Improving the Hardware Performance of Arithmetic Circuits using Approximate Computing

A Thesis Submitted to the
College of Graduate and Post Doctoral Studies
in Partial Fulfillment of the Requirements
for the Degree of Master's
in the Department of Electrical and Computer Engineering
University of Saskatchewan
Saskatoon

By
Elizabeth Adams

Permission to Use

In presenting this thesis/dissertation in partial fulfillment 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/dissertation in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis/dissertation 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/dissertation 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/dissertation.

Requests for permission to copy or to make other uses of materials in this thesis/dissertation 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
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9
Canada

Abstract

An application that can produce a useful result despite some level of computational error is said to be error resilient. Approximate computing can be applied to error resilient applications by intentionally introducing error to the computation in order to improve performance, and it has been shown that approximation is especially well-suited for application in arithmetic computing hardware. In this thesis, novel approximate arithmetic architectures are proposed for three different operations, namely multiplication, division, and the multiply accumulate (MAC) operation. For all designs, accuracy is evaluated in terms of mean relative error distance (MRED) and normalized mean error distance (NMED), while hardware performance is reported in terms of critical path delay, area, and power consumption.

Three approximate Booth multipliers (ABM-M1, ABM-M2, ABM-M3) are designed in which two novel inexact partial product generators are used to reduce the dimensions of the partial product matrix. The proposed multipliers are compared to other state-of-the-art designs in terms of both accuracy and hardware performance, and are found to reduce power consumption by up to 56% when compared to the exact multiplier. The function of the multipliers is verified in several image processing applications.

Two approximate restoring dividers (AXRD-M1, AXRD-M2) are proposed along with a novel inexact restoring divider cell. In the first divider, the conventional cells are replaced with the proposed inexact cells in several columns. The second divider computes only a subset of the trial subtractions, after which the divisor and partial remainder are rounded and encoded so that they may be used to estimate the remaining quotient bits. The proposed dividers are evaluated for accuracy and hardware performance alongside several benchmarking designs, and their function is verified using change detection and foreground extraction applications.

An approximate MAC unit is presented in which the multiplication is implemented using a modified version of ABM-M3. The delay is reduced by using a fused architecture where the accumulator is summed as part of the multiplier compression. The accuracy and hardware savings of the MAC unit are measured against several works from the literature, and the design is utilized in a number of convolution operations.

Acknowledgements

I would like to express my sincere appreciation to my supervisor Dr. Seok-Bum Ko for his continuous support throughout the course of my M.Sc. studies. His immense knowledge and experience in the field has given him an excellent eye for worthwhile research opportunities, and I can not thank him enough for the patience, compassion, and encouragement he has shown me. Without his guidance, this dissertation would not have been possible.

I would like to extend a special thanks to Suganthi Venkatachalam for her role as an excellent research partner. I deeply appreciate the patience she expressed while I was becoming oriented with our research as a new graduate student, and I am indebted for the countless occasions on which she took time out of her day to lend me a helping hand. It was because of her warmth that my transition to becoming a graduate student was so much less intimidating. I would also like to express sincere gratitude to my labmate Hao Zhang for his vast technical knowledge and the prompt assistance he always gave me when troubleshooting issues in the lab. My labmates have continuously taken the extra step to help me out when I need it, and I can not overstate how much I appreciate their support.

I thank my mom, my dad, my sister, and my boyfriend for their endless patience throughout my studies. It goes without saying that research can impose a sense of intimidation and frustration from time to time. I am so incredibly grateful for the support provided by my family and friends during those times that I was feeling under-confident. It was because of their encouraging words that I was also able to experience the joy and accomplishment of seeing my research through.

My sincere thanks to the Natural Sciences and Engineering Research Council of Canada (NSERC), the R&D program of the Korean Ministry of Trade, Industry and Energy (MOTIE), the Korea Institute for Industrial Economics and Trade (KIET), and the Department of Electrical and Computer Engineering, University of Saskatchewan.

To to all those who are patient and persistent in encouraging me,
you have shown me that I am much more capable than I thought myself to be.

Contents

List of Tables

List of Figures

List of Abbreviations

AAXD	Adaptive approximation-based divider [79].
ABM-M1	Approximate Booth multiplier model 1.
ABM-M2	Approximate Booth multiplier model 2.
ABM-M3	Approximate Booth multiplier model 3.
ABM-MAC	Approximate Booth multiplier-based MAC.
ABM1	Approximate Booth multiplier 1 [71].
ABM2-C9	Approximate Booth multiplier 2 with 9-bit truncation [71].
ADP	Area-delay product.
AP-MAC	Approximate MAC [87].
APP	Area-power product.
ARL	Approximate recoded logic.
ATC-MAC	Approximate tree compressor-based MAC [81].
AXDr	Approximate restoring divider [77].
AXDr1	Approximate restoring divider 1 [77].
AXDr2	Approximate restoring divider 2 [77].
AXDr3	Approximate restoring divider 3 [77].
AXHD	Approximate hybrid divider [66].
AXRD-M1	Approximate restoring divider model 1.
AXRD-M2	Approximate restoring divider model 2.
AXRDC	Approximate restoring divider cell.
CLA	Carry-lookahead adder.
CSA	Carry-select adder.
$\mathbf{D_{max}}$	Maximum error distance.
DRAM	Dynamic RAM.
ED	Error distance.
ER	Error rate.
EXRDC	Exact restoring divider cell.
FIR	Finite impulse response.
FPGA	Field-programmable gate array.
FS	Full substractor.

ISA Instruction set architecture.
LM Logarithmic multiplier.
LOD Leading-one detector.
LSB Least-significant bit.
MAC Multiply accumulate.
MADPP MRED-ADP product.
MAE Mean absolute error.
MBE Modified Booth encoder.
MED Mean error distance.
MPDPP MRED-PDP product.
MRED Mean relative error distance.
MSB Most-significant bit.
MSE Mean squared error.
NADPP NMED-ADP product.
NMED Normalized mean error distance.
NOD Nearest-one detector [64].
NPDPP NMED-PDP product.
NPU Neural processing unit.
PDP Power-delay product.
PPG Partial product generator.
PPG-1S 1-signal partial product generator.
PPG-2S 2-signal partial product generator.
PSNR Peak signal-to-noise ratio.
R4ABM Approximate radix-4 Booth multiplier [70].
RAD256 Approximate multiplier with radix-256 LSB encoding [72].
SSIM Structural similarity index measurement.
VOS Voltage overscaling.

Chapter 1

Introduction

This chapter provides an introduction to the topics discussed in this thesis. Section 1.1 presents the motivation behind the research, Section 1.2 features a high-level survey of approximate computing techniques, Section 1.3 describes the novel research contributions presented in this thesis, Section 1.4 lists the publications and submissions made during the course of the M.Sc. study, and Section 1.5 details the organization of the thesis.

1.1 Motivation

While energy consumption has always been an important metric in the design of computing systems, a number of recent trends have led power efficiency to become the paramount concern in the field. Computing systems are becoming increasingly mobile, and there is a strong demand for high performance computing on power-constrained devices. Even in the context of more traditional devices that do not rely on battery power, emerging high performance applications, such as multimedia streaming and machine learning, require increasingly large datasets to be processed with high efficiency.

1.1.1 Trends in Semiconductor Scaling

In 1965, Gordon Moore observed that the number of transistors per integrated circuit was doubling every year, and he predicted that this trend would continue for at least the next decade [1]. A decade later and the co-founder of Intel, Moore revised his forecast to project that the doubling would occur every two years [2]. Now widely known as Moore's law, his projection has held strong for decades, perhaps having acted as a sort of self-fulfilling prophecy due to the fact that it has been widely used as a guide for establishing timelines and setting targets in the semiconductor industry. In 1974, Robert Dennard observed that power density remained constant as transistor dimensions shrank, a phenomenon referred to as Dennard scaling [3]. Dennard proposed a method of scaling voltage and current proportional to transistor dimensions, thereby yielding higher speed and reduced power consumption. It is crucial to note that Moore's law alone does not provide any insight into performance. While Moore's law predicts that transistors will shrink, it is

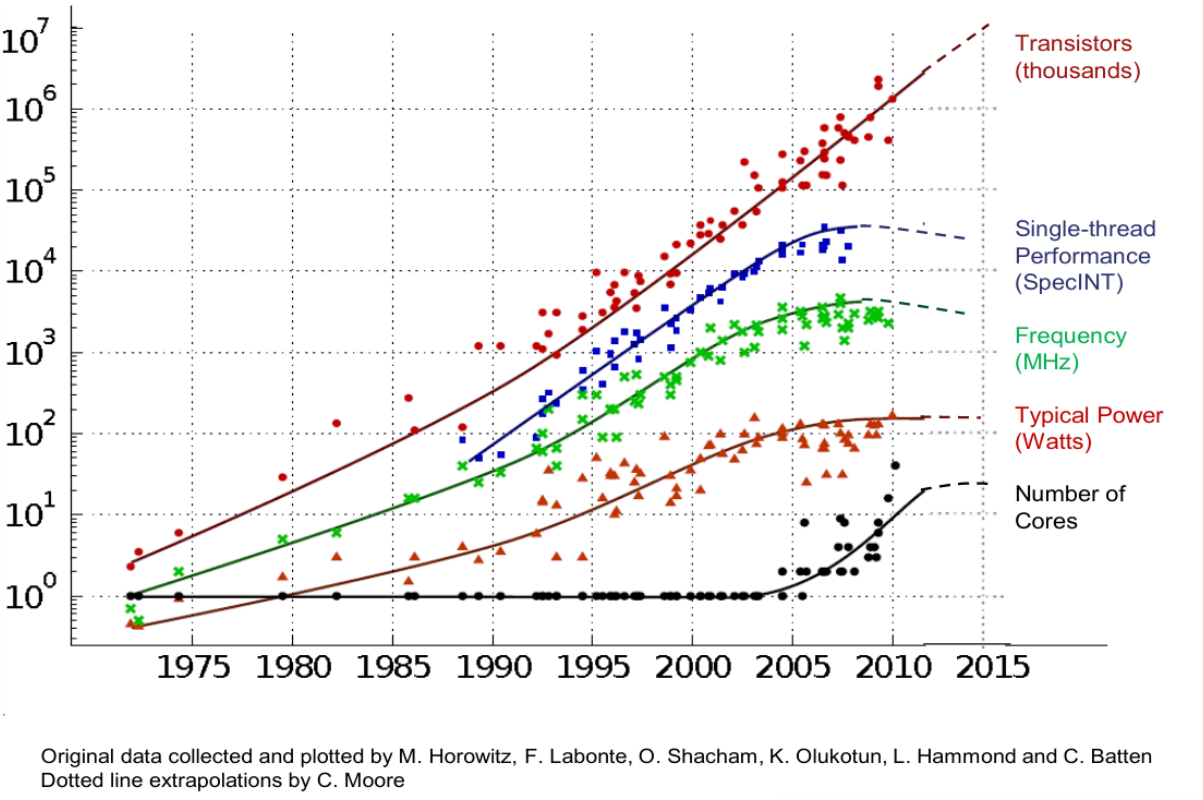


Fig. 1.1. Decay of Dennard scaling in recent years. [5]

Dennard scaling that connects the reduction of transistor dimensions with smaller energy consumption and higher clock frequencies.

The scaling of transistor density, clock frequency, and power consumption from 1975 to 2015 is shown in Fig. 1.1. For many decades, Dennard scaling allowed for increasing clock frequencies and decreasing power consumption to match the rate of Moore's law. However, as visible from Fig. 1.1, Dennard scaling began to break down in 2005, despite the fact that transistors are still shrinking. The cause of this breakdown lies in an oversight made by Dennard when formulating his scaling technique, in which he assumed that threshold voltage would scale with operating voltage. While sub-threshold voltage leakage was rather negligible in 1974, transistors have now scaled to the point where sub-threshold leakage has a major impact on overall chip power, thereby presenting serious challenges with regards to further reducing operating voltage [4]. As a result, while transistor sizes continue to shrink in accordance with Moore's law, we can no longer rely on performance gains provided by Dennard scaling, and thus transistor size is no longer a reliable representative of single-core computing performance.

Even if sub-threshold leakage can be addressed, there are additional issues associated with the fact that performance improvements under Dennard scaling intrinsically rely on Moore's law. While there is still much debate as to how quickly the rate of progress will saturate, transistor dimensions are already approaching atomic measurements and can only become so small before they are fundamentally limited by physical constraints [6].

In a 2005 interview with Techworld, Moore confirmed that the eventual obsolescence of the law is unavoidable:

> It can't continue forever. The nature of exponentials is that you push them out and eventually disaster happens.
>
> In terms of [transistor] size you can see that we're approaching the size of atoms which is a fundamental barrier, but it'll be two or three generations before we get that far—but that's as far out as we've ever been able to see. We have another 10 to 20 years before we reach a fundamental limit. By then they'll be able to make bigger chips and have transistor budgets in the billions. [7]

The International Technology Roadmap for Semiconductors has used Moore's law as a primary driving force behind their industry roadmaps since 1998. However, in 2016, with the general acknowledgement that Moore's law is slowing down, a final roadmap was published and the organization was succeeded by the International Roadmap for Devices and Systems, whose aim is to provide a more generalized approach to roadmapping [8], [9].

Historically, the semiconductor industry has considered transistor miniaturization to be among the most effective design paradigms in terms of generally improving computing power. While there is still some room to further shrink transistors, the breakdown of Dennard scaling and deceleration of Moore's law indicate that we must look beyond transistor scaling if we hope to continue driving performance. This presents a unique opportunity in that there is an unprecedented push to explore alternate methods of improving computational performance. This push to develop new techniques for improving performance has been a significant driving force behind the emergence of approximate computing as a field of research.

1.1.2 Trends in Data Processing Applications

An application is said to be error resilient if it can produce useful results despite some level of computational error. In [10], several application characteristics corresponding to error resilience are identified. The ability to handle input noise is indicative of error resilience, due to the fact that noise is essentially the consistent occurrence of small, unpredictable errors. Similarly, the ability to handle redundant inputs also improves error resilience, as it suggests that certain computations may be entirely skipped over without significant issue. The absence of a unique golden output also corresponds to error resilience, where a perfect output may be lacking due to the occurrence of multiple equally-desirable outcomes, or because the optimality of the result is inherently unknown. Applications also tend to be error resilient if their result is interpreted by limited human perception, e.g. a computer will analyze an image and detect erroneous pixels with much greater speed and accuracy

than a human. Finally, an application may be error resilient due to certain algorithmic features that favor the mitigation of errors. For example, any application implementing iterative refinement will be error resilient because errors generated are naturally reduced by the algorithm.

The decreasing cost of storage, commodification of data sensors, and widespread use of complex computational models for a variety of applications has lead to the emergence of data-intensive computing as a field of research. Popular applications involving large datasets include multimedia processing, data mining, data analytics, neural networks, and computer vision. In [11], Gorton defines data-intensive computing as "managing, analyzing, and understanding data at volumes and rates that push the frontier of current technologies." By definition, conventional processing techniques tend to be ineffective in handling data-intensive applications, and thus new approaches must be explored. Many data-intensive applications are error resilient since input noise and redundant data are so common, and the lack of a golden output is also not unusual. Additionally, applications such as machine learning make use of highly iterative techniques, and therefore exhibit a high degree of error tolerance. The prevalence of error resilient applications has been a significant driving factor behind the development of approximate computing techniques, and the unique challenges associated with data-intensive processing deepens this motivation.

1.2 Approximate Computing as a Field of Research

Approximate computing in the most general sense involves the intentional introduction of error to a computation in order to improve performance. This section provides a survey of approximation techniques applied at all levels of the computing hierarchy. Software-level techniques are discussed in Section 1.2.1, architecture-level techniques are reviewed in Section 1.2.2, and circuit-level techniques are examined in Section 1.2.3. Because this thesis focuses on the application of approximate computing to arithmetic hardware, a study of these techniques is omitted from Section 1.2.3, and a comprehensive literature review is instead provided in Chapter 2.

1.2.1 Software-Level Techniques

Software approximation encompasses a wide variety of methods at varying levels of abstraction. At the higher level, approximation-aware programming languages enable the programmer to directly indicate acceptable levels of accuracy for various computations executed in their program. In the programming language Eon [12], paths through a program occupy different energy states set by the programmer, which generally correspond to rate of execution priority. Eon's automatic energy management then dynamically adapts these states according to currently available and predicted energy levels. In [13], a Java

extension EnerJ uses type qualifiers to declare data for which approximate computations may be performed. These type qualifiers indicate to the system how the data should be handled, where approximate data is mapped to low-power storage, operated on using low-power computations, etc. The programming language Rely proposed in [14] supports quantitative reliability specifications for results produced by a function, which essentially defines the minimum acceptable reliability for the function to be called. A static quantitative reliability analysis is used to verify the quantitative requirements on the reliability of a program, effectively verifying that the program satisfies its reliability specification when executed on the underlying unreliable hardware.

Approximate computing can also be applied in software at lower levels of abstraction. Loop perforation is a technique in which computation loops are transformed to execute only a subset of their iterations. In [15], a criticality testing phase filters out all critical loops to identify tunable loops, i.e. any loop whose perforation does not result in unacceptable accuracy degradation. Loop perforation is also utilized in [16], in which the accuracy loss produced by approximation is modelled and subsequently used to make approximation decisions, such as determining whether to terminate a loop early. In parallel systems, relaxed synchronization introduces approximation by waiving some of the synchronization requirements preventing concurrent accesses to shared data. Relaxed synchronization is used in [17] to improve per-core utilization for the computation of large-scale quadratic programming problems by relaxing the requirement that full synchronization be achieved after each iteration and additionally permitting mid-iteration synchronization.

1.2.2 Architecture-Level Techniques

Approximate computing can be applied at the architecture-level in a variety of ways. Memoization is an instruction-reuse technique in which the result of an instruction or group of instructions is stored so that it may be reused in the case of an identical instruction call. However, the storage of these results requires resources and therefore it is only beneficial if there is a certain level of data reuse. In fuzzy memoization, stored data may be reused not only in the case of identical inputs but also for similar inputs, allowing for a greater level of data reuse. The tolerance of multimedia applications is exploited in [18], where fuzzy memoization is used for floating-point operations, resulting in energy improvements of 12%. In the case of a load miss in a private cache, data typically needs to be retrieved from main memory or higher-level caches, resulting in additional energy consumption and latency. These costs can be mitigated using a technique called load value approximation [19] in which value patterns are learned so that an approximate data value may be generated in the case of a load miss, thereby avoiding the need to stall during the time it takes to fetch the data and improving speedup by an average of 8.5%.

While a variety of approximate accelerators have been proposed for use in specialized applications, the development of neural network accelerators has received notable atten-

tion. A general-purpose limited-precision code accelerator for error tolerant applications is introduced in [20] in which approximable regions of code are automatically transformed from a von Neumann model to an analog neural model. An approximate computing framework for artificial neural networks is proposed in [21] in which the impact of neurons on the output quality is characterized so that the computations and memory accesses for less critical neurons may be approximated. In [22], the Parrot transformation is presented which selects and trains a neural network to mimic a region of imperative code. Once the learning stage is complete, the compiler replaces the code with an instance of a low power accelerator, i.e. a neural processing unit (NPU). A new approach to implementing hardware-efficient large-scale neural networks is presented in [23] in which neural networks are approximated by firstly adapting the backpropagation technique to indicate the impact of approximating a given neuron, and secondly approximating the neurons found to have the smallest impact on output quality. In [24], an accelerator is used to address the issue of branch divergence in single instruction, multiple data architectures by training neural networks offline to approximate those regions of code with degraded performance due to branch divergence, and subsequently replacing the code regions with their neural network approximations. In [25], approximate programs are accelerated via a NPU implemented in a field-programmable gate array (FPGA), where the proposed accelerator is designed for use with a compiler workflow that automatically configures the neural network topology and weights, rather than the programmable logic itself.

A number of approximate general-purpose processors have been proposed in the literature. These architectures generally provide some degree of configurability as to allow for use not only in error-tolerant applications, but also generic applications with stricter accuracy requirements. An error resilient system architecture is proposed in [26], where high error resilience is achieved by mixing processor cores with varying reliability levels, utilizing error resilient algorithms at the core of probabilistic applications, and performing intelligent software optimizations. In [27], the minimum error-protection required for streaming applications is analyzed, and microarchitectural techniques are proposed to mitigate errors arising in a general-purpose processor built from an unreliable fabric. A processor is designed in [28] for use in recognition and mining, where a 2-D array of processing elements, a streaming memory hierarchy, and an interconnect network allow for the efficient execution of dominant computational kernels from a wide range of recognition and mining applications. This work is extended in [29] in which the authors present an automatic resilience characterization framework designed to quantitatively evaluate the intrinsic error resilience of a given application. This framework is used alongside accuracy-scalable hardware that can be dynamically configured at runtime according to application requirements and data characteristics. The authors extend this work again in [30], where they further refine the scalable effort design approach, stating that mechanisms for modulating computational effort should be sought at all levels of design abstraction,

emphasizing that maximal exploitation of error resilience requires the cross-layer optimization of scaling mechanisms identified at all layers of abstraction. An energy efficient, quality programmable vector processor is proposed in [31], where the notion of quality is explicitly codified in the instruction set architecture (ISA) via the use of instructions for which an associated quality field indicates the minimum level of accuracy that must be met during execution. In [32], the authors observe that the memory hierarchy tends to be overlooked in favor of applying approximation in the data path. In order to realistically model the relationship between performance improvements and accuracy, errors occurring both along the data path and in the memory hierarchy are analyzed using an energy model that expresses its findings in terms of operations per virtual Joule, a relative metric that is independent of implementation technology.

1.2.3 Circuit-Level Techniques

While the majority of circuit-level approximation techniques feature inexact arithmetic units as explored in Chapter 2, there are several other methods worth discussing here. While dynamic RAM (DRAM) memory tends to be fast, the storage capacitor within each memory cell must be periodically recharged for it to maintain its data value, resulting in additional power usage. Approximation can be applied by reducing the rate of DRAM refresh, thereby lowering power consumption at the risk of corrupting data stored in the DRAM cells. An application-level technique is proposed in [33] which allows developers to specify critical and non-critical data for their programs, where the storage location for the data is selected by the runtime system accordingly. The refresh rate for the memory containing non-critical data can then be reduced, resulting in lowered power consumption. In [34], embedded DRAM-based frame buffers are segmented into four tiers, each with a different refresh rate, reducing power consumption by an impressive 48% while maintaining adequate performance. Voltage overscaling (VOS) is a technique in which the supply voltage for a given circuit is reduced without modifying the operational frequency, lessening power consumption at the risk of analog values. An early example of approximate computing can be found in [35], where a digital signal processing block is approximated by pairing VOS with algorithmic noise tolerance schemes which perform error compensation. In [36], several meta-functions are characterized as computational kernels common to error resilient applications. An assortment of design techniques are utilized to allow the hardware implementation for these meta-functions to scale more gracefully under VOS. In [37], hardware for a proposed ISA extension utilizes dual-voltage operation, where high voltage is used for precise computations, and low voltage is used for inexact operations.

1.3 Contributions of the Thesis

The main contributions of this thesis are as follows:

1. **Approximate Booth multipliers**
   Two approximate partial product generators PPG-1S and PPG-2S are proposed which utilize only one or two encoding signals, respectively. The first multiplier ABM-M1 utilizes PPG-2S to approximate the partial products in its least-significant columns. The second multiplier ABM-M2 uses PPG-2S to replace the least-significant partial products of each row with a signal approximate partial product. The third multiplier ABM-M3 utilizes PPG-1S to replace all bits in each row, for a significance below a certain value, with a single approximate partial product. The designs are evaluated in terms of accuracy and hardware, and their operation is verified using several image processing applications.

2. **Approximate restoring dividers**
   Two approximate restoring dividers are proposed, where the first design AXRD-M1 utilizes a novel approximate restoring divider cell in its least-significant columns to reduce hardware cost. The second divider AXRD-M2 eliminates several rows from the cell array such that only a subset of the trial subtractions are computed, after which the partial remainder the divisor are rounded and encoded in order to estimate the remaining quotient bits. The accuracy and hardware performance of the dividers is investigated, and their functionality is verified using change detection and foreground extraction applications.

3. **Approximate MAC unit**
   The Booth multiplier ABM-M3 is used as a basis for designing an approximate Booth multiplier-based MAC (ABM-MAC) unit, where the critical path delay is reduced by fusing the accumulation with the multiplier compression. The accuracy and hardware performance of ABM-MAC are evaluated alongside several other approximate MAC units for benchmarking purposes. The accuracy-performance tradeoff is additionally evaluated using combined accuracy-hardware metrics, and the proposed design is functionally verified using a convolution application.

1.4 Publications and Submissions during M.Sc. Study

1.4.1 Published Journal

1. S. Venkatachalam, E. Adams, H. J. Lee, and S. Ko, "Design and analysis of area and power efficient approximate Booth multipliers," *IEEE Trans. Comput.*, vol. 68, no. 11, pp. 1697–1703, 2019. DOI: 10.1109/TC.2019.2926275

   A Major portion of this paper is included in Chapter 3: Approximate Booth Multipliers.

2. E. Adams, S. Venkatachalam, and S. Ko, "Approximate restoring dividers using inexact cells and estimation from partial remainders," *IEEE Trans. Comput.*, vol. 69, no. 4, pp. 468–474, 2020. DOI: 10.1109/TC.2019.2953751

   A Major portion of this paper is included in Chapter 4: Approximate Array-Based Restoring Dividers.

1.4.2 Published Conference

1. E. Adams, S. Venkatachalam, and S. Ko, "Energy-efficient approximate MAC unit," in *Proc. 2019 IEEE Int. Symp. Circuits and Syst. (ISCAS)*, 2019, pp. 1–4. DOI: 10.1109/ISCAS.2019.8701880

   A Major portion of this paper is included in Chapter 5: Approximate Fixed-Point MAC Unit.

