliers

Design (16-bit)	p	Area (μm^2)	Power (mW)	APP ($\mu m^2 \cdot$mW)
Exact radix-4	–	3904	1.309	5110
Exact radix-8	–	4401	1.369	6025
ABM-M1	12	3375	1.183	3993
	14	3371	1.162	3917
	16	3129	1.108	3467
ABM-M2	6	2379	0.835	1986
	8	2040	0.700	1428
	10	1596	0.529	844
ABM-M3	12	2625	0.941	2470
	14	2262	0.805	1821
	16	1848	0.649	1199
R4ABM1 [56]	12	3822	1.217	4651
	14	3610	1.148	4144
	16	3466	1.106	3833
ABM1 [78]	–	3399	1.149	3905
ABM2-C9 [78]	–	3025	1.083	3276

accumulation of the multiplier is performed with approximate adders and compressors for p=16, it has MRED and NMED values similar to ABM-M3 of p=16.

5.3.2 Hardware Measures

The three proposed designs are based off the radix-4 Booth multiplication algorithm. All three proposed designs exhibit significant area and power savings over the exact radix-4 multiplier. The proposed ABM-M1 designs allow for APP savings in the range of 22% to 32%, and the ABM-M3 designs provide APP savings in the range of 52% to 77%. ABM-M2 exhibits the most substantial APP reduction, with p values of 12, 14, and 16 corresponding to APP savings of 61%, 72%, and 83%, respectively.

The proposed designs also exhibit significant power and area savings over the R4ABM1 designs. As described in the error analysis, ABM-M1 and ABM-M3 compete best with the R4ABM1 design. ABM-M1 designs exhibit APP savings in the range of 5% to 14% and ABM-M3 in the range of 52% to 77% when compared to R4ABM1. The ABM1 and ABM2-C9 designs are based off the radix-8 Booth multiplication algorithm. Even when compared to the larger APP of the exact radix-8 multiplier, the APP reductions of ABM-M1 and ABM-M3 described previously are substantially larger than the ABM1 savings of 35% and ABM2-C9 savings of 36%. When partial product accumulation of the multiplier is performed with approximate adders and compressors for p=16, it has a APP saving of 61% when compared to radix-8 multiplier.

5.4 Applications

In this section, the proposed and existing approximate multipliers are tested with two applications—image multiplication and matrix multiplication.

5.4.1 Image Multiplication

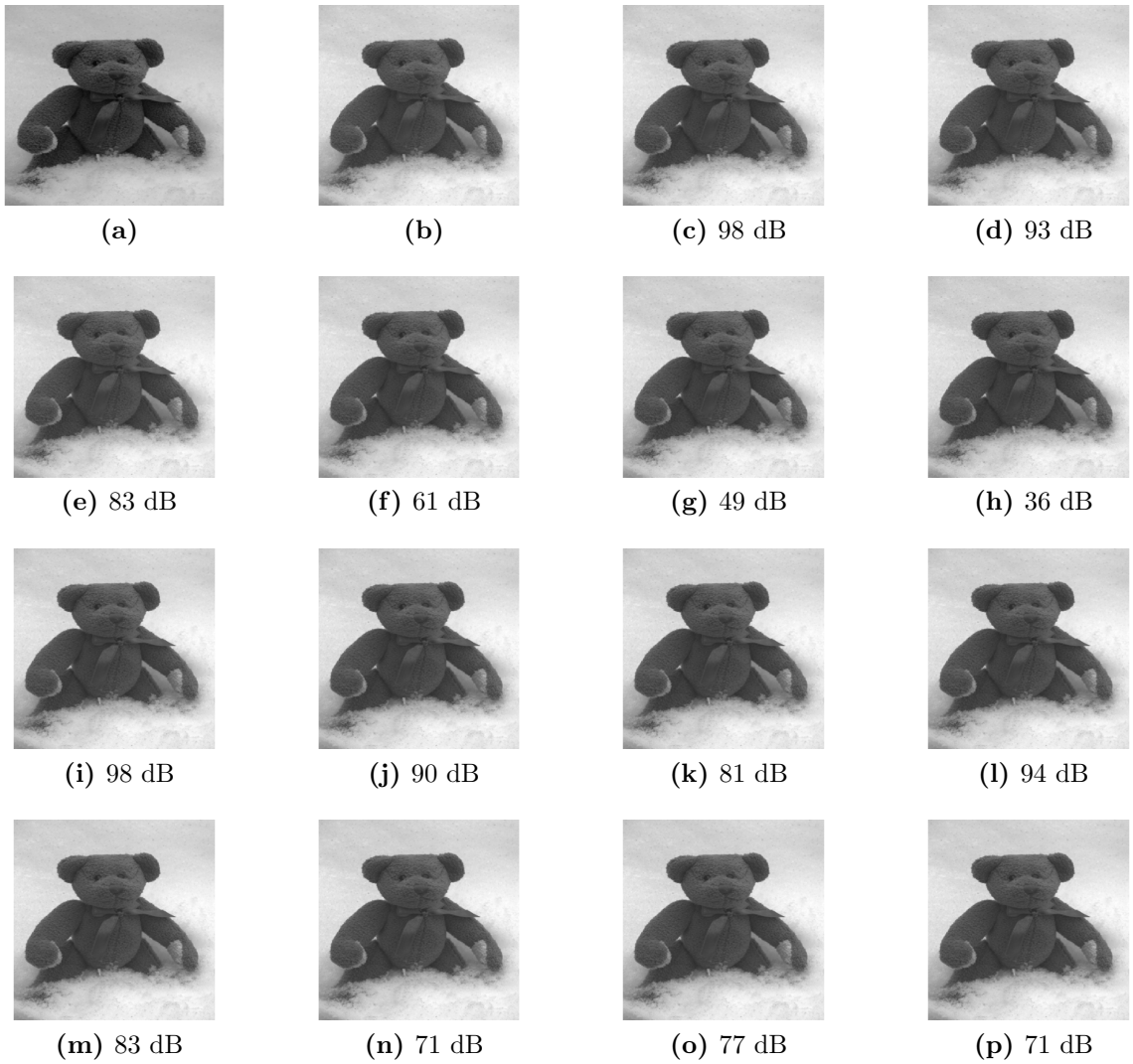
The discussed approximate Booth multipliers are applied to image multiplication. A 16-bit image is taken and its pixel values are shifted from the range $[0, 65535]$ to $[-32768, 32767]$.

The image is then multiplied by a constant on a pixel-by-pixel basis in order to brighten the image. The exact Booth multiplier, existing approximate Booth multipliers, and the proposed multipliers are used in this application by the use of MATLAB. To compare the image quality of exact and approximate Booth multiplier units, PSNR is used as a performance metric.

The resulting images of exact, proposed and existing multipliers and their corresponding PSNR values are shown in Figure 5.10. For all the models, the image quality deteriorates with increasing approximation factor. The results of this application show that ABM-M1 and ABM-M3 with approximation factors $p = 12$, 14, and 16 outperform R4ABM1 with equivalent approximation factors, ABM1, and ABM2-C9. Similarly, ABM-M2 exhibits better PSNR values for approximation factors $p = 6$ and 8. ABM-M1 with $p = 12$, 14, and 16, ABM-M3 with $p = 12$, 14, and 16, and ABM-M2 with $p = 6$ are suitable candidates for image processing applications which require negligible loss in image quality.

5.4.2 Matrix Multiplication

Matrix multiplication is a basic computing operation used in applications such as deep learning, convolution, and transforms in image and video processing, graphics, and robotic applications. Matrix multiplication is one of the most important kernel operations in these applications. In this work, 2×2 matrices are chosen for matrix multiplication. The existing and proposed multipliers are tested by taking 50,000 matrix multiplication test cases. MRED is taken as the error metric to assess the quality of approximate multipliers. MRED is the result of absolute difference between matrix multiplication results of approximate multiplier and exact multiplier, scaled by actual value of exact matrix multiplication result. The MRED values are listed in Table 5.5. It can be seen that the proposed multipliers perform better than existing state-of-the-art approximate Booth multipliers. The findings are consistent with MRED values discussed in Table 7.2.



(a) **(b)** **(c)** 98 dB **(d)** 93 dB

(e) 83 dB **(f)** 61 dB **(g)** 49 dB **(h)** 36 dB

(i) 98 dB **(j)** 90 dB **(k)** 81 dB **(l)** 94 dB

(m) 83 dB **(n)** 71 dB **(o)** 77 dB **(p)** 71 dB

Figure 5.10: (a) Input image. Images after image multiplication using (b) exact multiplier, the proposed multipliers with their PSNR values in dB (c) ABM-M1 (p=12) (d) ABM-M1 (p=14) (e) ABM-M1 (p=16) (f) ABM-M2 (p=6) (g) ABM-M2 (p=8) (h) ABM-M2 (p=10) (i) ABM-M3 (p=12) (j) ABM-M3 (p=14) (k) ABM-M3 (p=16), the existing comparison multipliers (l) R4ABM1 [56] (p=12) (m) R4ABM1 [56] (p=14) (n) R4ABM1 [56] (p=16) (o) ABM1 [78] (p) ABM-C9 [78]

Table 5.5: MRED values of proposed and existing approximate multipliers used in matrix multiplication application

Approximate Design (16-bit)	p	MRED (10^{-2})
ABM-M1	12	0.028
	14	0.034
	16	0.104
ABM-M2	6	1.830
	8	9.540
	10	21.800
ABM-M3	12	0.018
	14	0.048
	16	0.317
R4ABM1 [56]	12	0.058
	14	0.148
	16	0.514
ABM1 [78]	–	0.062
ABM2-C9 [78]	–	0.153

6 Approximate Sum of Products Designs Based on Distributed Arithmetic [1]

In this chapter, approximate sum of product units based on distributed arithmetic are proposed. Novel approximate sum of products designs are proposed using efficient distributed arithmetic structure. Approximation involves changes with respect to word length, number of lookup tables, and number of elements in the final accumulator. Three models are proposed. First model provides significant power reduction with lower MRED and NMED. Second and third models with increased area and power compared to first model provides better accuracy. In the proposed approximate structures, reductions in number of lookup tables, length of adders and accumulator size are employed for approximation. Compared to the exact sum of products unit, the proposed models have reduced circuit complexity.

Section 6.1 describes distributic arithmetic based sum of products computation. In section 6.2, three architectures are proposed. Section 6.3, approximate architectures are comparted with state of the art existing architectures. In section 6.4, one application on smoothing and one machine learning application - K-means clustering for color image compression are discussed.

[1] Major part of this work has been published in "Approximate Sum-of-Products Designs Based on Distributed Arithmetic," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 8, pp. 1604-1608, Aug. 2018. Authors: Suganthi Venkatachalam and Seok-Bum Ko.

6.1 Sum of Products Units Based on Distributed Arithmetic

Distributed arithmetic is a popular technique for implementing sum of products computations without the use of multipliers. Sum of products units based on distributed arithmetic are frequently used in filters and other DSP applications. The main advantage of distributed arithmetic is its high computational efficiency. Distributed arithmetic distributes multiply and accumulate operations across adders, lookup tables and final accumulation in such a way that conventional multipliers are not required.

Consider unsigned K elements of N-bits $a_1, a_2, \ldots, a_K$ and $b_1, b_2, \ldots, b_K$. To implement sum of products $a_1b_1 + a_2b_2 + \cdots + a_Kb_K$, the result y can be defined as

$$y = \sum_{n=0}^{N-1} x_n 2^n \tag{6.1}$$

where $x_n = \sum_{k=1}^{K} a_k b_{kn}$ represents summation of inputs $a_1, a_2, \ldots, a_K$ based on the elements $b_{1n}, b_{2n}, \ldots, b_{Kn}$ for bit position n. For example, for k=1, 2, 3 (three elements of inputs a and b) for bit position n, selection of summation of a inputs based on b is shown in Table 6.1. In Table 6.1, a_{ij} represents $a_i + a_j$. To implement this table, a lookup table can be used. All possible 2^K combinations of inputs a are stored in a lookup table. For a bit width of N, N lookup tables are required. To implement sum of products using distributed arithmetic, initially N-bit adders are required to perform $\sum_{k=1}^{K} a_k b_{kn}$. Then N lookup tables are required. The contents are chosen based on b_{kn} for n=$0, 1, 2, \ldots, N-1$. All contents are placed in final accumulator structure. The final accumulator stage has N elements. Hence sum of products structure based on distributed arithmetic requires adders to perform summation of a_k, lookup tables and a final accumulator. A general lookup table and its structure of sum of products based on distributed arithmetic for K=3 and N=16 is shown in Figure 6.1. Approximation in [85] involves changing the limits of equation 6.1 from m to N-1, where m is the number of lookup tables to be truncated.

