Approximate Computing Techniques For Accuracy-Energy Trade-offs

by
Soheil Hashemi

M.Sc., Brown University, Providence, RI, 2015
B.Sc., Sharif University of Technology, Tehran, Iran, 2013

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in School of Engineering at Brown University

PROVIDENCE, RHODE ISLAND

May 2018

This dissertation by Soheil Hashemi is accepted in its present form
by School of Engineering as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Recommended to the Graduate Council

Date_____ _____

Sherief Reda, Advisor

Date_____ _____

Harvey Silverman, Reader

Date_____ _____

Gabriel Taubin, Reader

Date_____ _____

Rodrigo Fonseca, Reader

Approved by the Graduate Council

Date_____ _____

Andrew G. Campbell, Dean of the Graduate School

Vitae

Soheil Hashemi was born in Tehran, Iran in 1990. He received his B.Sc. in Electrical Engineering from Sharif University of Technology in Spring of 2013. He received his M.Sc. in Electrical Science and Computer Engineering from Brown University continuing his studies toward a Ph.D. degree. His principal research areas include energy-efficient hardware design, approximate computing, hardware realization of machine learning applications, and deep learning.

Soheil_Hashemi@brown.edu
https://sites.google.com/view/soheil-hashemi
Brown University, RI, USA

Selected Publications:

1. S. Hashemi, H. Tann, and S. Reda, "BLASYS: Approximate Logic Synthesis Using Boolean Matrix Factorization", to appear in ACM/IEEE Design Automation Conference, San Francisco, CA, 2018.

2. S. Hashemi, H. Tann, F. Buttafuoco and S. Reda, "Approximate computing for biometrie security systems: A case study on iris scanning," Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 2018, pp. 319-324.

3. H. Tann, S. Hashemi, R. I. Bahar and S. Reda, "Hardware-software codesign of

accurate, multiplier-free Deep Neural Networks," 54th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, 2017, pp. 1-6.

4. S. Hashemi, N. Anthony, H. Tann, R. I. Bahar and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, Lausanne, 2017, pp. 1474-1479.

5. K. Nepal, S. Hashemi, H. Tann, R. I. Bahar and S. Reda, "Automated High-Level Generation of Low-Power Approximate Computing Circuits", in IEEE Transactions on Emerging Topics in Computing, 2016.

6. H. Tann, S. Hashemi, R. I. Bahar and S. Reda,"Runtime configurable deep neural networks for energy-accuracy trade-off," 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Pittsburgh, PA, 2016, pp. 1-10.

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

8. S. Hashemi, R. I. Bahar and S. Reda, "DRUM: A Dynamic Range Unbiased Multiplier for approximate applications," 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, 2015, pp. 418-425.

    - *ACM/IEEE William J. McCalla ICCAD Best Paper Award Candidate.*

Book Chapters:

1. S. Hashemi, H. Tann, and S. Reda, "Approximate Logic Synthesis Using Boolean

Matrix Factorization", to appear in Approximate Circuits: Circuits and Methodologies, Springer.

2. S. Hashemi, H. Tann, and S. Reda, "Approximate Multipliers and Dividers Using Dynamic Bit Selection", to appear in Approximate Circuits: Circuits and Methodologies, Springer.

3. H. Tann, S. Hashemi, and S. Reda, "Approximate Computing for Iris Recognition Systems", to appear in Approximate Circuits: Circuits and Methodologies, Springer.

4. H. Tann, S. Hashemi, and S. Reda, "Approximate Deep Neural Network Accelerators", to appear in Approximate Circuits: Circuits and Methodologies, Springer.

Acknowledgements

This thesis would not have been possible without the constant support, guidance and inspirations of many grateful individuals. First and foremost, I want to express my immense gratitude to my advisor and mentor Prof. Sherief Reda, whose guidance, support, and valuable insights during the course of my studies has made this thesis possible.

I would also like to thank my dissertation committee members, Prof. Harvey Silverman, Prof. Gabriel Taubin, and Prof. Rodrigo Fonseca, for agreeing to serve on the committee, for their time, and their valuable feedbacks on this thesis. Their comments and questions are invaluable to this thesis.

I am grateful to my co-authors, Prof. Iris Bahar for her help and guidance through different projects, my friends and lab mates, Hokchhay Tann, Nicholas Anthony, Dr. Kumud Nepal, and my advisor Prof. Sherief Reda at Brown University, and Francesco Buttafuoco from Polytechnic University of Turin. I owe much of my output over the past five years to their hard effort.

I would like to thank my friends and colleagues at SCALE lab, at Brown, and in Providence. Thank you for all the discussions, foosball games, fun gathering, and for making these past five years more memorable. I am also grateful for all my friends from Boston, to New Jersey and Maryland.

I also want to express my gratitude to my friend, lab mate, and roommate Reza Azimi, for being a positive force in my life for the past five years.

I would also like to thank Elham Saraee for years of support, lively discussions, and unforgettable adventures.

Last but not least, I would like to thank my parents and my brother, Soroush, for their love, unwavering support, and their invaluable advice throughout my life. None of what I have achieved would have been possible without what I have learned from them.

The research in this thesis is partially supported by NSF grant 1420864.

Abstract of "Approximate Computing Techniques For Accuracy-Energy Trade-offs" by Soheil Hashemi, Ph.D., Brown University, May 2018

Power efficiency has emerged as one of the main concerns in many digital design domains ranging from embedded and battery operated systems to data centers. As a result, in recent years, many different approaches have been aimed at lowering the power and resource footprints of computing systems. Approximate computing is one such emerging technique targeting error resilient applications and offering promising benefits. Approximate computing introduces design accuracy as a third orthogonal dimension to the conventional power/performance trade-offs. The underlying principal of such approach is that with relaxing the full accuracy requirement on the output, one can benefit from significant savings in hardware design metrics, such as power consumption, design area, and critical path delay. This paradigm, however, is only applicable to applications where an inherent tolerance to small and insignificant errors exists. Applications in domains of media processing, machine learning, and data mining are a few examples. Interestingly, with the recent growth in the number of data intensive and machine learning applications, the relevance of approximate computing has only increased.

This thesis makes the following contributions toward the advance of approximate computing techniques in different computing systems. First, we propose and evaluate a novel methodology for design of approximate arithmetic blocks, namely multipliers and dividers. Our methodology benefits from a dynamic truncation scheme, where the most important bits are guaranteed to be selected, as well as an unbiasing scheme, where the error distribution in balanced around zero to ensure both positive and negative errors. We show that our methodology can achieve up to 71.45% in power savings for an average error of 1.47%, and up to 70.81% in power savings for and average error of 3.08%, for the approximate multiplier and the approximate divider, respectively. Second, using Boolean matrix factorization (BMF), we devise an approximate synthesis methodology, where any

circuit can be reduced to an approximate variant in an automated fashion. We devise a BMF algorithm to give more weight to approximations on higher bit indices compared to their least significant counterparts. We evaluate our proposed methodology on six different circuits and show benefits of up to 47.55% in power reduction for a tight error bound of 5%. Next, and to showcase the applicability of approximate computing techniques to complex computing pipelines, we explore such techniques for case studies from two different application domains. Here, we first explore the energy-accuracy trade-offs while using different bit precisions and quantization techniques for deep learning applications. We devise training time techniques to minimize the accuracy degradation as a result of lower precisions. We also propose the utilization of augmented network topologies to claim back the accuracy degradation while still offering significant hardware benefits. As one data point, and for CIFAR-10 dataset, we report energy savings of up to 36% for an augmented network based on powers-of-two weight, and while maintaining classification accuracy. Finally, we evaluate the accuracy-performance frontiers of a biometric security system (more specifically an iris recognition system) by identifying approximation knobs throughout the processing pipeline and exploring the resulting design space. We deploy an end-to-end system, where images are captured using an infrared camera and processed through a HW/SW co-designed pipeline implemented on an FPGA board. Our introduced approximations result in $48\times$ speedup in runtime compared on an already hardware accelerated design while maintaining industry standard levels of accuracy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Characterization

Historically, with the advance of fabrication technology, more and more computational power has been available for the same hardware real estate. This trend based on the scaling in the fabrication technology size, enables larger numbers of computation elements for the same chip size. Established in 1965 by Gordon Moore, Moore's law states that the number of transistors in an integrated circuit is roughly doubled every 18 month [61]. In addition, in the performance and power efficiency domain, as the transistors decrease in size, the switching speed and power consumption both improve. Moore's law was derived by empirical data and while slowing in down in the recent years, has stayed relevant for decades. In 1974, Robert Dennard, complemented Moore's law for metal-oxide-semiconductor field-effect transistors (MOSFETs), by stating that along with the scaling in transistor size, both voltage and current scale proportionately to the length of the transistors, while power density stayed roughly constant [18]. Dennard scaling offered

the manufacturers the possibility of increasing the clock frequency significantly, therefore achieving significant performance improvements for similar power budgets.