2. S. Venkatachalam, E. Adams, and S. Ko, "Design of approximate restoring dividers," in *Proc. 2019 IEEE Int. Symp. Circuits and Syst. (ISCAS)*, 2019, pp. 1–5. DOI: 10.1109/ISCAS.2019.8702363

   A Major portion of this paper is included in Chapter 4: Approximate Array-Based Restoring Dividers.

1.4.3 Submitted Journal

1. E. Adams, S. Venkatachalam, H. J. Lee, and S. Ko, "Convolution using an approximate radix-4 Booth multiplier-based MAC unit," *IEEE Trans. Circuits Syst. I*, 2020, submitted.

   A Major portion of this paper is included in Chapter 5: Approximate Fixed-Point MAC Unit.

1.5 Organization of the Thesis

The thesis is organized as follows:

- **Chapter 1: Introduction** describes the motivation behind the research presented in this thesis, the history of approximate computing as a field of research, the contributions provided in this thesis, the publications and submissions made during the M.Sc. study, and the organization of this thesis.
- **Chapter 2: Review of Contemporary Approximate Arithmetic Designs** provides a review of contemporary works in the literature which feature approximate arithmetic hardware. The approximate designs reviewed include adders, multipliers, dividers, and MAC units.
- **Chapter 3: Low-Power Approximate Booth Multipliers** presents three inexact radix-4 Booth multipliers which utilize approximation techniques in the partial product generation stage via the use of two novel inexact partial product generators (PPGs).
- **Chapter 4: Approximate Array-Based Restoring Dividers** proposes two inexact array-based restoring dividers in which approximation is applied using inexact divider cells as well as quotient estimation from rounded and encoded values.
- **Chapter 5: Approximate Fixed-Point MAC Unit** presents an inexact MAC unit which utilizes one of the PPGs introduced in Chapter 3 alongside other approximation techniques.
- **Chapter 6: Conclusions and Future Work** provides a summary of this thesis and discusses potential future works.

Chapter 2

Review of Contemporary Approximate Arithmetic Designs

This chapter provides an overview of approximate arithmetic hardware in the literature. Approximate adders are reviewed in Section 2.1, inexact multipliers are surveyed in Section 2.2, inexact dividers are discussed in Section 2.3, and approximate MAC units are examined in Section 2.4. For each type of arithmetic unit, the most prominent approximate designs in the literature are explored.

2.1 Approximate Adders

The half-adder and full-adder architectures are simple and make use of few logic gates. Thus, the works in the literature that propose approximate half- and full-adder designs generally involve modifying the adder architecture at the transistor-level to improve hardware efficiency. In [43], the logical complexity of the conventional mirror adder is reduced to generate four approximate designs exhibiting a substantial reduction in power dissipation. Similarly, approximate *XOR*/*XNOR*-based adders are introduced in [44], where the adder designs are based on *XOR*/*XNOR*-gates modified at the transistor-level for which the multiplexer operation is implemented using pass transistors.

A wide variety of inexact multi-bit adders utilizing gate-level approximation can be found in the literature. Considering the addition of n-bit operands X and Y, a conventional parallel adder generally computes a given carry-out c_i as a function of all previous input bits, i.e. $c_i = f(x_{i-1}, y_{i-1}, \ldots, x_0, y_0)$, meaning that the critical path delay of the adder is proportional to $\log n$. One of the earliest approximate adders in the literature is a modified parallel adder introduced by Lu et al. in [45], for which the carry of a given stage considers only up to the previous k inputs, i.e. $c_i = f(x_{i-1}, y_{i-1}, \ldots, x_{i-k}, y_{i-k})$ where $0 \leq k \leq i + 1$. Consequently, the critical path delay of the adder is proportional to $\log k$, and the selection of $k = \sqrt{n}$ reduces the delay by half. A fast inexact adder based on the carry-lookahead adder (CLA) architecture is proposed in [46], where a kill signal indicates that the carry bit be set to zero in the case that both input bits are zero, resulting in the

dependence of the current carry on the previous carry only if neither kill nor generate are high. This inexact adder is then used as a component in a reliable variable latency adder which utilizes error detection and error recovery to guarantee a correct result. In [47], an inexact adder is proposed in which the addition is split so that the k most-significant bits are computed using a precise adder and the remaining $n-k$ bits are computed in parallel by OR-ing the respective input bits.

Numerous approximate adders in the literature make use of segmentation, a technique in which the inputs of a parallel addition are divided into k-bit segments and the carry-out for a given segment is computed using only its k input bits. Zhu et al. [48] introduce an error tolerant adder whose inaccurate part is comprised of a carry-free addition block and a control block, where the carry-free addition utilizes a modified XOR-gate. In [49], this work is extended with the proposal of a second adder architecture in which the carry propagation path is split into several segments over which carry propagation is performed concurrently. A modified version of this design is also presented in [49], where the carry generators of each segment are cascaded as to improve accuracy in the most-significant block at the expense of increased delay. This work is again extended in [50], where a third error tolerant adder utilizes a selector circuit to determine the number of bits to be divided into the accurate and inaccurate parts. Zhu et al. extend this work a final time in [51] by proposing a carry-select adder (CSA)-based design in which the addition is segmented into k-bit blocks whose carry chains are each broken into two stages, thereby improving the accuracy of the most-significant block without incurring additional delay. In [52], a novel function speculation technique is proposed which takes advantage of the low probability of long carry chains. The addition operands are segmented into windows of k consecutive input bits, where the carry-out of each window is speculated using only the k input bits of the window. A variable latency CSA is also proposed in [52], in which a fast addition is performed by utilizing speculation-based adders.

The literature also features adder designs in which accuracy can be configured at runtime. In [53], the proposed accuracy-configurable adder is divided into four stages, i.e. an approximate addition followed by three error correction stages, where the error correction stages can be selectively enabled or disabled to provide different levels of computational accuracy. In [54], an approximate adder is proposed where the addition operands are split into k-bit segments each implemented by a k-bit adder. The adder units are connected using multiplexers which select a carry-in from either the previous adder unit or the carry-in prediction component of the given adder unit, allowing for runtime configurability.

2.2 Approximate Multipliers

The application of approximate computing to multipliers has been widely explored in the literature. Section 2.2.1 discusses approximate multiplier designs derived from the AND-

array architecture, Section 2.2.2 reviews logarithmic multipliers, Section 2.2.3 surveys approximate Booth multipliers, and Section 2.2.4 explores multipliers utilizing approximate compression techniques.

2.2.1 Approximate *AND*-Array Multipliers

For an n-bit multiplication $X \times Y$, the *AND*-array multiplier architecture produces a partial product matrix for which the partial product bit pp_{ij} corresponding to the jth bit of the ith partial product is generated using a single *AND*-gate to compute $x_i y_j$. In [55], an error tolerant multiplier is proposed in which the input operands are split into a higher-order part and a lower-order part. A reduced width multiplier is used to compute the product of the higher-order bits, while the remaining product bits are approximated according to the position of the leading-one within the lower-order bits of each operand. The carry propagation path in the resulting architecture is limited to that of the higher-order part, and the reduction of the partial product matrix results in substantial power savings. An underdesigned multiplier architecture is proposed in [56], where an inaccurate 2×2 multiplier block is used to implement larger multipliers. Tunable accuracy can then achieved by selectively replacing inaccurate 2×2 blocks with the accurate version. In [57], a novel inaccurate 4:2 counter is used to build an approximate 4×4 Wallace multiplier. The 4×4 multiplier can then be used to build arbitrarily large multipliers for which power consumption is substantially reduced. An accurate version of the multiplier is also proposed, where error detection and error correction compensate fully for all inaccuracies while only marginally increasing area and power consumption. Similarly, an approximate Wallace-tree multiplier is proposed in [58] in which a $2n \times 2n$ multiplication is implemented using an accurate $n \times n$ multiplier for the most-significant block and three inaccurate $n \times n$ multipliers for the remaining blocks. Additionally, carry-in precomputation is used to exploit the fact that the occurrence of at least two 1s in the critical column of the partial product matrix results in a carry of at least 1 being propagated to the next column. A dynamic segmentation method proposed in [59] extracts a continuous k-bit segment from an n-bit operand for $k \geq n/2$, where the segment may only start from one of several fixed positions dependent on the location of the leading-one in the operand. The multiplication is then implemented using a $k \times k$ multiplier, several multiplexers, and a few additional gates, resulting in a significant power reduction over the conventional architecture. In [60], a multiplier is designed to have an unbiased error distribution, leading to lower computational errors during application. The multiplier utilizes a leading-one detector (LOD) to locate the leading-one in each operand, and then selects the following $k-2$ bits, where k is a designer-defined value selected in accordance with accuracy requirements. Error is reduced by assuming a uniform error distribution and setting the value of the remaining lower bits to the nearest one-hot median value. An inexact multiplier is proposed in [61] in which the operands are rounded to their nearest

power of two, allowing for a simplified multiplication to be performed using three shift operations and two addition/subtraction operations. Three hardware implementations are presented which utilize computation blocks for sign detection, rounding, shifting, addition, subtraction, and sign selection.

2.2.2 Logarithmic Multipliers

The logarithmic multiplier (LM), originally introduced in [62], converts the multiplication operands to inexact logarithmic numbers so that an inexact product may be computed using only bitshifting and addition. More specifically, the binary operands of the multiplication $X \times Y$ can be expressed as

$$\log_2 X = k_1 + \log_2(1 + x_1), \quad \text{where } 0 \leq x_1 < 1, \tag{2.1}$$

$$\log_2 Y = k_2 + \log_2(1 + x_2), \quad \text{where } 0 \leq x_2 < 1, \tag{2.2}$$

in which k_1 and k_2 are the characteristics of X and Y, respectively, meaning that they indicate the position of the leading-one in the unsigned binary representations. The product P and its logarithm can then be expressed as

$$P = X \times Y = 2^{k_1+k_2}(1 + x_1) \times (q + x_2), \tag{2.3}$$

$$\log_2 P = k_1 + k_2 + \log_2(1 + x_1) + \log_2(1 + x_2). \tag{2.4}$$

As $\log_2(1 + x) \approx x$ when $0 \leq x < 1$, the logarithmic product can be approximated as

$$\log_2 P \approx k_1 + k_2 + x_1 + x_2. \tag{2.5}$$

The computation described in (2.1)–(2.5) is implemented by performing leading-one detection on each operand, converting the operands to their logarithmic equivalents, summing $k_1 + x_1$ and $k_2 + x_2$, and converting the result to its binary form. The LM does not generate partial products and thus does not require the use of partial product generators or partial product accumulation trees, allowing for substantially reduced hardware complexity. The LM generates significant errors, so contemporary works have aimed to refine the design and improve its accuracy. In [63], the size of the approximate fraction in the LM is kept proportional to its precision via the use of truncation. The least-significant bits are rounded off in the approximate log and anti-log conversions, resulting in reduced hardware costs. In [64], $\log_2 x$ is approximated by rounding x to its nearest power-of-two rather than the highest power-of-two smaller than or equal to x. The multiplication can

be rewritten as

$$X = m_1 + q_1, \quad \text{where } m_1 = 2^{k_1}, \tag{2.6}$$

$$Y = m_2 + q_2, \quad \text{where } m_2 = 2^{k_2}, \tag{2.7}$$

$$X \times Y \approx (2^{k_1+k_2} + q_2 2^{k_1} + q_1 2^{k_2}) + \cancel{q_1 q_2}. \tag{2.8}$$

As indicated in (2.8), the least-significant term is ignored as approximation error. Based on the LOD structure, a nearest-one detector (NOD) circuit determines the nearest power-of-two for each operand. The required number of shifts is then determined using a priority encoder according to the output of the NOD. The three terms are then summed using a combination of exact and approximate adders.

A LM implementation utilizing an iterative approach is proposed in [65], where a correction term is computed and added to the approximate product to increase accuracy. The correction term is computed concurrent to the product, and therefore no additional delay is incurred. A new truncation scheme is introduced to reduce the area overhead of the iterative error correction. Designs for both iterative and non-iterative LMs are proposed in [66], where inexact adders are utilized during the mantissa addition. The lower-part *OR*-adder from [47], the third inexact mirror adder design from [43], and a novel inexact adder architecture are utilized in the mantissa addition, allowing for a substantial reduction in power-delay product (PDP).

2.2.3 Approximate Booth Multipliers

The radix-4 Booth multiplication algorithm, also known as the modified Booth algorithm, is a popular multiplication architecture in which the number of partial products is reduced by encoding the multiplicand as a set of radix-4 digits. The multiplicand is examined in overlapping 3-bit groups $\{y_{2i+1}, y_{2i}, y_{2i-1}\}$, and each group is encoded as a radix-4 value in the range $[-2, 2]$ via the use of a modified Booth encoder (MBE) circuit. Each encoded digit is expressed using three encoding signals, i.e. neg_i, two_i, and $zero_i$. A PPG takes as input the encoding signals along with multiplier bits x_j and x_{j-1} to compute the jth bit of the ith partial product pp_{ij}. For a signed $n \times n$ multiplication, the height of the resulting partial product matrix is reduced to $\lceil n/2 \rceil$. Because the partial product generation of the Booth multiplier has additional complexity, a number of works apply approximation to the MBE or the PPG to reduce hardware cost. In [67], a MBE is proposed in which four novel encoding signals are generated. The delay incurred in the encoder is equal to that of the fast conventional encoder, and the decoder requires two fewer transistors when compared to the conventional design. The structure of the partial product matrix is also modified by combining the least-significant bit (LSB) of each row with its corresponding sign-correction term as to make the matrix more regular. The final summation is performed using a novel adder which combines the properties of the conditional-sum adder and the

conditional-carry adder. An approximate Wallace-Booth multiplier is proposed in [68], where an inexact MBE, an approximate 4:2 compressor, and an approximate tree structure are utilized. Rather than generating encoding signals and passing them to the PPG, the approximate MBE computes the partial product bit in terms of three multiplier bits and a single multiplicand bit, generating errors in only 2/16 input combinations. The approximate 4:2 compressor utilized during compression is the second design from [69]. Finally, the approximate tree structure eliminates a compression stage by utilizing the approximate compressor in the $n/2$ least-significant columns of the matrix and eliminating the most-significant sign-correction term. An approximate Booth multiplier is presented in [70], where two inexact encoding algorithms are utilized alongside an approximate Wallace tree. Both of the proposed encoders express the partial product as a direct function of the multiplier and multiplicand bits, introducing error in 4/32 and 8/32 cases, respectively. Similar to [68], the number of compression stages in the Wallace tree is reduced by ignoring the most-significant sign-correction term.

The literature also features a number of approximate multipliers that make use of higher-radix Booth encoding. The radix-8 Booth algorithm tends to be less popular due to the complexity in generating partial products corresponding to $\pm 3X$. In [71], an approximate 2-bit adder is designed for calculating this partial product by summing $\times 1X$ and $\times 2X$. Two approximate radix-8 multipliers utilizing this adder are proposed, where each multiplier is implemented for several truncation widths. In [72], a hybrid high-radix encoding scheme is used to improve multiplier performance. In the proposed encoding scheme, the n-bit multiplicand is divided into two parts: the higher-order part of $n-k$ bits, and the lower-order k-bit part, where the configuration parameter k is an even number such that $k \geq 4$. The higher-order part is encoded using radix-4 Booth encoding to maintain accuracy, while the lower-order part utilizes a high radix-2^k encoding to reduce hardware cost. Three multipliers are implemented for which the lower-order part is encoded using radix-64, radix-256, and radix-1024 encoding, respectively. Compared to the accurate radix-4 multiplier, the proposed designs reduce area and power consumption by up to half while producing an error with a Gaussian distribution and a near-zero average.

2.2.4 Multiplication using Approximate Compressors

Partial product accumulation is often the most resource-intensive stage of the multiplication operation. Thus, a number of approximate compressors have been proposed for use in reducing the complexity of the accumulation tree. In [69], two inexact 4:2 compressors are proposed and subsequently verified for use in partial product accumulation. The first approximate compressor leverages the fact that the *carry* output for an exact compressor is equal to c_{in} in 24 of 32 input states by simplifying *carry* to c_{in}. Because the *carry* output has a higher weight than the other outputs and produces a difference

of 2 when incorrect, partial error compensation is provided by setting the *sum* output to 0 in the case that $c_{in} = 1$. The c_{out} output is also modified to compensate for errors associated with *carry* and *sum*. The result of these modifications is an incorrect output in 12/32 cases. Because the *carry* and c_{out} outputs have the same weight, their equations are simply interchanged in the second compressor design, resulting in an output error rate of 4/16. Several $n \times n$ multipliers are implemented, where exact compressors are replaced with one of the proposed compressors for either the entire partial product matrix or the $n - 1$ least-significant columns. The use of approximate compressors in the $n - 1$ least-significant columns provides a good balance between improving hardware performance and maintaining accuracy. Three approximate 4:2 compressors are proposed in [73], where the c_{in} and c_{out} signals are ignored. The inexact compressors are utilized in a 8×8 multiplier with Dadda-tree accumulation, where the four least-significant product bits are truncated and the next four least-significant product bits are accumulated using the proposed compressors. In [74], the compressor designs from [69] and [73] are modified for use alongside error recovery modules, resulting in a multiplier design with higher accuracy, smaller area, and lower power consumption. Approximate designs for a 2:1 compressor, 3:2 compressor, 4:2 compressor, 5:3 compressor, and 6:3 compressor are introduced in [75], where the optimal allocation of approximate compressors for a given multiplier is determined using a novel algorithm. The proposed compressors and allocation algorithm are tested for several operand lengths and are shown to provide substantial reductions in power and delay.

2.3 Approximate Dividers

Division as an operation is not nearly as ubiquitous as multiplication. In the an average computer program, multiplications occur with a much greater frequency than divisions. Because of its lower utilization, the design of approximate dividers has received relatively little attention. Additionally, while the partial products of a multiplication can be generated in parallel, division is a sequential operation and thus is more challenging to approximate, due to the fact that the error produced at a given stage will propagate to all following stages. Division is commonly implemented using an array-based architecture which implements either a restoring or non-restoring division algorithm. A restoring $2n/n$ division performs the operation over i iterations, where each iteration involves the subtraction of the divisor from the shifted partial remainder to produce a trial difference. If the result of the subtraction is non-negative, the quotient bit for the current iteration is set to 1 and the partial remainder is loaded with the value of the trial difference. If the subtraction produces a negative result, the quotient bit is set to 0 and the partial remainder is restored to its prior value. In non-restoring division, the result of the subtraction instead selects a quotient bit from the set $\{-1, 1\}$ so that restoring the value of

the partial remainder is not necessary. The non-restoring division architecture requires an additional remainder correction circuit to correct the final remainder in the case that it is negative. Array-based division utilizes a matrix of divider cells, where each row computes an iteration of the division. In [76], three transistor-level approximations of the full-subtractor are presented. Three corresponding designs are then proposed in which the conventional subtractor of the divider cell is replaced with the approximate subtractor for a portion of the array, where four different cell-replacement schemes are explored. The authors of [76] extend their work to restoring dividers in [77], where the inexact subtractor cells proposed in [76] are utilized in three restoring divider designs. In [78], the proposed divider computes the division of a n-bit dividend by a $n/2$-bit divisor by utilizing a LOD to dynamically select for each operand a predefined number of consecutive bits starting at the leading-one, which are then routed to an accurate core divider. The parameter k refers to the number of bits that are divided in the core divider, meaning that k dividend bits and $k/2$ divisor bits are selected. Priority encoders and multiplexers are used to route the selected bits to the core divider to compute a $k/2$-bit quotient which is then adjusted to a width of $n/2$ using a barrel shifter. A similar design is presented in [79], where LODs are used to select a portion of the bits from each input operand for routing to an accurate reduced-width divider. An additional error-correction block utilizes *OR*-gates to recover the error introduced by the shifter. In [66], the architectures of the logarithmic divider and the conventional array-based divider are combined in an approximate hybrid divider. Restoring divider cells are used to generate the most-significant bits (MSBs) of the quotient as to maintain accuracy, while the use of logarithmic division to compute the LSBs reduces hardware cost.

2.4 Approximate MAC Units

While approximate computing has been widely applied to atomic arithmetic operations, comparatively little work has been done to utilize approximation techniques in the MAC unit hardware. In [80], an inexact MAC unit is proposed in which the partial product terms of the multiplier are compressed using *OR*-gates to implement approximate counters, and a compensation term is introduced to improve accuracy. Designs are implemented for three levels of truncation, and the function of the proposed MAC unit is verified using a Gaussian kernel application. An inexact MAC unit is proposed in [81], which utilizes an approximate compression tree to improve hardware performance. The MAC operation is divided into two stages: (1) *AND*-based partial product generation, and (2) partial product accumulation via the compression tree. The proposed design implements a fused MAC architecture, where the previous accumulator value is inserted at the bottom of the generated partial product matrix so that it may be accumulated during partial product compression. The compression tree performs an approximate compression

over three stages to produce an intermediate product and two accuracy compensation vectors. The intermediate product and compensation vectors are subsequently compressed into a sum and carry, which are then summed to generate the final result. The functionality of the proposed MAC unit is verified in a Gaussian smoothing application, as well as a convolution operation within a neural network.

Chapter 3

Approximate Booth Multipliers

In this chapter, approximation is applied to the radix-4 Booth multiplication architecture to produce three inexact multiplier designs. Approximation is introduced in the partial product generation stage of the multipliers using two novel inexact PPGs which utilize a reduced number of encoding signals. In two of the proposed multipliers, the use of the approximate PPGs results in the reduction of the partial product matrix. The designs are evaluated alongside other state-of-the-art approximate multipliers, where MRED and NMED are used to measure accuracy, and hardware performance is evaluated in terms of area, power consumption, and area-power product (APP). The multipliers are utilized in applications for image transformation, matrix multiplication, and a finite impulse response (FIR) filter, where application performance is evaluated using peak signal-to-noise ratio (PSNR), MRED/NMED, and mean squared error (MSE), respectively.

Author contributions are firstly outlined in Section 3.1. Section 3.2 provides a background of radix-4 Booth multipliers, and Section 3.3 presents the inexact multiplier designs. Accuracy and hardware performance are evaluated in Section 3.4, and signal processing applications are presented in Section 3.5.

3.1 Author Contributions

S.V. conceived of and designed the proposed multiplier models. E.A. provided minor design contributions to the proposed models. S.V. and E.A. carried out the implementation of benchmarking models from the literature. S.V. and E.A. carried out the functional simulations and synthesis simulations. S.V. and E.A. analyzed the accuracy and hardware metrics of the implemented models. S.V. carried out the simulations for the signal processing applications. S.V. and E.A. wrote the manuscript in consultation with S.K.

Table 3.1. Radix-4 Booth encoding where $(pp_{ij})_4$ is the partial product as a radix-4 digit

Input			Encoded Signals			$(pp_{ij})_4$
y_{2i+1}	y_{2i}	y_{2i-1}	neg_i	two_i	$zero_i$	
0	0	0	0	0	1	$+0$
0	0	1	0	0	0	$+X$
0	1	0	0	0	0	$+X$
0	1	1	0	1	0	$+2X$
1	0	0	1	1	0	$-2X$
1	0	1	1	0	0	$-X$
1	1	0	1	0	0	$-X$
1	1	1	0	0	1	-0

3.2 Radix-4 Booth Multipliers

Consider the $2n$-bit product P of a signed $n \times n$ multiplication with multiplier X and multiplicand Y given by

$$X = -x_{n-1}2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i, \tag{3.1}$$

$$Y = -y_{n-1}2^{n-1} + \sum_{i=0}^{n-2} y_i 2^i, \tag{3.2}$$

$$P = -p_{2n-1}2^{2n-1} + \sum_{i=0}^{2n-2} p_i 2^i. \tag{3.3}$$

The radix-4 Booth multiplication algorithm, also known as the modified Booth algorithm, reduces the number of partial products by encoding the multiplicand as a set of radix-4 digits. The encoding process is described in Table 3.1. First, the multiplicand is examined in overlapping 3-bit groups $\{y_{2i+1}, y_{2i}, y_{2i-1}\}$ as shown in Fig. 3.1, where the least-significant group uses only two bits of the multiplicand and assumes the third bit to be zero. Each 3-bit group occupies an unsigned value in the range $[0, 3]$ and is inputted to a MBE which computes the equivalent radix-4 encoded value, a signed number in the range $[-2, 2]$. Each encoded digit can be represented using three encoding signals neg_i, two_i, and $zero_i$ defined as

$$neg_i = y_{2i+1} \cdot \overline{y_{2i}} \cdot y_{2i-1}, \tag{3.4}$$

$$two_i = \overline{y_{2i+1}} \cdot y_{2i} \cdot y_{2i-1} + y_{2i+1} \cdot \overline{y_{2i}} \cdot \overline{y_{2i-1}}, \tag{3.5}$$

$$zero_i = y_{2i+1} \cdot y_{2i} \cdot y_{2i-1} + \overline{y_{2i+1}} \cdot \overline{y_{2i}} \cdot \overline{y_{2i-1}}. \tag{3.6}$$

Each bit of the partial product is computed via a PPG whose circuit schematic is

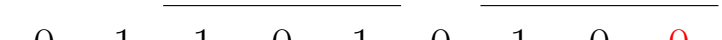


Fig. 3.1. Example of overlapping encoding groups for an 8-bit multiplicand, where red indicates the zero padding bit.

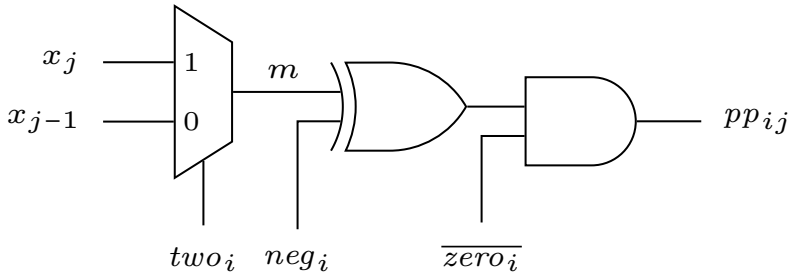


Fig. 3.2. Gate-level schematic for exact PPG.