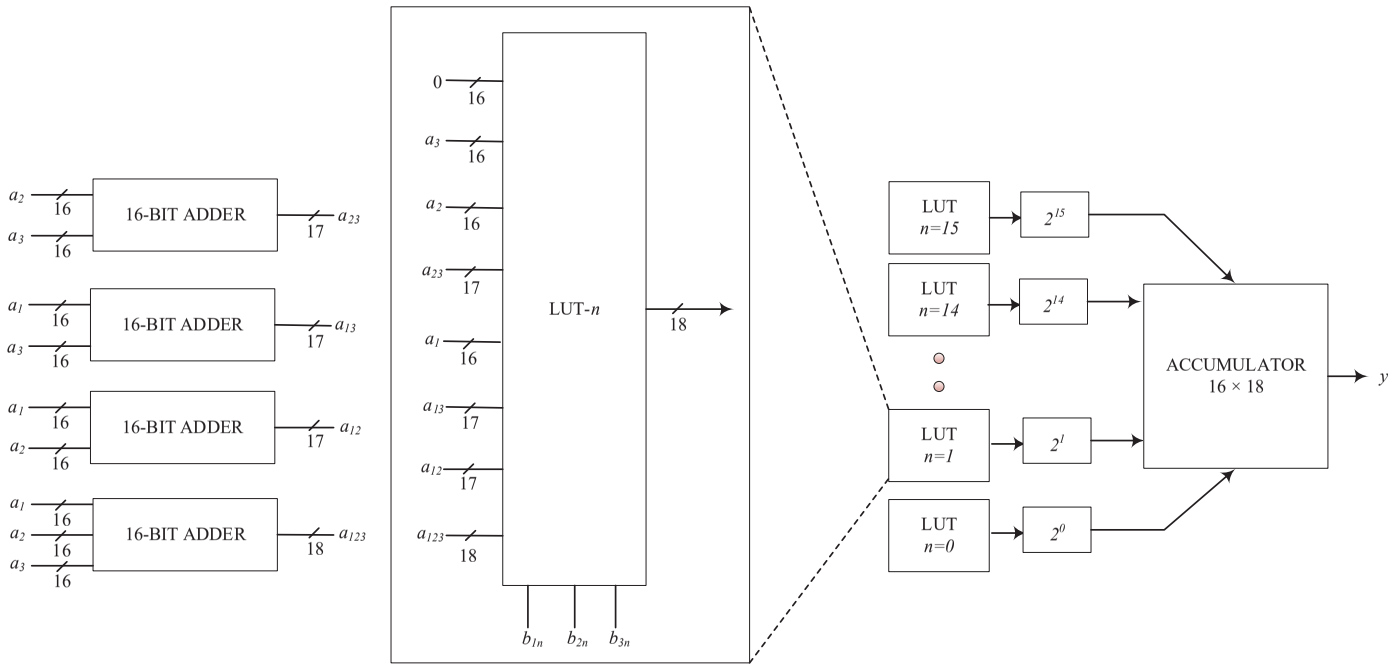


Figure 6.1: Lookup table and corresponding exact sum of products structure for K=3 and N=16

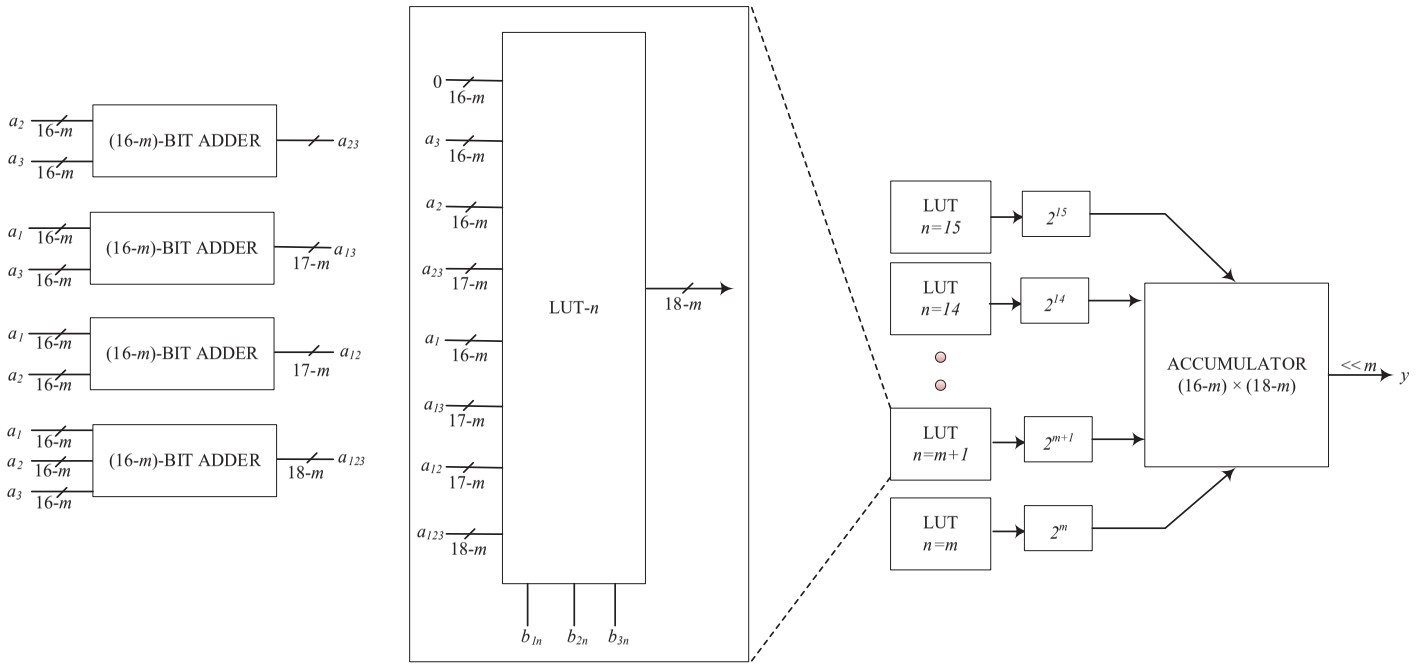


Figure 6.2: Approximate lookup table and corresponding approximate sum of products (ASOP1) structure for K=3 and N=16

Table 6.1: Lookup table contents of sum of products for K=3

b_{1n}	b_{2n}	b_{3n}	Contents
0	0	0	0
0	0	1	a_3
0	1	0	a_2
0	1	1	a_{23}
1	0	0	a_1
1	0	1	a_{13}
1	1	0	a_{12}
1	1	1	a_{123}

6.2 Proposed Approximate Sum of Products Architectures

In our work, K is 3 and N is 16. For conventional implementation of sum of products unit based on parallel distributed arithmetic [84], three two-input 16 bit adders, one three-input 16 bit adder, 16 lookup tables with 8 cases, and final accumulator with 16 elements are required. In our approximation models, hardware requirements are considerably reduced. Three models of approximate sum of products (ASOP) - ASOP1, ASOP2 and ASOP3 are proposed.

6.2.1 Proposed Approximate Sum of Products Model ASOP1

In approximate model 1, K is 3 and N is reduced. m bits at the least significant part of a_k and b_k for k=1, 2, 3 are truncated. m=8, 6, and 4 bits are implemented. For this implementation, three two-input 16-m bit adders, one three-input 16-m bit adder, 16-m lookup tables with 8 cases, and final accumulator with 16-m elements are required. This considerably reduces the hardware utilization at all the levels. The approximate model with reduced elements is

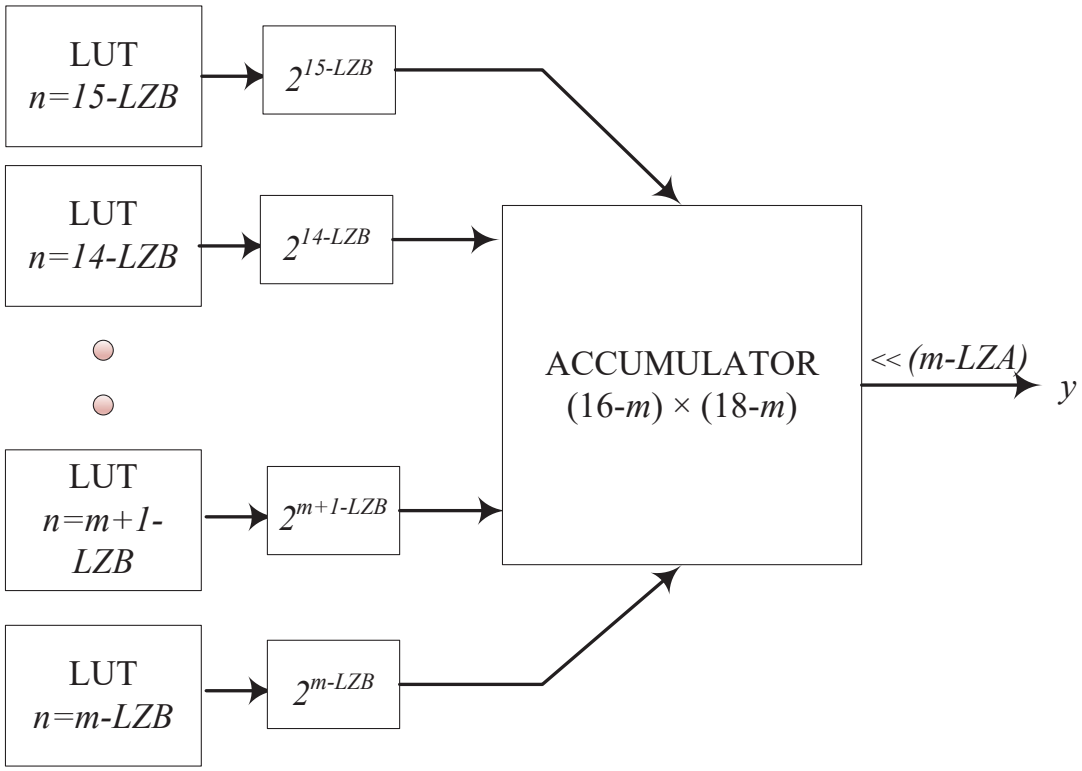


Figure 6.3: Approximate lookup table and corresponding approximate sum of products (ASOP2) structure for K=3 and N=16

shown in Figure 6.2. In [85], by implementing equation 6.1 with limits m to N-1, the number of look up tables reduces to 16-m and 16-m elements are sent to the final accumulator (16-m $\times$ 18). It should be noted that in ASOP1, the number of input bits to the adders is reduced, which further reduces complexity of accumulator (16-m $\times$ 18-m), compared to [85].

6.2.2 Proposed Approximate Sum of Products Model ASOP2

ASOP2 is similar to ASOP1 with the addition of m-bit leading one predictor. This increases the accuracy, and more suitable for DSP application which will be discussed later in this section. In our method, leading one prediction of a_k and b_k for k=1, 2, 3 requires OR

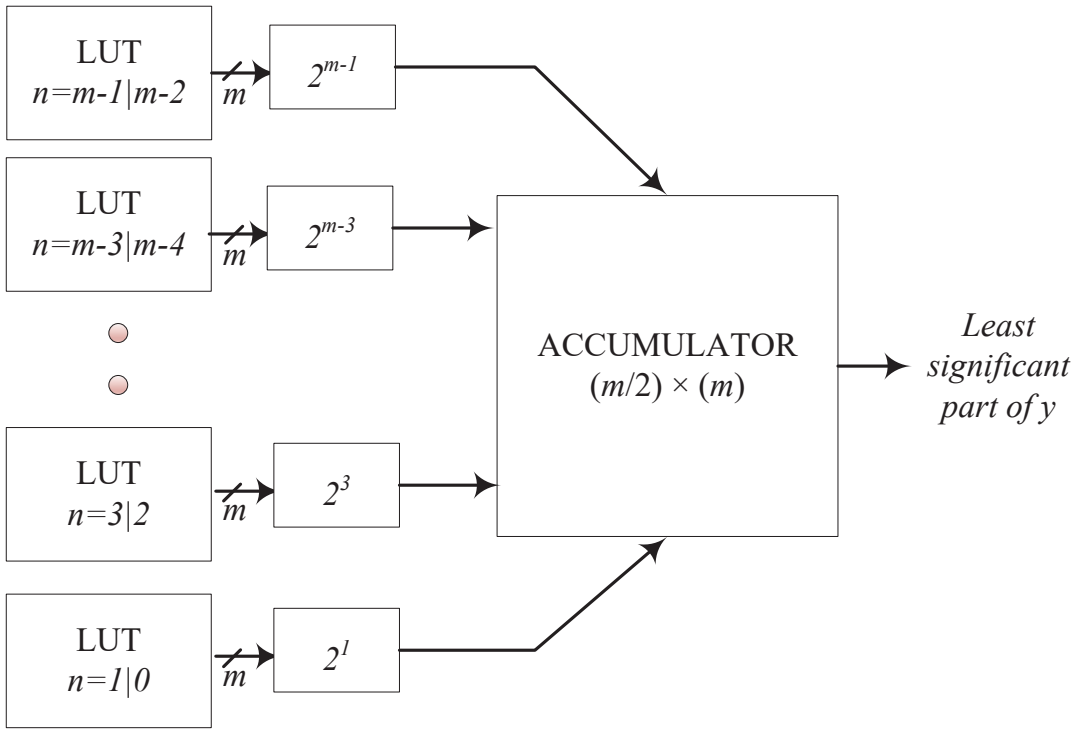


Figure 6.4: Least significant part of the approximate sum of products (ASOP3) structure

operation of most significant m bits of a_k and b_k for k=1, 2, 3 followed by priority encoder. The function of OR gates can be given as $a_{m_{or}} = a_{1m}|a_{2m}|a_{3m}$ and $b_{m_{or}} = b_{1m}|b_{2m}|b_{3m}$ where km represents first m bits of kth element, for m=4, 6 or 8. After the leading one prediction, ASOP1 structure is used for the computation of elements starting from the leading one position. Steps followed in ASOP2 can be illustrated with example as follows

- Consider the input elements as

  a_1="0011001000101110"

  a_2="0001011000101011"

  a_3="0010011001101000"

  b_1="0001001011101001"

b_2="0001101000101110"

b_3="0000101011101011".

- For m=4, the combination of first four bits using OR gate is computed.

  $a_{m_{or}}$=0011

- In this example, leading one predictor predicts zeros in first two bits of bit positions '15' and '14' of a_1, a_2 and a_3. From the next consecutive bit position '13', information is trimmed.

- 12-bit (16-m) information starting from bit position '13' to '2' of a_1, a_2 and a_3 ("110010001011", "010110001010", "100110011010") are taken and fed to the inputs of the lookup tables.

- For m=4, $b_{m_{or}}$=0001, leading one predictor detects zeros in first three bits of bit positions '15', '14' and '13' of b_1, b_2 and b_3.