While these laws have provided a roadmap for the semiconductor industry for decades, in recent years, the scaling of the fabrication technology has been less than ideal. While the number of transistors are still increasing in similar chip footprints, the rate of scaling has slowed down to doubling approximately every 2.5 to 3 years [13]. In addition, unforeseen difficulties with deep submicron process nodes, such as increased leakage power and thermal runaway issues, have put a halt on the Dennard scaling. These issues have limited the increase in the clock frequency effectively curbing the single core advances and pushing the manufacturers toward multicore architectures. Since 2006, multicore and manycore architectures have provided performance improvements by exploiting the parallelism in applications and multiple contexts [20]. Lack of parallelism as well as thermal considerations lingering from the single core era, however, have once again slowed the computing performance gain initially offered by multicore.

In addition to power density and thermal issues, which has come to be known as the power wall, the widespread utilization of battery operated devices has created yet another motive for low power high efficiency designs. The significance of low-power design in such embedded systems is especially boldened by the reality that power consumption directly maps to battery life and, therefore, to longer operation time between charges [72]. As the multicore paradigm, due to scalability, appears to be a temporary solution for the power issues, and with higher efficiency requirement due to broad adoption of hand held devices, a shift toward dark silicon, system on chip (SoC), and emerging low-power design paradigms has been the hardware design community's response to the aforementioned thermal and power issues.

One such emerging energy-efficient design paradigm is approximate computing. Ap-

proximate computing proposes to intentionally introduce small and insignificant amounts of inaccuracies into the computational pipeline, in favor of significant reductions in design area, design complexity, power consumption, or critical path delay. Such approximations are only appealing if deployed in applications where the introduction of inaccuracies in the result will not significantly degrade the quality of service (QoS). Many applications, by their own nature, are prime candidate for such techniques. Examples of error tolerant applications include applications with results interpreted by human perception, lacking a global best result, using noisy input, or with redundancies in the input data [29]. Furthermore, with the exponential growth in machine learning and data oriented applications, the number of applications amenable to approximate computing has only increased. The benefits of approximate computing also stem from the fact that they offer benefits in many different aspects of the hardware design simultaneously. Approximate computing techniques have been proposed and utilized in different components of the design stack ranging from application level [73, 21, 87], down to architectural [26, 42, 44, 54, 65, 91] and device level [12, 24, 46, 67].

One specific aspect of architectural approximate computing which has attracted a lot of attention is approximate arithmetic design [27, 42, 45, 52, 54, 62]. The promise of approximate arithmetic is that by design of efficient and flexible approximate arithmetic (such as adders, multipliers and dividers), one can readily utilize these designs as building blocks for many different applications from different domains or as part of an approximate ALU in a CPU or GPU. As an example, an approximate multiplier can simply be plugged in, instead of the the accurate counterpart, in many different applications ranging from communication applications, digital signal processing (DSP) applications, computer vision applications, or machine learning applications without requiring any tailoring for each individual application.

Another direction to approximate logic design is approximate synthesis for arbitrary

logic circuits. Here, the research has focused on developing methodologies that can approximate any input circuit into an approximate variant and in an automated fashion [65, 88, 91]. Such methodologies do not require circuit specific knowledge and offer the flexibility of operating on different circuits with different characteristics.

Finally, while approximate computing subcircuits have shown promising benefits with small reductions in result accuracy, end-to-end evaluation of approximate components in a complete pipeline can provide great insights into the applicability of such approximate computing techniques in real world applications. Such end-to-end systems have been considered and evaluated previously [69] and show immense potential for reducing the design footprint of different hardware pipelines albeit at the cost of small accuracy degradations. Next, a summary of the contributions made in this thesis is provided.

1.2 Major Contributions of This Thesis

In this section, we summarize our contributions, as they relate to different components of approximate computing and low-power design paradigm.

1. **A Novel Dynamic Methodology for Design of Approximate Multipliers and Dividers:** In this thesis, in Chapter 3, we propose a novel truncation-based methodology for design of approximate arithmetic logic. We exploit the fact that in binary operations not all bits of a input operand are equally significant. Therefore, we proposed to devise steering logic that can detect and forward the most important bits of each operand to a smaller, hence less expensive, accurate arithmetic core. We design our hardware to use leading one detectors resulting in a dynamic approach, where our approximate methodology can always capture the most relevant components of

each input operand. Further, to improve the result accuracy, for each arithmetic operation we ensure a zero-centered error distribution, therefore enabling the errors to cancel each other out instead of accumulate in typical multiply-accumulate operations. In addition, we explore the degree of dynamic truncation as a design time approximation parameter offering a wide range of trade-offs between accuracy and design metrics. We fully implement and evaluate our approach for two of the most computationally expensive arithmetic operations, namely multiplication and division. Extensive experiments are performed and the approximate arithmetic logic is evaluated in terms of both accuracy and design metrics such as design area, total power consumption, and critical path delay. We showcase designs with power savings of up to 71.45% for an average absolute error of 1.47%, for the case of the approximate multiplier, and power savings of up to 70.81% for an average absolute error of 3.08% for the approximate divider. Finally, to show the applicability of such methodology in real world applications, we evaluate each approximate operation on three different benchmarks from domains of signal processing, computer vision, and machine learning. Applications results highlight significant benefits while introducing negligible accuracy degradations.

2. **Automated Boolean Level Approximate Logic Synthesis:** Non-negative matrix factorization (NNMF) is an algorithm where an input matrix ($\mathbf{M}$) is factorized into two matrices ($\mathbf{B}$ and $\mathbf{C}$) with smaller dimensions, where all elements of the three matrices are non-negative and such that $\mathbf{M} \approx \mathbf{BC}$. Boolean matrix factorization (BMF), as a special case of NNMF, further restricts the elements to be of Boolean representation, allowing only '1's and '0's. A wide range of applications in domains of language processing, and data mining, to name a few, utilize BMF algorithms where a low-dimensional representation of the data is required. In this case, $\mathbf{B}$ encodes the available classes, or groups, while each element of $\mathbf{C}$ represents the existence (or non-existence) of the corresponding element in an specific input. In

this thesis, in Chapter 4, we propose to utilize BMF to reduce a complex arbitrary circuit into an approximate variant in an automated fashion. Our technique maps the input circuit to its corresponding truth table and feeds the truth table to the BMF algorithm, effectively breaking the input circuit into a cascade of a compressor and a decompressor circuit. Our methodology benefits from significant flexibility in accuracy-design metric trade-offs as the latent dimension (i.e. the factorization degree) behaves as an approximation knob, enabling different trade-offs. To navigate the complex design space of different approximation degrees, we devise a simple heuristic where the best combination of approximation degrees for different circuit components are found in an automated fashion. We implement and evaluate our methodology on a broad range of combinational logic and show significant benefits of up to 47.55% in total power consumption for an average error bound of 5%.