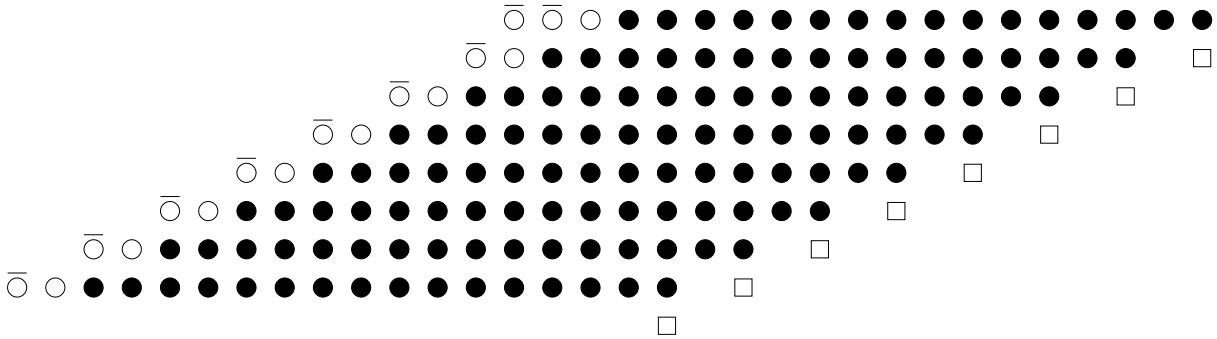


Fig. 3.3. Partial product matrix of the exact 16×16 radix-4 Booth multiplier. ●: a partial product. ○: a sign-extension term. $\overline{\text{○}}$: an inverted sign-extension term. □: a sign-correction term.

shown in Fig. 3.2. The generated partial product is given by

$$\begin{aligned} m &= (x_j \cdot \overline{two_i}) + (x_{j-1} \cdot two_i), \\ pp_{ij} &= \overline{zero_i} \cdot (neg_i \oplus m). \end{aligned} \tag{3.7}$$

While a traditional *AND*-array based multiplier produces a $n \times n$ partial product matrix, the radix-4 Booth algorithm reduces the height of the matrix to $\lceil n/2 \rceil$ as visible in Fig. 3.3.

3.3 Approximate Radix-4 Booth Multipliers

Because of the complexity associated with multiplicand encoding, Booth multipliers are well-suited to applying approximation techniques in the partial product generation hardware. An exact radix-4 PPG requires all three encoding signals, i.e. neg_i, two_i, and $zero_i$, to generate the partial product. In approximate Booth multiplier model 1 (ABM-M1) and approximate Booth multiplier model 2 (ABM-M2), an approximate PPG is used where only two of the three signals, namely neg_i and two_i, are utilized. In approximate Booth

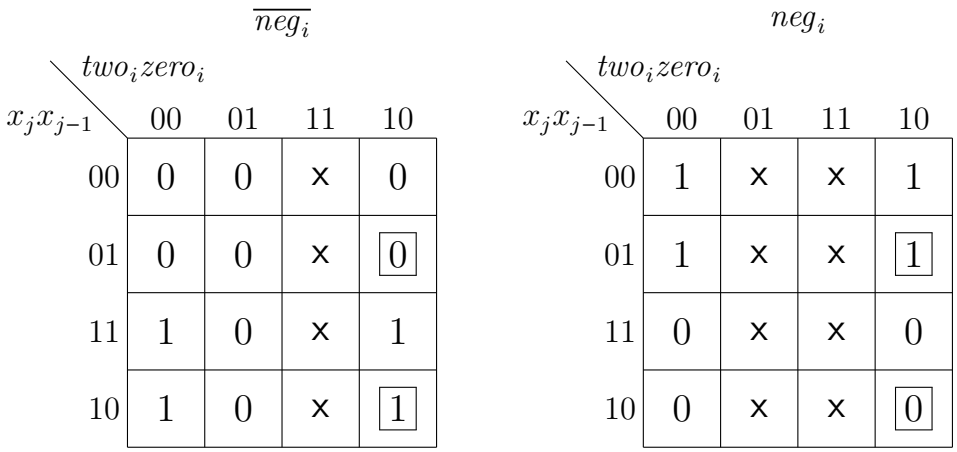


Fig. 3.4. Karnaugh-map for PPG-2S, where $\boxed{1}$ represents a change from 0 to 1 and $\boxed{0}$ represents a change from 1 to 0.

multiplier model 3 (ABM-M3), an additional inexact PPG is proposed in which only the signal $zero_i$ is utilized. In ABM-M1, approximation is also applied in the partial product accumulation by combining the sign-correction terms cor_i with their respective columns in the partial product matrix, thereby reducing its height. In ABM-M2 and ABM-M3, the width of the partial product matrix is reduced by replacing several exact PPGs with a single approximate PPG for multiple rows of the matrix.

The term approximation factor k is used to refer to the magnitude of approximation for a given design. ABM-M1 and ABM-M3 employ a column-wise approximation, where k refers to the number of columns in which approximate PPGs are used. ABM-M2 utilizes a diagonal-wise approximation for which k implies the number of exact PPGs in each matrix row that are replaced with a single inexact PPG. For ABM-M1 and ABM-M3, designs for $k = 4, 8, 12, 16$ are implemented. Because the reduction technique in ABM-M2 approximates a greater number of elements for a given k when compared to the other designs, ABM-M2 models for $k = 2, 4, 6, 8$ were selected for implementation.

3.3.1 Approximate Booth Multiplier Model 1 (ABM-M1)

The logic of the exact PPG is modified according to the Karnaugh-map [82] shown in Fig. 3.4 in which 4/32 entries are modified. The resulting architecture is a 2-signal partial product generator (PPG-2S), where the two utilized encoding signals are neg_i and $zero_i$. The circuit schematic for PPG-2S is shown in Fig. 3.5 and can be expressed as

$$pp_{ij} = \overline{x}_j \cdot neg_i + x_j \cdot \overline{neg_i} \cdot \overline{zero_i}. \tag{3.8}$$

When compared to the exact PPG, the PPG-2S circuit does not require a multiplexer or an *XOR*-gate, and the output can be expressed using only two *AND*-gates and one *OR*-gate. The error difference between the exact PPG and PPG-2S is given in Table 3.2. Since the two_i signal is absent in PPG-2S, the $+2X$ and $-2X$ cases are replaced with

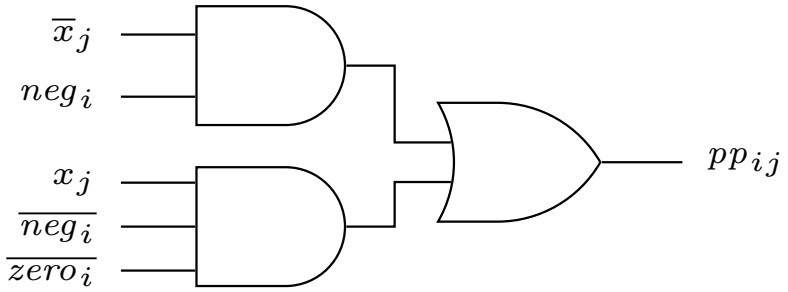


Fig. 3.5. Circuit-level schematic for PPG-2S.

Table 3.2. Accurate radix-4 encoding vs. approximate encoding via PPG-2S

			PPG					**PPG-2S**					
neg_i	two_i	$zero_i$	$pp_{ij}\|x_jx_{j-1}$				$(pp_{ij})_4$	$pp_{ij}\|x_jx_{j-1}$				$(pp_{ij})_4$	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	$+1X$	0	0	1	1	$+1X$	0
0	0	0	0	0	1	1	$+1X$	0	0	1	1	$+1X$	0
0	1	0	0	1	0	1	$+2X$	0	0	1	1	$+1X$	1
1	1	0	1	0	1	0	$-2X$	1	1	0	0	$-1X$	-1
1	0	0	1	1	0	0	$-1X$	1	1	0	0	$-1X$	0
1	0	0	1	1	0	0	$-1X$	1	1	0	0	$-1X$	0
0	0	1	0	0	0	0	-0	0	0	0	0	-0	0

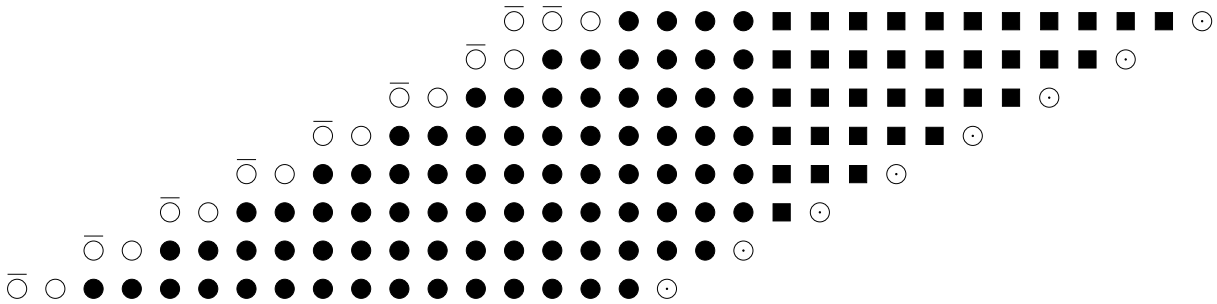
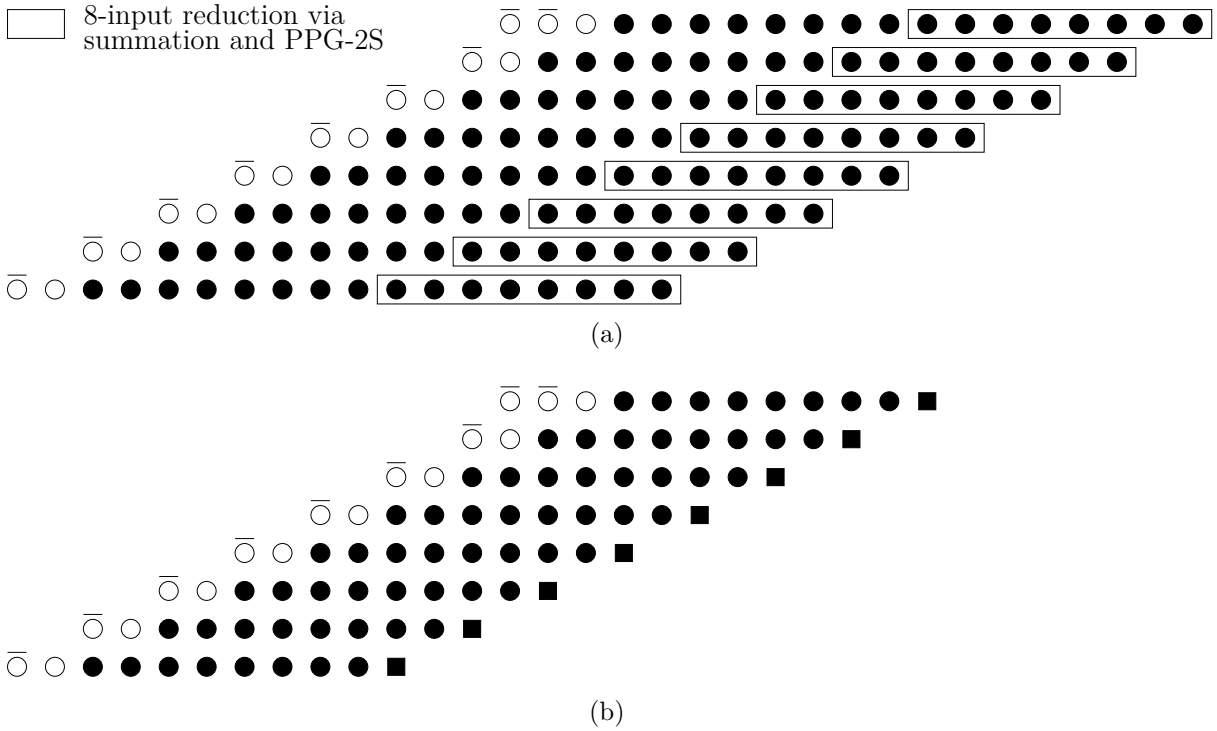


Fig. 3.6. Partial product matrix of ABM-M1 for k = 12. ●: an exact partial product. ■: an inexact partial product generated by PPG-2S. ○: a sign-extension term. $\overline{\text{○}}$: an inverted sign-extension term. ⊙: the result of *OR*-ing each sign-correction term with the above partial product.

$+1X$ and $-1X$, respectively, resulting in two cases with an error distance of ± 1.

The partial product matrix of ABM-M1 for k = 12 is shown in Fig. 3.6. In the ABM-M1 design, all partial products with a significance less than k are approximated using PPG-2S, and all remaining partial products are generated using the exact PPG. To reduce the height of the matrix, each sign-correction term is combined with the least-significant bit in its corresponding row using an *OR*-gate.



(a)

(b)

Fig. 3.7. Partial product matrix of ABM-M2 for $k = 8$, where (a) indicates the row-wise groups over which x_j is summed, and (b) shows the reduced matrix. ●: an exact partial product. ■: an inexact partial product generated by PPG-2S. ○: a sign-extension term. $\overline{\text{○}}$: an inverted sign-extension term.

3.3.2 Approximate Booth Multiplier Model 2 (ABM-M2)

The ABM-M2 design differs from ABM-M1 such that, in every row of the partial product matrix, the k least-significant exact PPGs are replaced with a single PPG-2S. The k LSBs of input X are summed to produce x_{sum}, and a value $x_{\forall j \in (0,k-1)}$ is generated by comparing x_{sum} to the median value for a k-bit number according to

$$\begin{aligned} x_{sum} &= \sum_{j=0}^{k-1} x_j, \\ x_{\forall j \in (0,k-1)} &= \begin{cases} 1, & x_{sum} > k/2 \\ 0, & x_{sum} \leq k/2 \end{cases}. \end{aligned} \tag{3.9}$$

The approximate partial product $pp_{i,\forall j \in (0,k-1)}$ for each row i is then generated using PPG-2S, where $x_{\forall j \in (0,k-1)}$ is supplied as the x_j signal.

The reduction of the partial product matrix is illustrated in Fig. 3.7 for $k = 8$. The 8 LSBs of X are summed to produce x_{sum}. From x_{sum}, the value $x_{\forall j \in (0,7)}$ is found as per (3.9), which is then used to generate for each row i the approximate partial product $pp_{i,\forall j \in (0,7)}$, where neg_i, $zero_i$, and $x_{\forall j \in (0,7)}$ serve as the inputs to PPG-2S.

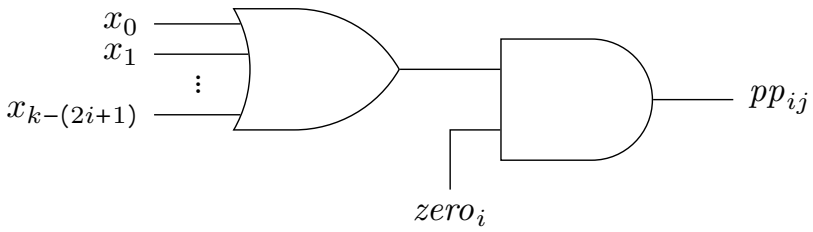


Fig. 3.8. Circuit-level schematic for PPG-1S.

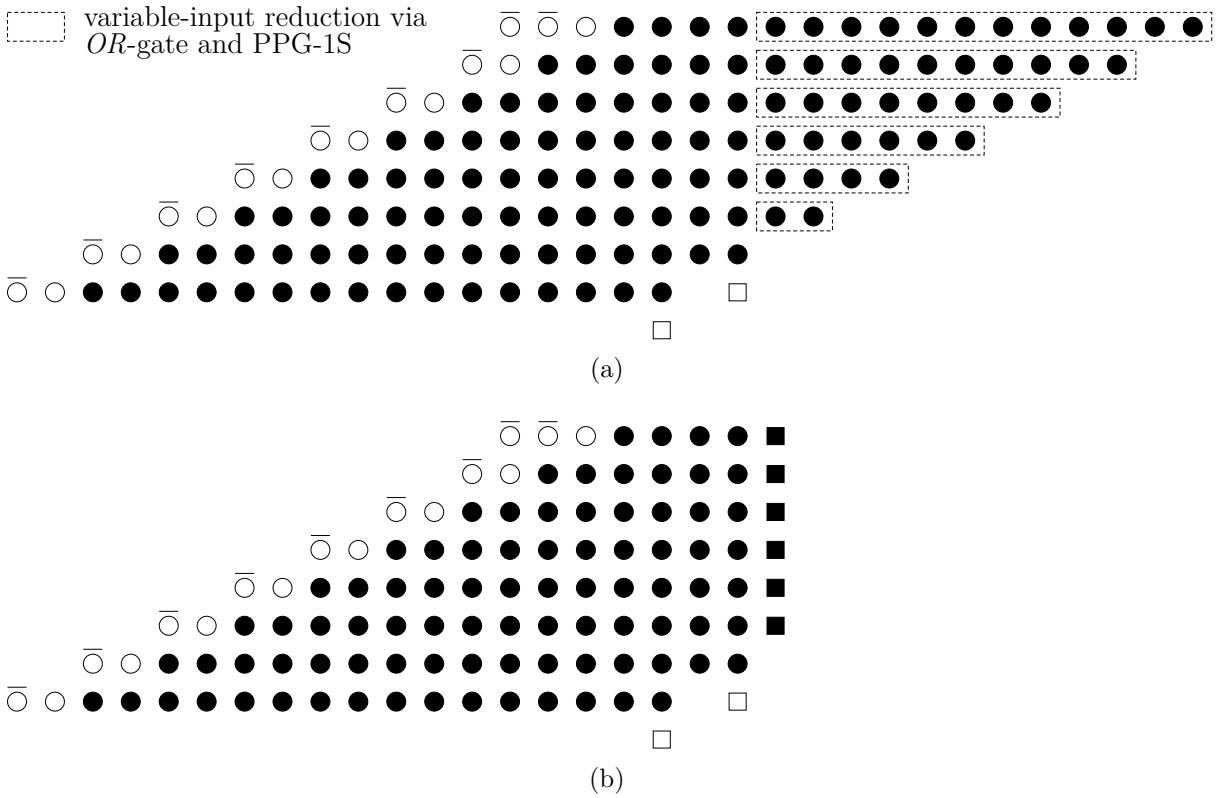


Fig. 3.9. Partial product matrix of ABM-M3 for k = 12, where (a) indicates the row-wise groups over which x_j is reduced, and (b) shows the reduced matrix. ●: an exact partial product. ■: an inexact partial product generated by PPG-1S. ○: a sign-extension term. $\overline{\text{○}}$: an inverted sign-extension term. □: a sign-correction term.

3.3.3 Approximate Booth Multiplier Model 3 (ABM-M3)

In the ABM-M3 design, all partial products with significance less than k are reduced to a single column of approximate partial products. Considering the exact partial product matrix in Fig. 3.3, for a given row i, let l be the number of bits with a significance less than k. For a row i, $x_{\forall j \in (0,k-(2i+1))}$ is generated by *OR*-ing the l least-significant bits of X. The approximate partial product for the row i is then generated by the use of the 1-signal partial product generator (PPG-1S), whose circuit schematic is shown in Fig. 3.8. PPG-1S takes in the signal $zero_i$ and the result of the *OR* operation to produce the approximate partial product for that row.

Consider the partial product matrix for k = 12 shown in Fig. 3.9. For the row i = 3,

Table 3.3. Accuracy and hardware metrics for approximate multipliers

Design	k	**Error Metrics**		**Hardware Metrics**		
		MRED (10^{-2})	NMED (10^{-6})	Area (μm^2)	Power (mW)	APP ($\mu m^2 \cdot$ mW)
Exact radix-4	—	—	—	3851	1.3	5006
Exact radix-8	—	—	—	4116	1.3	5351
ABM-M1	4	0.011	5.079	3578	1.3	4651
	8	0.012	5.089	3419	1.2	4103
	12	0.016	5.491	3218	1.2	3862
	16	0.079	20.800	3160	1.1	3476
ABM-M2	2	0.040	15.250	3148	1.1	3463
	4	0.165	64.626	2797	1.0	2797
	6	0.663	266.100	2333	0.8	1866
	8	2.689	1087.270	2015	0.7	1411
ABM-M3	4	0.007	0.005	3747	1.3	4871
	8	0.010	0.106	3563	1.3	4632
	12	0.020	2.002	2669	1.0	2669
	16	0.340	36.150	1830	0.7	1281
R4ABM [70]	4	0.012	5.714	4037	1.3	5248
	8	0.013	5.835	4008	1.3	5210
	12	0.038	8.430	3640	1.2	4368
	16	0.473	64.190	3362	1.1	3698
ABM1 [71]	—	0.040	19.360	3589	1.2	4307
ABM2-C9 [71]	—	0.089	44.500	2820	1.0	2820
RAD256 [72]	—	0.277	167.800	1977	0.7	1384

x_j ranging from 0 to $k-7$ are passed to the $(k-6)$-input *OR*-gate. The output of this *OR*-gate $x_{\forall j \in (0,k-7)}$ and the signal $\overline{zero}_3$ are inputted to an *AND*-gate which computes the approximate partial product $pp_{3,\forall j \in (0,k-7)}$. Similarly, for $i = 0$, input x_j from 0 to $k-1$ and $zero_0$ are considered. For the row $i = 1$, input x_j from 0 to $k-3$, and $zero_1$ are considered. For the row $i = 2$, input x_j from 0 to $k-5$ and $zero_2$ are considered, and so on.

3.4 Experimental Results

In addition to the proposed designs, several state-of-the-art approximate Booth multipliers are implemented for benchmarking purposes. In [70], an approximate radix-4 Booth multiplier (R4ABM) is proposed in which a novel inexact PPG does not take in any recoding signals, but rather operates only on bits of the multiplicand and multiplier. The approximate Booth multiplier 1 (ABM1) presented in [71] is a radix-8 based design which approximates partial products corresponding to $\pm 3X$ using an inexact 2-bit adder. ABM2-C9 is a modification of ABM1, where error compensation is utilized in the approx-

imate 2-bit adder alongside 9-bit truncation. In [72], the RAD256 multiplier uses radix-4 encoding to compute higher-order partial products, while radix-256 encoding is used to compute lower-order partial products. Because the designs from [71] are based on the radix-8 architecture, an exact radix-8 multiplier is implemented along with the radix-4 design to provide fair benchmarking.

All designs are implemented in Verilog HDL, and functional verification is performed using a SystemVerilog testbench. Python scripts are used to process the testbench outputs. All multipliers are tested on an identical set of one million random input pairs with a uniform distribution. The multipliers are synthesized using Synopsys Design Compiler for the TSMC 65 nm technology library. The Design Compiler simulations use an operating voltage of 1 V and an operating temperature of 25 °C. The largest critical path delay of all the designs, i.e. that of the exact radix-8 multiplier, was measured to be 1.2 nm, and this delay value was then used as the timing constraint when simulating all other models. The error characteristics of the implemented designs are reported in Table 3.3, where the error metrics computed are MRED and NMED as defined in [83]. Area, power consumption, and APP are the metrics used to evaluate hardware performance.

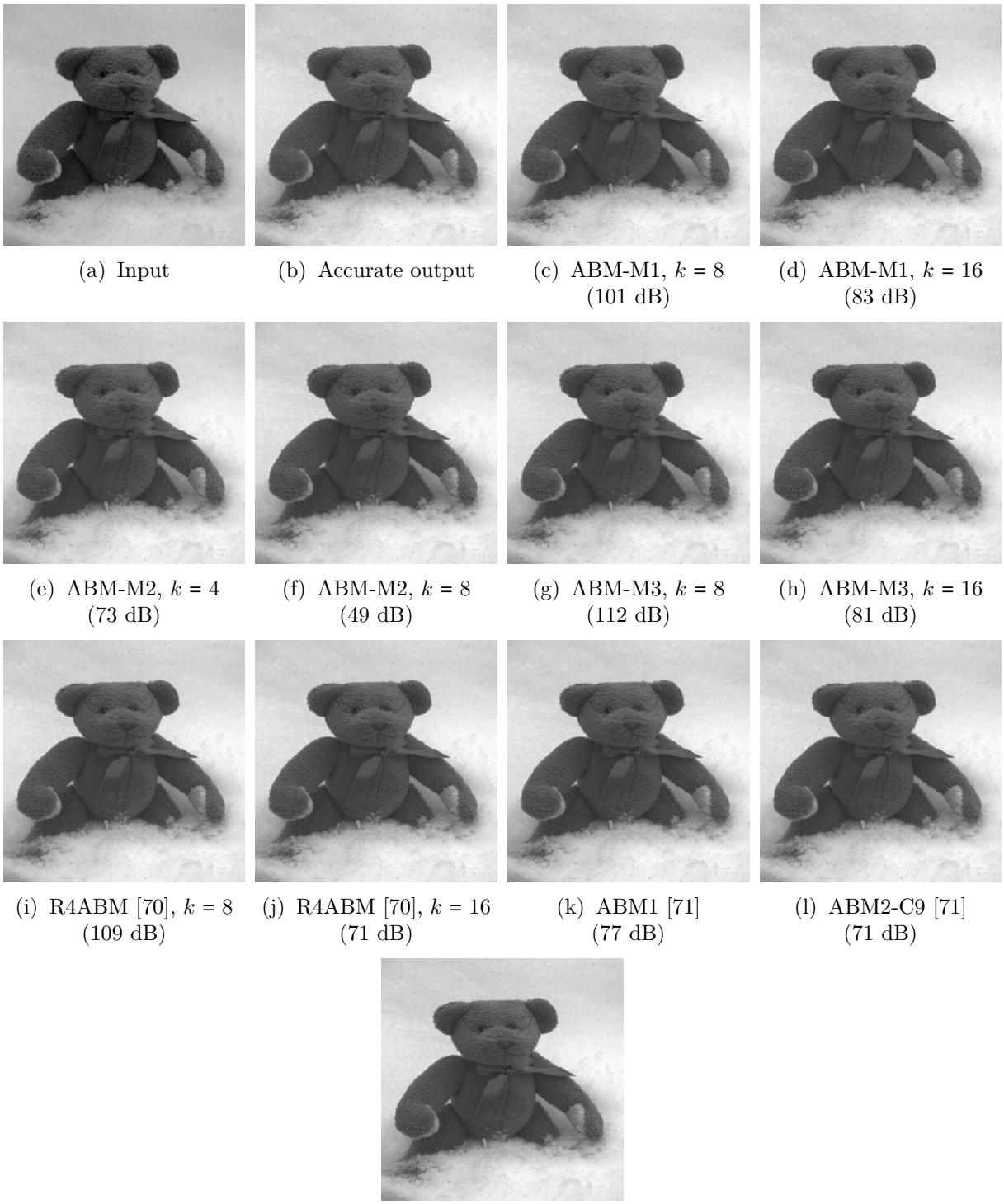
ABM-M2 achieves an accuracy similar to that of ABM1 and ABM2-C9 [71], but is outperformed by R4ABM [70]. ABM-M1 and ABM-M3 exhibit the lowest MRED values, and ABM-M3 achieves the lowest NMED values. All proposed designs exhibit significant area and power savings over the exact radix-8 multiplier. ABM-M1 provides APP savings in the range of 13% to 35%, and ABM-M3 provides APP savings in the range of 9% to 76%. ABM-M2 exhibits the most substantial APP reduction, with $k = 2, 4, 6, 8$ corresponding to improvements of 35%, 48%, 65% and 74%, respectively.