- 12-bit (16-m) information starting from bit position '12' to '1' of b_1, b_2 and b_3 ("100101110100", "110100010111", "010101110101") are taken and fed as control signals of lookup tables.

The overall structure of ASOP2 is given in Figure 6.3, where LZA refers to leading zeros in $a_{m_{or}}$ and LZB refers to leading zeros in $b_{m_{or}}$. ASOP2 reduces the negative effects of truncation, especially when there is information only in least significant parts of the inputs. In DSP applications, pixel values are highly correlated and number of initial zeros of a_k and b_k for k=1, 2, 3 have high chances of being same. Using OR gate for combining the elements and using a leading one predictor afterward reduces hardware resources to be used.

6.2.3 Proposed Approximate Sum of Products Model ASOP3

In ASOP1, the least significant part m=8, 6, and 4 bits are truncated. In ASOP1, m bits are truncated from the 18 bit outputs of the lookup table contents. And also, m control signals b_{1n}, b_{2n}, b_{3n} of the lookup table for $n = 0, 1, ..m-1$ are truncated.

In ASOP3, instead of truncation, approximation is employed. Lookup table output contents are divided into $18-m$ bits and m bits. The inputs b are divided to $16-m$ group and m group. ASOP1 is used for the first $16-m$ group. For the least m bits group of b_k

for k=1, 2, 3, the control signals are grouped in pair. m lookup tables are reduced to $m/2$ tables. The additional hardware required for ASOP3 is given in Figure 6.4.

- Consider the input elements as

  a_1="0011001000101110"

  a_2="0001011000101011"

  a_3="0010011001101000"

  b_1="0001001011101001"

  b_2="0001101000101110"

  b_3="0000101011101011".

- For m=4, a_{23}, a_{13}, a_{12} and a_{123} are calculated.

- Except for least m bits, other bits are given to ASOP1 structure, and 12-bit (16-m) information starting most significant bit of b_1, b_2 and b_3 are taken and fed as control signals of lookup tables.

- For the least significant bits calculation, least significant m bits of a_{23}, a_{13}, a_{12} and a_{123} are used as inputs to the lookup table.

- The number of lookup tables are reduced by half, by ORing each pair of control signals. In this scenario, for lookup table of $n = 1 \mid 0$, the control signals would be 111.

6.3 Results and Discussion

Exact and approximate sum of products units described in Verilog HDL are implemented in TSMC 65nm library using Synopsys compiler at the typical process corner. The exact and approximate sum of products units are modeled for N=16. The efficiency of the proposed sum of products units is compared with existing approximate sum of products models [85, 62, 100, 54]. Approximate design model of parallel implementation of [85] is used to design truncated sum of products unit with least significant bits truncated in the lookup tables part. Two versions of truncation- with 10 bits truncated and 8 bits truncated are designed (TRUNC1 and TRUNC2). [62, 100, 54] propose approximate multipliers. These

Table 6.2: Implementation results of exact, existing and proposed approximate sum of products units

Sum of Products Type	Area (μm^2)	Delay (ns)	Power (mW)
Exact unit	11120	0.9	4.72
ASOP1 (m=8)	4030	0.7	1.44
ASOP1 (m=6)	5648	0.7	2.28
ASOP1 (m=4)	7030	0.8	2.92
ASOP2 (m=8)	7526	0.9	2.15
ASOP2 (m=6)	8960	1	3.17
ASOP2 (m=4)	10167	1	3.98
ASOP3 (m=8)	5790	0.7	1.97
ASOP3 (m=6)	6484	0.8	2.39
ASOP3 (m=4)	8812	0.8	3.56
TRUNC1 [85]	5538	0.7	2.11
TRUNC2 [85]	7110	0.7	2.72
ACM-SOP [62]	9779	0.6	2.06
PROB-SOP [100]	9784	0.6	2.94
PERF-SOP [54]	7185	0.7	2.44

approximate multipliers are used in construction of sum of products units. In [62], approximate compressor is used to construct 16- bit multipliers, approximate compressors based multipliers (ACM) and they are added to compute SOP (ACM-SOP). In [100], probabilistic multiplier of Chapter 4 is analyzed (PROB). For our comparison, three 16-bit probabilistic multipliers are designed and they are added to form SOP (PROB-SOP). Partial product perforation technique discussed in [54] is used to design perforated multiplier for j=2, k=8 and a Dadda tree structure is used to accumulate reduced partial product structure. The 16 bit multipliers are constructed using this technique for SOP (PERF-SOP). Error analysis of the approximate sum of products models are done using MATLAB with inputs of 1 million uniform random variable combination.

Table 6.2 compares all designs in terms of area, delay and power. ASOP1 with m=8 offers area, delay and power savings upto 64%, 22% and 70% respectively over the exact sum of product unit. ASOP2 saves up to 32% in area and 54% in power. ASOP3 saves up to 48%, 22% and 58% in area,delay and power respectively. While ASOP1 and ASOP3 version has delay improvement ranging from 11% to 22%, ASOP2 versions has increase in delay due to leading one prediction. However, they have significant reduction in area and power, and better accuracy as seen in Table 6.3.

From Table 6.3, error metrics MRED and NMED of approximate units along with their area power product (APP) and delay power product (PDP) can be seen. When compared to TRUNC1 unit, ASOP1 (m=8) has improved APP and PDP of 50% and 30% respectively, whereas ASOP3 (m=8) have slightly better APP and PDP. MRED of ASOP1 (m=8) and ASOP3 (m=8) are one order of magnitude better than TRUNC1 whereas their NMEDs are 50% and 61% lower respectively. Compared to TRUNC2, ASOP1 (m=6) has a 33% better APP, 16% better PDP, 51% and 49% lower MRED and NMED, respectively. All versions of ASOP have better MRED and NMED than ACM-SOP, PROB-SOP and PERF-SOP. Compared to ACM-SOP, PROB-SOP and PERF-SOP, ASOP1 (m=8, 6), ASOP2 (m=8), ASOP3 (m=8,6) have improved APP, and ASOP1 (m=8) have improved PDP. ASOP1, ASOP2 and ASOP3 of m=4 have the best MRED and NMED values. Considering all the cases, proposed sum of products units has better area and power savings and better accuracy compared to existing approximate designs.

ASOP1 (m=8) has the least APP and PDP of all, with its MRED and NMED better than TRUNC1, ACM-SOP, PROB-SOP and PERF-SOP. ASOP1 (m=6) and ASOP3 (m=6) have better APP, MRED and NMED compared to TRUNC2, ACM-SOP, PROB-SOP and PERF-SOP. ASOP1 (m=8,6) and ASOP3 (m=8) perform better than PROB-SOP and PERF-SOP in all four aspects. While other ASOPs have higher APP and PDP, compared to existing designs, they are to be used in applications demanding reduced error. However with ASOP1 of m=8,6, ASOP2 of m=8 and ASOP3 of m=8,6, it is proved that the proposed model has better results in design and error metrics compared to existing designs.

In Figure 6.5, APP and MRED for uniform random variables are sorted in ascending order for the proposed 9 approximate sum of products designs and five approximate models from the literature. ASOP1 (m=8) has the best area power product, whereas ASOP3 (m=4) has the best MRED in terms of uniform random variables.

6.4 Applications

6.4.1 Image Processing - Gaussian Filtering

Gaussian filtering [101] is a popular filtering technique used in signal processing applications such as image smoothing, edge detections and texture segmentation. Image smoothing is performed to reduce the image noise by using convolution. In this work, 3×3 gaussian filter is used to reduce noise of input image. The hardware of 3×3 filter requires 3 sum of products units.16- bit image with gaussian noise is taken for analysis. Smoothing is performed by performing 2-dimensional convolution operation over every pixel of image. 3×3 gaussian filter can be given as

$$\begin{bmatrix} 0.077847 & 0.123317 & 0.077847 \\ 0.123317 & 0.195346 & 0.123317 \\ 0.077847 & 0.123317 & 0.077847 \end{bmatrix}$$

Exact and approximate sum of products units are used to gaussian convolution operation on the image. Mean square error is found between the smoothed image using exact sum of

Table 6.3: Error metrics, area power product and power delay product of exact, existing and proposed approximate sum of products units

Sum of Products Type	MRED	NMED	APP	PDP
Exact unit	-	-	52486.4	4.248
ASOP1 (m=8)	9.76×10^{-3}	1.94×10^{-3}	5803.2	1.008
ASOP1 (m=6)	2.42×10^{-3}	4.80×10^{-4}	12877.4	1.596
ASOP1 (m=4)	5.76×10^{-4}	1.14×10^{-4}	20527.6	2.336
ASOP2 (m=8)	8.37×10^{-3}	1.81×10^{-3}	16180.9	1.935
ASOP2 (m=6)	2.07×10^{-3}	4.48×10^{-4}	28403.2	3.17
ASOP2 (m=4)	4.88×10^{-4}	1.06×10^{-4}	40464.7	3.98
ASOP3 (m=8)	7.94×10^{-3}	1.54×10^{-3}	11406.3	1.379
ASOP3 (m=6)	1.97×10^{-3}	3.80×10^{-4}	15496.8	1.912
ASOP3 (m=4)	4.69×10^{-4}	9.06×10^{-5}	31370.7	2.848
TRUNC1 [85]	1.95×10^{-2}	3.90×10^{-3}	11685.2	1.477
TRUNC2 [85]	4.89×10^{-3}	9.73×10^{-4}	19339.2	1.904
ACM-SOP [62]	3.30×10^{-1}	3.89×10^{-2}	20144.7	1.236
PROB-SOP [100]	5.30×10^{-2}	1.35×10^{-2}	28765	1.764
PERF-SOP [54]	1.95×10^{-2}	3.89×10^{-3}	17531.4	1.708

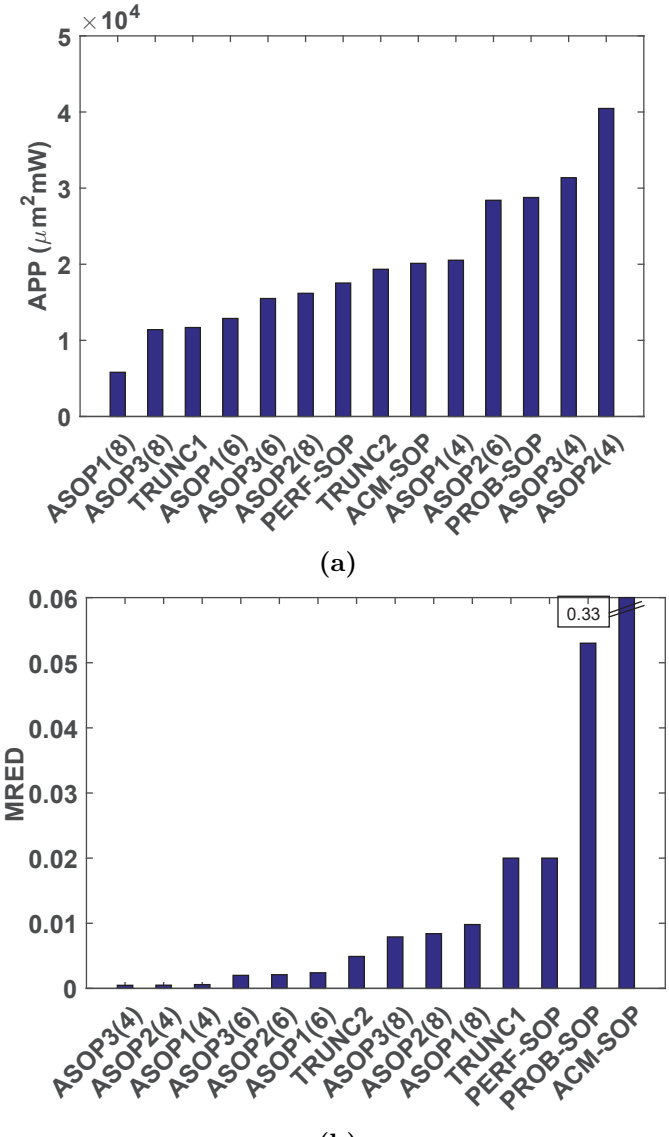


Figure 6.5: Ranked (a) APP (b) MRED

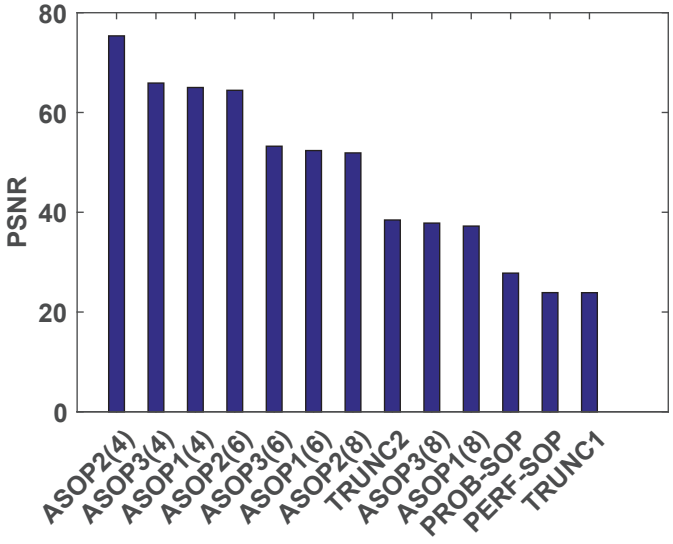


Figure 6.6: Ranked PSNR of approximate sum of products units in Gaussian filtering application

products unit and approximate sum of products units. To compare the image compression quality of exact and approximate sum of products units, PSNR is used.