3. **Application Case Study 1:** Deep neural networks produce the state of the art results in many complicated machine learning and computer vision applications. While heavily reliant on simple arithmetic, neural networks show the characteristics of an error tolerant application as, in many cases, they operate on noisy inputs and have an abundance of data redundancy in the intermediate layers. As a case study of the approximate computing techniques in complex computing systems, we investigate the design space of the deep learning applications using different bit precision and quantization schemes. We evaluate a comprehensive range of precisions, from floating point arithmetic, to multiple fixed-point, powers-of-two quantization, and binary and trinary network architectures. For each design point, we evaluate the design on three different network architectures and datasets, namely LeNet [47] on MNIST, ConvNet [77] on SVHN, and ALEXNet [41] on CIFAR-10. In addition, we investigate training time techniques for reducing the accuracy degradation as a result of lower precision. Finally, to show the benefits achievable using lower precision, we also propose use of augmented networks, i.e. slightly larger networks, with

lower precision delivering significant energy savings while increasing the accuracy qualities back to full precision levels. Our precision exploration along with the augmented network methodology, in the case of CIFAR-10, offers energy savings of up to 36% while maintaining classification accuracy.

4. **Application Case Study 2:** As another case study, we explore the accuracy energy trade-offs offered by approximate computing for a biometric security system. Biometric authentication applications have been present in daily life of millions of people where finger prints, iris signatures, and facial recognition systems have been intensively used in mobile phones, as well as many other security and identification applications. All such biometric identification applications rely on the scarcity of false positive (in the range of one in millions) while false negatives are not as catastrophic. Among the existing biometric methodologies, one of the most secure is iris recognition; where in a typical system false positive rate is a mere one in 1.5 billion. Here, we introduce and explore a wide range of approximation knobs in the computation pipeline, where signature accuracy can effectively be traded for energy and runtime benefits. We explore our methodologies in an end-to-end system where the images captured from an infrared camera sensor are passed through a pipeline with three different and computationally heavy processing components. In addition, for an effective design space exploration methodology we devise an algorithm based on reinforcement learning. We show runtime benefits of up to $48\times$ due to the introduction of the approximations, while adhering to the industry standard levels of accuracy.

The remainder of this thesis is organized as follows. Chapter 2 briefly presents the required background in digital circuit low-power design, provides an overview of current low-power design techniques, and presents an introducing to approximate computing. Here, we also summarize some of the previous work in approximate computing as they

relate to the work in this thesis. Next, Chapter 3 presents our methodology for approximate arithmetic design specifically for the case of approximate multipliers and dividers. In Chapter 4, we provide an in-depth description of our Boolean matrix factorization (BMF) based approximate logic synthesis along with our results. Two detailed case studies for exploring the energy-accuracy trade-offs of neural networks and biometric iris recognition systems are described, in detail, in Chapters 5 and 6, respectively. For each case study, we also provide a brief background for the computational pipeline. Finally, in Chapter 7, we highlight the main finding of these thesis and suggest new directions for research extensions to this work.

Chapter 2

Background

2.1 Low-Power Hardware Design

Power consumption of a large-scale digital chip can be divided into two components, namely dynamic power and static power. Dynamic power is the power consumed in a chip due to its switching activity in the transistors and interconnects. While dynamic power can further be divided into two contributing factor, namely logic activity power a short circuit power, here we neglect the short circuit power as it is overwhelmingly dominated by the logic activity power. Dynamic power can depend on intrinsic characteristics of the circuit, the runtime activity, or the operation circumstances. Formally, dynamic power is formulated as

$$P_{dynamic} = \frac{1}{2}\alpha C V_{dd}^2 f, \tag{2.1}$$

where α denotes the switching activity, C denotes the effective switching capacitance of the circuit, V_{dd} denotes the supply voltage, and f denotes the clocking frequency. As evident from the equation, one can reduce the dynamic power by reducing either of the switching activity, effective capacitance, supply voltage, or clock frequency. Furthermore, in typical operating conditions, frequency has a linear dependence to supply voltage. Therefore, in order for the circuit to switch faster, higher supply voltages are required. As such, supply voltage, V_{dd}, and the operating frequency, f, are the two main factors in dynamic power.

The second component in VLSI power is the static power. Static power is the leakage power that the transistors have even when the circuit does not switch or is not in use. The leakage power can further be divided to three main components, gate leakage, drain junction leakage, and sub-threshold leakage, where sub-threshold leakage is the dominating factor. Leakage power has exponential dependence on threshold voltage and temperature. The leakage power is given by,

$$P_{static} = V_{dd} \times I_{static}, \tag{2.2}$$

where I_{static} is formulated as,

$$I_{static} = I_s e^{\frac{V_{GS}-V_{th}}{nkT/q}} \left(1 - e^{\frac{V_{DS}}{kT/q}}\right). \tag{2.3}$$

Here, I_s is a constant dependent on transistor's dimensions and the process node, q is the electrical carrier charge, n denotes the subthreshold slope factor, k denotes the Boltzmann constant, T denotes the transistor's junction temperature, V_{th} denotes the transistor's

threshold voltage, and V_{GS} and V_{DS} denote the gate-source and drain-source voltages, respectively. Therefore, for a given technology node and transistor geometry, static power can be controlled by modifying the supply voltage, or the operating temperature.

Furthermore, while initially dynamic power overwhelmingly dominated the total power consumption, with the aggressive scaling in sub-100 nm technology nodes the contribution of the leakage power has constantly increased, such that controlling subthreshold current is considered one of the main challenges in MOSFET scaling in future technology nodes [66]. While many novel approaches have been introduces to reduce the leakage power (including the introduction of FinFETs), static power is expected to remain a major contributor to total power consumption necessitating the introducing of new directions for low-power design.

In low-power design, all available techniques effectively reduce one (or both) of these components. More specifically, for a given technology and process node, recent techniques aim to reduce one of the main contributing factors, namely capacitance, switching activity, operation frequency, supply voltage, or operation temperature. As an example, dynamic voltage and frequency scaling (DVFS) is a popular technique where the frequency and the supply voltage of a processor can be altered as needed based on the workload conditions. Here, as evident from the name, benefits in power can be achieved by dynamically reducing the voltage and the frequency when the application demands are not high. Note that DVFS offers benefits for both the static and the dynamic power. Power gating, as an alternative, suggests the isolation of the logic from the supply voltage rails when the subcircuit is not in use, effectively eliminating the static power [19]. Similarly, clock gating provides benefits in dynamic power by gating the clock of the logic to reduce the switching activity. Finally, multi-threshold designs have also been investigated.

While all these methods can provide significant improvements in energy efficiency,

power remains the main problem prohibiting more computational power or battery life. This fact, necessitates the exploration of novel and unexpected techniques to further improve the efficiency of computing systems. One such technique is approximate computing. Approximate computing has the benefit that it can provide improvement on many different aspects of the hardware design including static power, dynamic power, timing, design area and design complexity. Next, we will provide a detailed discussion of approximate computing as well as a summary of current techniques.

2.2 Approximate Computing

Approximate computing has emerged as a new paradigm in design of low-power computing systems, where the underlying application benefits from inherent tolerance to insignificant degrees of error. Approximate computing has been approached from many different directions and applied on many different layers of the computing stack. On the highest level, approximations into software programs have been proposed to reduce the runtime and complexity of applications with error resilience [63, 80]. To this end, techniques such as loop perforation [78], relaxation of data dependencies [56], and selectively skipping less important computation [6] have been proposed. Alternatively, Baek. *et al.* proposed a technique where computationally expensive functions are replaced with cost efficient variants [3]. Furthermore, Esmaeilzadeh *et al.* proposed a methodology in which a code segment deemed error resilient is transformed into an approximate version implemented using a neural network trained to mimic the original software [22].