3.5 Applications

Three signal processing applications are used to verify the function of the proposed multipliers, namely image transformation, matrix multiplication, and a FIR filter implementation. Designs for $k = 8, 16$ are tested for ABM-M1 and ABM-M3, while ABM-M2 is tested for $k = 4, 8$. Simulations were performed in ModelSim, and MATLAB was used to generate input data and assess output quality.

3.5.1 Image Transformation

A 16-bit image is selected and its pixel values are shifted from the range $[0, 65535]$ to $[-32768, 32767]$. The pixel values of the image are then multiplied by a constant to provide a brightening effect. The outputs computed for the implemented designs are shown in Fig. 3.10, where PSNR is used to express the quality of the result. For all models, image quality deteriorates with increasing approximation factor. ABM-M3 for $k = 8, 16$ outperforms R4ABM for equivalent approximation factors, as well as ABM1,



(a) Input (b) Accurate output (c) ABM-M1, k = 8 (101 dB) (d) ABM-M1, k = 16 (83 dB)

(e) ABM-M2, k = 4 (73 dB) (f) ABM-M2, k = 8 (49 dB) (g) ABM-M3, k = 8 (112 dB) (h) ABM-M3, k = 16 (81 dB)

(i) R4ABM [70], k = 8 (109 dB) (j) R4ABM [70], k = 16 (71 dB) (k) ABM1 [71] (77 dB) (l) ABM2-C9 [71] (71 dB)

(m) RAD256 [72] (63 dB)

Fig. 3.10. Input image and generated outputs for image transformation application.

and ABM2-C9. Because ABM-M2 has lower accuracy, it only produces competitive PSNR values for k = 4.

Table 3.4. MRED and NMED of multipliers for matrix multiplication

Design	k	MRED (10^{-3})	NMED (10^{-5})
ABM-M1	8	0.308	0.51
	16	0.337	1.006
ABM-M2	4	3.274	6.074
	8	51.774	97.726
ABM-M3	8	0.004	0.005
	16	1.186	1.747
R4ABM [70]	8	0.382	0.577
	16	4.233	6.499
ABM1 [71]	—	1.858	1.617
ABM2-C9 [71]	—	2.447	3.341
RAD256 [72]	—	3.140	8.531

3.5.2 Matrix Multiplication

Matrix multiplication is widely used in applications such as deep learning, image and video processing, and robotics. This application multiplies two 5×5 matrices, and performance is measured by calculating the MRED and NMED of the outputted matrix. The MRED and NMED values computed for the implemented designs are summarized in Table 3.4, where ABM-M1 and ABM-M3 produce competitive results.

3.5.3 FIR Filter Implementation

The finite impulse response (FIR) is widely used in digital signal processing to compute frequency response. In this application, an electrocardiograph wave with high frequency noise is passed through a 41-tap low-pass FIR filter with a cut-off frequency of 45 Hz and a sampling frequency of 200 Hz. In Fig. 3.11, the input signal and the filtered signal are shown in both the time and frequency domains. While FIR filtering utilizes both addition and multiplication, only the multiplication operations are modified to implement the inexact multiplier designs. Performance is evaluated using MSE as summarized in Table 3.5, where the proposed designs achieve high strong performance.

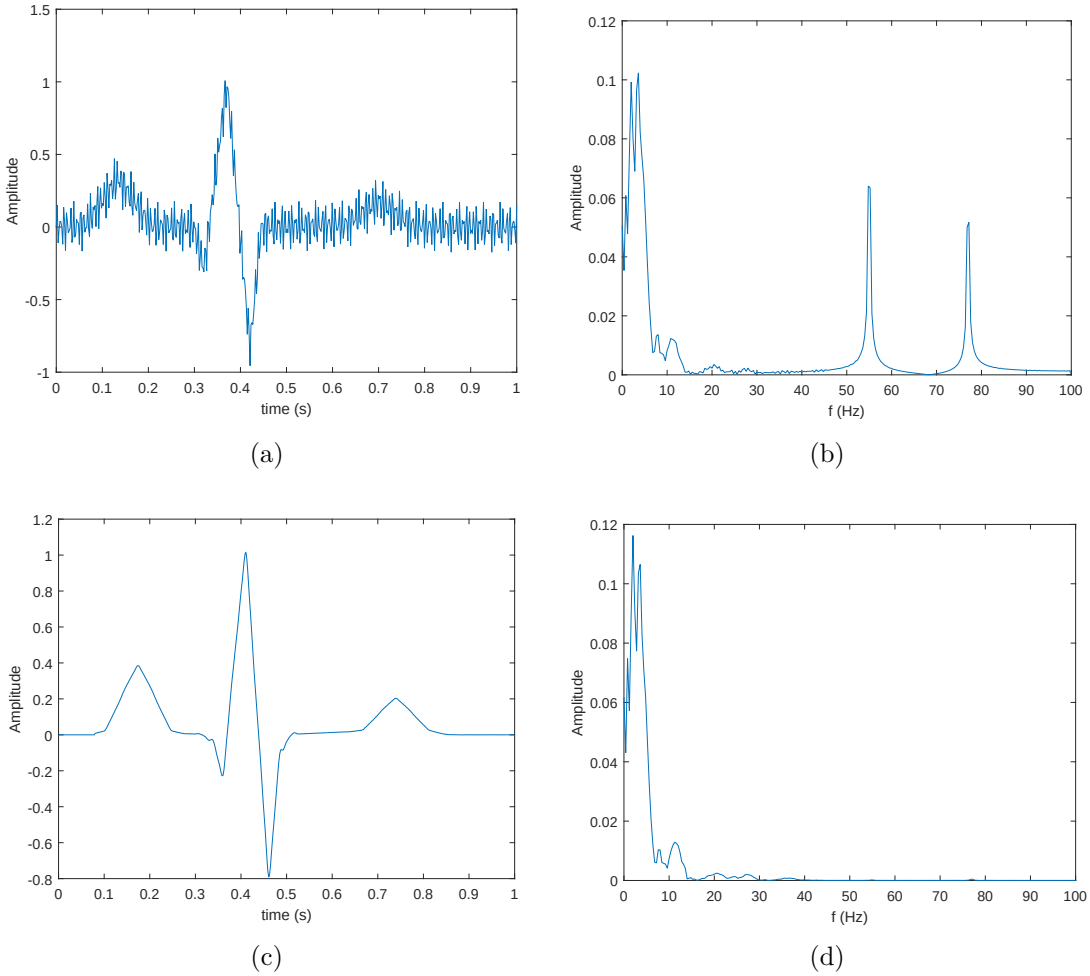


Fig. 3.11. Input electrocardiograph signal in the (a) time domain and (b) frequency domain. Signal outputted by the low-pass FIR filter in the (a) time domain and (b) frequency domain.

Table 3.5. MSE of multipliers in FIR filter application

Design	k	MSE
ABM-M1	8	2.546×10^{-9}
	16	6.328×10^{-8}
ABM-M2	4	1.044×10^{-6}
	8	2.673×10^{-4}
ABM-M3	8	3.570×10^{-12}
	16	2.391×10^{-7}
R4ABM [70]	8	7.747×10^{-11}
	16	1.819×10^{-5}
ABM1 [71]	—	7.291×10^{-9}
ABM2-C9 [71]	—	2.897×10^{-7}
RAD256 [72]	—	1.919×10^{-6}

Chapter 4

Approximate Array-Based Restoring Dividers

In this chapter, approximation is applied to the array-based restoring divider architecture to produce two inexact divider designs. The first divider features a novel approximate restoring divider cell which is used to replace the conventional cells in several columns of the array. Only a subset of the trial subtractions are performed in the second divider, where encoding and rounding are used to estimate the remaining quotient bits. The designs are evaluated alongside other state-of-the-art approximate dividers. Accuracy is measured using MRED and NMED, and hardware performance is expressed in terms of delay, area, power consumption, PDP, and area-delay product (ADP). The ability of each design to balance hardware reduction with accuracy is evaluated by computing a number of hybrid accuracy-hardware metrics. The dividers are then verified in two image processing applications, namely change detection and foreground extraction, where performance is measured in terms of PSNR and structural similarity index measurement (SSIM).

Section 4.1 describes the author contributions, Section 4.2 provides an outline of the conventional array-based restoring divider architecture, and the approximate restoring dividers are proposed in Section 4.3. An accuracy and hardware analysis are performed in Section 4.4, and the image processing applications are presented in Section 4.5.

4.1 Author Contributions

S.V. conceived of and designed the first proposed divider model, while E.A. conceived of and designed the second proposed divider model. S.V. and E.A. carried out the implementation of benchmarking models from the literature. S.V. carried out the functional simulations and synthesis simulations for the first model as presented in the conference manuscript. S.V. analyzed the accuracy and hardware metrics for the first model as presented in the conference manuscript. S.V. carried out the image processing simulations for the first model as presented in the conference manuscript. E.A. carried out the functional simulations and the synthesis simulations for both models as presented in the

journal manuscript. E.A. analyzed the accuracy and hardware metrics for both models as presented in the journal manuscript. E.A. analyzed the accuracy-hardware tradeoff performance for both models as presented in the journal manuscript. E.A. carried out the image processing simulations for both models as presented in the journal manuscript. S.V. and E.A. wrote the conference and journal manuscripts in consultation with S.K.

4.2 Exact Restoring Dividers

Binary division generally involves the division of a $2n$-bit dividend z by a n-bit divisor d to produce a n-bit quotient q and a n-bit remainder s, where $s < d$. Restoring division involves the repeated subtraction of the divisor from the shifted partial remainder, where the value of the partial remainder is restored if the result of the subtraction is negative. In more formal terms, each iteration i involves the subtraction of the shifted divisor $2^n d$ from the shifted partial remainder $2s^{(j-1)}$ to produce a trial difference $2s^{(j-1)} - q_{n-i}(2^n d)$. If the result of the subtraction is non-negative, the quotient bit q_{n-i} is set to 1 and the next partial remainder $s^{(i)}$ is loaded with the value of the trial difference. If the subtraction produces a negative result, the quotient bit q_{n-i} is set to 0 and the partial remainder is restored to the prior shifted value $s^{(i)} = 2s^{(j-1)}$. After $i = n$ iterations, all quotient bits have been selected and the remainder s is set to the most-significant n bits of the final partial remainder $s^{(n)}$.

Restoring division is commonly implemented using an array-based architecture. A $2n/n$ division makes use of an array of n rows, where each row contains n restoring divider cells and a single *OR*-gate. Each exact restoring divider cell (EXRDC) consists of a full substractor (FS) and a multiplexer, as shown in Fig. 4.1. The multiplexer is used to select the difference in the case of a non-negative partial remainder and otherwise selects the prior value in the case of a restored partial remainder. The architecture of the 8/4 array-based restoring divider is illustrated in Fig. 4.2.

4.3 Approximate Restoring Dividers

An exact $2n/n$ division requires $n \times n$ restoring divider cells as well as n *OR*-gates. As n grows, the hardware required to compute the division quickly becomes expensive. This section describes two approximate divider architectures which compute an inexact division through the use of reduced hardware. Approximate restoring divider model 1 (AXRD-M1) replaces some of the exact divider cells with approximate divider cells whose logic expressions are simplified. In approximate restoring divider model 2 (AXRD-M2), several rows of the array are eliminated and the corresponding quotient bits are instead approximated by rounding and encoding the divisor and partial remainder. As the ratio of approximate cells to exact cells in AXRD-M1 increases, the accuracy decreases and

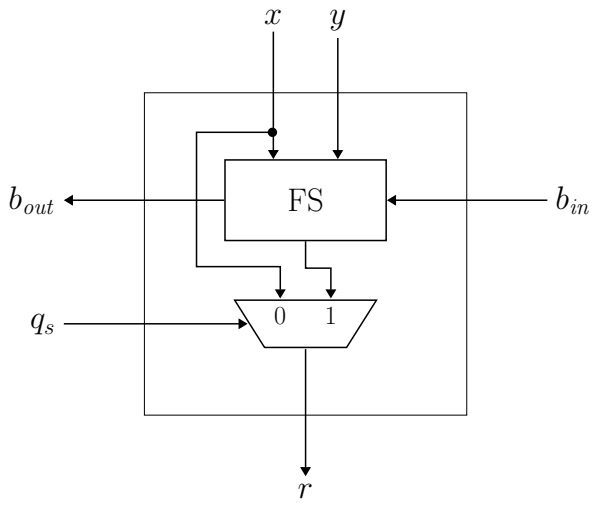


Fig. 4.1. Circuit-level schematic for EXRDC.

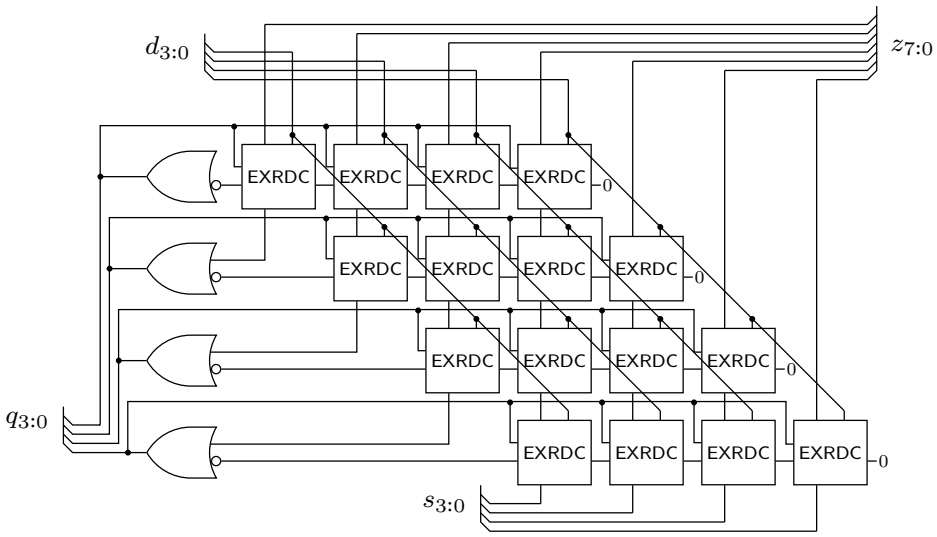


Fig. 4.2. Architecture of exact 8/4 restoring divider.

the hardware savings improve. A similar tradeoff occurs in AXRD-M2 as the number of eliminated rows increases. The term approximation factor k is used to refer to the magnitude of approximation for a given design. In AXRD-M1, k expresses the number of columns for which exact cells have been replaced with approximate cells. In AXRD-M2, k indicates the number of rows that have been eliminated from the array.

It should be noted that, while AXRD-M1 computes an inexact remainder, AXRD-M2 does not. The reason for this is related to the fact that the quotient is generally much more useful than the remainder in the majority of approximate division applications [76]. The architecture of AXRD-M1 is structured such that an approximate remainder is computed alongside an approximate quotient without any additional overhead. The architecture of AXRD-M2 differs in that additional gates would be needed to compute an approximate remainder. Considering that approximate remainders have such limited use, it was decided that circuitry devoted solely to remainder computation would not be added to AXRD-M2

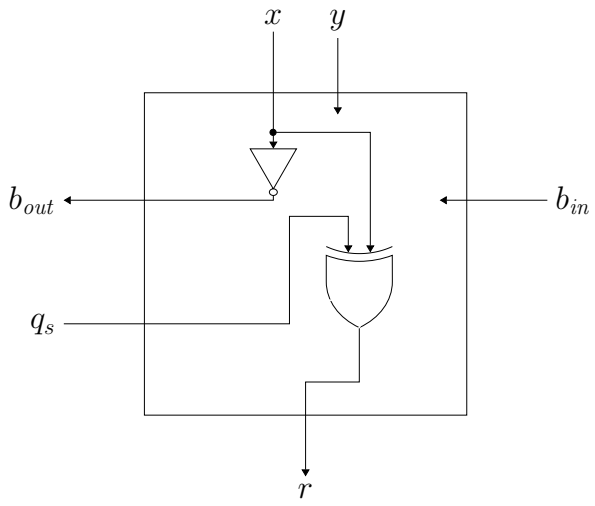


Fig. 4.3. Circuit-level schematic for AXRDC.

as to maximize hardware savings.

4.3.1 Approximate Restoring Divider Model 1 (AXRD-M1)

This section introduces the architecture of the proposed AXRD-M1 design. As described above, approximation is applied by replacing all EXRDCs in the k least-significant columns of the array with the proposed approximate restoring divider cell (AXRDC). The outputs of EXRDC can be expressed as

$$s = q_s d + \overline{q_s} x, \tag{4.1}$$

$$b_{out} = \overline{x \oplus y} \cdot b_{in} + \overline{x} y, \tag{4.2}$$

and the equivalent circuit diagram is shown in Fig. 4.1. The circuitry of the proposed AXRDC is shown in Fig. 4.3 and the equivalent logic expressions are

$$s = q_s \oplus x, \tag{4.3}$$

$$b_{out} = \overline{x}, \tag{4.4}$$

from which it can be seen that s and b_{out} are independent of y and b_{in}. The high-level architecture of AXRD-M1 for $k = 8$ is shown in Fig. 4.4. For a $2n/n$ division and approximation factor k, the number of approximated cells is given by

$$\begin{cases} \dfrac{k(k+1)}{2}, & \text{for } k <= n \\ n(2k-n+1) - \dfrac{k(k+1)}{2}, & \text{for } k > n \end{cases}. \tag{4.5}$$

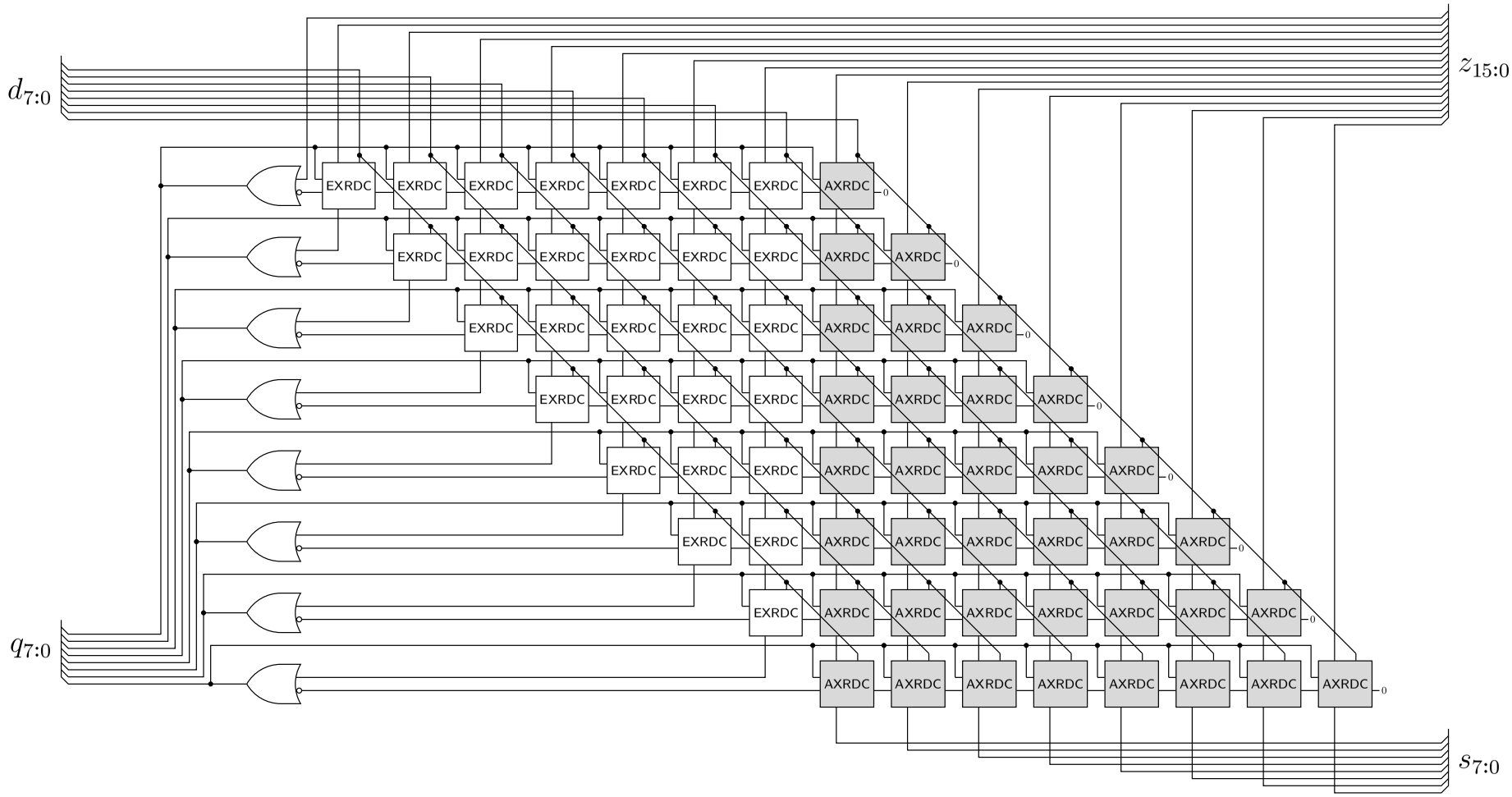


Fig. 4.4. Architecture of AXRD-M1 for k = 8 where approximate cells are colored gray.

4.3.2 Approximate Restoring Divider Model 2 (AXRD-M2)

This section describes the architecture of the proposed AXRD-M2. The design utilizes the exact architecture to perform the first $n-k$ trial subtractions, producing the first k quotient bits and the partial remainder $s^{(n-k)}$. The positions of the leading-one in both the divisor d and the partial remainder $s^{(n-k)}$ are used to generate encoded values u and v, which are then used to approximate the remaining quotient bits. The high-level architecture of AXRD-M2 is shown in Fig. 4.5.

Independent of k, the three most-significant bits of the divisor $d_{n-1:n-3}$ are examined by d-LOD. The location of the leading-one (or its absence) is used to round the divisor according to

$$d_{rnd} = \begin{cases} 2^7 = 128, & d_{n-1:n-3} = 1\text{xx} \\ 2^6 = 64, & d_{n-1:n-3} = 01\text{x} \\ 2^5 = 32, & d_{n-1:n-3} = 001 \\ 2^4 = 16, & d_{n-1:n-3} = 000 \end{cases}. \tag{4.6}$$

More precisely, the location of the leading-one is determined and the value of the rounded divisor d_{rnd} is computed by keeping the leading-one and setting all other bits to zero. In the case that the leading-one is absent, the leading-one is assumed to be at bit-position $n-4$ and the value of d_{rnd} consequently becomes 16.