The original input image and resultant images after image smoothing using exact and approximate sum of product units with their PSNR values are shown in Figure 6.7. ACM-SOP has a tendency to produce non-zero results for zero inputs, and produces sum of products output value exceeding 65535, so it is not included in the figure and tables. ASOP versions have significantly higher PSNR compared to existing approximate units, and it can be noticed that ASOP2 in particular achieves the high PSNR values. ASOP2 versions perform significantly better than other designs due to the presence of leading one predictor. Also, ASOP1 and ASOP3 versions where approximation is applied to 6 bits and 4 bits (m=6 and m=4), perform better than TRUNC1 and TRUNC2, where 10 bits and 8 bits are truncated respectively. Compared to sum of products units made of individual 16- bit approximate multipliers i.e., ACM-SOP, PROB-SOP and PERF-SOP, proposed ASOP units, TRUNC1 and TRUNC2 have better PSNR. ASOP2 (m=4) version has the highest PSNR of all approximate designs.

In Figure 6.6, PSNR of image smoothing application are sorted in descending order for the approximate units. ASOP2 (m=4) has the best PSNR in terms of Gaussian image processing application.

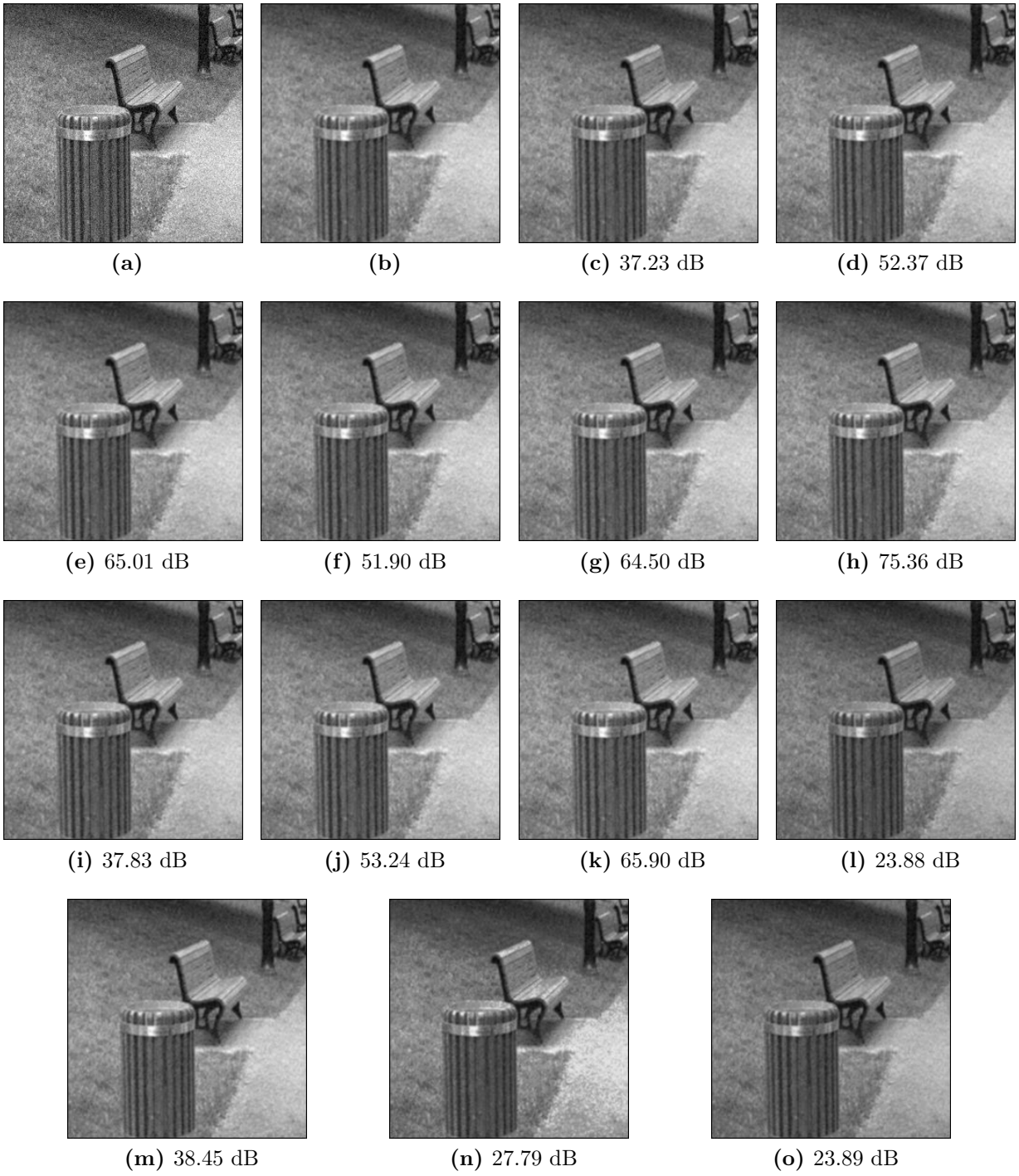
6.4.2 Color Compression- K-means Clustering

K-means clustering algorithm is a well known machine learning algorithm. It is used to group similar multi-dimensional data points based on the a distance measure. Sufficient number of initial cluster points are chosen from the input, and the distance between existing points and chosen cluster points are calculated. Based on minimum distance, each point is allocated to a cluster. In our application, K-means steps can be given as

- Colored images are stored with color information of 3 bytes in each pixel. In this work, an RGB (Red, Green, Blue) image is taken. These images are stored as compressed image after K-means clustering color compression.
- Random initialization of K-clusters from the data points in the input image is performed.
- With Euclidean distance measurement, each data point is assigned to the nearest cluster value.
- The final compressed image has pixel values from one of the cluster points where it finds its closest match.

K-means algorithm is widely used in image processing for compression of color images. Approximate sum of products units are used to perform distance measures. PSNR is based on MSE found between resulting image of exact input image before compression and the images generated from k-means algorithm after compression.

The original input image and resultant images after color compression using exact and approximate sum of product units are shown in Figure 6.8. Their NMED, MRED and PSNR values are given in Table 6.4. ASOP versions have significantly higher PSNR compared to existing approximate units, and it can be noticed that they achieve the same PSNR as achieved by the exact unit. ASOP2 versions behave better in this application, whereas ASOP3 has better results with uniforms random variables. ASOP2 (m=4) version has lower MRED and NMED of all approximate designs.



(a) **(b)** **(c)** 37.23 dB **(d)** 52.37 dB

(e) 65.01 dB **(f)** 51.90 dB **(g)** 64.50 dB **(h)** 75.36 dB

(i) 37.83 dB **(j)** 53.24 dB **(k)** 65.90 dB **(l)** 23.88 dB

(m) 38.45 dB **(n)** 27.79 dB **(o)** 23.89 dB

Figure 6.7: (a) Input noisy image. Images after gaussian smoothing using (b) exact multiplier (c) ASOP1 (m=8) (d) ASOP1 (m=6) (e) ASOP1 (m=4) (f) ASOP2 (m=8) (g) ASOP2 (m=6) (h) ASOP2 (m=4) (i) ASOP3 (m=8) (j) ASOP3 (m=6) (k) ASOP3 (m=4) (l) TRUNC1 (m) TRUNC2 (n) PROB-SOP (o) PERF-SOP

Table 6.4: Error metrics and PSNR of exact, existing and proposed approximate sum of products units used in K-means application

Sum of Products Type	MRED	NMED	PSNR (dB)
Exact unit	—	—	31.65
ASOP1 (m=8)	$5.52\text{x}10^{-2}$	$1.29\text{x}10^{-3}$	31.65
ASOP1 (m=6)	$1.66\text{x}10^{-2}$	$3.23\text{x}10^{-4}$	31.65
ASOP1 (m=4)	$5.23\text{x}10^{-3}$	$8.16\text{x}10^{-5}$	31.65
ASOP2 (m=8)	$6.22\text{x}10^{-3}$	$1.10\text{x}10^{-3}$	31.65
ASOP2 (m=6)	$1.41\text{x}10^{-3}$	$2.72\text{x}10^{-4}$	31.65
ASOP2 (m=4)	$2.82\text{x}10^{-4}$	$6.42\text{x}10^{-5}$	31.65
ASOP3 (m=8)	$4.40\text{x}10^{-2}$	$9.58\text{x}10^{-4}$	31.65
ASOP3 (m=6)	$1.33\text{x}10^{-2}$	$2.38\text{x}10^{-4}$	31.65
ASOP3 (m=4)	$3.40\text{x}10^{-3}$	$5.66\text{x}10^{-5}$	31.65
TRUNC1 [85]	$1.15\text{x}10^{-1}$	$2.61\text{x}10^{-3}$	31.56
TRUNC2 [85]	$3.41\text{x}10^{-2}$	$6.50\text{x}10^{-4}$	31.65
PROB-SOP [100]	$4.83\text{x}10^{-2}$	$6.57\text{x}10^{-3}$	31.64
PERF-SOP [54]	$1.14\text{x}10^{-1}$	$2.60\text{x}10^{-3}$	31.61

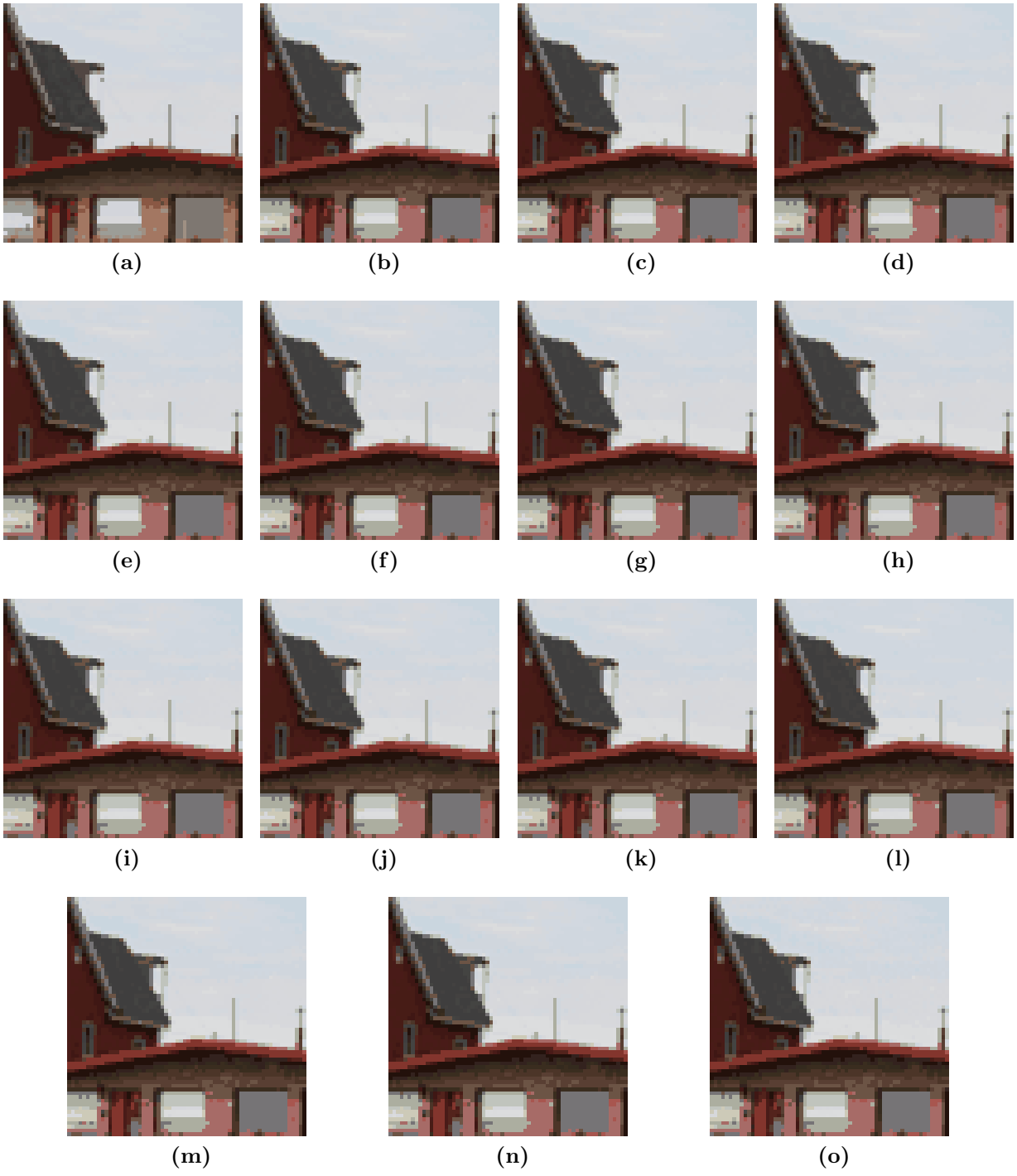


Figure 6.8: (a) Input image. Image after K-means clustering using (b) exact multiplier (c) ASOP1 (m=8) (d) ASOP1 (m=6) (e) ASOP1 (m=4) (f) ASOP2 (m=8) (g) ASOP2 (m=6) (h) ASOP2 (m=4) (i) ASOP3 (m=8) (j) ASOP3 (m=6) (k) ASOP3 (m=4) (l) TRUNC1 (m) TRUNC2 (n) PROB-SOP (o) PERF-SOP

7 Design of Approximate Restoring Dividers

[1]

In this work, two approximation models are proposed for restoring divider. In the first design, approximation is performed at circuit level, where approximate divider cells are utilized in place of exact ones by simplifying the logic equations. In the second model, restoring divider is analyzed strategically and number of restoring divider cells are reduced by finding the portions of divisor and dividend with significant information. Two models of hardware-efficient approximate dividers AD-M1 and AD-M2 are proposed. AD-M1 is based on replacing the exact individual restoring cell in the restoring divider, which is a combination of a subtractor and multiplexer, with an approximate cell. In AD-M2, the number of bits in the dividend and divisor are reduced based on the leading one position in divisor.