Approximate instruction set architectures (ISA) have also been proposed. In this approach, generally, a generic processor is designed where a subset of operations can be mapped to approximate assembly level instructions. An instruction set architecture with

approximate operation support in proposed by Sampson *et al.* where the programmer can provide type annotations [73]. On a similar approach, an ISA that relies on the compiler to detect operations suitable for approximation is proposed by Esmaeilzadeh *et al.*. In this work, a high level microarchitecture supporting dual-voltage is also proposed where higher voltage is used for accurate operations while a lower voltage is used for approximate ones [21]. Venkataramani *et al.* introduced a quality programmable vector processor, with hardware support managing to translate instruction-level quality requirements into energy benefits [87].

On lower levels, approximate logic architectures have also been investigated [27, 42, 45, 52, 54, 62]. These methodologies aim at delivering power benefits by reducing the logic complexity of the design through removing a less prominent portion of the logic. Therefore, in this approach, the approximation error is added and controlled by construction. In addition, while in the approximate computing paradigm, these techniques have mostly focused on reducing power consumption, they commonly also deliver benefits in design area, critical path delays, and memory footprint.

One major subset of research in this area in approximate arithmetic design. Here, efficient approximate arithmetic logic, e.g. adders, multipliers, dividers, etc., are designed where these designs can later serve as the building blocks of more complicated logic circuits. Many approximate adders have been proposed [5, 26, 34, 39]. Gupta *et al.* proposed several approximate adder designs that, by removing some of the logic used in a traditional mirror adder, achieved improved power, area, and performance [27].

Furthermore, design of approximate multipliers has been investigated intensively [36, 52, 95]. Mahdiani *et al.* propose a bio-inspired approach, where the addition results within the multiplier were approximated by using OR gates for the lower part of the inputs [54]. Babic *et al.* [2] proposed a pipelined log-based approximation where the classical Mitchell

multiplier [60] is utilized in an iterative approach to improve its accuracy. An error-tolerant multiplier design is proposed in [45] where the multiplication is divided into accurate (multiplication based) and inaccurate (non-multiplication based) parts. Liu *et al.* [52] proposed an approximate multiplier with configurable error recovery using fast approximate adders for partial product addition stage. Kulkarni *et al.* proposed an under-designed 2×2 approximate multiplier block to generate partial products, where larger multipliers are built using the inaccurate block to calculate the partial products which are then added together to generate the final result [42]. Narayanamoorthy *et al.* proposed a static truncation based approach where the higher, middle, or lower portions of the inputs are used to approximately calculate the result [62].

While the majority of research in approximate arithmetic, due to their higher utilization, has been focused on adders and multipliers, other building blocks have also been investigated. Approximate subtractor cells [9, 8] and approximate dividers [96, 85, 93] are a few examples.

A different prospective to changing the underlying architectures, is approximate logic synthesis. In this paradigm, methodologies for automated design of approximate logic from arbitrary accurate counterparts are investigated [65, 88, 91, 70, 90, 57, 86]. In SALSA, a systematic approach for approximate circuit synthesis is proposed [88]. The idea is to create a difference circuit that compares the QoR between the original circuit and the approximated circuit. The don't cares of the outputs of the approximate circuit – which are internal nodes in the difference circuit – with respect to outputs of the difference circuit can be used to simplify the approximate circuit using regular logic synthesis techniques. This approach has been extended in ASLAN [70] to model error arising over multiple cycles. ASLAN also uses a circuit block exploration method that identifies the impact of approximating the combinational blocks and then uses a gradient-descent approach to find good approximations for the entire circuit. In SASIMI [90], a technique

is proposed to identify similar signals, such that their values agree over a large number of input test cases, and then substitute one for the other, simplifying the logic. A logic synthesis formulation proposed by Miao *et al.* uses a two-level logic synthesis approach that incorporates constraints on error deviation, and then a heuristic is used to solve the synthesis formulation [57]. Evolutionary techniques have been also explored [86]. Further, ABACUS seeks to generate variants of an input high-level Verilog description file by applying a set of possible transformations, such as bit width truncation, operand simplification and variable-to-constant substitution, to generate a set of mutant approximate circuit variants [65]. A multi-objective design space exploration technique is used to identify the best set of approximate variants. Recently, a new technique is proposed to raise the level of abstraction by synthesizing approximate circuit directly from C descriptions [50]. High-level synthesis in conjunction with approximations on the critical path can yield additional savings through voltage scaling [64, 50].

From a different prospective, in recent years, design and exploration of approximate memory subsystems has also been proposed [35]. Sampson *et al.* proposed a technique for approximate multi-level cells by reducing the number of programming pulses required for writing the data [74]. A methodology for construction of quality aware DRAMs using sub-optimal refresh rates is proposed by Raha *et al.* [68].

Approximation techniques on circuit level have been also proposed. One of the main approaches is voltage over-scaling (VOS) where the supply voltage is reduced to below safe working threshold in an attempt to save power [12, 24, 46, 67]. Inaccuracies are introduced into the circuit as the lower voltage results in timing violations in the circuit. While voltage over-scaling offers the benefit of flexibility on using the same hardware component in accurate or approximate modes, resulting timing errors occur on the most critical paths of the circuit. In arithmetic logic, these paths usually compute the most significant bits, therefore, VOS can result in significant and nondeterministic errors.

While many advances have been made in the approximate computing paradigm, most of the work evaluate the quality-energy trade-offs of a single module or algorithm in isolation. Optimal benefits in an end-to-end system can only be explored if the approximate computing techniques utilize all parts of the system pipeline where trade-offs are evaluated in connection with each other. Recently, Raha *et al.* proposed a full-system approximate design using a smart camera system as a case study [69]. In their system, approximations are introduced using camera resolution scaling, reducing memory refresh rate, and computation skipping.

Chapter 3

Dynamic Approximate Arithmetic Logic

3.1 Introduction

We describe our technique for design of low cost, low power, approximate multipliers and dividers in this chapter. Our technique targets error resilient applications and maintains desirable features that will facilitate its utilization in a broad range of applications. Our technique achieves significant savings in design metrics while introducing small amounts of error. Our work, as presented in this thesis, has previously been published in [31] and [32]. We summarize our contributions in this chapter as follows.

1. We devise a dynamic methodology where the most important bits of each of the operands are selected and the arithmetic complexity is reduced significantly by discarding the lower, less significant bits, as well as the leading zeros.

2. For both arithmetic operations, our proposed methodology provides a wide range of fine-grain trade-offs between accuracy and design metric by introducing a design

time approximation parameter. This enables the adaptation of the methodology for specific application requirements.

3. Our methodology utilizes a smaller accurate arithmetic block at its core, therefore it offers the flexibility to use any arithmetic implementation as necessitated by other design factors.

4. Our methodology ensures a zero balanced error distribution for approximate arithmetic, effectively preventing the accumulation of errors when the operation is performed repeatedly. In other words, our approximate design produces both positive and negative errors providing the opportunity of errors canceling each other out.

The rest of this chapter is organized as follows. In Section 3.2, we describe our methodology for design of the approximate arithmetic blocks and the intuition behind our approach. Later, in Section 3.3 we thoroughly evaluate the accuracy and design characteristics of the proposed approximate arithmetic blocks. Here, and for each operation, we initially, focus on an standalone approximate operator, and then, we report the benefit in the context of real world applications. Finally, we summarize this chapter's contribution in Section 3.4.