The seven most-significant bits of the partial remainder $s^{(n-k)}_{n-1:n-7}$ are similarly examined by s-LOD to produce a rounded partial remainder $s^{(n-k)}_{rnd}$ according to

$$s^{(n-k)}_{rnd} = \begin{cases} 2^{n+k-1}, & s^{(n-k)}_{n+k-1:n+k-7} = 1\text{xxxxxx} \\ 2^{n+k-2}, & s^{(n-k)}_{n+k-1:n+k-7} = 01\text{xxxxx} \\ 2^{n+k-3}, & s^{(n-k)}_{n+k-1:n+k-7} = 001\text{xxxx} \\ 2^{n+k-4}, & s^{(n-k)}_{n+k-1:n+k-7} = 0001\text{xxx} \\ 2^{n+k-5}, & s^{(n-k)}_{n+k-1:n+k-7} = 00001\text{xx} \\ 2^{n+k-6}, & s^{(n-k)}_{n+k-1:n+k-7} = 000001\text{x} \\ 2^{n+k-7}, & s^{(n-k)}_{n+k-1:n+k-7} = 0000001 \\ 0, & s^{(n-k)}_{n+k-1:n+k-7} = 0000000 \end{cases}. \tag{4.7}$$

It should be noted that the width of $s^{(n-k)}$ is $n+k$ bits. In the case that the leading-one is absent, the partial remainder is rounded to zero. To provide a more tangible example,

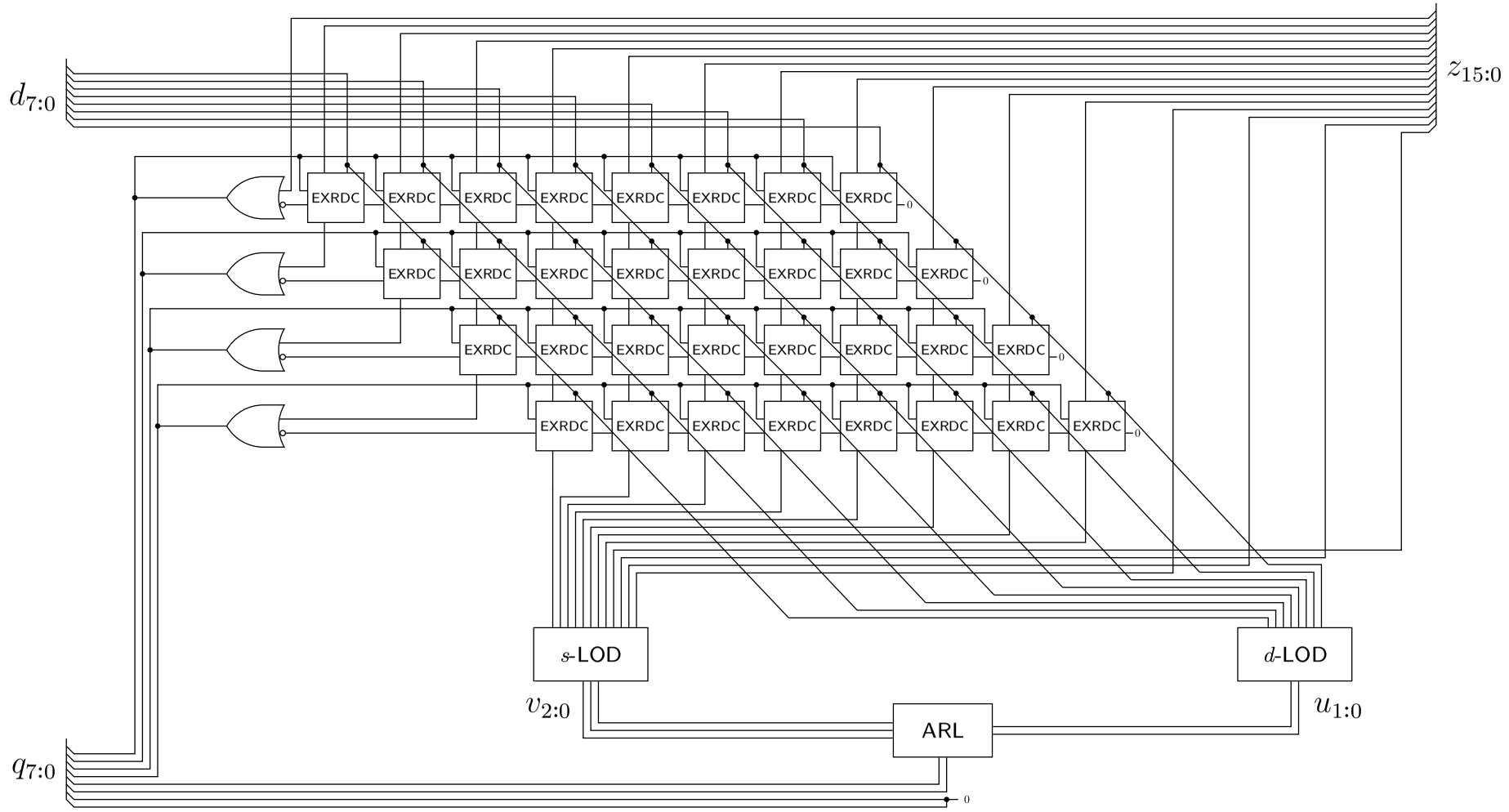


Fig. 4.5. Architecture of AXRD-M2 for $k = 4$.

Table 4.1. Divisor rounding and encoding to generate u

$d_{n-1:n-3}$	d_{rnd}	u_1	u_0
000	16	0	0
001	32	0	1
01x	64	1	0
1xx	128	1	1

Table 4.2. Partial remainder rounding and encoding to generate v

$s^{(n-k)}_{n+k-1:n+k-7}$	$s^{(n-k)}_{rnd}$	v_2	v_1	v_0
0000000	0	0	0	0
0000001	2^{n+k-7}	0	0	1
000001x	2^{n+k-6}	0	1	0
00001xx	2^{n+k-5}	0	1	1
0001xxx	2^{n+k-4}	1	0	0
001xxxx	2^{n+k-3}	1	0	1
01xxxxx	2^{n+k-2}	1	1	0
1xxxxxx	2^{n+k-1}	1	1	1

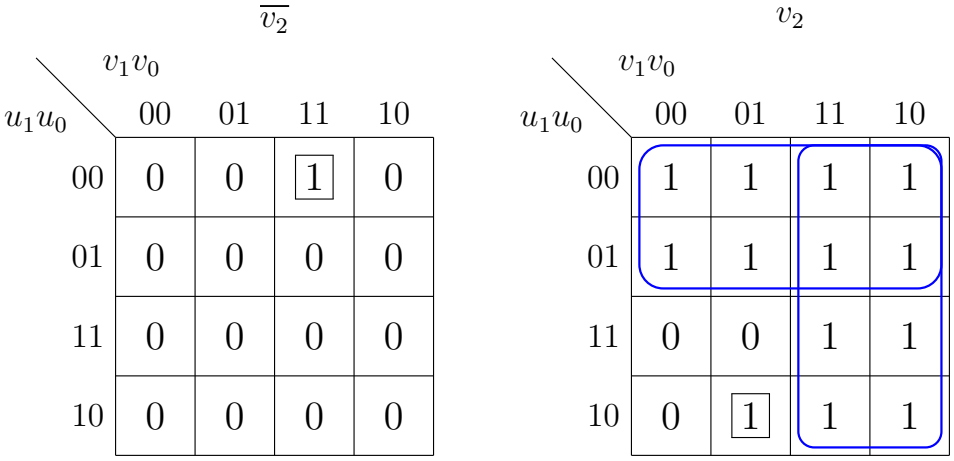
k = 4 can be used with (4.7) to generate

$$s^{(4)}_{rnd} = \begin{cases} 2^{11} = 2048, & s^{(4)}_{11:5} = 1\text{xxxxxx} \\ 2^{10} = 1024, & s^{(4)}_{11:5} = 01\text{xxxxx} \\ 2^{9} = 512, & s^{(4)}_{11:5} = 001\text{xxxx} \\ 2^{8} = 256, & s^{(4)}_{11:5} = 0001\text{xxx} \\ 2^{7} = 128, & s^{(4)}_{11:5} = 00001\text{xx} \\ 2^{6} = 64, & s^{(4)}_{11:5} = 000001\text{x} \\ 2^{5} = 32, & s^{(4)}_{11:5} = 0000001 \\ 0, & s^{(4)}_{11:5} = 0000000 \end{cases}, \tag{4.8}$$

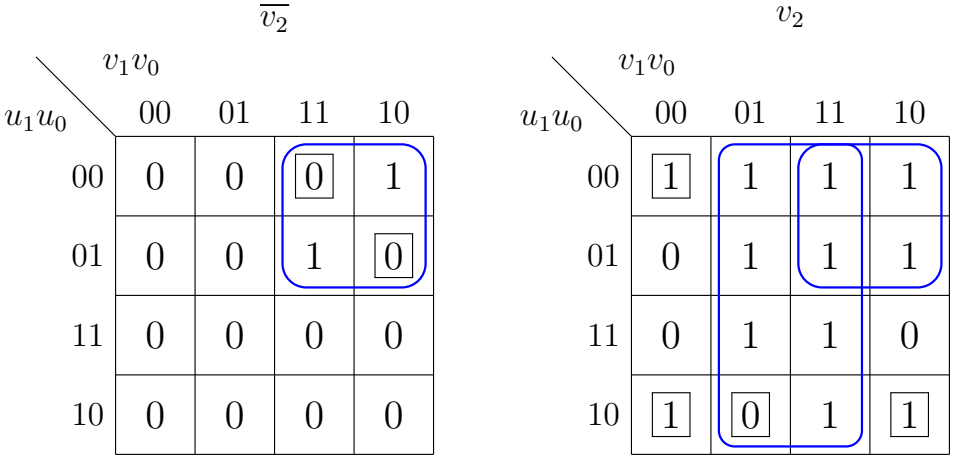
where the seven most-significant bits of $s^{(4)}$ are examined to produce $s^{(4)}_{rnd}$.

The rounded divisor d_{rnd} and rounded partial remainder $s^{(n-k)}_{rnd}$ are then used to generate encoded values u and v, respectively, where u is a 2-bit number and v is a 3-bit number. The process of rounding the divisor and partial remainder as well as generating u and v is illustrated in Table 4.1 and Table 4.2.

The quotient bits that have yet to be computed are approximated by dividing the rounded partial remainder by the rounded divisor and therefore can be expressed in terms



(a) Karnaugh-map for q_3



(b) Karnaugh-map for q_2

Fig. 4.6. Karnaugh-map for further approximating q_3 and q_2 in AXRD-M2, where $\boxed{1}$ represents a change from 0 to 1, $\boxed{0}$ represents a change from 1 to 0, and a blue box indicates approximated terms.

of u and v. However, these expressions are dependent on five input variables and are still more complex than is ideal. This is remedied by applying further approximation to the logic expressions for the remaining quotient bits $q_{k-1:0}$ via the use of Karnaugh-maps [82]. For $k = 4$, the Karnaugh-maps for q_3 and q_2 are shown in Fig. 4.6a and Fig. 4.6b, respectively, and the corresponding logic expressions are given by

$$q_3 = u_2u_1 + u_2\overline{v_1} \tag{4.9}$$

$$q_2 = u_2u_0 + u_1\overline{v_1}. \tag{4.10}$$

Table 4.3 demonstrates the process of rounding the divisor and partial remainder, generating u and v, and computing the remaining inexact quotient bits. Table 4.3 also compares the quotient value q_{acc} resulting from the division of the rounded partial remainder by the rounded divisor with the quotient value q_{app} derived from the Karnaugh-maps in Fig. 4.6. Error is exhibited only by q_{app} and specifically in cases where q_{acc} is larger than 15, due

Table 4.3. Use of encoded values u and v to produce approximate quotient bits for $k = 4$

$s_{rnd}^{(n-k)}$	d_{rnd}	Encoding		Quotient Values		
		$u_{2:0}$	$v_{1:0}$	q_{acc}	q_{app}	ED
0	16	000	00	0	0	0
0	32	000	01	0	0	0
0	64	000	10	0	0	0
0	128	000	11	0	0	0
32	16	001	00	2	2	0
32	32	001	01	1	1	0
32	64	001	10	0	0	0
32	128	001	11	0	0	0
64	16	010	00	4	4	0
64	32	010	01	2	2	0
64	64	010	10	1	1	0
64	128	010	11	0	0	0
128	16	011	00	8	8	0
128	32	011	01	4	4	0
128	64	011	10	2	2	0
128	128	011	11	1	1	0
256	16	100	00	16	15	1
256	32	100	01	8	8	0
256	64	100	10	4	4	0
256	128	100	11	2	2	0
512	16	101	00	32	15	17
512	32	101	01	16	15	1
512	64	101	10	8	8	0
512	128	101	11	4	4	0
1024	16	110	00	64	15	49
1024	32	110	01	32	15	17
1024	64	110	10	16	15	1
1024	128	110	11	8	8	0
2048	16	111	00	128	15	113
2048	32	111	01	64	15	49
2048	64	111	10	32	15	17
2048	128	111	11	16	15	1

to the fact that the approximation expresses the inexact quotient using only four bits. It should be noted that q_{acc} in this context is only "accurate" and not "exact" because approximation has already been applied when rounding the partial remainder and divisor. As visible in Fig. 4.5, the approximate recoded logic (ARL) block takes u and v as inputs and computes the inexact quotient bits according to (4.9) and (4.10).

For any approximate arithmetic circuit, the magnitude of approximation applied to the design is primarily limited by the ceiling on acceptable error metrics defined by the intended application. During the design of AXRD-M2, competitive error metrics were only achieved when eliminating no more than $n/2$ array rows, and thus larger approximation factors were not explored. Approximation factors of $k = 2, 3, 4$ exhibited a desirable balance of hardware savings and accuracy, and those designs were therefore selected for implementation. Only the architecture for $k = 4$ is illustrated in full (see Fig. 4.5), but the designs for $k = 2, 3$ can be expressed using the logic provided for $k = 4$. More specifically, the logic expression of q_3 for $k = 4$ is equivalent to that of q_2 for $k = 3$, and q_1 for $k = 2$. Similarly, the expression of q_2 for $k = 4$ is equal to q_1 for $k = 3$, and q_0 for $k = 2$. In the $k = 4$ design, the two least-significant quotient bits $q_1 q_0$ are truncated, as is the least-significant quotient bit q_0 for $k = 3$.

4.4 Experimental Results

This section provides an evaluation of the proposed dividers in terms of accuracy and hardware efficiency. Several state-of-the-art approximate dividers are selected from the literature for use in benchmarking. In [77], three inexact full-subtractors are used to implement approximate restoring divider (AXDr) models #1–3, where several cell replacement schemes are explored. In [79], an adaptive approximation-based divider (AAXD) is presented in which the operand bits are selectively trimmed according to the position of the leading-one, allowing for the division to be computed using a shifter and a reduced-width divider. An approximate hybrid divider (AXHD) is proposed in [66] in which EXRDCs are used to compute the MSBs of the quotient while the remaining LSBs are computed using a logarithmic division scheme. In the context of AXDr, AAXD, and AXHD designs, approximation factor k indicates the number of approximated columns via triangular replacement, the bitwidth of the pruned divisor, and the approximation depth as defined by the authors, respectively. Approximation factors were selected such that the proposed designs and the benchmarking designs exhibit similar error metrics, allowing for fair comparison.

4.4.1 Accuracy Evaluation

All models are implemented using Verilog HDL, and functional verification is implemented using a SystemVerilog testbench. For all models, accuracy was evaluated using exhaustive analysis. Python scripts are used to generate inputs and calculate accuracy metrics. The accuracy of the approximate dividers is evaluated using MRED and NMED as defined in [83], as well as error rate (ER) and maximum error distance ($\mathrm{D_{max}}$). Simulation results are summarized in Table 4.4, where accuracy measurements are provided for both quotient and remainder, where applicable.

Table 4.4. Accuracy and hardware metrics for approximate dividers

Design	k	**Quotient Accuracy**				**Remainder Accuracy**				**Hardware Metrics**				
		ER (%)	MRED (10^{-2})	NMED (10^{-2})	D_{max}	ER (%)	MRED (10^{0})	NMED (10^{0})	D_{max}	Delay (ns)	Area (μm^2)	Power (mW)	PDP (pJ)	ADP ($\mu m^2 \cdot$ns)
Exact	—	—	—	—	—	—	–	–	–	5.05	827.6	0.3703	1.870	4180
AXRD-M1	4	9.26	0.226	0.037	7	84.99	1.662	0.088	253	4.51	735.8	0.3325	1.500	3319
	6	23.47	0.494	0.101	31	94.25	2.269	0.175	255	4.05	635.0	0.2809	1.138	2572
	8	66.29	1.909	0.449	127	98.40	2.893	0.261	255	3.15	472.0	0.1856	0.585	1487
AXRD-M2	2	40.95	1.066	0.163	3	–	–	–	–	3.79	625.7	0.2554	0.968	2371
	3	64.73	2.214	0.351	7	–	–	–	–	3.15	523.8	0.2032	0.640	1650
	4	81.35	4.248	0.759	15	–	–	–	–	2.53	420.8	0.1525	0.386	1055
AXDr1 [77]	6	9.12	0.182	0.038	12	67.10	0.873	0.111	252	4.69	802.8	0.3472	1.628	3765
	8	50.93	1.207	0.292	51	86.97	3.103	0.292	255	4.53	778.7	0.3195	1.447	3527
	10	80.94	4.323	1.320	116	92.48	3.786	0.320	255	4.36	755.3	0.2854	1.244	3293
AXDr2 [77]	6	50.47	1.272	0.251	63	88.61	2.351	0.258	255	3.59	700.9	0.2957	1.062	2516
	8	96.43	6.543	1.395	255	96.98	4.407	0.329	255	2.66	567.7	0.2213	0.589	1510
	10	99.41	25.640	5.177	255	98.22	4.424	0.333	255	1.56	336.6	0.1039	0.162	525
AXDr3 [77]	6	11.95	0.211	0.049	21	84.01	1.748	0.120	254	5.02	721.8	0.3135	1.574	3623
	8	48.39	0.961	0.257	85	93.76	2.542	0.278	255	5.37	630.0	0.2538	1.363	3383
	10	78.64	3.251	0.967	119	95.98	2.265	0.278	255	5.75	568.8	0.2079	1.195	3271
AAXD [79]	5	73.74	1.521	0.716	14	–	–	–	–	3.23	870.1	0.2518	0.813	2811
	4	84.49	3.121	1.462	27	–	–	–	–	2.48	693.0	0.1666	0.413	1717
	3	91.06	6.343	2.966	49	–	–	–	–	1.81	580.3	0.1143	0.207	1050
AXHD [66]	12	15.86	3.946	0.019 [1]	4080	–	–	–	–	2.15	821.2	0.1473	0.317	1766
	14	15.86	3.946	0.058	16320	–	–	–	–	2.16	775.4	0.1435	0.310	1675
	16	15.86	3.946	0.208	65280	–	–	–	–	2.22	760.3	0.1421	0.315	1688

[a] AXHD achieves very low NMED values largely due to the fact that a $2n/n$ division in this design produces a $2n$-bit quotient rather than a n-bit quotient, meaning that NMED for this design is generated by dividing mean error distance (MED) by $2^{2n}-1$ rather than $2^{n}-1$, resulting in a substantially smaller value.

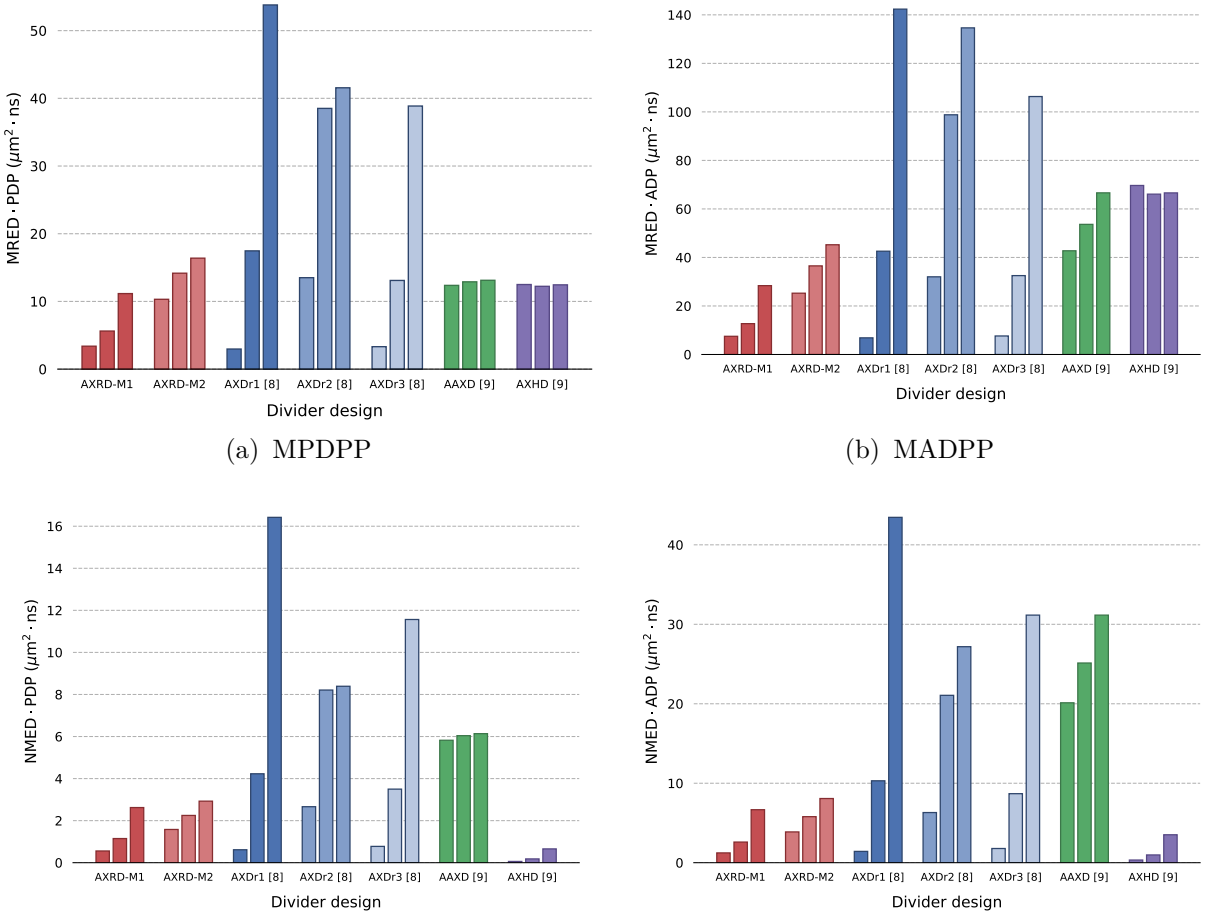
Table 4.4 shows that AXRD-M1, AXDr1 [77], and AXDr3 [77] perform the best in terms of quotient MRED, with the lowest MRED being achieved by AXDr1 for $k = 6$. Considering quotient NMED, the highest-performing designs are AXRD-M1, AXDr1 [77], AXDr3 [77], and AXHD [66], where AXHD for $k = 12$ has the lowest NMED value. It is important to note that the low NMED values achieved by AXHD are largely due to the fact that a $2n/n$ division in this design produces a $2n$-bit quotient rather than a n-bit quotient, meaning that NMED for this design is generated by dividing MED by $2^{2n} - 1$ rather than $2^n - 1$, resulting in a substantially smaller value. The lowest quotient ER is demonstrated by AXRD-M1 for $k = 4$ and AXDr1 [77] for $k = 6$, where both designs achieve similar values. AXRD-M1, AXRD-M2, and AAXD [79] have low D_{max}, and AXRD-M2 obtains especially small values.

The sequential nature of division means that errors introduced in one stage will be propagated to the next stage. Because the remainder is computed at the very final stage, approximate remainders tend to have poor accuracy. Only a subset of the implemented designs compute a remainder, namely AXRD-M1 and AXDr [77]. As reflected in Table 4.4, the remainder accuracy metrics for these designs is generally quite poor, although this is not of great concern. Regardless, it is worth mentioning that the smallest remainder MRED and NMED are achieved by AXDr1 [77] and AXRD-M1, respectively. All dividers that compute a remainder are capable of producing maximal or almost-maximal error distances in the remainder output.

4.4.2 Hardware Evaluation

All dividers are synthesized for the TSMC 65 nm process technology using Synopsys Design Compiler. For all simulations, an operating voltage of 1 V and an operating temperature of 25 °C are used. Synopsys Design Compiler is used to report critical path delay, circuit area, and power dissipation, while PDP and ADP are manually computed to provide a more comprehensive measurement of hardware efficiency. Simulation results are summarized in Table 4.4.

The proposed AXRD-M1 and AXRD-M2 provide reductions in critical path delay of up to 38% and 49%, respectively, when compared to the exact design. The AXDr [77] designs have relatively high delays, especially AXDr3 whose delay actually surpasses that of the exact divider for $k > 6$. The most impressive delay metrics are achieved by AAXD [79] and AXHD [66], where AAXD for $k = 3$ demonstrates a reduction of 64%. In terms of circuit area, AXRD-M1, AXRD-M2, and AXDr2 [77] exhibit the most substantial reductions. AXRD-M1 and AXRD-M2 achieve area savings of up to 43% and 46%, respectively, when compared to the exact divider, while AXDr2 [77] for $k = 10$ has the lowest circuit area of all implemented designs with a reduction of 59%. It should be noted that AAXD [79] reports an area larger than that of the exact divider for $k = 5$. The greatest reduction in power dissipation is achieved by AXRD-M1, AXRD-M2, AAXD [79], and AXHD [66]. When



(a) MPDPP

(b) MADPP

(c) NPDPP

(d) NADPP

Fig. 4.7. Graphs illustrating accuracy-hardware tradeoff metrics for the implemented dividers.

compared to the exact divider, AXRD-M1 and AXRD-M2 reduce power consumption by up to 50% and 57%, respectively, while the largest reduction of 69% is obtained by AAXD [79] for $k = 3$.

A greater understanding of the hardware performance for a given design can be obtained by considering the combined hardware metrics shown in Table 4.4, namely PDP and ADP. The most impressive PDP values are those for AXRD-M2, AAXD [79], and AXHD [66]. The proposed design AXRD-M2 provides a PDP reduction of up to 79%, relative to the exact divider, and the smallest PDP for all implemented designs is achieved by AAXD for $k = 3$, with a reduction of 89%. The designs exhibiting the lowest ADP values are AXRD-M2, AXDr2 [77], and AAXD [79], where the proposed AXRD-M2 has a reduction of up to 75%, and the lowest ADP is obtained by AXDr2 [77] for $k = 10$ with a reduction of 87%.