Some contributions of this work can be listed as

- Circuit based approximation is proposed in AD-M1. Restoring division is analyzed at the basic cell level, logic table of each cell is analyzed and approximation is proposed at the cell level, by altering few entries in the truth table.

- Strategy based approximation is proposed in AD-M2. Input bits are reduced by considering only the more relevant bits in divisor and dividend and the overflow error in the quotient is reduced by using an equality detector.

- An approximation factor p is used. Three values of p are used in each model, which enables the designer to adopt a scalable approach and comprehensive analysis. In the first model, p is the number of columns where exact cells are replaced with approximate

[1] Major part of this work has been submitted in "Design of Approximate Restoring Dividers," *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, Japan, 2019. Authors: Suganthi Venkatachalam, Elizabeth Adams and Seok-Bum Ko.

ones. In the second model, p is the number of bits reduced in dividend and divisor by using a leading-one predictor in divisor.

- The proposed and existing approximate divider structures are tested with an image division application to detect changes from one scenario to another.

In section 7.1, design of the exact restoring divider is analyzed and proposed circuit based and strategy based approximate models are discussed. Design and error metrics of proposed models are compared with existing approximate models and exact design in section 7.2. In the context of application, proposed approximate division models are utilized in motion detection application in section 7.3 and found to exhibit high quality results.

7.1 Proposed Models of Approximate Restoring Division

7.1.1 Exact Restoring Divider

Restoring division consists of series of subtraction and shifting operations. Dividend A of $2n$ bits and divisor B of n bits can be given as

$$\begin{aligned} A &= a_{2n-1} \cdot a_{2n-2} \cdots a_0 \\ B &= b_{n-1} \cdot b_{n-2} \cdots b_0 \end{aligned} \tag{7.1}$$

An 8-bit dividend/4-bit divisor version of restoring array divider is shown in Figure 7.1, where each row performs a trial subtraction. The result of trial subtraction being positive or negative determines the quotient bit and the partial remainder. Although the array divider looks similar to an array multiplier, the latency of the array divider is much higher because of ripple-borrow subtraction in each row, and each row has to wait for the execution of the previous row.

In a $2n/n$ restoring divider with a quotient of n bits, the basic rule to avoid overflow is that the most significant n bits of the $2n$ bits wide dividend should be less than n bits of divisor [92]. As shown in Figure 7.1, in each row, n bits of subtrahend are subtracted from

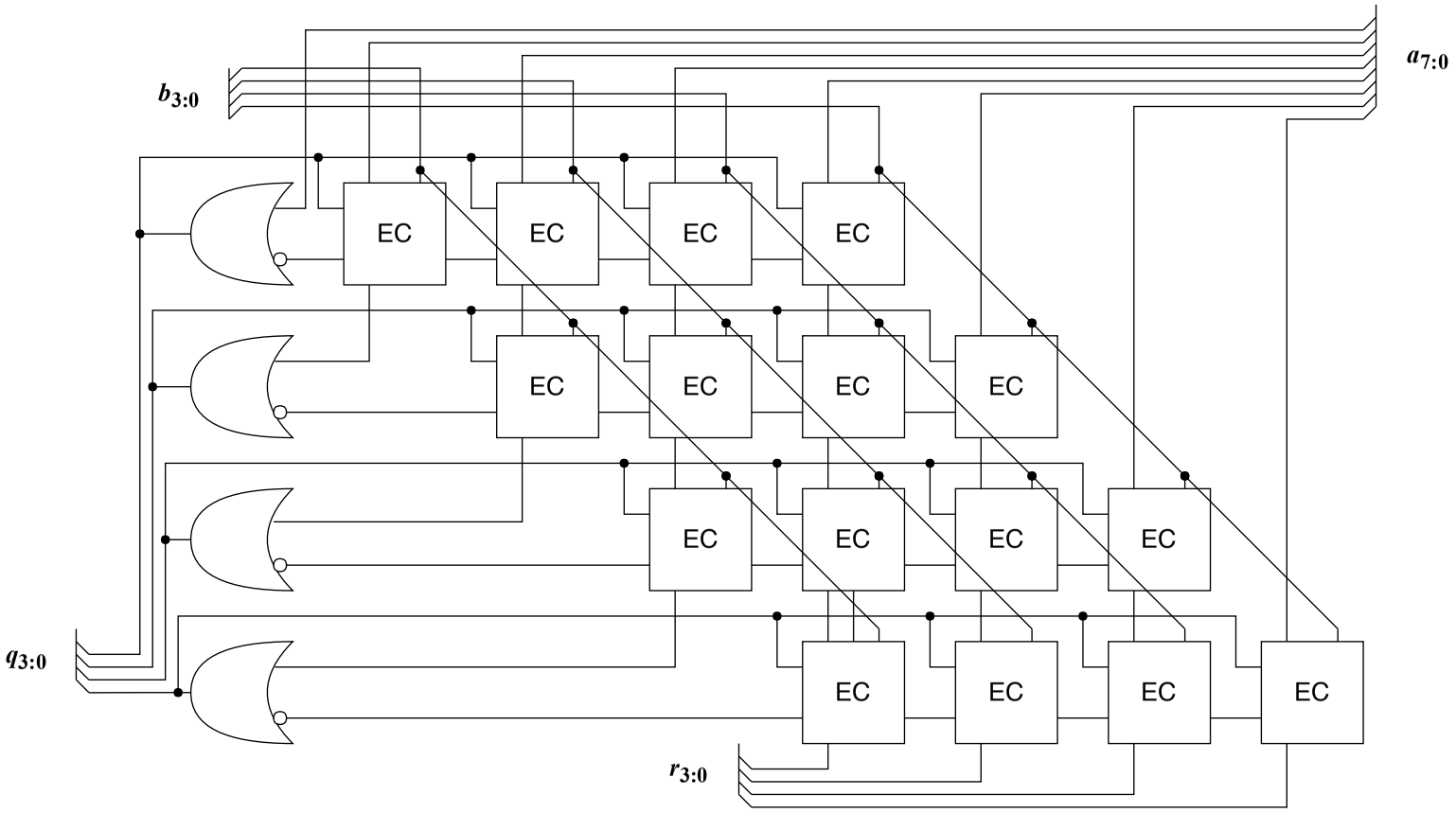


Figure 7.1: Circuit schematic for exact divider with 8-bit dividend and 4-bit divisor.

$n + 1$ bits of minuend. Subtrahend is always the n-bit divisor B. In the first row, most significant $n + 1$ bits of dividend A are taken as minuend. The quotient bit is 1 if there is no borrow after subtraction or if the most significant bit of minuend is 1, otherwise the quotient is 0. Each quotient bit decides whether the partial remainder of the respective stage is the result of subtraction (when the quotient bit is 1) or the least n bits of minuend. In the following row, the partial remainder with the consecutive dividend bit forms the minuend and steps are repeated until n bits of quotient are obtained.

7.1.2 Approximate Restoring Divider - Model 1

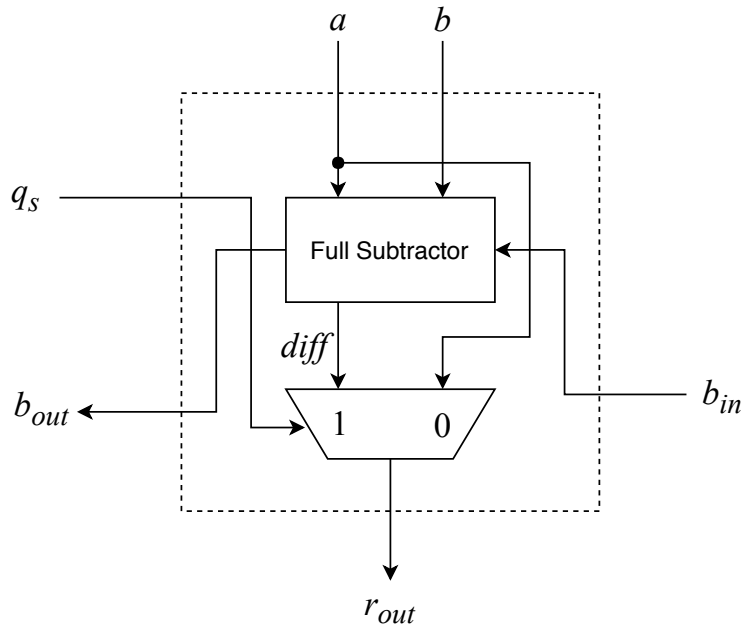


Figure 7.2: Circuit diagram for exact cell EC.

In the first approximation model of the restoring divider, some of the exact restoring cells are replaced with approximate restoring cells. Each exact cell in restoring division is made of a full subtractor and a 2-1 multipliexer as shown in Figure 7.2. The two outputs of the exact cell can be given by equations 7.2 and 7.3.

$$b_{out} = b_{in}\overline{(a \oplus b)} + \overline{a}b \tag{7.2}$$

$$r_{out} = q_s diff + \overline{q_s}a \tag{7.3}$$

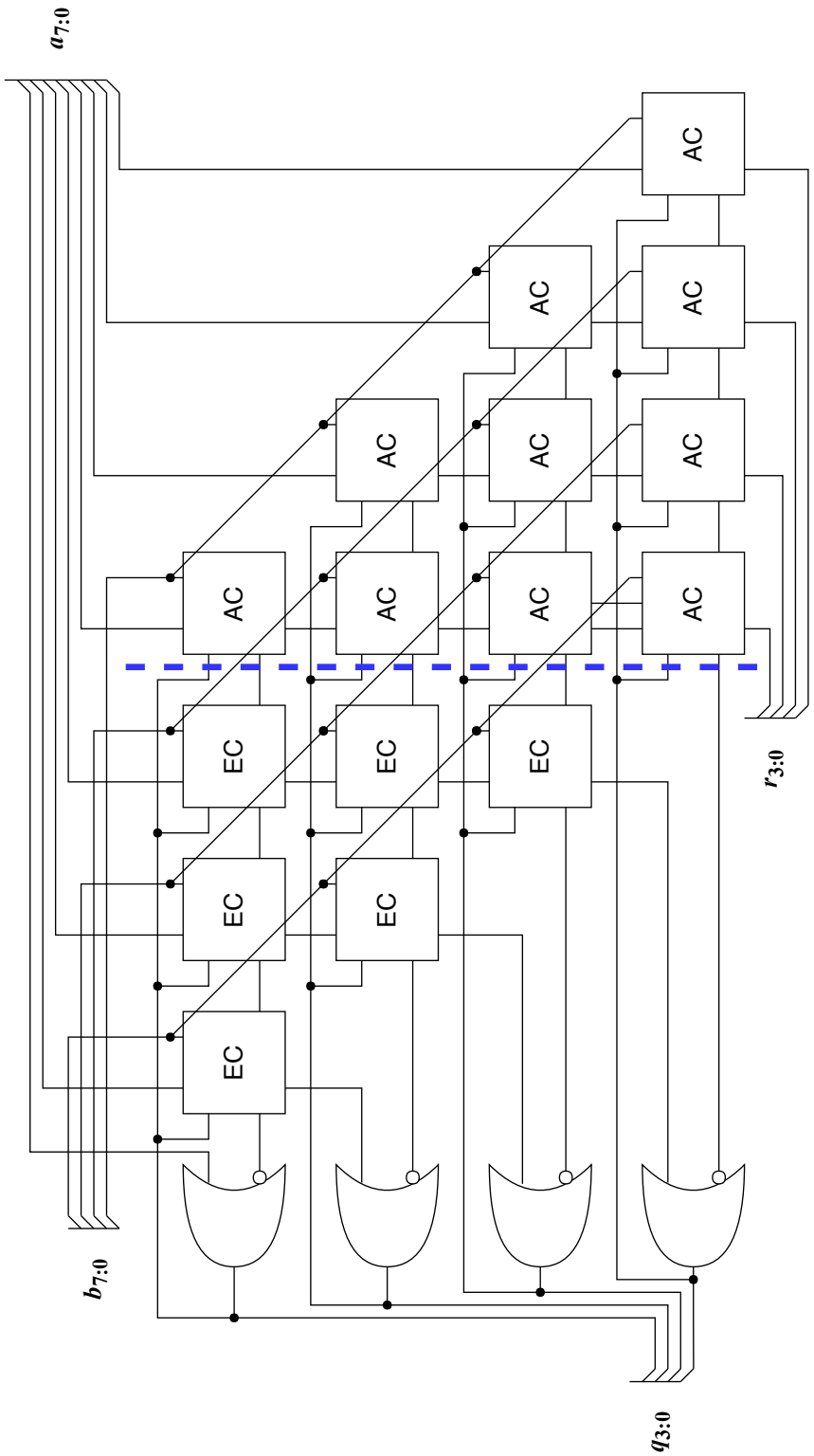


Figure 7.3: Circuit schematic for AD-M1 with 8-bit dividend and 4-bit divisor for $p = 4$.

Table 7.1: Comparison of outputs for exact cell and approximate cell.