3.2 Approximate Arithmetic Methodology

The approximate methodology proposed in this chapter exploits the realization that not all bits of a number, in our case a binary number, have similar significance. For example in an 16-bit unsigned number a '1' in most significant bit (MSB) has a value of 2^{15} while a '1' in the least significant bit (LSB) adds a value of 2^0 or 1. Further, as demonstrated in the example, this significance grows exponentially as we move to higher index bits. Our

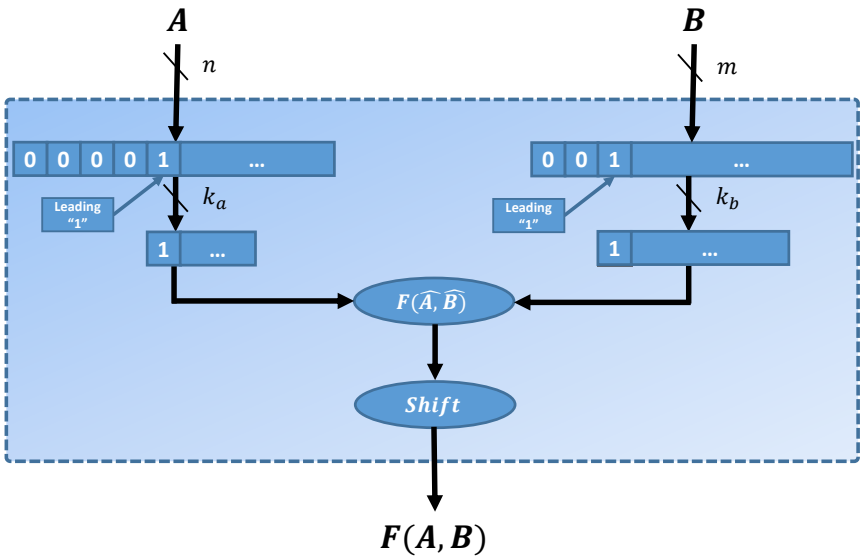


Figure 3.1: The general methodology proposed in this chapter. Here, each operand is dynamically approximated and the computation is performed accurately on the approximate operands.

methodology proposes to take advantage of this fact and limit the number of bits used for actual computation. In other words, the approximate methodology selects a subset of bits most representative of each operand and forwards them to the accurate core arithmetic.

Within such framework, the main question is how to select the best possible bits for computation. Here, we advocate a dynamic approach where we utilize leading on detectors (LOD) to zoom in on the most important bits of each operand. More specifically, we propose to select a chunk of each operand starting from the leading one. Regardless of input values, such an approach has the benefit that the maximum error can be bounded. The basic idea of our methodology is demonstrated in Figure 3.1. Here, $\hat{A}$ and $\hat{B}$ represent approximated operands A and B respectively, while $F(\cdot,\cdot)$ represents the operation.

On the other hand, in hardware implementation domain, our method proposes to design an approximate arithmetic building block by reducing a large and expensive arithmetic operation to some steering logic (responsible for finding and routing the most impor-

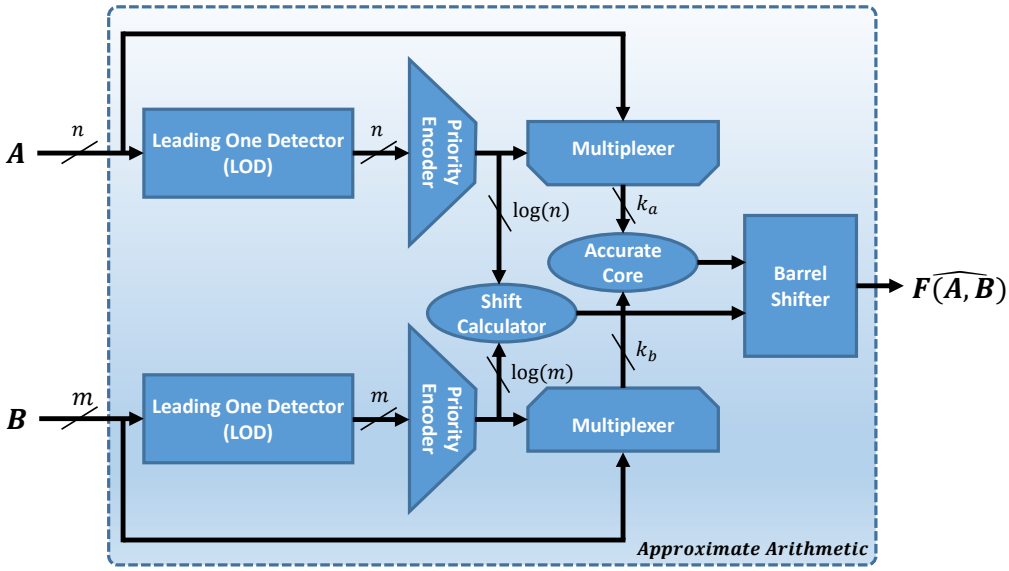


Figure 3.2: The generic schematic of the proposed methodology. $\widehat{F(A,B)}$ represents the approximate result.

tant bits) and a significantly smaller exact arithmetic (responsible for the actual calculation of the approximate operands). Such methodology is justified in any arithmetic or combinational logic where the overhead of the routing logic is significantly less demanding than processing the entire operands.

Figure 3.2 shows the generic schematic of such an implementation. The hardware design is composed of two parts: a *steering logic* and an *arithmetic logic*. The steering logic is responsible for dynamically selecting the right range of the input operands (e.g. k_a and k_b bits) and feeding them to the arithmetic logic, and afterwards shifting the calculated result from the arithmetic core to the right index. Therefore, the steering logic is comprised of LOD blocks, encoders, multiplexers, and a barrel shifter, while the arithmetic core is simply an accurate, smaller implementation of the arithmetic operation.

In our implementation, each input operand is first fed to a LOD circuit where the location of the leading one is identified. The output of the LOD block is an one hot

encoded number indicating the location of the most significant one. Next, the encoder components translate the detected locations to binary representations. These numbers are then used to select the relevant bits from each operand using multiplexers as well as to calculate the number of shifts required. The selected bits for each operand are then fed to the accurate computation core to generate the partial result. One key benefit of our architecture is that it does not restrict the designer from using their own preferred design as the smaller core arithmetic. Finally, a barrel shifter shifts the output of the arithmetic core to the correct index based on the location of the leading ones. For each operand, if the input operand is small enough to be represented with k_a(or k_b) bit without approximation, the steering logic simply forwards the input exactly as is to the core arithmetic.

To further improve the accuracy, one can ensure an *unbiased* error distribution. Unbiased here means that the error distribution is centered around '0', generating both negative and positive errors. The main benefit of an unbiased approximate logic is that some errors can potentially cancel each other out rather than accumulate. For both our approximate designs and based on the operation, we slightly adjust our bit selection scheme to maintain an unbiased error distribution. As the unbiasing method is operation dependent, we will further discuss our approach later on.

Furthermore, another important benefit of this methodology is that as the input size grows, the dynamic nature of our approach, while maintaining accuracy, leads to more design savings. This results in an approximate methodology that is highly scalable to higher input widths. In the proposed design, and for the operations considered, while the complexity of the steering logic grows as a factor of the input size, to maintain accuracy, the core arithmetic does not need to grow. Therefore, as we move to higher input widths, within the same accuracy bounds, the power and area benefits only increase. Table 3.1 summarizes the relationship between the steering and the arithmetic logic area and the size of the input using $O(\cdot)$ notation for the proposed design. In Section 6.4.2 we will

Table 3.1: Area growth as a function of n and k for the proposed designs compared against accurate designs.

Multiplier	Steering Logic	Arithmetic Logic	k
Accurate Arithmetic (n)	-	$O(n^2)$	-
Poprosed Methodology(k,n)	$O(n \log n)$	$O(k^2)$	$k \sim Const.$

provide experimental data supporting our argument.