4.4.3 Accuracy-Performance Tradeoff

As shown in Table 4.4, the smallest MRED is achieved by AXRD-M1, AXDr1 [77], and AXDr3 [77], which exhibit similar values for $k = 4, 6, 6$, respectively. Among these designs,

Table 4.5. Accuracy-hardware tradeoff metrics for approximate dividers

Design	k	**Tradeoff Metrics**			
		MPDPP (fJ)	MADPP ($\mu m^2 \cdot ns$)	NPDPP (fJ)	NADPP ($\mu m^2 \cdot ns$)
AXRD-M1	4	3.388	7.497	0.558	1.234
	6	5.624	12.715	1.149	2.598
	8	11.158	28.375	2.623	6.669
AXRD-M2	2	10.314	25.266	1.581	3.874
	3	14.169	36.524	2.248	5.794
	4	16.390	45.231	2.927	8.077
AXDr1 [77]	6	2.968	6.862	0.616	1.425
	8	17.468	42.572	4.229	10.308
	10	53.794	142.361	16.422	43.458
AXDr2 [77]	6	13.506	32.015	2.663	6.312
	8	38.514	98.804	8.210	21.062
	10	41.558	134.635	8.390	27.182
AXDr3 [77]	6	3.315	7.633	0.776	1.788
	8	13.099	32.515	3.499	8.686
	10	38.864	106.330	11.565	31.641
AAXD [79]	5	12.371	42.750	5.823	20.123
	4	12.895	53.637	6.041	25.130
	3	13.123	66.630	6.137	31.156
AXHD [66]	12	12.497	69.670	0.060	0.337
	14	12.232	66.097	0.181	0.978
	16	12.449	66.608	0.657	3.517

AXRD-M1 exhibits the smallest delay, AXRD-M1 and AXDr3 [77] achieve the smallest area metrics, and AXDr3 [77] features the most impressive power savings. The smallest NMED is achieved by AXRD-M1, AXDr1 [77], AXDr3 [77], and AXHD [66], where similar values are exhibited for $k = 4, 6, 6, 12$, respectively. Among these designs, AXRD-M1 and AXDr3 [77] provide a modest advantage in area savings, while AXHD [66] achieves substantial improvements over the others designs in terms of both delay and power consumption.

The accuracy and hardware metrics reported in Table 4.4 vary significantly according to implemented design and approximation factor. In order to obtain a better understanding of the general ability of a divider to improve hardware performance while maintaining accuracy, an analysis of the accuracy-performance tradeoff of each design is needed. Several works have made use of hybrid accuracy-hardware metrics to serve this purpose [66], [76], [77], [84]–[86]. To provide a thorough evaluation of all implemented designs, four hybrid accuracy-hardware metrics are proposed, namely MRED-PDP prod-

uct (MPDPP) [86], MRED-ADP product (MADPP), NMED-PDP product (NPDPP), and NMED-ADP product (NADPP). Fig. 4.7 illustrates the relationship between the hybrid metrics and approximation factor, where the metrics are summarized in Table 4.5. The lowest MPDPP and MADPP are achieved by AXDr1 [77], while the lowest NPDPP and NADPP correspond to AXHD [66]. As per the discussion in Section 4.4.1, it is unsurprising that AXHD [66] has such low values for these metrics, considering the dependence on NMED. While AXDr1 [77] has very low MPDPP and MADPP for $k = 6$, it can be seen in Fig. 4.7 that these values increase rapidly with k, and a similar trend is visible in AXDr3 [77]. The proposed designs AXRD-M1 and AXRD-M2 exhibit hybrid metrics on par with the highest-achieving of the implemented designs. Additionally, it is apparent from Fig. 4.7 that the proposed dividers provide an advantage over the benchmarking designs in that accuracy metrics scale well with k.

4.5 Applications

The proposed designs were tested using two image processing applications, namely change detection and foreground extraction. Performance is evaluated in terms of both PSNR and SSIM, where SSIM can better separate structural information from image distortion.

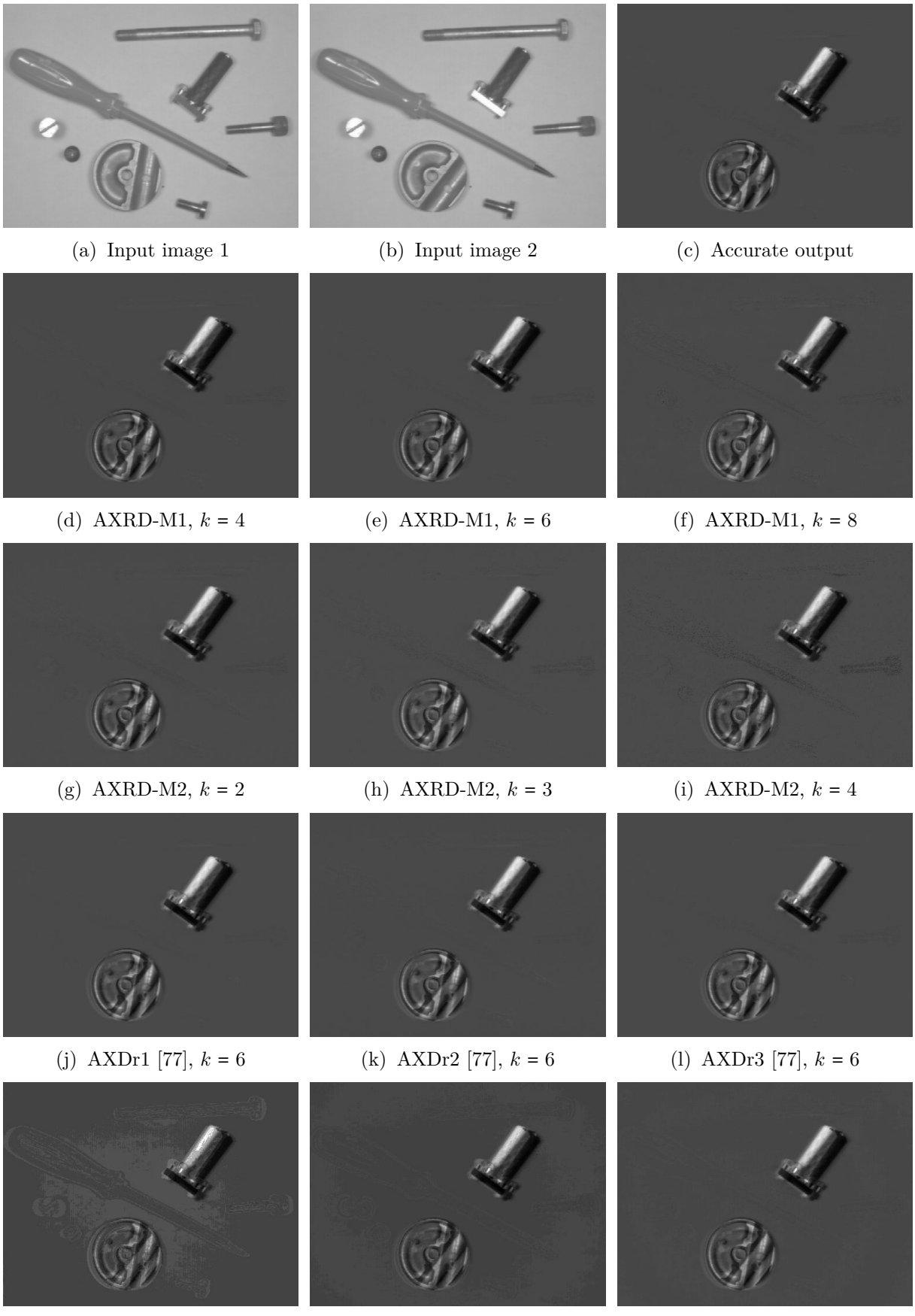
4.5.1 Change Detection

Considering two images of equal size, the change between these images can be detected by performing a per-pixel division, where the output image is constructed from the computed quotient values. Dividers are used to detect the change between two 8-bit grayscale images as shown in Fig. 4.8. The PSNR and SSIM values computed for the dividers are summarized in Table 4.6.

As visible in Fig. 4.8d and 4.8e, the proposed dividers produce images that are highly similar to that generated by the exact divider. The images most closely resembling the accurate output were generated by AXRD-M1 AXRD-M2 and AXDr [77]. AXDr3 [77] produces the best result with a PSNR and SSIM of 60.9 dB and 0.9997, respectively. AXRD-M1 achieves very similar performance, with a PSNR and SSIM of 60.2 dB and 0.9996, respectively.

4.5.2 Foreground Extraction

Given an image with a visually disruptive background variation, the division of the image by its background allows for the foreground to be extracted. Approximate dividers are used to extract the foreground of the 8-bit grayscale image shown in Fig. 4.9a by dividing it by its background as shown in Fig. 4.9b. The image in Fig. 4.9a is multiplied by a factor of 64 prior to division to improve precision. Application performance is again measured



(a) Input image 1 (b) Input image 2 (c) Accurate output

(d) AXRD-M1, k = 4 (e) AXRD-M1, k = 6 (f) AXRD-M1, k = 8

(g) AXRD-M2, k = 2 (h) AXRD-M2, k = 3 (i) AXRD-M2, k = 4

(j) AXDr1 [77], k = 6 (k) AXDr2 [77], k = 6 (l) AXDr3 [77], k = 6

(m) AAXD [79], k = 3 (n) AAXD [79], k = 4 (o) AAXD [79], k = 5

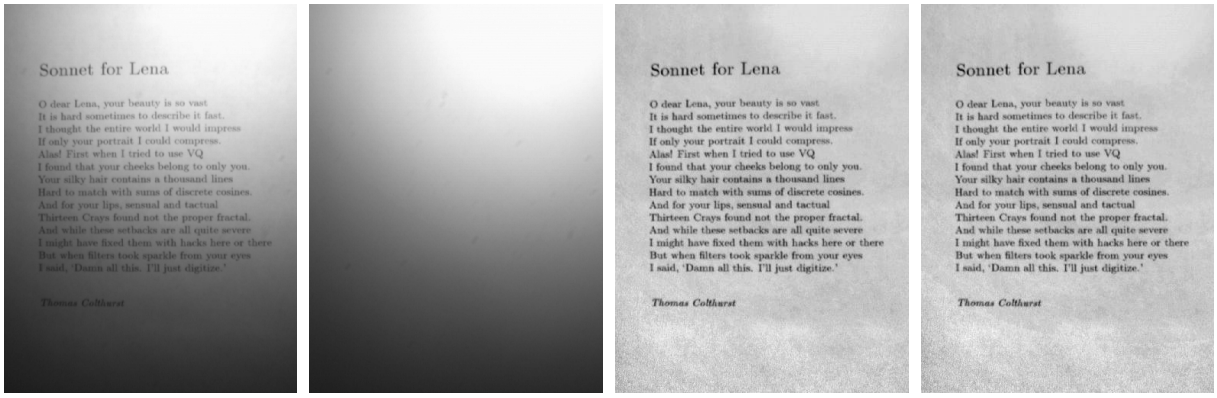
Fig. 4.8. Change detection results computed using various dividers.

Table 4.6. Application metrics for approximate dividers

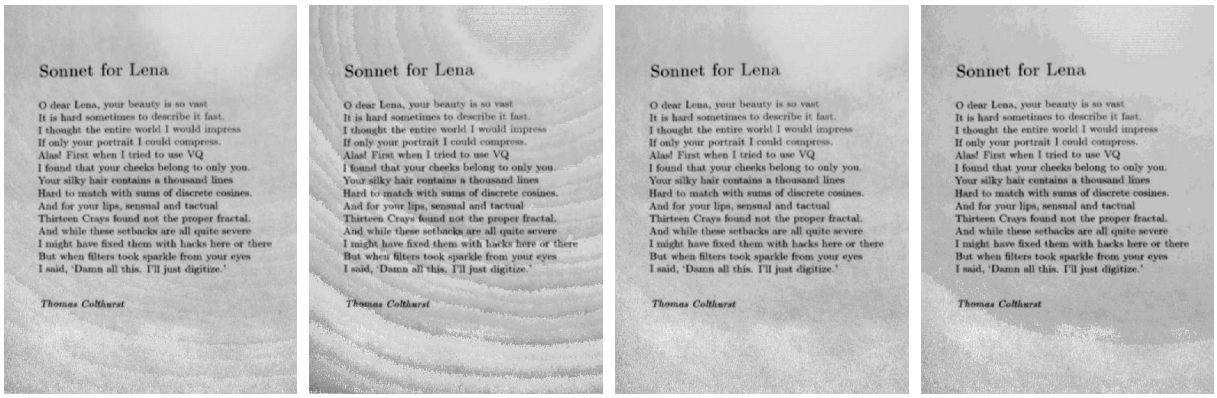
Design	k	**Change Detection**		**Background Removal**	
		PSNR (dB)	SSIM	PSNR (dB)	SSIM
AXRD-M1	4	60.2	0.9996	51.8	0.9957
	6	57.7	0.9993	48.9	0.9920
	8	50.7	0.9959	42.1	0.9765
AXRD-M2	2	53.7	0.9983	48.5	0.9927
	3	51.3	0.9972	44.4	0.9887
	4	45.3	0.9893	37.6	0.9739
AXDr1 [77]	6	57.4	0.9993	49.1	0.9936
	8	52.2	0.9982	42.6	0.9868
	10	40.8	0.9614	36.1	0.9440
AXDr2 [77]	6	51.9	0.9981	42.9	0.9901
	8	41.6	0.9912	35.3	0.9593
	10	29.8	0.9262	29.6	0.8309
AXDr3 [77]	6	60.9	0.9997	57.1	0.9988
	8	54.8	0.9986	46.4	0.9859
	10	45.2	0.9841	36.7	0.9219
AAXD [79]	5	46.9	0.9949	45.0	0.9857
	4	39.5	0.9829	40.7	0.9683
	3	36.8	0.9474	35.1	0.9257
AXHD [66]	12	46.3	0.9959	42.7	0.9874
	14	46.3	0.9959	42.7	0.9874
	16	46.3	0.9959	42.7	0.9874

using PSNR and SSIM, where the computed metrics are summarized in Table 4.6.

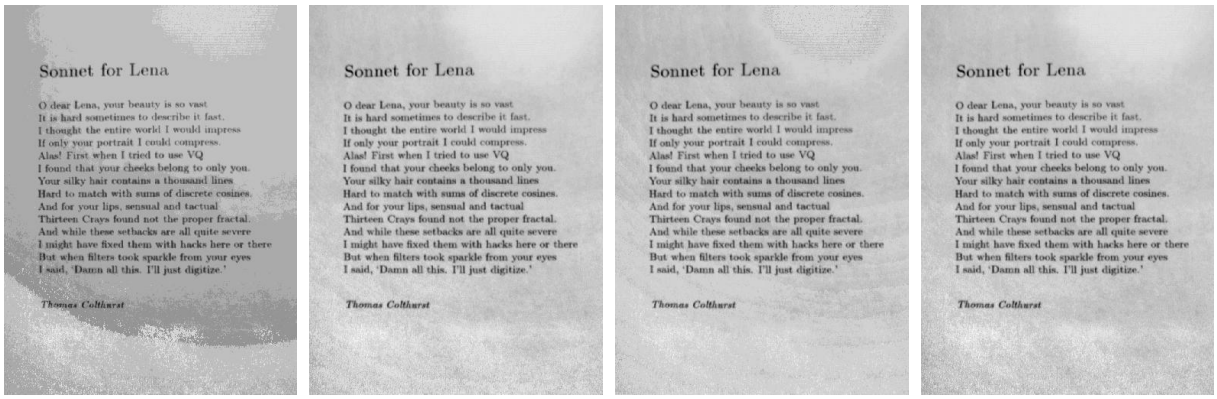
Most approximate dividers produce outputs similar to that of the exact divider. The proposed dividers AXRD-M1 and AXRD-M2 produce good results for $k = 4, 6$ and $k = 2, 3$, respectively. The output image of highest quality was produced by AXDr3 [77] for $k = 6$, where the corresponding PSNR and SSIM are 57.1 dB and 0.9988, respectively. Comparatively, AXRD-M1 for $k = 4$ achieves competitive metrics with a PSNR and SSIM of 51.8 dB and 0.9957, respectively.



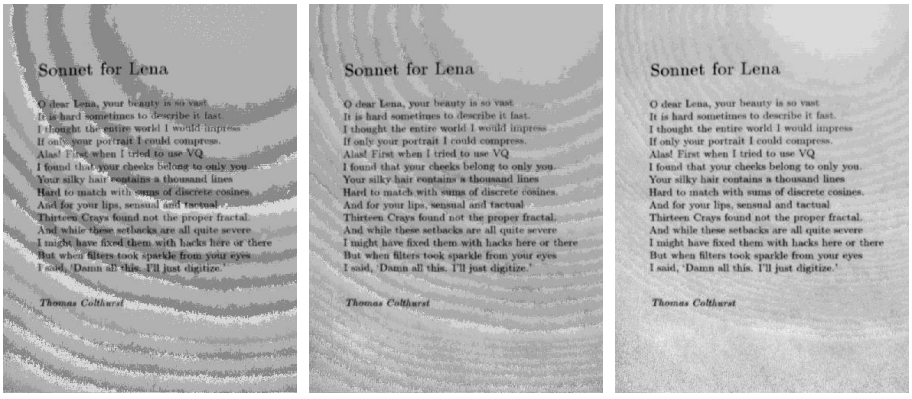
(a) Input image (b) Background (c) Accurate output (d) AXRD-M1, $k = 4$



(e) AXRD-M1, $k = 6$ (f) AXRD-M1, $k = 8$ (g) AXRD-M2, $k = 2$ (h) AXRD-M2, $k = 3$



(i) AXRD-M2, $k = 4$ (j) AXDr1 [77], $k = 6$ (k) AXDr2 [77], $k = 6$ (l) AXDr3 [77], $k = 6$



(m) AAXD [79], $k = 3$ (n) AAXD [79], $k = 4$ (o) AAXD [79], $k = 5$

Fig. 4.9. Foreground extraction results computed using various dividers.

Chapter 5

Approximate Fixed-Point MAC Unit

This chapter proposes an approximate MAC unit whose architecture is based on the ABM-M3 multiplier presented in Chapter 3. Several other approximation techniques are utilized alongside the ABM-M3 architecture to propose a novel MAC design. The design is compared with other state-of-the-art inexact MAC designs in terms of error and hardware metrics. Accuracy is evaluated using MRED, NMED, and mean absolute error (MAE), while hardware performance is measured by area and power consumption. Hybrid accuracy-hardware metrics are computed to provide insight into the ability of each design to reduce hardware costs while maintaining accuracy. Finally, the MAC units are verified using a Gaussian blur application for 5×5 and 7×7 filters, where performance is measured in terms of PSNR and SSIM.

The author contributions are outlined in Section 5.1. A brief review of the ABM-M3 architecture from Chapter 3 is given in Section 5.2, and the architecture of the proposed MAC unit is presented in Section 5.3. Accuracy and hardware performance are analyzed in Section 5.4, and Section 5.5 provides a discussion of the Gaussian blur applications.

5.1 Author Contributions

E.A. conceived of and designed the proposed MAC model which utilizes techniques originally presented by S.V. in the multiplier work. E.A. carried out the implementation of benchmarking models from the literature. E.A. carried out the functional simulations and synthesis simulations. E.A. analyzed the accuracy and hardware metrics of the proposed model. E.A. analyzed the accuracy-hardware tradeoff performance of the proposed model. E.A. carried out the Gaussian blur applications by simulating convolution operations. E.A. wrote the (unpublished) manuscript in consultation with S.V. and S.K.

5.2 Approximate Booth Multiplier (ABM-M3)

The architecture of the mantissa multiplier for the proposed approximate MAC unit is based on the approximate 8-bit ABM-M3 from Chapter 3 with approximation factor $k = 6$.

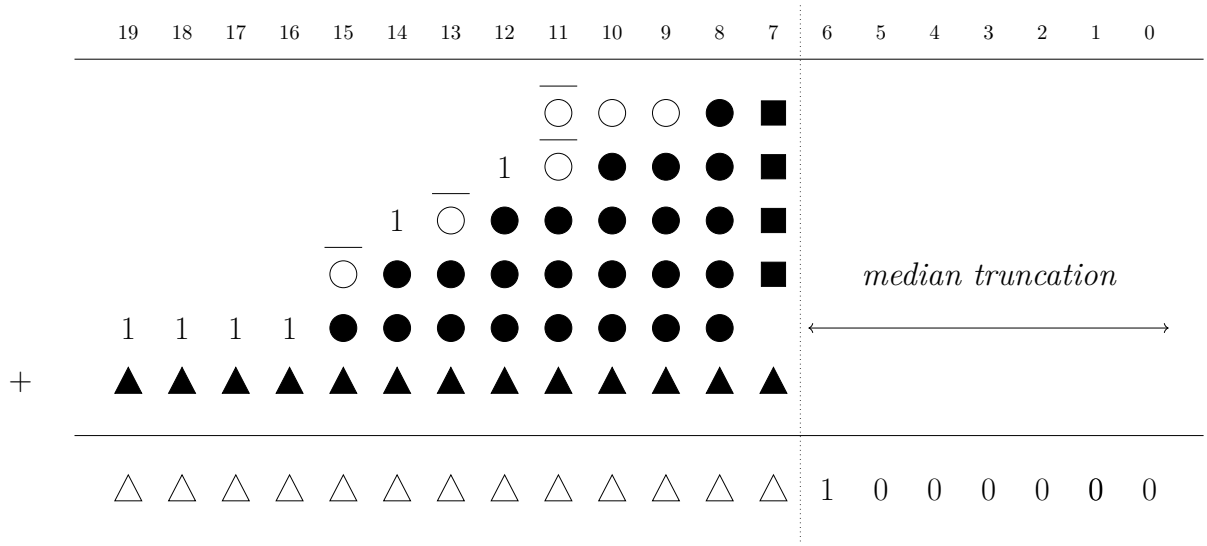


Fig. 5.1. Bit matrix of ABM-MAC in which the partial product bits generated by the multiplication are summed along with the accumulator value in a single computation. The $k = 6$ least-significant bits of the result are truncated to the median value. ●: exact partial products, ○: sign-extension terms, ▲: inexact partial products, ■: accumulator bits from the prior computation, □: newly-computed accumulator value.

In the ABM-M3 design, all partial products with significance less than k are reduced to a single column of approximate partial products. For a given partial product matrix row i, let l be the number of bits with a significance less than k. For a row i, $x_{\forall j \in (0,k-(2i+1))}$ is generated by *OR*-ing the l least-significant bits of X. The approximate partial product for the row i is then generated by the use of PPG-1S which takes in the signal $zero_i$ and the result of the *OR*-operation to compute the approximate partial product for that row.

5.3 Approximate Booth Multiplier-Based MAC (ABM-MAC)

The architecture of the proposed ABM-MAC consists of three stages: (1) the inputs are clocked to registers, (2) the multiplication of the inputs and the summation of the product with the accumulator value are computed as a single fused operation, and (3) the result is clocked to the output register. Fig. 5.1 illustrates the bit matrix that is operated on during the fused operation. The method of generating partial products is as described in Section 5.2, where the partial products in the $k = 6$ least-significant columns are compressed row-wise into single inexact partial products via PPG-1S. The remaining $2n - k$ columns are generated using the exact PPG. As visible in Fig. 5.1, selecting an approximation factor of $k = 6$ alleviates the need to compute any sign-correction terms cor_i. This is not immediately advantageous during the partial product generation since $cor_i = neg_i$ in this design and neg_i still requires computation. However, considering that partial product compression tends to be the most resource-intensive portion of the computation, it is beneficial to reduce the number of terms to be accumulated.

5.3.1 Partial Product Compression

The accumulator value is included in the initial matrix as the bottom-most row in Fig. 5.1, effectively fusing the partial product compression with the accumulation operation. During the initial design of the unit, hardware simulations quickly indicated that fusing the operations was the better decision. It may seem that choosing to fuse the operations would not have a substantial effect on the efficiency of the model, especially considering that both the fused and non-fused units contain precisely the same number of half- and full-adders. However, the ordering of compression stages can have a significant impact on critical path delay. The reason the proposed MAC unit is poorly-suited for implementation via a non-fused architecture is related to the fact that it utilizes a Booth-based multiplication algorithm. In a traditional *AND*-array multiplier, the partial products are each generated using only a single *AND*-gate and can be computed in parallel as soon as the inputs become available. Conversely, in the radix-4 Booth multiplier, the multiplicand must first be encoded in terms of radix-4 digits before partial products may be generated. Additionally, the PPGs in a Booth multiplier are significantly more complex than the single *AND*-gate that is needed to generate a partial product in the *AND*-array multiplier. Because of these factors, the partial products in a Booth multiplier are generated with some additional delay. This is counteracted by the fact that the radix-4 Booth algorithm reduces the number of partial product terms by half, or almost half in the case of an unsigned multiplier, therefore greatly reducing the hardware needed for the accumulation. While the reduced partial product matrix can provide significant advantages in terms of hardware reduction, there are still cases in which the increased delay associated with partial product generation can be problematic. During the initial development of ABM-MAC, the combination of delays introduced during partial product generation and the latency incurred during accumulation result in a critical path delay that was actually slightly larger than that of the exact MAC unit. The partial product matrix generated by a non-fused ABM-MAC has a height of $h = 5$. Considering that the Dadda compression algorithm defines the height of the matrix after the next compression stage j as

$$h_{j+1} = 2 \cdot \left\lfloor \frac{h_j}{3} \right\rfloor + h_j \bmod 3, \tag{5.1}$$

it can be deduced that a total of four compression stages are needed to reduce the matrix into a single product term p. However, since the operation is not fused, p needs to be additionally be summed with the accumulator z. As a result, there are effectively a total of five compression stages required to compute the MAC operation. Additionally, it should be noted that the latter accumulation stages tend to be slower than earlier stages. Each compression stage produces a matrix of reduced height but greater width and, as a result, the length of the carry-propagate chain increases with each compression. Considering all

of this, it becomes clear that this design is better suited for implementation via a fused operation. In this fused operation, z is included as the bottom-most row of the partial product matrix which now has an initial height of $h = 6$. Due to the occurrence of the floor function in (5.1), the operation can still be computed using only four compression stages. The use of the fused operation results in a critical path delay that meets the timing constraints derived from the delay of the exact MAC unit while also reducing hardware cost. These hardware savings are examined further in Section 5.4.2.