				Exact			Approximate	
q_s	a	b	b_{in}	*diff*	b_{out}	r_{out}	b_{out}	r_{out}
0	0	0	0	0	0	0	$\boxed{1}$	0
0	0	0	1	1	1	0	1	0
0	0	1	0	1	1	0	1	0
0	0	1	1	0	1	0	1	0
0	1	0	0	1	0	1	0	1
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	0	1
0	1	1	1	1	1	1	$\boxed{0}$	1
1	0	0	0	0	0	0	$\boxed{1}$	$\boxed{1}$
1	0	0	1	1	1	1	1	1
1	0	1	0	1	1	1	1	1
1	0	1	1	0	1	0	1	$\boxed{1}$
1	1	0	0	1	0	1	0	$\boxed{0}$
1	1	0	1	0	0	0	0	0
1	1	1	0	0	0	0	0	0
1	1	1	1	1	1	1	$\boxed{0}$	$\boxed{0}$

where b_{out} is the borrow-out of the restoring cell and r_{out} is the partial remainder bit. The exact results are given in the table 7.1. As it can be seen from the table, when some values of b_{out} and r_{out} are modified, logic equations can be simplified as given in equations 7.4 and 7.5. Modified cells are represented in square boxes in table 7.1. For approximate divider model 1 (AD-M1) with approximation factor p, $p(p+1)/2$ exact restoring cells are replaced with approximate restoring cells. A 8/4 AD-M1 with $p = 4$ is shown in Figure 7.3.

$$b_{out} = \overline{a} \tag{7.4}$$

$$r_{out} = q_s\overline{a} + \overline{q_s}a = q_s \oplus a \tag{7.5}$$

7.1.3 Approximate Restoring Divider - Model 2

In approximate divider model 2 (AD-M2), a $2n/n$ divider is reduced to a $(2n-p)/(n-p)$ divider. p number of cells are reduced in each row by reducing the number of bits in the divisor by a factor p. The first step is reducing the n-bit divisor to a $n-p$-bit divisor. A leading-one detector is used in first p bits of B to truncate p bits. After finding the leading-one starting bit-position (sbp), B can be trimmed to

$$B_{mod} = b_{sbp} \cdot b_{sbp-1} \cdot b_{sbp-2} \cdot \cdot \cdot \cdot b_{sbp-((n-p)-1)} \tag{7.6}$$

Similarly, A is trimmed as

$$A_{mod} = a_{(sbp+n)} \cdot a_{(sbp+n)-1} \cdot a_{(sbp+n)-2} \cdot \cdot \cdot \cdot a_{(sbp+n)-((2n-p)-1)} \tag{7.7}$$

The number of bits trimmed at the beginning and end of the divisor and dividend are the same, since the first n bits of $2n$ bits wide A are less than n bits wide B. After trimming the bits, overflow is possible if the most significant $n-p$ bits of A are equal to $n-p$ bits of B. To compensate for overflow, an equality detector is used and, when equality is detected, quotient bits are set to 1. The remainder is shifted left by $sbp-((n-p)-1)$ bits. A 8/4 AD-M2 with p=2 is shown in Figure 7.4.

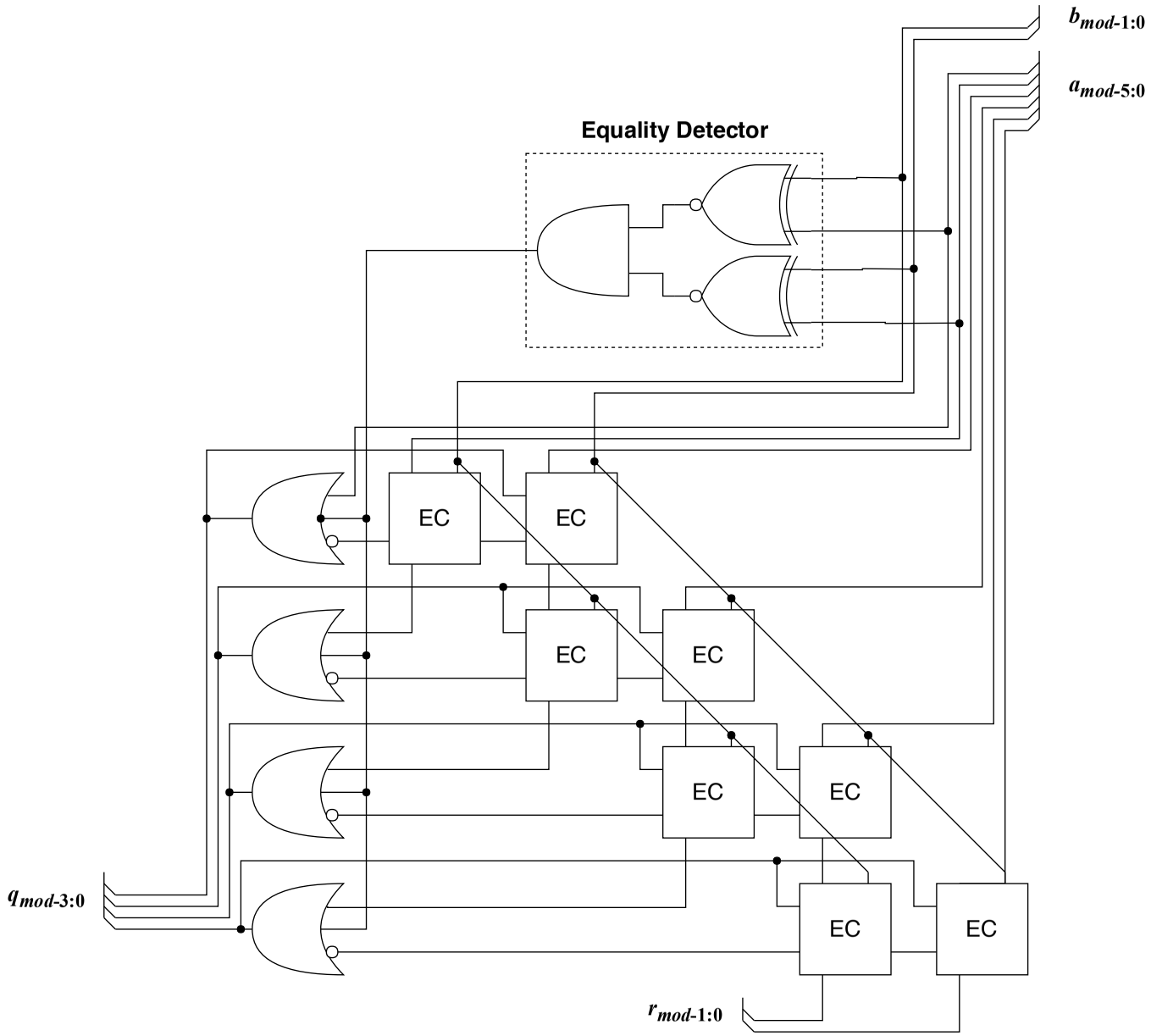


Figure 7.4: Circuit schematic for AD-M2 with 8-bit dividend and 4-bit divisor for $p = 2$.

7.2 Results and Discussion

This section compares the error and design metrics of the proposed divider designs as well as the primary competitor design introduced in [94]. All designs were implemented in Verilog and verified using the ModelSim HDL simulator. Error metrics were generated using a SystemVerilog testbench and a Python script. Area and power consumption metrics were generated using Synopsys Design Compiler using the TSMC 65 nm library at typical process corner.

Table 7.2: MRED and NMED values of proposed and competing approximate 16-bit dividend and 8-bit divisor dividers

Design	p	Q-MRED (10^{-2})	Q-NMED (10^{-2})	R-MRED (10^{0})	R-NMED (10^{0})
AD-M1	4	0.226	0.037	1.662	0.088
	6	0.494	0.101	2.269	0.175
	8	1.909	0.449	2.893	0.261
AD-M2	2	0.702	0.343	1.465	0.157
	3	1.764	0.858	1.808	0.190
	4	3.958	1.900	1.926	0.207
AXDr1 [94]	8	1.207	0.292	3.103	0.292
AXDr2 [94]	8	3.378	0.717	3.183	0.273
AXDr3 [94]	8	0.961	0.257	2.542	0.278

The error metrics calculated for all designs are summarized in Table 7.2. The models

AXDr1, AXDr2 and AXDr3 of [94] are implemented with a triangle replacement scheme and compared with proposed models. The accuracy evaluation for AD-M1 produces a competitive quotient MRED that is significantly lower than that of all competitor designs. Notably, the Q-MRED for approximation factors $p = 4, 6$ are an order of magnitude smaller than that of competitor designs AXDr1 and AXDr2. Quotient NMED values for $p = 4, 6$ are similarly competitive, and $p = 4$ produces a Q-NMED an order of magnitude smaller than that of all competitor designs. All three approximation factors for AD-M1 have remainder NMED values lower than all competitor designs. R-NMED for $p = 4$ is notably an order of magnitude smaller than any other design.

Although the quotient error metrics for AD-M2 are larger than AD-M1, they are still competitive for approximation factors of $p = 2, 3$. AD-M2 exhibits notably small error metrics for the remainder; designs for all three approximation factors produce R-MRED and R-NMED values smaller than all competitor designs.

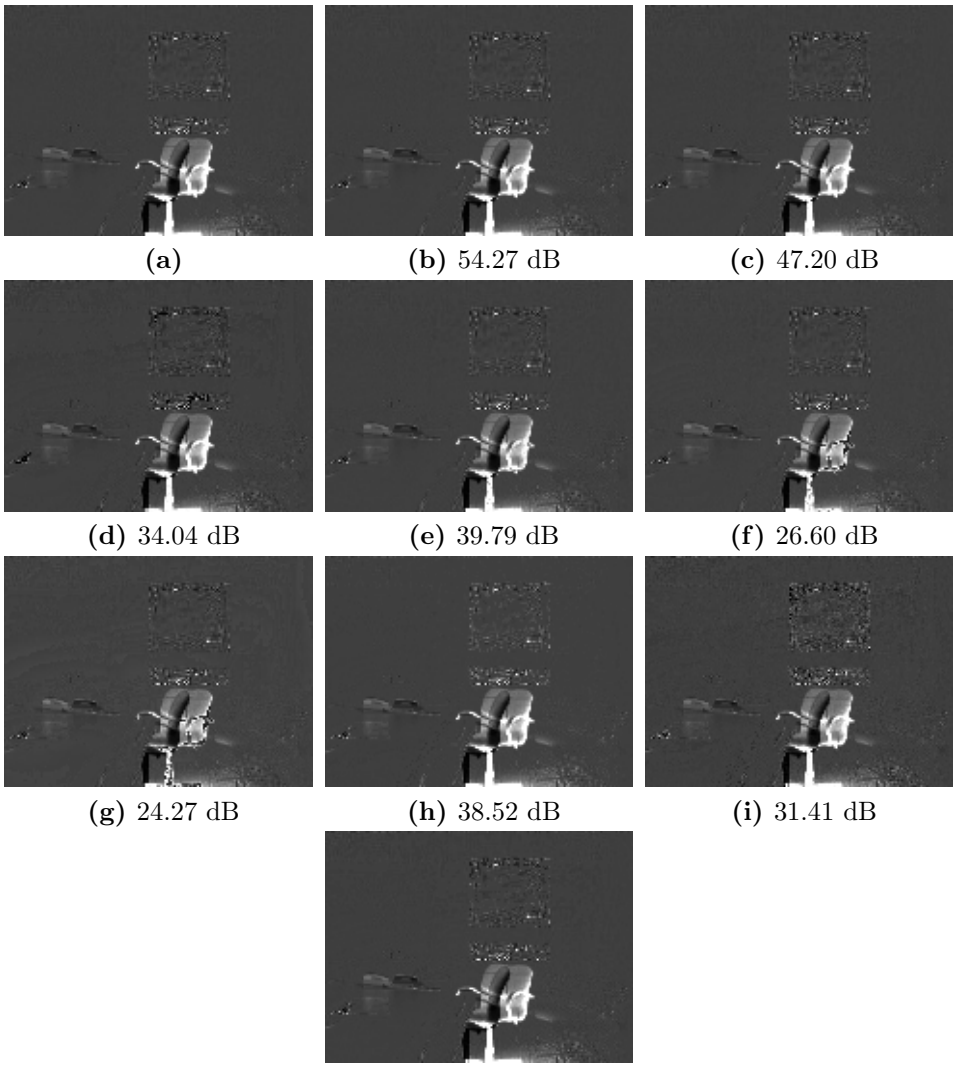
Table 7.3 describes the area, power, area-power product and power-delay product for proposed and competitor designs, synthesized at critical path delay of each design. AD-M1 exhibits APP reduction up to 83% and PDP reduction up to 71% when compared to the exact divider. Approximation factors $p = 6, 8$ for AD-M1 have smaller APP and PDP than all competitor designs. AD-M2 has area and power values that are competitive for approximation factors $p = 3, 4$. Notably, AD-M2 with $p = 4$ has an area-power product of 672, which is an 83% reduction from the exact design and a reduction of at least 56%.

(a) **(b)**

Figure 7.5: Two input images for change detection application

Table 7.3: Area, power, and area-power product values of proposed and competing 16-bit dividend and 8-bit divisor approximate dividers

Design	p	Area (μm^2)	Delay (ns)	Power (mW)	APP ($\mu m^2 \cdot mW$)	PDP (pJ)
Exact	-	3448	1.8	1.12	3848	2.02
AD-M1	4	2585	1.6	0.84	2178	1.34
	6	1946	1.5	0.68	1326	1.03
	8	1426	1.3	0.46	662	0.58
AD-M2	2	2294	2.0	0.74	1688	1.43
	3	2327	1.7	0.67	1568	1.13
	4	1587	1.7	0.42	672	0.74
AXDr1 [94]	8	2885	1.7	0.95	2754	1.63
AXDr2 [94]	2	1877	1.9	0.79	1490	1.51
AXDr3 [94]	8	2137	1.7	0.69	1482	1.19



(a) **(b)** 54.27 dB **(c)** 47.20 dB

(d) 34.04 dB **(e)** 39.79 dB **(f)** 26.60 dB

(g) 24.27 dB **(h)** 38.52 dB **(i)** 31.41 dB

(j) 38.65 dB

Figure 7.6: Change detection results using (a) Exact divider. Approximate divider models with the corresponding PSNR values (b) AD-M1 (p=4) (c) AD-M1 (p=6) (d) AD-M1 (p=8) (e) AD-M2 (p=2) (f) AD-M2 (p=3) (g) AD-M2 (p=4) (h) AXDr1 (p=8) (i) AXDr2 (p=8) (j) AXDr3 (p=8)

7.3 Application- Change Detection

Change detection is to detect relative motion of an object with respect to its surroundings or vice-versa. In general, change detection is an initial pre-processing step in computer vision and is used in applications such as intrusion detection, smart environment, activity localization, tracking, medical diagnosis and remote sensing. Changes can be short-term as in intrusion detection or long-term as in rate of growth in remote sensing, and change detection helps in finding region of interest. Here, exact and approximate dividers are used to find changes in two given images. Input images are given in Figure 7.5. The first image is multiplied by 64 and divided by the second image on a pixel-by-pixel basis, which results in highlighting the region of interest. A Matlab environment is used to process the images.