As previously discussed, the arithmetic operation being approximated needs to be complex enough for savings in the core arithmetic to justify the steering logic. Therefore, a dynamic approach, as discussed in this chapter, is not suitable for simpler logic such as adders and subtractors. Multiplier and dividers on the other hand, offer significant enough benefits to justify our methodology. While the general approach for both approximate multiplier and the approximate divider is as previously described, for each implementation we introduce slight modifications to the algorithm to improve the results based on the characteristics of the operations. Next, we will discuss specifics of each arithmetic operation, including the bit selection and unbiasing.

3.2.1 Approximate Multiplier Design

In this subsection, we provide the details of the proposed method when deployed as an approximate multiplier. In this operation, assuming each operand has n bits, our design uses two Leading One Detector (LOD) circuit blocks to dynamically locate the most significant '1' in each of the two operands as illustrated in Figure 3.3. For each operand, the location of the most significant '1' is then used to select the following $k-2$ consecutive number of bits based on the required accuracy. Here, k is a designer-defined value which specifies the bandwidth used in the core accurate multiplier.

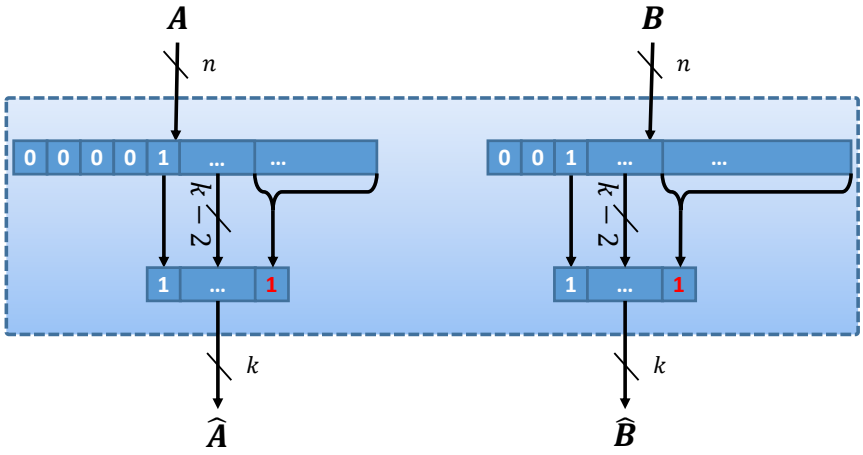


Figure 3.3: The input operand approximation for the proposed approximate multiplication. The red '1's show the unbiasing bits and $\hat{A}$ and $\hat{B}$ represent the approximated operands.

As mentioned before, in the case of the approximate multiplier and to improve the error characteristics, for each operand we deploy an *unbiasing bit* to ensure a zero-centered (balanced) error distribution. In order to do so we modify our bit selection scheme slightly to approximate the value of the truncated bits by their expected value. Thus, we unbias each approximate operand by reducing the lower bits to a '1' at the most significant lower bits index (shown in red in the figure) and '0's for the rest. Finally, to generate the approximate operands, the tailing '0's are truncated resulting in k-bit approximate operands. These k-bit operands are then forwarded to the inputs of a $k \times k$ accurate multiplier. Note that as a multiplier has commutative property and the operands are treated equally, in our approach we approximated them similarly as well.

Figure 3.4 illustrates a numerical example of the approximate multiplication with input size 16 and $k = 6$. In this figure, bold numbers represent the selected bits that will be routed to the arithmetic logic as they are while the red bits are used in the unbiasing process. In this example, the unbiasing results in significant improvements in accuracy at no overhead. Using the unbiasing bit, the relative error is 0.27%, while an approximate multiplier with the same size core multiplier without the unbinding bit (therefore approximate

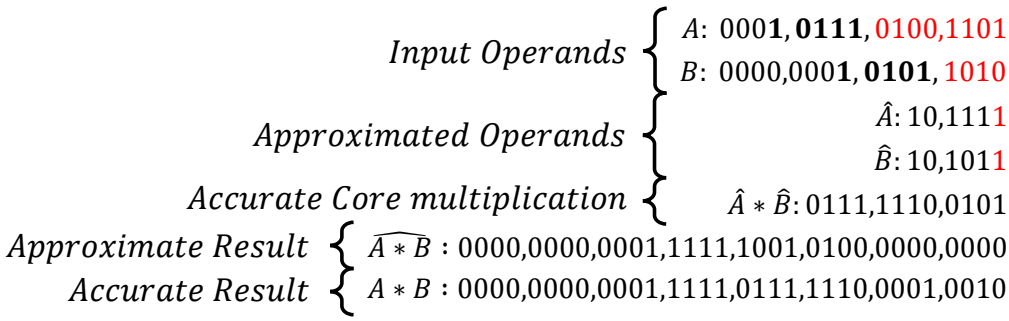


Figure 3.4: A numerical example demonstrating the operation of the approximate multiplier with a relative error of 0.27%. The unbiasing is shown in red, while the selected bits are shown in bold.

the lower bits with zero) has a 1.86% relative error.

3.2.2 Approximate Divider Design

In this subsection, we describe, in more detail, the workings of our proposed approximate divider. In order to be consistent with literature, we maintain a $2/1$ ratio between the dividend and the divisor. Therefore, in our divider design, we represent the dividend with n bits while representing the divisor with $n/2$ bits. While this ratio is kept constant throughout this chapter, the proposed approximate methodology can be readily used to implement dividers with any random input widths without restriction.

As in the case of approximate multiplier, for designing approximate dividers LODs are utilized to locate the indices of the leading ones and to select the most important bits of each operand accordingly. In the case of the divider, however, multiplexers select k bits and $k/2$ bits from the dividend and the divisor respectively. In the case of approximate divider, however, in contrast to the approximate multiplier underestimating both operands can result in both underestimation and overestimation of the result. Therefore simple truncation of lower bits, leads to an unbiased error distribution. Therefore, we opt to

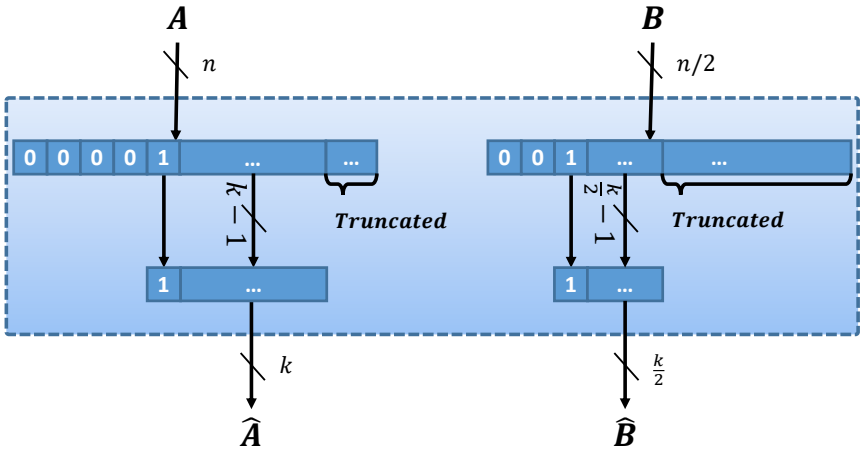


Figure 3.5: The input operand approximation for the proposed approximate divider. The lower bits are simply truncated. $\hat{A}$ and $\hat{B}$ represent the approximated operands.

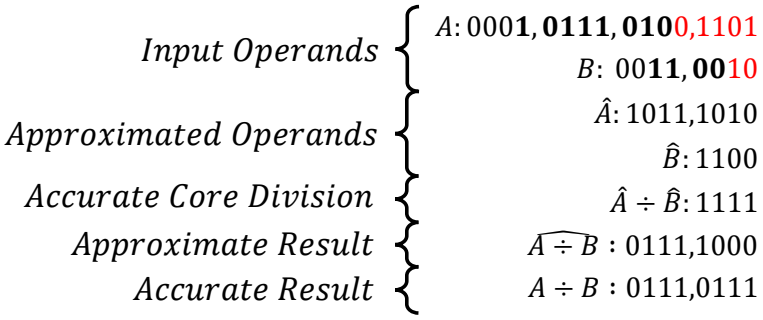


Figure 3.6: A numerical example demonstrating the operation of the approximate divider with a relative error of -0.84%. The truncated bits are shown in red, while the selected bits are shown in bold.

use this scheme as it allows for one more operand bit to be forward to the accurate core. Figure 3.5 shows the divider steering logic in action.