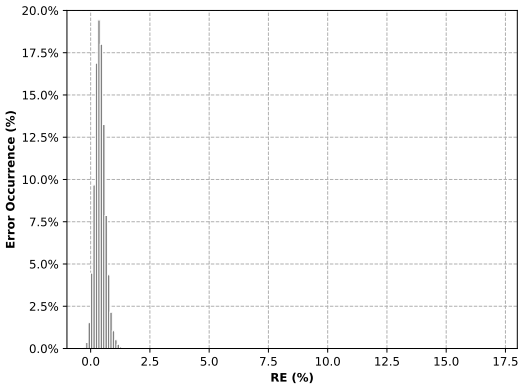
5.3.2 Median Value Truncation

While MAE is an important metric in accuracy analysis, the distribution of the error function is of special significance for the MAC unit due to the fact that it repeatedly accumulates the results of its computations. Even if the magnitude of the error produced is not large, the error observed in the output will grow with each accumulation if the mean of the error function is not centered near zero. It is desirable in this architecture to have an even-sided error function, i.e. one that tends to produce positive errors as often as it produces negative errors. Truncation can become a potential issue since rounding everything to zero will underestimate the result. Of course, the accuracy is affected not only by the truncation error but also by any other error function that results from additional approximation techniques. Thus, while it may not be immediately evident which rounding scheme is ideal for minimizing the error of a given design, it is worth investigating alternate techniques to use in place of traditional truncation.

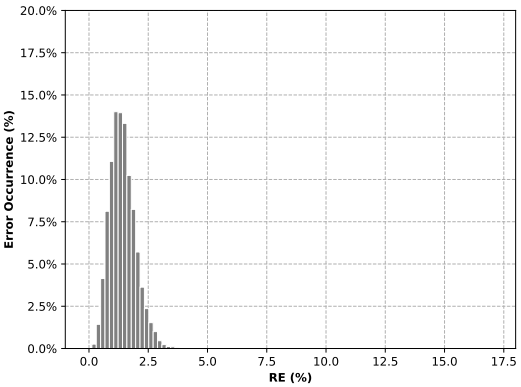
The error E produced by an approximate computation for an inexact result a and correct result b is given by $E_{(a,b)} = a - b$, and thus relative error RE can be expressed as

$$RE(a, b) = \frac{a - b}{b} = \frac{E}{b}. \tag{5.2}$$

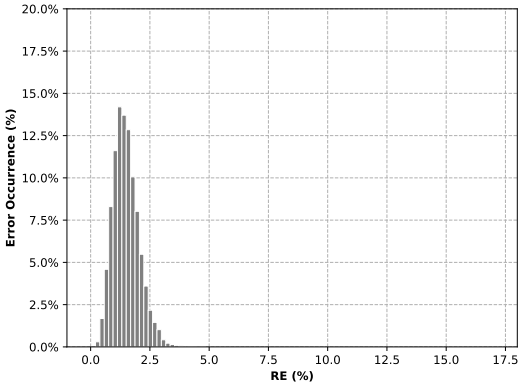
Because relative error is not a magnitude but rather indicates positive or negative direction, its distribution can be analyzed to gain insight into the ability of a design to a provide an even-sided error function. The distribution of RE for ABM-MAC is shown in Fig. 5.2a. The error distribution was calculated on data derived from computing 10000 randomized operations each involving 25 accumulations. It may also be noted that this is the same data used to perform the accuracy analysis provided in Section 5.4.1. While not perfectly zero-centered, it can be seen that the distribution occupies the region near zero. This distribution was considered alongside other accuracy evaluation techniques when determining the truncation method that will result in the highest computation accuracy. Despite the fact that the MAC unit still slightly overestimates the operation result, it was determined though experimentation that truncating to the median value (i.e. 7b1000000) minimizes error. The effect of error distribution on MAC unit accuracy is discussed further in Section 5.4.1.



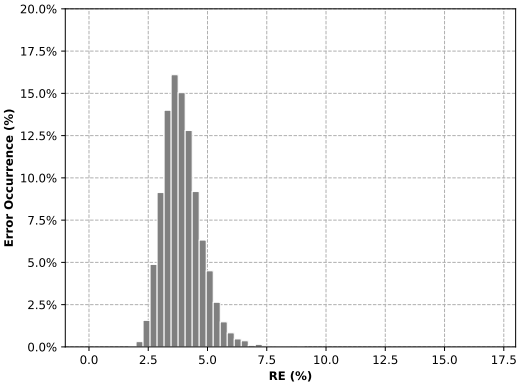
(a) ABM-MAC



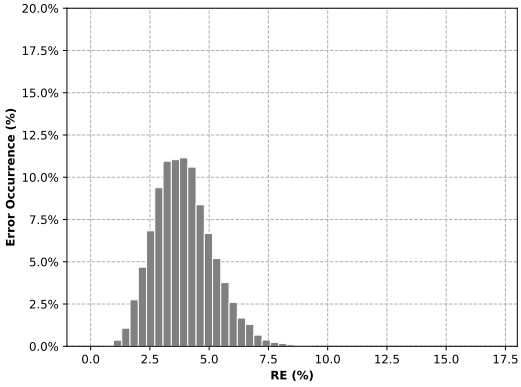
(b) Design #1 [80]



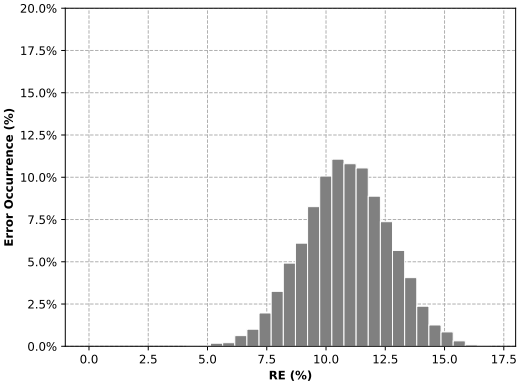
(c) Design #2 [80]



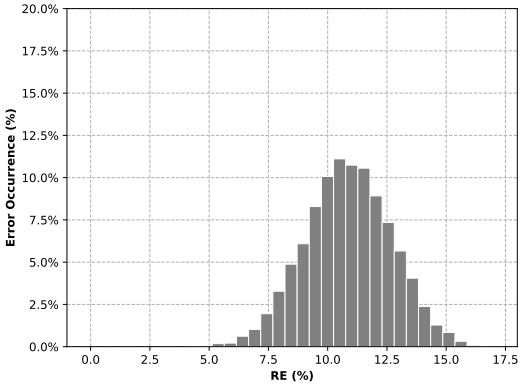
(d) Design #3 [80]



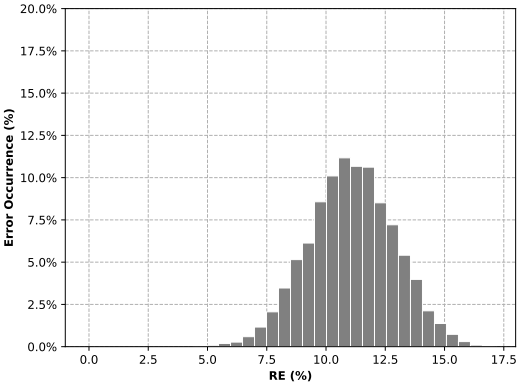
(e) ATC-MAC [81]



(f) AP-MAC (k = 4) [87]



(g) AP-MAC (k = 6) [87]



(h) AP-MAC (k = 8) [87]

Fig. 5.2. Relative error distribution for implemented MAC units.

Table 5.1. Accuracy and hardware metrics for approximate MAC units

Design	Accuracy Metrics			Hardware Metrics	
	MRED (10^{-2})	NMED (10^{-2})	MAE (10^{4})	Area (μm^2)	Power (mW)
Exact MAC	—	—	—	2224.8	2.2482
ABM-MAC	0.391	0.281	0.422	1279.4	1.5084
Design #1 [80]	1.423	1.050	1.557	1420.9	2.0723
Design #2 [80]	1.493	1.100	1.585	1216.8	1.7035
Design #3 [80]	3.925	2.859	2.640	734.0	1.1395
ATC-MAC [81]	3.946	2.954	5.145	1046.5	1.8310
AP-MAC (k = 4) [87]	10.858	8.176	9.651	1776.6	1.7391
AP-MAC (k = 6) [87]	10.861	8.179	9.651	1454.4	1.6581
AP-MAC (k = 8) [87]	11.082	8.337	9.715	1212.1	1.6107

5.4 Experimental Results

This section provides an evaluation of the proposed design in terms of both accuracy and hardware by comparing its performance to other approximate MAC units from the literature. Works used for benchmarking include: (1) Designs #1–3 from [80], (2) the approximate tree compressor-based MAC (ATC-MAC) from [81], and (3) the approximate MAC (AP-MAC) from [87]. The AP-MAC design presented in [87] is accuracy-configurable via the selection of an approximation factor. Several approximation factors were considered, and it was decided that AP-MAC would be analyzed for $k = 4, 6, 8$, as the resulting models produced accuracy and hardware metrics that made for a fair comparison with the other designs. It should be noted that [87] provides two primary contributions: a novel approximate MAC unit, and a data reuse method for use in the given neural network which aims to reduce the number of MAC computations. The data reuse method is designed specifically for the neural network proposed in [87] and moreover is outside of the scope of this work, so only the approximate MAC unit architecture is implemented for use in benchmarking. It should also be noted that all designs were modified as to allow for accumulation up to a bitwidth of 20. This is because a bitwidth of 20 is needed to perform the filtering applications discussed in Section 5.5.

The results of the accuracy and hardware evaluation are summarized in Table 5.1. All designs are implemented in Verilog HDL and synthesized using Synopsys Design Compiler for the TSMC 65 nm process technology library. Synthesis simulations are configured to use an operating voltage of 1 V and an operating temperature of 25 °C. Functional verification is performed using SystemVerilog testbenches. All data processing and error analysis is performed using Python scripting.

5.4.1 Accuracy Analysis

A set of 10000 random 8-bit input operations is used for testing, where each operation involves resetting the model to clear the accumulator and then performing 25 MAC operations before reading out the result. The input set size of 25 operations was chosen as to imitate a 5×5 filter. MRED, NMED, and MAE are selected to measure accuracy as they provide insight into both the magnitude and the distribution of the error. Because a 20-bit accumulator is used, values for NMED are calculated by normalizing the MED by 2^{19}. Relative error is also computed so that the error distribution of each design may be analyzed. Table 5.1 shows that the proposed ABM-MAC performs best across all accuracy metrics compared to other designs. AP-MAC [87] performs relatively poorly in terms of accuracy for all k. While Design #1 and Design #2 from [80] perform somewhat competitively, the proposed ABM-MAC achieves lower error metrics by nearly an order of magnitude for all three measurements.

Error distribution can provide key insight into the reason for which a MAC unit exhibits a given accuracy. The distribution of the relative error for all models is illustrated in Figure 5.2. All distributions are Gaussian in nature, although the mean and deviation vary greatly among designs. The feature that seems to be most evidently related to model accuracy is the distribution mean. AP-MAC [87] performs poorly in terms of accuracy, and the mean of its error distribution is ~10%. Design #3 [80] and ATC-MAC [81] exhibit similar error metrics and their distribution means both lie near ~3.5%. Design #1 and #2 [80] achieve decent accuracy and have a mean error of ~1.5%. Finally, with a error distribution mean of ~0.35%, ABM-MAC exhibits strong accuracy metrics. While the cause for each model to generate the given distribution is complicated, the likely reason for ABM-MAC to produce a distribution closer to zero lies in the fact that it relies on Booth-based multiplication. Due to the nature of Booth multiplication terms, the approximated partial product terms in ABM-MAC may belong to a partial product row that is either positive or negative. The approximation techniques used in ABM-MAC do not affect whether partial products are considered positive or negative, and thus the error produced by ABM-MAC has opportunities to be both positive and negative. The ability of the Booth-based architecture to generate a more balanced error distribution makes it well-suited for use in approximate MAC units.

5.4.2 Hardware Analysis

The maximum critical path delay of the exact MAC unit was measured to be 0.74 ns, and this value was then set as the clock period for all other simulations. The area and power measurements for all simulations are summarized in Table 5.1, and the percentage improvements over the exact design are illustrated in Fig. 5.3. Design #3 from [80] exhibits hardware metrics which are superior to those for all other designs. The ATC-

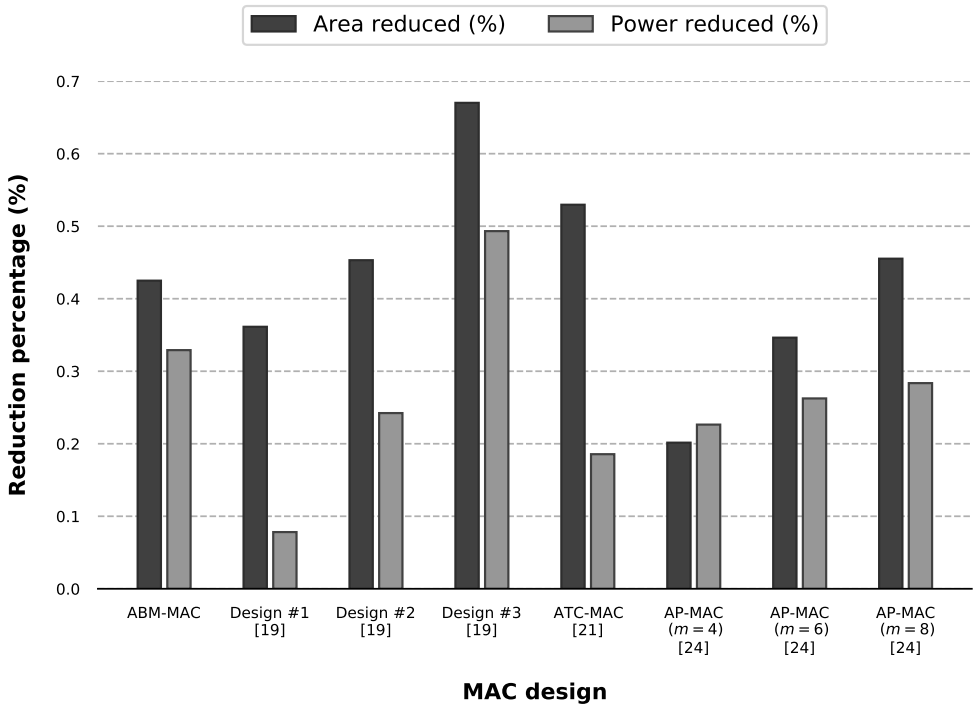


Fig. 5.3. Hardware savings for approximate MAC units compared to the accurate design.

MAC design interestingly performs very well in terms of area but somewhat poorly in power consumption. The proposed ABM-MAC performs similarly to Design #2 [80], except Design #2 achieves slightly better area metrics, and ABM-MAC achieves better power consumption. Design #1 provides moderate area savings and negligible power savings, which is unsurprising since it introduces a low level of approximation. AP-MAC for $k = 8$ achieves a strong area reduction, but otherwise the AP-MAC designs perform poorly in terms of hardware performance when compared to the other models.

Hardware simulations can demonstrate the advantage of choosing a fused-operation architecture for ABM-MAC. The non-fused MAC unit was synthesized and its hardware metrics are compared to those of the fused MAC design. The non-fused design actually failed to meet the 0.74 ns timing constraint set by the exact MAC unit and thus, to provide a fair comparison of the fused and non-fused models, the critical path delay of the unfused design was determined to be 0.78 ns, and this delay was then set as the clock period for the simulations used in this comparison. The percentage area and power saved for both the fused and non-fused ABM-MAC are illustrated in Fig. 5.4. It can be seen that, while the designs perform nearly identically in terms of power, the fused model reduces circuit area by ~15%. Considering that the result computed in both architectures is identical, it is clear that the fused model is advantageous in this application.

5.4.3 Accuracy-Performance Tradeoff Evaluation

Evaluating accuracy and hardware usage individually is critical in determining the overall level of performance for a given arithmetic unit. However, a holistic opinion cannot

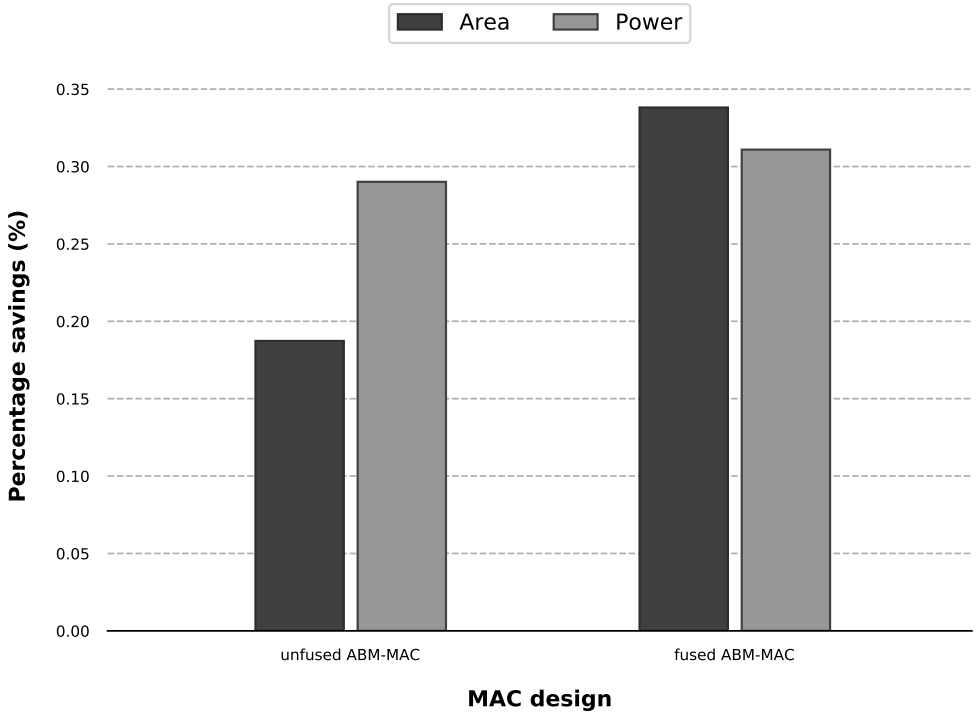


Fig. 5.4. Comparison of hardware savings delivered by fused vs. non-fused ABM-MAC.

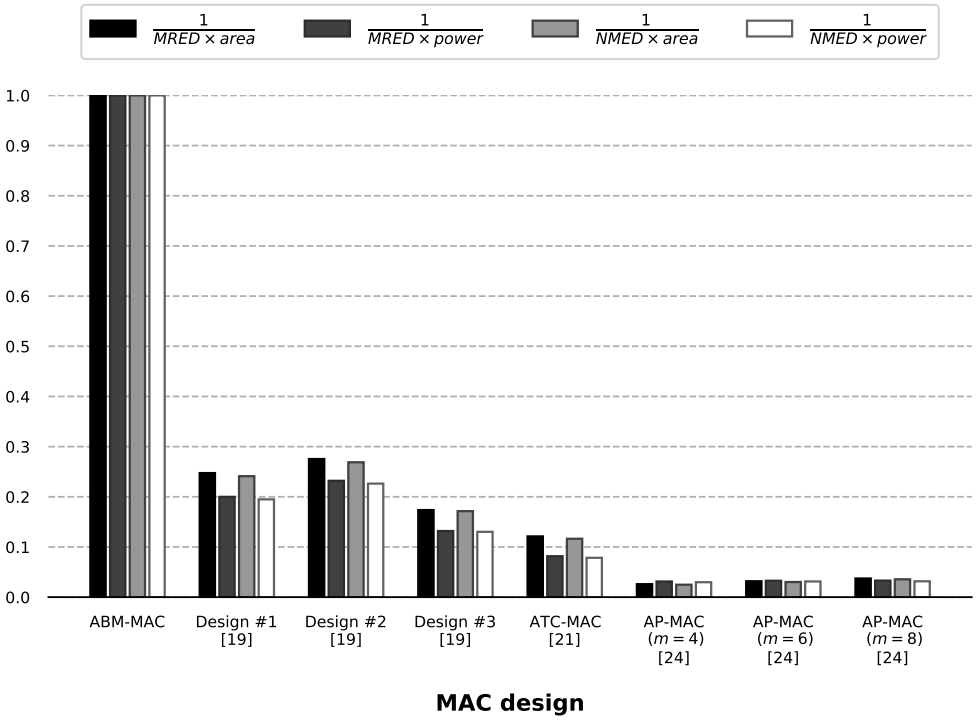


Fig. 5.5. Inverse multiplication of error metrics (MRED and NMED) by hardware parameters (area and power) to evaluate tradeoff.

be formed unless we can understand the relationship maintained between accuracy and hardware performance for the given model: a quality design will achieve a balance between reducing hardware complexity and maintaining an acceptable level of accuracy. This relationship is illustrated for the proposed model and benchmarking designs in Fig. 5.5.

The data plotted is the inverse of MRED/NMED multiplied by area/power, corresponding to four sets of metrics for which a larger value indicates a greater ability to both maintain low computational error and provide strong hardware savings. For each set, all data points are normalized by the maximum value.

Fig. 5.5 highlights a strength of the proposed design. While ABM-MAC achieved neither the smallest area metric nor the smallest power metric among the implemented models, the plots shown indicate the ability of the proposed design to produce significant power savings while maintaining high levels of accuracy. Comparatively, Designs #1–3 [80] as well as ATC-MAC [81] provide larger hardware savings but at the cost of degraded accuracy. Among the AP-MAC [87] designs, the $k = 8$ model achieves low area and moderate power savings; however, the AP-MAC [87] designs involve a high level of approximation and are therefore limited by low accuracy. The ability of ABM-MAC to provide competitive hardware savings while maintaining high accuracy indicates that it provides novel contribution in the context of state-of-the-art inexact MAC units.

5.5 Gaussian Blur Application

The MAC unit is widely utilized in both general matrix multiplication and convolution. The implemented MAC units are evaluated for performance in a Gaussian blurring application in which an 8-bit grayscale image is blurred by convolving it with a discrete Gaussian filter. The application consists of two testbenches: one in which the blur is applied to 256×256 images using a 5×5 kernel, and the other for which a 7×7 kernel operates on images of size 512×512. Convolution using discrete Gaussian kernels results in a large accumulated number which must be rescaled down to fit back into 8-bits at the end of computation. This is performed by simply dividing the outputs by a constant dependent on the particular kernel being used. A Python script was used to read the image file and produce an equivalent data file for quick reading. Simulations were performed using ModelSim, and Python scripting was used to transform the raw output data to an image and subsequently compute performance metrics. The performance of each design is indicated by the level of degradation in the output image when compared to the accurate output, and thus PSNR and SSIM are used to for evaluation. The performance metrics measured for all designs are summarized in Table 5.2.

The 5×5 filter application utilizes the kernel shown in Fig. 5.6, where $1/273$ is the scaling factor used to scale the large accumulated output of the convolution back down to an 8-bit number. Convolutions are performed for several input images. The input images, the accurate outputs, and the outputs produced by ABM-MAC are shown in Fig. 5.7. From Fig. 5.7, it can be seen that there is no visible difference between the images produced by ABM-MAC and those computed accurately. The 7×7 kernel testbench utilizes the kernel shown in Fig. 5.8 to perform the convolution. The input images, the

$$\frac{1}{273} \times \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Fig. 5.6. Discrete 5×5 Gaussian filter.

Table 5.2. PSNR and SSIM measurements for Gaussian blur application

(a) 5×5 filter

Design	PSNR (dB)				SSIM			
	Clk	Chm	Aer	Cam	Clk	Chm	Aer	Cam
ABM-MAC	41.9	40.5	40.5	41.1	0.985	0.981	0.990	0.980
Design #1 [80]	27.6	37.4	32.6	38.3	0.979	0.985	0.977	0.986
Design #2 [80]	27.4	36.4	32.2	37.2	0.978	0.985	0.977	0.985
Design #3 [80]	27.1	27.5	27.4	27.5	0.943	0.928	0.950	0.844
ATC-MAC [81]	39.6	38.5	38.2	39.4	0.989	0.987	0.992	0.987
AP-MAC (k = 4) [87]	28.5	31.4	29.1	30.7	0.982	0.977	0.981	0.981
AP-MAC (k = 6) [87]	28.5	31.4	29.1	30.6	0.982	0.977	0.981	0.980
AP-MAC (k = 8) [87]	28.0	29.7	28.1	29.3	0.979	0.972	0.978	0.975

(b) 7×7 filter

Design	PSNR (dB)				SSIM			
	Boa	Brb	Air	Glh	Boa	Brb	Air	Glh
ABM-MAC	43.1	44.1	43.4	44.9	0.992	0.991	0.992	0.991
Design #1 [80]	37.6	34.7	27.9	34.7	0.988	0.985	0.983	0.985
Design #2 [80]	37.0	34.3	27.8	34.3	0.988	0.985	0.983	0.985
Design #3 [80]	27.3	27.3	27.3	27.1	0.973	0.968	0.973	0.968
ATC-MAC [81]	38.3	36.8	35.2	36.1	0.987	0.983	0.984	0.982
AP-MAC (k = 4) [87]	30.3	29.4	27.7	29.4	0.967	0.965	0.965	0.961
AP-MAC (k = 6) [87]	30.3	29.3	27.7	29.4	0.967	0.965	0.965	0.961
AP-MAC (k = 8) [87]	29.6	28.8	27.7	28.9	0.966	0.962	0.963	0.959

accurate results, and the approximate results are shown in Fig. 5.9.

Table 5.2a summarizes the PSNR and SSIM results computed for all MAC units for each of the four test images in the 5×5 blur application, where the image names are abbreviated as: Clock → Clk, Chemical plant → Chm, Aerial → Aer, and Cameraman → Cam.