Final images after division are shown in Figure 7.6 with their corresponding PSNR values. As the approximation factor increases, the quality decreases for each model. AD-M2 with $p = 3, 4$ has poor performance with loss of quality especially in arms and legs of the change detected armchair region. AD-M1 with $p = 4$ and AD-M2 with $p = 2$ exhibits better PSNR than other designs and can be used in applications demanding high quality.

8 Conclusions and Future Research

8.1 Summary and Conclusions

Motivated by the advantages of approximation in power hungry, error tolerant applications and the need of mobile energy efficient designs in today's world, designs of approximate arithmetic circuits are investigated in this thesis. The major areas covered in this thesis are investigation of approximate designs for arithmetic circuits and their real-life applications, mostly focusing on image processing applications including convolution operations and a machine learning algorithm - K-means clustering. In addition, our approximate arithmetic circuits designs also find wide range of applications in artificial intelligence, data mining, object recognition and computer vision. Investigation of approximation in deep learning is to be conducted in near future.

Previous approximate designs of arithmetic circuits and possible improvements on existing designs are studied during this research. Our ideas to further improve the performance of approximate arithmetic circuits are investigated during the research. One main challenge was to find areas of performance optimization with a better trade-off of accuracy.

In the proposed approximation in array multiplication, two variants of approximate multipliers are proposed, where approximation is applied in all n bits in Multiplier1 and only in $n-1$ least significant part in Multiplier2. Multiplier1 and Multiplier2 achieve significant reduction in area and power consumption compared with exact designs. With area power product savings being 87% and 58% for Multiplier1 and Multiplier2 with respect to exact multipliers, they also outperform in area power product in comparison with existing approximate designs. They are also found to have better precision when compared to existing approximate multiplier designs. The proposed multiplier designs can be used in error tolerant applications saving significant area and power with negligible loss in output quality.

Three models of approximate Booth multipliers are proposed, differing according to the signals used in the partial product generator. An exact partial product generator requires at least three encoded signals resulting from multiplier bit groupings to generate an exact partial product matrix. Here, we investigate reducing the number of recoded signals, reduced complexity of the partial product generator, and introducing approximation in the partial product matrix. In ABM-M1, the depth of the partial product matrix is reduced to simplify the partial product accumulation. A partial product generator that uses only two signals is used in ABM-M1 and ABM-M2, and a partial product generator that uses only one signal is used in ABM-M3. An approximation factor p is used to indicate the imprecision of each model of the proposed multipliers. When compared to the exact Booth multiplier, ABM-M1 exhibits power savings ranging from 9.6% to 15% with corresponding MRED values in the range of 1.6×10^{-4} to 7.9×10^{-4}. Similarly, ABM-M2 has a reduction of power consumption in the range of 36% to 60% for MRED values of 6.6×10^{-3} to 1.1×10^{-1}, and ABM-M3 exhibits power savings of 28% to 50% for MRED values in the range of 2.0×10^{-4} to 3.4×10^{-3}. Furthermore, the proposed multipliers are tested using image multiplication and matrix multiplication applications. It is found that of all the proposed multipliers, ABM-M1 and ABM-M3 have better accuracy and provide better results than existing approximate multipliers.

Three models of efficient approximate sum of products are proposed in this work. Model 1 (ASOP1) employs truncation, model 2 (ASOP2) employs leading one predictor for approximation, and model 3 (ASOP3) provides approximation in lower significant part. Area power trade-off with error analysis is analyzed and it is found that our proposed models have better area power product compared to exact and existing approximate designs and with lower error metrics compared to existing models. ASOP1, ASOP2 and ASOP3 models with highest accuracy achieve up to 61%, 23% and 40% improved area power product respectively compared to the exact sum of products unit. The proposed sum of products designs can be used in applications where some loss of precision in the computation is possible with higher improvement in area and power.

Two approximate restoring dividers AD-M1 and AD-M2 are proposed. AD-M1 introduces approximation to the p least-significant columns by replacing $p(p-1)/2$ exact cells

with approximate cells. The approximate cells have simplified circuitry which therefore allows for reduction of design metrics in the AD-M1 divider. The AD-M2 design introduces approximation by eliminating p cells of each row by reducing in the divider and making use of an equality-detector circuit for the most-significant $n - p$ bits of a and b. Reducing the number of cells in the divider design reduces energy consumption. Both dividers are used to demonstrate a change detection application.

8.2 Future Research

8.2.1 Approximation in Deep Learning Applications

Deep learning networks are machine learning models which are on the rise. They are made of cascaded structures of many processing layers, with different levels of abstraction [103]. Deep networks have been proven to be effective in a wide spectrum of computationally intensive tasks including image processing [104, 105], medical imaging [106], audio processing [107], big-data analytics [108], machine translation [109], recommender systems [111], and speech recognition [110]. They achieve excellent accuracy with their larger increased network depths. For example, in [105], when the network depth is increased from 11 layers to 19 layers, error rate significantly drops in image recognition tasks. This improvement comes with a price. Larger network results in more parameters, memory bandwidth and hardware resources. As the depth increases, training deep networks becomes a problem. Deep architecture uses millions of parameters. In a MNIST dataset experiment, deep networks such as Fitnets networks uses up to 9M (million) parameters and Highway networks uses up to 2.3M parameters [112]. Various thin networks called FitNets to reduce number of computations are analyzed in [113]. In [113], training complexity of deep networks are compared. A deep network called Teacher network with 5 layers which uses around 9M parameters with 725M multiplications is compared with proposed FitNets. Even the thin networks consume millions of parameters and operations, for instance, FitNet 1 with 11 layers consume 250K parameters with 30M multiplications and FitNet4 with 19 layers uses around 2.5M parameters and 382M multiplications.

The main operation during training and inference is matrix multiplication which consists of multiplications and additions. With massive demand of hardware resources, network bandwidth of storage and transmission, deep computing becomes an ideal candidate for approximation. Considering these limitations of deep learning, the plan is to analyze combination of approximations including

- Hardware level approximation in multiply accumulate units in matrix multiplication operations which can be efficient during training and inference.
- Huge memory requirement in deep learning causes latency when data being transferred from memory to main processor. To solve this issue, research on Processor In Memory (PIM) is going on in our current research.
- Loop perforation approximations where number of iterations are reduced with a trade-off of negligible precision for efficiency are to be researched in different image recognition applications.
- Sparsity in deep learning is to be analyzed, which means to differentiate necessary and unnecessary forwarding of activations to succeeding layers in deep networks. Finding out sparse kernels can be used to apply relaxed approximations.

Deep learning finds various range of applications and has numerous bottlenecks. It can benefit to a large extent from approximate computing.

Bibliography

[1] https://www.purdue.edu/newsroom/releases/2013/Q4/approximate-computing-improves-efficiency,-saves-energy.html

[2] G. E. Moore, "Cramming more components onto integrated circuits," Reprinted from *Electronics*, volume 38, number 8, April 19, 1965 in *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33-35, Sept. 2006.

[3] G. E. Moore, "Progress in digital integrated electronics," Reprinted from Technical Digest, *International Electron Devices Meeting, IEEE*, 1975, pp. 11-13, in *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 36-37, Sept. 2006.

[4] https://www.seattletimes.com/business/forty-years-of-moores-law/

[5] https://en.wikipedia.org/wiki/Moore%27s_law

[6] https://www.technologyreview.com/s/601102/intel-puts-the-brakes-on-moores-law/

[7] "INTEL CORP, FORM 10-K (Annual Report), Filed 02/12/16 for the Period Ending 12/26/15"

[8] E. Track, N. Forbes and G. Strawn, "The End of Moore's Law," in *Computing in Science & Engineering*, vol. 19, no. 2, pp. 4-6, Mar-Apr. 2017

[9] https://steveblank.com/2018/09/12/the-end-of-more-the-death-of-moores-law/

[10] R. H. Dennard, F. H. Gaensslen, Hwa-Nien Yu, V. L. Rideout, E. Bassous and A. R. Leblanc, "Design Of Ion-implanted MOSFET's with Very Small Physical Dimensions," in *Proceedings of the IEEE*, vol. 87, no. 4, pp. 668-678, April 1999. This paper is reprinted from *IEEE Journal of Solid state circuits*, vol. SC-9, no. 5, pp-256-268, October 1974.

[11] http://www.gotw.ca/publications/concurrency-ddj.htm

[12] M. Bohr, "A 30 Year Retrospective on Dennard's MOSFET Scaling Paper," in *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 11-13, Winter 2007.

[13] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *IEEE Micro*, vol. 32, no. 3, pp. 122-134, May-June 2012.

[14] V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX, 2013, pp. 1-9.

[15] W. El-Harouni, S. Rehman, B. S. Prabakaran, A. Kumar, R. Hafiz and M. Shafique, "Embracing approximate computing for energy-efficient motion estimation in high efficiency video coding," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Lausanne, 2017, pp. 1384-1389.

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

[17] J. Bornholt, T. Mytkowicz and K. S. McKinley, "Uncertain T: A first-order type for uncertain data," in *ACM SIGARCH Computer Architecture News ASPLOS'14*, vol. 42, issue 1, pp. 51-66, March 2014.

[18] C. H. Stapper and R. J. Rosner, "Integrated circuit yield management and yield analysis: development and implementation," in *IEEE Transactions on Semiconductor Manufacturing*, vol. 8, no. 2, pp. 95-102, May 1995.

[19] B. R. Gaines, "Stochastic computing," *Proceedings of the AFIPS Spring Joint Computer Conference*, Atlantic city, New Jersy, 1967, pp. 149–156.

[20] P. K. Gupta and R. Kumaresan, "Binary multiplication with PN sequences," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 4, pp. 603-606, April 1988.

[21] M. van Daalen, P. Jeavons, J. Shawe-Taylor and D. Cohen, "Device for generating binary sequences for stochastic computing," in *Electronics Letters*, vol. 29, no. 1, pp. 80-, Jan 1993.

[22] R. Wang, J. Han, B. F. Cockburn and D. G. Elliott, "Stochastic Circuit Design and Performance Evaluation of Vector Quantization for Different Error Measures," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, pp. 3169-3183, Oct. 2016.

[23] H. Aliee and H. R. Zarandi, "Fault tree analysis using stochastic logic: A reliable and high speed computing," *2011 Proceedings - Annual Reliability and Maintainability Symposium*, Lake Buena Vista, FL, 2011, pp. 1-6.

[24] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," *2008 45th ACM/IEEE Design Automation Conference*, Anaheim, CA, 2008, pp. 648-653.

[25] B. D. Brown and H. C. Card, "Stochastic neural computation. I. Computational elements," in *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891-905, Sept. 2001.

[26] J. A. Dickson, R. D. McLeod and H. C. Card, "Stochastic arithmetic implementations of neural networks with in situ learning," *IEEE International Conference on Neural Networks*, San Francisco, CA, USA, 1993, pp. 711-716, vol.2.

[27] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing", *ACM Transactions on Embedded Computing Systems (TECS) - Special Section on Probabilistic Embedded Computing*, vol. 12, no. 92, May 2013.

[28] K. Rana and S. Thakur, "Data compression algorithm for computer vision applications: A survey," *2017 International Conference on Computing, Communication and Automation (ICCCA)*, Greater Noida, 2017, pp. 1214-1219.

[29] Z. Vasicek, V. Mrazek and L. S. Brno, "Towards low power approximate DCT architecture for HEVC standard," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Lausanne, 2017, pp. 1576-1581.

[30] H. A. F. Almurib, T. N. Kumar and F. Lombardi, "Approximate DCT Image Compression Using Inexact Computing," in *IEEE Transactions on Computers*, vol. 67, no. 2, pp. 149-159, Feb. 2018.

[31] A. Shmilovici, Y. Kahiri, I. Ben-Gal and S. Hauser, "Measuring the Efficiency of the Intraday Forex Market with a Universal Data Compression Algorithm," *Computational Economics, Springer*, vol. 33, issue 2, pp. 131-154, March 2009.

[32] D. Sculley and C. E. Brodley, "Compression and machine learning: a new perspective on feature space vectors," *Data Compression Conference (DCC'06)*, Snowbird, UT, 2006, pp. 332-341.

[33] S. Liu, K. Pattabiraman, T. Moscibroda and B. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning," *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XVI, ACM*, California, USA, 2011, pp. 213–224.

[34] A. Sampson, J. Nelson, K. Strauss and L. Ceze, "Approximate storage in solid-state memories," *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Davis, CA, 2013, pp. 25-36.

[35] H. Esmaeilzadeh, A. Sampson, L. Ceze and D. Burger, "Neural Acceleration for General-Purpose Approximate Programs," *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Vancouver, BC, 2012, pp. 449-460.

[36] L. Leem, H. Cho, J. Bau, Q. A. Jacobson and S. Mitra, "ERSA: Error Resilient System Architecture for probabilistic applications," *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, Dresden, 2010, pp. 1560-1565.