Figure 3.6 illustrates a numeric example demonstrating the operation of the proposed approximate divider with input size $16/8$ and $k = 8$. Here, bold numbers represent the selected bits that will be routed to the arithmetic logic as they are while the red bits are simply truncated. In this example, the approximate result has a relative error of -0.84%.

3.2.3 Support for Negative Arithmetic

While in this chapter we focus on unsigned operations, our proposed approximate design can support signed operation with a straight-forward extension. For this purpose, pre-processing logic can be added in order to convert the signed operands to unsigned inputs using twos complement before forwarding them to an unsigned approximate arithmetic block. Furthermore, in the applications with enough error tolerance, the two's complement method can be replaced by simple bit inversion to skip the long carry chain of the needed addition by '1' and improve the delay. The sign signal for the result is then calculated separately and the output will be negated if necessary. These scheme adds small area and power overheads and therefore should only be deployed if necessitated by the application.

3.3 Experimental Results

In this section, we thoroughly evaluate our proposed methodology on both the approximate multiplier and the approximate divider. For our empirical evaluations we consider both computational accuracy as well as hardware design metrics such as design area and power consumption. All designs are coded in Verilog and synthesized using an industry strength 65-nm standard cell library in the typical operation corner. We use Synopsys Design Compiler for synthesis and MentorGraphics Modelsim for accuracy simulations. For both arithmetic, we evaluate the design both as an standalone hardware block and as an arithmetic block integrated within multiple applications from different domains.

For the standalone results, two sets of randomly and uniformly generated input vectors are used to evaluate computational accuracy results. The accuracy performance is reported

in respect to an accurate arithmetic counterpart. For reporting error on standalone application, we define maximum error distance as,

$$Max\ ED = Max_{i \in I}(|App(A,B) - Acc(A,B)|) \tag{3.1}$$

where $App(A,B)$ represents the approximate result, $Acc(A,B)$ represents the accurate result, and I denotes the testset. We define average absolute error as,

$$Average\ Absolute\ Error = \frac{1}{Size(I)} \sum_{i \in I} \frac{|App(A,B) - Acc(A,B)|}{Acc(A,B)} \tag{3.2}$$

and error bias as,

$$Error\ Bias = \frac{1}{Size(I)} \sum_{i \in I} \frac{App(A,B) - Acc(A,B)}{Acc(A,B)}\ . \tag{3.3}$$

We also report the standard deviation for each design. For application analysis, on the other hand, we use application specific quality metrics. Next, we report the results obtained for each approximate design first as an standalone unit, and later as part of a complex datapath.

Table 3.2: Accuracy results for standalone approximate multiplier using different k ($n = 16$).

	range					
	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$
Max. Error %	56.25	26.56	12.86	6.31	3.1	1.54
Average Abs. Error %	11.90	5.89	2.94	1.47	0.73	0.37
Error Bias %	2.08	0.53	-0.14	-0.04	0.01	0.01
Standard Deviation %	14.75	7.26	3.61	1.80	0.90	0.45

3.3.1 The Approximate Multiplier

Standalone Multiplier Results

As mentioned in Section 3.2, the proposed methodology offers a range of trade-offs between accuracy and design benefits by changing the number of bits forwarded to the accurate core (k). Therefore, k is a design time approximation knob which should be determined based on the application requirements. Furthermore, theoretically, the number of bits selected can vary from 1 to n, therefore offering a wide range of trade-offs to choose from.

Here, as a first set of experiments we fix the value of n ($n = 16$) and evaluate the performance of the multiplier as a factor of k. The results are summarized in Table 3.2. Here, we provide maximum error, average absolute error, error bias, and standard deviation. Note that, here, to provide a better sense of significance the maximum error distance is normalized by the accurate value. As expected, with increase in the value of k all error metrics decrease. Furthermore, the error follows an interesting trend where the errors are roughly halved for each bit increase in the value of k.

Figure 3.7 shows the design savings offered by approximation methodology for the same experiment. Here, we show the total power (including both static power and dynamic

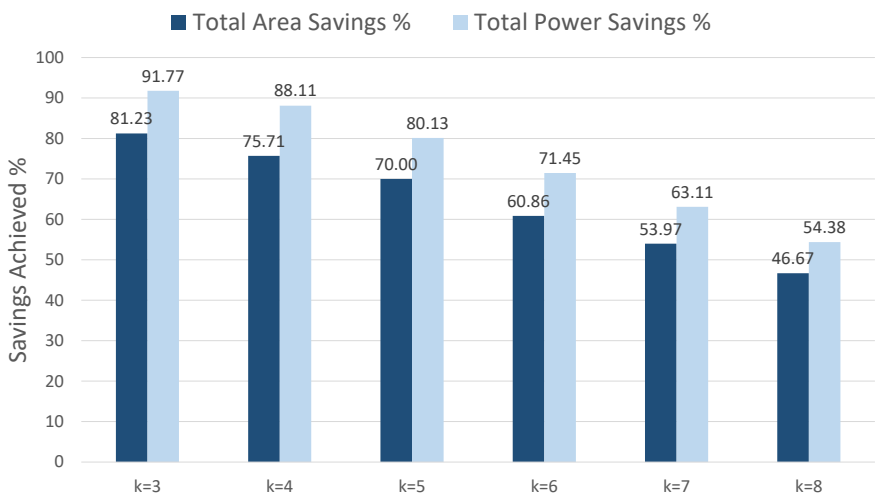


Figure 3.7: Total power and area savings as a function of k for the standalone approximate multiplier. ($n = 16$)

power), and the total design area. Again, as expected, the design area and power consumption increase as a factor of k. As demonstrated, significant savings can be achieved while introducing insignificant amounts of error. As an example, with an average absolute error of 1.47%, $k = 6$ offers up to 61% in area savings and up to 71% in power savings.

Figure 3.8 shows the fitted Gaussian error distribution of the proposed multiplier,with $n = 16$, and for different values of k. As shown in the plot, our approximate multiplier shows a Gaussian like error distribution, resulting from the unbiasing bit. Therefore, our unbiased error distribution will compensate for some of the error when utilized within real applications as will be demonstrated in next section.

Next, we consider the impact of the input size on the performance of the approximate multiplier. Table 3.3 summarizes the accuracy results for three different cases, namely $n = 16$, $n = 24$, and $n = 32$. In these experiments we chose $k = 6$ to demonstrate the behavior of the approximate design solely as a factor of n. It can be seen from the table that the dynamic nature of our approach prevents the error from degrading as the input size is increased. As a result, when moving to larger multipliers, for the same error

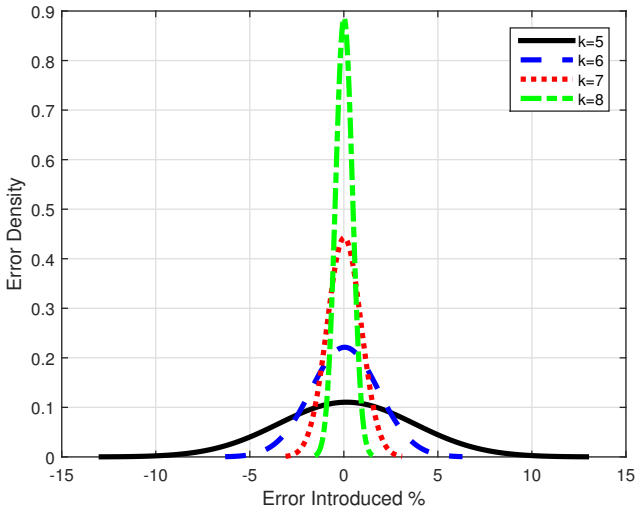


Figure 3.8: The fitted error distributions for the proposed multiplier for different k ($n = 16$).