Fig. 5.7. Images used in 5×5 Gaussian blur application from top to bottom: clock, chemical plant, aerial, and cameraman. The first column contains original images used as input, the second column is the result from convolution computed using exact operations, and the third column contains the outputs of the convolution computed by ABM-MAC.

$$\frac{1}{1003} \times \begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ \hline 0 & 3 & 13 & 22 & 13 & 3 & 0 \\ \hline 1 & 13 & 59 & 97 & 59 & 13 & 1 \\ \hline 2 & 22 & 97 & 159 & 97 & 22 & 2 \\ \hline 1 & 13 & 59 & 97 & 59 & 13 & 1 \\ \hline 0 & 3 & 13 & 22 & 13 & 3 & 0 \\ \hline 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ \hline \end{array}$$

Fig. 5.8. Discrete 7×7 Gaussian filter.

Firstly, it is worth noting that not all designs performed consistently across all four test images. More specifically, Design #1 and #2 from [80] compute PSNRs from over a 10 dB range. While Design #1 and #2 have moderate performance, Design #3 does not perform adequately in this application. On the other hand, ATC-MAC [81] performs very well for all input images in terms of both PSNR and SSIM. AP-MAC [87] exhibits poor accuracy metrics when compared to the other designs, so it is not surprising that it produces relatively low PSNR and SSIM values. The proposed ABM-MAC consistently achieves the best PSNR values amongst the implemented designs. While ATC-MAC [81] does achieve the highest SSIM, the SSIM values for ABM-MAC differ from those of ATC-MAC by less than one percent.

Table 5.2b summarizes the PSNR and SSIM results computed for all MAC units for each of the four test images in the 7×7 blur application, where the image names are abbreviated as: Boat $\rightarrow$ Boa, Barbara $\rightarrow$ Brb, Airplane $\rightarrow$ Air, and Goldhill $\rightarrow$ Glh. There are substantial differences between the metrics derived from the 5×5 filter application and those derived using the 7×7 filter. Firstly, the PSNR values in the 7×7 application improved for all models. In the case of Design #1–3 [80], PSNR values improved by a value of 10 dB. The SSIM metrics remained very similar between both testbenches. The performance for all MAC units relative to one another is quite similar to that of the previous testbench. However, one difference is that ABM-MAC now reports the best SSIM values for all four images.

Fig. 5.9. Images used in 7×7 Gaussian blur application from top to bottom: boat, barbara, airplane, and goldhill. The first column contains original images used as input, the second column is the result from convolution computed using exact operations, and the third column contains the outputs of the convolution computed by ABM-MAC.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

While Dennard scaling enabled impressive performance improvements for decades, its decay has highlighted the need to explore alternate methods of improving performance beyond the scope of transistor miniaturization. The emergence of data-heavy error resilient applications has enabled the development of approximate computing as a paradigm for improving performance, where approximation techniques are largely independent of semiconductor technology. While approximation can be applied at any layer of the computing hierarchy, the development of approximate arithmetic hardware has been proven to be an especially effective method of reducing unnecessary hardware cost.

In Chapter 3, three approximate Booth multipliers were proposed alongside two inexact PPGs. The proposed multipliers were compared to other state-of-the-art designs in terms of both accuracy and hardware performance. Compared to the exact design, power is reduced by up to 15% for ABM-M1, 46% for ABM-M2, and 46% for ABM-M3. When compared to the benchmarking models, ABM-M1 and ABM-M3 achieve the lowest error metrics, and the highest area and power savings are achieved by ABM-M3. The multipliers are utilized in three applications, i.e. image transformation, matrix multiplication, and a FIR filter implementation. For all three applications, the best performance was typically achieved by ABM-M1 and ABM-M3.

In Chapter 4, two approximate restoring dividers are proposed. The first divider utilizes a proposed inexact divider cell in its least-significant columns to reduce hardware cost. The second model performs only a subset of the trial subtractions, after which the partial remainder and the divisor are rounded and encoded for use in generating the remaining quotient bits. The dividers were evaluated according to their accuracy and hardware performance alongside several benchmarking divider designs. When compared to the exact divider, power is reduced by 46% for AXRD-M1 and 57% for AXRD-M2. When compared with the benchmarking designs, the proposed dividers achieve a strong balance between accuracy and hardware savings.

In Chapter 5, ABM-MAC is proposed in which the multiplication architecture is based

on ABM-M3. The proposed design was compared with several other inexact MAC units to evaluate its accuracy and hardware performance. Compared to the exact design, area is reduced by 42% and power is reduced by 33%. Compared to the benchmarking designs, while the proposed MAC unit exhibits relatively competitive hardware metrics, its primary advantage is its high accuracy. Finally, the function of ABM-MAC is verified by implementing convolution operations for both 5×5 and 7×7 kernels.

6.2 Future Work

The development of low-level approximate arithmetic units has received vast attention in the literature, likely in part due to the fact that such units are generic and can therefore be utilized in a variety of error tolerant applications. While the flexibility associated with basic arithmetic blocks may be convenient, additional hardware savings are sacrificed to provide this generalization. To be more specific, approximation techniques can generally be classified in one of two ways. Domain-specific techniques are intended for use in a certain data domain and therefore hold prior expectations on what sort of data will be received. On the other hand, general-purpose techniques have no prior knowledge or conceptions of the data and therefore can be used with any dataset. Domain-specific techniques have an advantage in that approximation may be applied according to the expected characteristics of the data. While general-purpose techniques are still useful, domain-specific approximation can achieve impressive performance improvements, and thus there is a growing interest in the development of application-specific approximate hardware.

A number of works featuring domain-specific techniques were discussed in Section 1.2, where these works mostly featured acceleration techniques for use in neural network applications [20]–[25]. While the aforementioned works are historically the most prevalent examples of approximate accelerators in the literature, recent literature features a number of neural network architectures which utilize a variety of approximation techniques to achieve impressive hardware reduction. Quantization is a popular technique for reducing precision in which data is mapped to a smaller set of quantization levels. Quantization can be applied to just the network weights, or to both the weights and activations. Additionally, quantization can be uniform, meaning that quantization levels are evenly spaced, or it may be nonuniform with varying space between values. Neural networks typically require less precision during inference than is needed during training, so quantization is often performed once the network training is complete. A very simple quantization scheme involves mapping floating-point data to dynamic fixed-point data of a smaller width [88], [89], where 8 bits seems to be a good choice for maintaining reasonable precision. Some works have employed very aggressive quantization schemes resulting in ternary [90], [91] or even binary [89], [90], [92]–[96] values. Other works feature the use of floating-point

data during training, which is then quantized to integer values for use in inference [97]. A popular nonuniform quantization technique is pruning, in which it is assumed a large number of weights in the network are redundant and can therefore be set to zero [98], [99].

While quantization is generally quite effective in reducing the data load, there are a number of more sophisticated approximation techniques that may be utilized in neural accelerators. In [100], an approximate training technique is proposed in which two methods are executed simultaneously. While one method actively searches for a network parameter distribution with high error tolerance, the other passively learns resilient weights by numerically incorporating the noise distribution associated with the inexact hardware during the forward pass. This framework is then incorporated into an accuracy-scalable accelerator with the aim of achieving high energy efficiency. The design presented in [101] utilizes a combination of several approximation techniques including quantization, the use of an approximate multiplier architecture, and the solving of optimization algorithms to determine the decisions to make that will correspond with the smallest amount of error.

While there are numerous applications that are well-suited to approximation, it seems that neural networks are notably receptive to the use of approximation techniques. Even if state-of-the-art hardware is available, the performance of neural network applications can be continuously improved by identifying and removing unnecessary computation through the use of approximate computing techniques.

References

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, 1965.

[2] G. E. Moore, "Progress in digital integrated electronics," in *Tech. Dig. 1975: Int. Electron Devices Meeting*, 1975, pp. 11–13.

[3] R. H. Dennard, F. H. Gaensslen, H. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE J. Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[4] M. Bohr, "A 30 year retrospective on Dennard's MOSFET scaling paper," *IEEE Solid-State Circuits Soc. Newslett.*, vol. 12, no. 1, pp. 11–13, 2007.

[5] C. Moore, "Data processing in exascale-class computer systems," Presented at Salishan Conf. High Speed Comput. 2011. [Online]. Available: `https://www.lanl.gov/conferences/salishan/salishan2011/3moore.pdf` (visited on Dec. 20, 2020).

[6] T. N. Theis and H.-S. P. Wong, "The end of Moore's law: A new beginning for information technology," *Computing in Science & Engineering*, vol. 19, no. 2, pp. 41–50, 2017.

[7] M. Dubash. (Apr. 2005). "Moore's law is dead, says Gordon Moore," [Online]. Available: `http://news.techworld.com/operating-systems/3477/moores-law-is-dead-says-gordon-moore` (visited on Oct. 8, 2020).

[8] I. Cutress. (Jul. 2016). "Looking to the future: International technology roadmap for semiconductors 2.0," [Online]. Available: `https://www.anandtech.com/show/10525/ten-year-anniversary-of-core-2-duo-and-conroe-moores-law-is-dead-long-live-moores-law/7` (visited on Oct. 8, 2020).

[9] *IEEE Rebooting Computing Initiative, Standards Association, and Computer Society introduce new International Roadmap for Devices and Systems to set the course for end-to-end computing*, Press Release, IEEE, May 2016. [Online]. Available: `https://rebootingcomputing.ieee.org/images/files/pdf/rc_irds.pdf` (visited on Oct. 8, 2020).

[10] D. J. Pagliari, M. Poncino, and E. Macii, "Energy-efficient digital processing via approximate computing," in *Smart Systems Integration and Simulation*, N. Bombieri, M. Poncino, and G. Pravadelli, Eds. Springer, 2016, ch. 4, pp. 55–89.

[11] I. Gorton, "Software architecture challenges for data intensive computing," in *Proc. 7th Work. IEEE/IFIP Conf. Softw. Architecture (WICSA 2008)*, Vancouver, Canada, 2008, pp. 4–6.

[12] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. Corner, and E. Berger, "Eon: A language and runtime system for perpetual systems," in *Proc. 5th Int. Conf. Embedded Networked Sensor Syst. (SenSys)*, 2007, pp. 161–174.

[13] A. Sampson, W. Dietl, E. Fortuna, and D. Gnanapragasam, "EnerJ: Approximate data types for safe and general low-power computation," in *Proc. 32nd ACM SIGPLAN Conf. Program. Lang. Des. and Implementation (PLDI)*, 2011, pp. 164–174.

[14] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware," in *Proc. 2013 ACM SIGPLAN Int. Conf. Object Oriented Program., Syst., Lang., and Appl. (OOPSLA)*, 2013, pp. 33–52.

[15] S. Sidiroglou, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proc. 19th ACM SIGSOFT Symp. and 13th Eur. Conf. Found. Softw. Eng. (ESEC/FSE)*, 2011, pp. 124–134.

[16] W. Baek and T. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," in *Proc. 2010 ACM SIGPLAN Conf. Program. Lang. Des. and Implementation (PLDI)*, vol. 45, 2010, pp. 198–209.

[17] K. Lee, R. Bhattacharya, J. Dass, V. N. S. Prithvi Sakuru, and R. N. Mahapatra, "A relaxed synchronization approach for solving parallel quadratic programming problems with guaranteed convergence," in *Proc. 2016 IEEE Int. Parallel and Distrib. Process. Symp. (IPDPS)*, 2016, pp. 182–191.

[18] C. Alvarez, J. Corbal, and M. Valero, "Fuzzy memoization for floating-point multimedia applications," *IEEE Trans. Comput.*, vol. 54, no. 7, pp. 922–927, 2005.

[19] J. S. Miguel, M. Badr, and N. E. Jerger, "Load value approximation," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2014, pp. 127–139.

[20] R. S. Amant, A. Yazdanbakhsh, J. Park, *et al.*, "General-purpose code acceleration with limited-precision analog computation," in *Proc. 2014 ACM/IEEE 41st Int. Symp. Comput. Architecture (ISCA)*, 2014, pp. 505–516.

[21] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "ApproxANN: An approximate computing framework for artificial neural network," in *Proc. 2015 Des., Automat. & Test in Europe (DATE)*, 2015, pp. 701–706.

[22] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proc. 2012 45th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2012, pp. 449–460.

[23] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "AxNN: Energy-efficient neuromorphic systems using approximate computing," in *Proc. 2014 IEEE/ACM Int. Symp. Low Power Electron. and Des. (ISLPED)*, 2014, pp. 27–32.

[24] B. Grigorian and G. Reinman, "Accelerating divergent applications on SIMD architectures using neural networks," in *Proc. 2014 IEEE 32nd Int. Conf. Comput. Des. (ICCD)*, 2014, pp. 317–323.

[25] T. Moreau, M. Wyse, J. Nelson, *et al.*, "SNNAP: Approximate computing on programmable SoCs via neural acceleration," in *Proc. 2015 IEEE 21st Int. Symp. on High Perform. Comput. Architecture (HPCA)*, 2015, pp. 603–614.

[26] H. Cho, L. Leem, and S. Mitra, "ERSA: Error resilient system architecture for probabilistic applications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 4, pp. 546–558, 2012.

[27] Y. Yetim, M. Martonosi, and S. Malik, "Extracting useful computation from error-prone processors for streaming applications," in *Proc. 2013 Des., Automat. & Test in Europe (DATE)*, 2013, pp. 202–207.

[28] V. K. Chippa, H. Jayakumar, D. Mohapatra, K. Roy, and A. Raghunathan, "Energy-efficient recognition and mining processor using scalable effort design," in *Proc. IEEE 2013 Custom Integr. Circuits Conf. (CICC)*, 2013, pp. 1–4.

[29] V. K. Chippa, S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing: An integrated hardware approach," in *Proc. 2013 Asilomar Conf. Signals, Syst. and Comput. (ACSSC)*, 2013, pp. 111–117.

[30] V. K. Chippa, D. Mohapatra, K. Roy, S. T. Chakradhar, and A. Raghunathan, "Scalable effort hardware design," *IEEE Trans. VLSI Syst.*, vol. 22, no. 9, pp. 2004–2016, 2014.

[31] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proc. 2013 46th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2013, pp. 1–12.

[32] P. Düben, Parishkrati, S. Yenugula, *et al.*, "Opportunities for energy efficient computing: A study of inexact general purpose processors for high-performance and big-data applications," in *Proc. 2015 Des., Automat. & Test in Europe (DATE)*, 2015, pp. 764–769.

[33] S. Liu, K. Pattabiraman, T. Moscibroda, and B. Zorn, "Flicker: Saving refresh-power in mobile devices through critical data partitioning," Microsoft Research, Redmond, WA, Tech. Rep. MSR-TR-2009-138, Oct. 2009.

[34] K. Cho, Y. Lee, Y. H. Oh, G. Hwang, and J. W. Lee, "eDRAM-based tiered-reliability memory with applications to low-power frame buffers," in *Proc. 2014 IEEE/ACM Int. Symp. Low Power Electron. and Des. (ISLPED)*, 2014, pp. 333–338.

[35] R. Hegde and N. R. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *Proc. 1999 Int. Symp. Low Power Electron. and Des. (ISLPED)*, 1999, pp. 30–35.

[36] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *Proc. 2011 Des., Automat. & Test in Europe (DATE)*, 2011, pp. 1–6.

[37] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *Proc. 17th Int. Conf. Architectural Support for Program. Lang. and Operating Syst. (ASPLOS)*, 2012, pp. 301–312.

[43] 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.

[44] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate XOR/XNOR-based adders for inexact computing," in *Proc. 2013 13th IEEE Int. Conf. Nanotechnol. (IEEE-NANO 2013)*, 2013, pp. 690–693.

[45] S. Lu, "Speeding up processing with approximation circuits," *Computer*, vol. 37, no. 3, pp. 67–73, 2004.

[46] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *Proc. 2008 Des., Automat. & Test in Europe (DATE)*, 2008, pp. 1250–1255.

[47] 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*, vol. 57, no. 4, pp. 850–862, 2010.

[48] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE Trans. VLSI Syst.*, vol. 18, no. 8, pp. 1225–1229, 2010.

[49] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application (isic)," in *Proc. 2009 12th Int. Symp. Integr. Circuits*, 2009, pp. 69–72.

[50] N. Zhu, W. L. Goh, and K. S. Yeo, "Ultra low-power high-speed flexible probabilistic adder for error-tolerant applications," in *Proc. 2011 International SoC Design Conference (ISOCC)*, 2011, pp. 393–396.

[51] N. Zhu, W. L. Goh, G. Wang, and K. S. Yeo, "Enhanced low-power high-speed adder for error-tolerant application," in *Proc. 2010 International SoC Design Conference (ISOCC)*, 2010, pp. 323–327.

[52] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in *Proc. 2012 Des., Automat. & Test in Europe (DATE)*, 2012, pp. 1257–1262.

[53] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proc. 49th Annu. Des. Automat. Conf. (DAC)*, 2012, pp. 820–825.

[54] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proc. 2013 IEEE/ACM Int. Conf. Computer-Aided Des. (ICCAD)*, 2013, pp. 48–54.

[55] K. Kyaw, W. Goh, and K. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *Proc. 2010 IEEE Int. Conf. Electron Devices and Solid-State Circuits (EDSSC)*, 2010, pp. 1–4.

[56] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. 2011 24th Int. Conf. VLSI Des. (VLSID)*, 2011, pp. 346–351.

[57] C. Lin and I. Lin, "High accuracy approximate multiplier with error correction," in *Proc. 2013 IEEE 31st Int. Conf. Comput. Des. (ICCD)*, 2013, pp. 33–38.

[58] K. Bhardwaj, P. S. Mane, and J. Henkel, "Power- and area-efficient approximate Wallace tree multiplier for error-resilient systems," in *Proc. 15th Int. Symp. Quality Electro. Des. (ISQED)*, 2014, pp. 263–269.

[59] 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 Trans. VLSI Syst.*, vol. 23, no. 6, pp. 1180–1184, 2015.

[60] S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: A dynamic range unbiased multiplier for approximate applications," in *Proc. 2015 IEEE/ACM Int. Conf. Computer-Aided Des. (ICCAD)*, 2015, pp. 418–425.

[61] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," *IEEE Trans. VLSI Syst.*, vol. 25, no. 2, pp. 393–401, 2017.

[62] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Trans. Electron. Comput.*, vol. EC-11, no. 4, pp. 512–517, 1962.

[63] M. B. Sullivan and E. E. Swartzlander, "Truncated logarithmic approximation," in *Proc. 2013 IEEE 21st Symp. Comput. Arithmetic (ARITH)*, 2013, pp. 191–198.

[64] M. S. Ansari, B. F. Cockburn, and J. Han, "A hardware-efficient logarithmic multiplier with improved accuracy," in *Proc. 2019 Des., Automat. & Test in Europe (DATE)*, 2019, pp. 928–931.

[65] S. E. Ahmed, S. Kadam, and M. B. Srinivas, "An iterative logarithmic multiplier with improved precision," in *Proc. 2016 IEEE 23nd Symp. Comput. Arithmetic (ARITH)*, 2016, pp. 104–111.

[66] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications," *IEEE Trans. Circuits Syst. I*, vol. 65, no. 9, pp. 2856–2868, 2018. DOI: 10.1109/TCSI.2018.2792902.

[67] W. Yeh and C. Jen, "High-speed Booth encoded parallel multiplier design," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 692–701, 2000.

[68] L. Qian, C. Wang, W. Liu, F. Lombardi, and J. Han, "Design and evaluation of an approximate Wallace-Booth multiplier," in *Proc. 2016 IEEE Int. Symp. Circuits and Syst. (ISCAS)*, 2016, pp. 1974–1977.

[69] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 984–994, 2015. DOI: 10.1109/TC.2014.2308214.

[70] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, "Design of approximate radix-4 Booth multipliers for error-tolerant computing," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1435–1441, 2017. DOI: 10.1109/TC.2017.2672976.

[71] H. Jiang, J. Han, F. Qiao, and F. Lombardi, "Approximate radix-8 Booth multipliers for low-power and high-performance operation," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2638–2644, 2016. DOI: 10.1109/TC.2015.2493547.

[72] V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi, "Approximate hybrid high radix encoding for energy-efficient inexact multipliers," *IEEE Trans. VLSI Syst.*, vol. 26, no. 3, pp. 421–430, 2018. DOI: 10.1109/TVLSI.2017.2767858.

[73] Z. Yang, J. Han, and F. Lombardi, "Approximate compressors for error-resilient multiplier design," in *Proc. 2015 IEEE Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnol. Syst. (DFTS)*, 2015, pp. 183–186.

[74] M. Ha and S. Lee, "Multipliers with approximate 4–2 compressors and error recovery modules," *IEEE Embedded Syst. Lett.*, vol. 10, no. 1, pp. 6–9, 2018.

[75] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro, and N. Petra, "Approximate multipliers based on new approximate compressors," *IEEE Trans. Circuits Syst. I*, vol. 65, no. 12, pp. 4169–4182, 2018.

[76] L. Chen, J. Han, W. Liu, and F. Lombardi, "Design of approximate unsigned integer non-restoring divider for inexact computing," in *Proc. 25th Ed. Great Lakes Symp. VLSI (GLVLSI)*, 2015, pp. 51–56.

[77] L. Chen, J. Han, W. Liu, and F. Lombardi, "On the design of approximate restoring dividers for error-tolerant applications," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2522–2533, 2016. DOI: 10.1109/TC.2015.2494005.

[78] S. Hashemi, R. I. Bahar, and S. Reda, "A low-power dynamic divider for approximate applications," in *Proc. 2016 53nd ACM/EDAC/IEEE Des. Automat. Conf. (DAC)*, 2016, pp. 1–6.

[79] H. Jiang, L. Liu, F. Lombardi, and J. Han, "Adaptive approximation in arithmetic circuits: A low-power unsigned divider design," in *Proc. 2018 Des., Automat. & Test in Europe (DATE)*, 2018, pp. 1411–1416. DOI: 10.23919/DATE.2018.8342233.

[80] D. Esposito, A. G. M. Strollo, and M. Alioto, "Low-power approximate MAC unit," in *Proc. 2017 13th Conf. Ph.D. Res. in Microelectron. and Electron. (PRIME)*, 2017, pp. 81–84.

[81] T. Yang, T. Sato, and T. Ukezono, "A low-power approximate multiply-add unit," in *Proc. 2nd Int. Symp. Devices, Circuits and Syst. (ISDCS)*, 2019, pp. 1–4.

[82] M. Karnaugh, "The map method for synthesis of combinational logic circuits," *Trans. Amer. Inst. Elect. Eng. Part I: Commun. Electron.*, vol. 72, no. 5, pp. 593–599, 1953.

[83] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1760–1771, 2013.

[84] B. Shao and P. Li, "A model for array-based approximate arithmetic computing with application to multiplier and squarer design," in *Proc. 2014 IEEE/ACM Int. Symp. Low Power Electron. and Des. (ISLPED)*, 2014, pp. 9–14.

[85] W. Liu, T. Cao, P. Yin, *et al.*, "Design and analysis of approximate redundant binary multipliers," *IEEE Trans. Comput.*, vol. 68, no. 6, pp. 804–819, 2019.

[86] M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han, "Low-power approximate multipliers using encoded partial products and approximate compressors," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 404–416, 2018.

[87] Y. Lu, W. Shan, and J. Xu, "A depthwise separable convolution neural network for small-footprint keyword spotting using approximate MAC unit and streaming convolution reuse," in *Proc. 2019 IEEE Asia Pacific Conf. Circuits and Syst. (APCCAS)*, 2019, pp. 309–312.

[88] Y. Ma, N. Suda, Y. Cao, J. Seo, and S. Vrudhula, "Scalable and modularized RTL compilation of convolutional neural networks onto FPGA," in *Proc. 2016 26th Int. Conf. Field Programmable Logic and Appl (FPL)*, 2016, pp. 1–8.

[89] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. (2016). "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients." arXiv: `1606.06160 [cs.NE]`.

[90] F. Li, B. Zhang, and B. Liu. (2016). "Ternary weight networks." arXiv: `1605.04711 [cs.CV]`.

[91] C. Zhu, S. Han, H. Mao, and W. J. Dally. (2017). "Trained ternary quantization." arXiv: `1612.01064 [cs.LG]`.

[92] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. (2016). "XNOR-Net: ImageNet classification using binary convolutional neural networks." arXiv: `1603.05279 [cs.CV]`.

[93] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Benigo. (2016). "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1." arXiv: `1602.02830 [cs.LG]`.

[94] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Benigo. (2016). "Quantized neural networks: Training neural networks with low precision weights and activations." arXiv: `1609.07061 [cs.NE]`.

[95] Y. Wang, J. Lin, and Z. Wang, "An energy-efficient architecture for binary weight convolutional neural networks," *IEEE Trans. VLSI Syst.*, vol. 26, no. 2, pp. 280–293, 2018.

[96] L. Yang, D. Bankman, B. Moons, M. Verhelst, and B. Murmann, "Bit error tolerance of a CIFAR-10 binarized convolutional neural network processor," in *Proc. 2018 IEEE Int. Symp. Circuits and Syst. (ISCAS)*, 2018, pp. 1–5.

[97] B. Jacob, S. Kligys, B. Chen, *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. 2018 IEEE/CVF Conf. Comput. Vision and Pattern Recognit. (CVPR)*, 2018, pp. 2704–2713.

[98] T. Yang, Y. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. 2017 IEEE Conf. Comput. Vision and Pattern Recognit. (CVPR)*, 2017, pp. 6071–6079.

[99] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. 2017 IEEE Int. Conf. Comput. Vision (ICCV)*, 2017, pp. 1398–1406.

[100] X. He, W. Lu, G. Yan, and X. Zhang, "Joint design of training and hardware towards efficient and accuracy-scalable neural network inference," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 4, pp. 810–821, 2018.

[101] V. Mrazek, Z. Vasicek, L. Sekanina, M. A. Hanif, and M. Shafique, "ALWANN: Automatic layer-wise approximation of deep neural network accelerators without retraining," in *Proc. 2019 IEEE/ACM Int. Conf. Computer-Aided Des. (ICCAD)*, 2019, pp. 1–8.