[37] Y. Liu, T. Zhang and K. K. Parhi "Computation error analysis in digital signal processing systems with overscaled supply voltage," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no.4, pp. 517–526, April 2010.

[38] D. Mohapatra, V. K. Chippa, A. Raghunathan and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," *2011 Design, Automation & Test in Europe*, Grenoble, 2011, pp. 1-6.

[39] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner and E. D. Berger "Eon: A language and runtime system for perpetual systems," *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, SenSys'07, ACM*, Sydney, Australia, 2007, pp. 161–174.

[40] 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 *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'11, ACM*, San Jose, California, 2011, pp. 164–174.

[41] M. Carbin, S. Misailovic and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA'13, ACM*, Indiana, USA, 2013, pp. 33–52.

[42] W. Baek and T. M. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," *ACM SIGPLAN Notices*, vol. 45, no. 6, pp. 198-209, June 2010.

[43] S. Misailovic, M. Carbin, S. Achour, Z. Qi and M. C. Rinard, "Chisel: Reliability and accuracy-aware optimization of approximate computational kernels," *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA'14*, Oregon, USA, 2014, pp. 309–328.

[44] S. Misailovic, S. Sidiroglou, H. Hoffmann and M. Rinard, "Quality of service profiling," *2010 ACM/IEEE 32nd International Conference on Software Engineering*, Cape Town, 2010, pp. 25-34.

[45] M. Samadi, D. Jamshidi, J. Lee and S. Mahlke, "Paraprox: Pattern-based approximation for data parallel applications," *Proceedings of the 19th International Conference*

on Architectural Support for Programming Languages and Operating Systems, ASPLOS'14, ACM, Utah, USA, 2014, pp. 35–50.

[46] M. Rinard, "Probabilistic accuracy bounds for fault-tolerant computations that discard tasks," *Proceedings of the 20th Annual International Conference on Supercomputing, ICS'06, ACM*, 2006, pp. 324–334.

[47] J. S. Miguel, M. Badr and N. E. Jerger, "Load Value Approximation," *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Cambridge, 2014, pp. 127-139.

[48] M. Sutherland, J. S. Miguel and N. E. Jerger, "Texture Cache Approximation on GPUs," *Workshop on Approximate Computing Across the Stack (WAX)*, Portland, Oregan, 2015.

[49] A. Rahimi, L. Benini and R. K. Gupta, "Spatial Memoization: Concurrent Instruction Reuse to Correct Timing Errors in SIMD Architectures," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 12, pp. 847-851, Dec. 2013.

[50] G. Keramidas, C. Kokkala and I. Stamoulis "Clumsy Value Cache: An Approximate Memoization Technique for Mobile GPU Fragment Shaders," *Workshop On Approximate Computing (WAPCO)*, Amsterdam, 2015.

[51] P. Kulkarni, P. Gupta and M. Ercegovac, "Trading Accuracy for Power with an Underdesigned Multiplier Architecture," *2011 24th Internatioal Conference on VLSI Design*, Chennai, 2011, pp. 346-351.

[52] J. Ma, K. Man, T. Krilavicius, S. Guan and T. Jeong, "Implementation of High Performance Multipliers Based on Apprxoimate Compressor Design", *International Conference on Electrical and Control Technologies (ECT)*, 2011.

[53] C.-H. Lin and C. Lin, "High accuracy approximate multiplier with error correction," in *IEEE 31st International Conference on Computer Design*, Asheville, NC, 2013, pp. 33-38.

[54] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris and K. Pekmestzi, "Design-efficient approximate multiplication circuits through partial product perforation," *IEEE Transactions on VLSI systems*, vol. pp , no. 99, pp. 1-13, 2016.

[55] S. A. White., "Applications of distributed arithmetic to digital signal processing: A tutorial review," in *IEEE ASSP Magazine*, vol. 6, pp. 4-19, Jul. 1989.

[56] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han and F. Lombardi, "Design of Approximate Radix-4 Booth Multipliers for Error-Tolerant Computing," in *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1435-1441, Aug. 1 2017.

[57] H. Jiang, J. Han, F. Qiao and F. Lombardi, "Approximate Radix-8 Booth Multipliers for Low-Power and High-Performance Operation," in *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2638-2644, Aug. 1 2016.

[58] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124-137, 2013.

[59] K. Du, P. Varman and K. Mohanram,, "High performance reliable variable latency carry select addition," 2012 *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2012, pp. 1257-1262.

[60] Ning Zhu, W. L. Goh and K. S. Yeo, "An enhanced low-power high-speed Adder For Error-Tolerant application," *Proceedings of the 2009 12th International Symposium on IC*, Singapore, 2009, pp. 69-72.

[61] 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 Trans. Circuits Syst. I: Reg. Papers*, vol. 57, no. 4, pp. 850-862, 2010.

[62] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Transactions on Computers,*, vol. 64, no. 4, pp. 984-994, 2015.

[63] N. Maheshwari, Z. Yang, J. Han and F. Lombardi, "A Design Approach for Compressor Based Approximate Multipliers," *2015 28th International Conference on VLSI Design*, Bangalore, 2015, pp. 209-214.

[64] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE Transactions on VLSI systems*, vol. 23, no. 6, pp. 1180-1184, 2015.

[65] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *2014 DATE Conference and Exhibition*, Dresden, 2014, pp. 1-4.

[66] R. Marimuthu, Y.E. Rezinold, and P. Mallick, "Design and Analysis of Multiplier Using Approximate 15-4 Compressor," *IEEE Access*, vol. 5, pp. 1027-1036, 2017

[67] J. Miao, K. He, A. Gerstlauer and M. Orshansky, "Modeling and synthesis of quality-energy optimal approximate adders," *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, 2012, pp. 728-735

[68] Khaing Yin Kyaw, Wang Ling Goh and Kiat Seng Yeo, "Low-power high-speed multiplier for error-tolerant application," *2010 IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC)*, Hong Kong, 2010, pp. 1-4.

[69] R. Venkatesan, A. Agarwal, K. Roy and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 667-673.

[70] S. J. Jou, M. Tsai, and Y. Tsao, "Low-error reduced-width Booth multipliers for DSP applications," *IEEE Transactions on Circuits and Systems*, vol. 50, no. 11, pp. 1470-1474, 2013.

[71] K. J. Cho, K. C. Lee, J. G. Chung and K. K. Parhi, "Design of low-error fixed-width modified Booth multiplier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 5, pp. 522-531, May 2004.

[72] J. Wang, S. Kuang and S. Liang, "High-Accuracy Fixed-Width Modified Booth Multipliers for Lossy Applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 1, pp. 52-60, Jan. 2011.

[73] Y. Chen and T. Chang, "A High-Accuracy Adaptive Conditional-Probability Estimator for Fixed-Width Booth Multipliers," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 3, pp. 594-603, March 2012.

[74] M. J. Schulte and E. E. Swartzlander, "Truncated multiplication with correction constant," *Proceedings of IEEE Workshop on VLSI Signal Processing*, Veldhoven, 1993, pp. 388-396.

[75] E. J. King and E. E. Swartzlander Jr., "Data dependent truncated scheme for parallel multiplication," in *Proc. 31st Asilomar Conf. Signals, Circuits Syst.*, 1998, pp. 1178-1182.

[76] Hong-An Huang, Yen-Chin Liao and Hsie-Chia Chang, "A self-compensation fixed-width Booth multiplier and its 128-point FFT applications," *2006 IEEE International Symposium on Circuits and Systems*, Island of Kos, 2006, pp. 4 pp.-3541.

[77] C. Y. Li, Y. H. Chen, T. Y. Chang and J. N. Chen, "A Probabilistic Estimation Bias Circuit for Fixed-Width Booth Multiplier and Its DCT Applications," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 4, pp. 215-219, April 2011.

[78] H. Jiang, J. Han, F. Qiao and F. Lombardi, "Approximate Radix-8 Booth Multipliers for Low-Power and High-Performance Operation," in *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2638-2644, Aug. 1 2016.

[79] N.H.E. Weste, K. Eshraghian, *Principles of CMOS VLSI Design*. Addison-Wesley Publishing Company, 1993.

[80] E. de Angel and E. E. Swartzlander, "Low power parallel multipliers," *VLSI Signal Processing*, IX, San Francisco, CA, 1996, pp. 199-208.

[81] N. Ahmed, T. Natarajan and K. R. Rao, "Discrete Cosine Transform," in *IEEE Transactions on Computers*, vol. C-23, no. 1, pp. 90-93, Jan. 1974.

[82] S. Venkatachalam, H. J. Lee and S. B. Ko, "Power Efficient Approximate Booth Multiplier," *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, Italy, 2018, pp. 1-4.

[83] Lingfeng Yuan, Sridhar Sana, Hardy J Pottinger and Vittal S Rao, "Distributed arithmetic implementation of multivariable controllers for smart structural systems," *IOPscience*, Jan. 2000.

[84] W. Li, J. B. Burr and A. M. Peterson, "A fully parallel VLSI implementation of distributed arithmetic," *IEEE International Symposium on Circuits and Systems*, Espoo, Finland, 1988, vol.2, pp. 1511-1515.

[85] R. Amirtharajah and A. P. Chandrakasan, "A micropower programmable DSP using approximate signal processing based on distributed arithmetic," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 2, pp. 337-347, Feb. 2010.

[86] S. W. Heo, "Power Optimization of Sum-of-Products Design in Signal Processing Applications," Ph. D Thesis, University of California, 2014 .

[87] D. Esposito, A. G. M. Strollo and M. Alioto, "Low-power approximate MAC unit," *2017 13th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, Giardini Naxos, 2017, pp. 81-84.

[88] S. F. Obermann and M. J. Flynn, "Division algorithms and implementations," in *IEEE Transactions on Computers*, vol. 46, no. 8, pp. 833-854, Aug 1997.

[89] M. D. Ercegovac and J. Muller, "Digit-recurrence algorithms for division and square root with limited precision primitives," *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, Pacific Grove, CA, USA, 2003, pp. 1440-1444 Vol.2.

[90] M. Ercegovac and T. Lang, "Digital Arithmetic". Morgan Kaufmann Publishers, 2004.

[91] M.J. Schulte, J. Omar, and E.E. Swartzlander, "Optimal Initial Approximations for the Newton-Raphson Division Algorithm," *Computing*, vol. 53, pp. 233-242, 1994.

[92] B. Parhami, "Computer Arithmetic- Algorithms and Hardware designs", published by Oxford University Press, 2000.

[93] S. Hashemi, R. I. Bahar and S. Reda, "A low-power dynamic divider for approximate applications," *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX, 2016, pp. 1-6.

[94] L. Chen, J. Han, W. Liu and F. Lombardi, "On the Design of Approximate Restoring Dividers for Error-Tolerant Applications," in *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2522-2533, Aug. 1 2016.

[95] L. Chen, W. Liu, J. Han, P. A. Montuschi and F. Lombardi, "Design, Evaluation and Application of Approximate High-Radix Dividers," in *IEEE Transactions on Multi-Scale Computing Systems.* Accepted for inclusion in a future issue of this journal.

[96] H. Jiang, L. Liu, F. Lombardi and J. Han, "Adaptive approximation in arithmetic circuits: A low-power unsigned divider design," *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2018, pp. 1411-1416.

[97] R. Zendegani, M. Kamal, A. Fayyazi, A. Afzali-Kusha, S. Safari and M. Pedram, "SEERAD: A high speed yet energy-efficient rounding-based approximate divider," *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2016, pp. 1481-1484.

[98] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Transactions on Computers*, vol. 63, no. 9, pp. 1760-1771, 2013.

[99] S. Suman, F. A. Hussain, A. S. Malik, N. Walter, K. L. Goh, I. Hilmi, and S. Ho, "Image enhancement using geometric mean filter and gamma correction for WCE iamges," in *21st International Conference, Neural Information Processing, ICONIP, Springer*, 2014, pp. 276-283.

[100] S. Venkatachalam and S. B. Ko, "Design of Power and Area Efficient Approximate Multipliers," *IEEE Transactions on VLSI systems*, vol. 25, no. 5, pp. 1782-1786, May 2017.

[101] J. Babaud, A. P. Witkin, M. Baudin and R. O. Duda, "Uniqueness of the Gaussian Kernel for Scale-Space Filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 1, pp. 26-33, Jan. 1986.

[102] S. F. Obermann and M. J. Flynn, "Division algorithms and implementations," in *IEEE Transactions on Computers*, vol. 46, no. 8, pp. 833-854, Aug 1997.

[103] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436-444, May 2015.

[104] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 770-778.

[105] K. Simonyan and A. Zisserman "Very deep convolutional networks for large-scale image recognition," arXiv:1409.1556, 2014.

[106] A. A. A. Setio et al., "Pulmonary Nodule Detection in CT Images: False Positive Reduction Using Multi-View Convolutional Networks," in emphEEE Transactions on Medical Imaging, vol. 35, no. 5, pp. 1160-1169, May 2016.

[107] K. M. Hermann et al., "Teaching Machines to Read and Comprehend," arXiv:1506.03340, 2015.

[108] M. A. Alsheikh, D. Niyato, S. Lin, H. P. Tan and Z. Han, "Mobile big data analytics using deep learning and apache spark," *IEEE Network*, vol. 30, no. 3, pp. 22-29, May-June 2016.

[109] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," arXiv:1609.08144, 2016.

[110] W. Xiong et al., "The microsoft 2016 conversational speech recognition system," *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, LA, 2017, pp. 5255-5259.

[111] https://medium.com/@libreai/a-glimpse-into-deep-learning-for-recommender-systems-d66ae0681775

[112] R. K. Srivastava, K, Greff and J. Schmidhuber "Training Very Deep Networks," arXiv:1507.06228, 2015.

[113] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, Y. Bengio, "FitNets: Hints for Thin Deep Nets", arXiv:1412.6550, 2014.