Table 3.3: Standalone approximate multiplier accuracy for different input size ($k = 6$).

	multiplier size		
	$n = 16$	$n = 24$	$n = 32$
Max. Error %	6.31	6.31	6.31
Average Abs. Error %	1.466	1.467	1.467
Error Bias %	-0.043	-0.033	-0.033
Standard Deviation %	1.803	1.803	1.803

characteristics the design benefits only increase rendering our approach highly scalable. Figure 3.9 shows the benefits achievable in hardware metrics while changing the value of n.

Table 3.4 highlights the significance of the design parameter benefits obtained while introducing small errors into the results. In this table we also include the total power, design area, and the critical path delay values for both the accurate and approximate designs. Here, for $k = 6$ savings of more that 70% in area and power are achieved with a mere 1.47% average absolute error. We also report the critical path delay of both accurate and approximate multipliers and show a speedup of $1.89\times$.

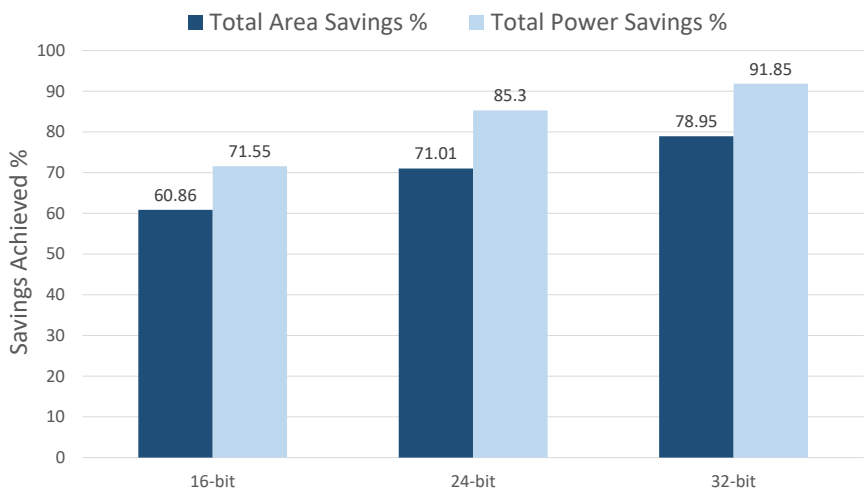


Figure 3.9: Area and power savings as a function of input size for the standalone approximate multiplier (k=6).

Table 3.4: Design Comparison of the proposed multiplier to other published work.

Multiplier Design	Max. ED	Average Abs. Error	Error Bias	Area (um^2)	Power (mW)	Area Savings	Power Savings	Critical Path (ns)
Accurate	-	-	-	2165	1.04	-	-	3.61
Approximate	6.31%	1.47%	-0.04%	649.4	0.296	70%	71.45%	1.91

Multiplier Application Results

As demonstrated in the previous subsection, the proposed methodology offers significant benefits as an standalone multiplier. The degree of benefits and errors, however, can significantly differ when the multiplier is utilized within an application. Furthermore, in approximate arithmetic with a biased distribution, errors can accumulate further degrading the application QoR. In this section, we evaluate our approximate divider within applications. We will also show how the introduction of the unbiasing '1' can eliminate accumulating errors and therefore improve the accuracy performance of the multiplier.

We evaluate the proposed multiplier using three applications from different domains. Our chosen applications are image filtering and JPEG compression from image processing

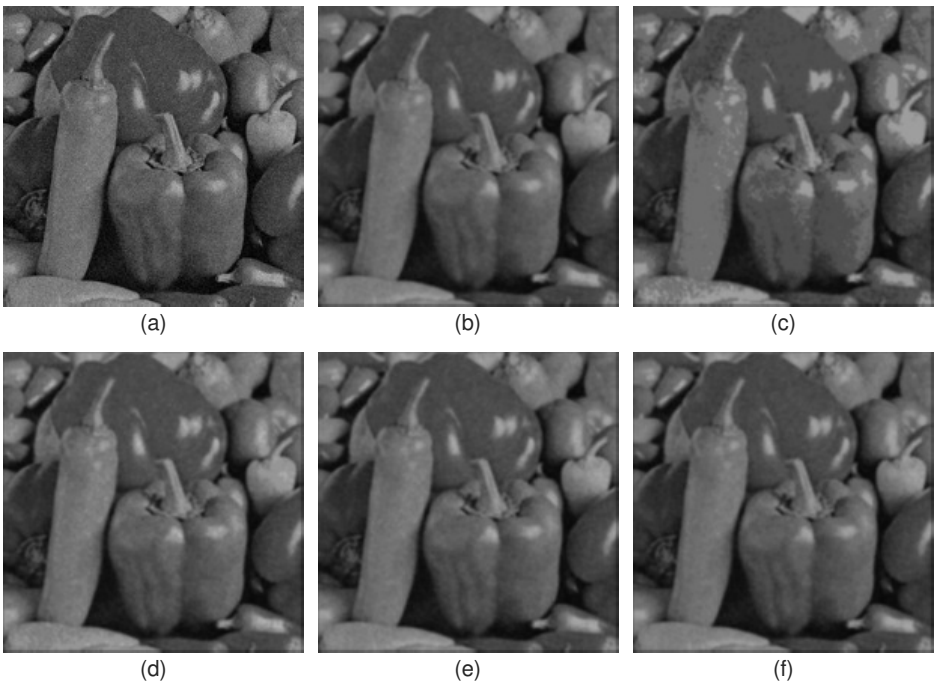


Figure 3.10: Gaussian filtering results for different values of k. (a) Input image; (b) Filtered with accurate multiplier; (c) $k = 3$, $PSNR = 34.58$ dB; (d) $k = 4$, $PSNR = 34.39$ dB; (e) $k = 5$, $PSNR = 42.36$ dB; (f) $k = 6$, $PSNR = 54.57$ dB.

domain and a perceptron classifier from the data classification domain. As before, we use Verilog to implement the applications and DC compiler for synthesis. For accuracy results, we model the applications and the multipliers in MATLAB using fixed point simulations.

As our first application, an image is convolved by a Gaussian-based smoothing kernel. We use a 7×7 kernel and use 16-bit fixed point arithmetic in the processing pipeline. The input image is a 200×200 greyscale image with 16-bit pixels and the approximate multiplier is used to replace all the multipliers in the convolution accelerator.

We visualize the approximate output resulting from different values of k. Figure 3.10 shows input image, the accurate result, and the approximate results from $k = 3$ to $k = 6$. As demonstrated, using different values for k, our methodology enables a wide range of quality energy trade-offs. Here, the accuracy metric is computed in reference to the

(a) (b)

Figure 3.11: JPEG compression algorithm. (a) Compressed using accurate multiplier, $PSNR = 26.17dB$; (b) Compressed using $k = 6$, $PSNR = 26.03dB$.

accurate output.

We also utilize our approximate multiplier in a JPEG compression pipeline. Figure 3.11 compares, visually, the result of the proposed approximated design, with $k = 6$, to an accurate multiplier on a test image when using 20 coefficients with a 0.53% degradation in PSNR. As demonstrated in the figure, the quality reduction is barely noticeable.

Finally, we evaluate the performance of the proposed approximate multiplier when utilized within a perceptron classifier. We use a simple classification task where 1000 two-dimensional point from classes $\{$-1,1$\}$ are classified. The error rate (ER) is defined as the percentage of mismatch between classification output and the ground truth. The results are shown in Figure 3.12. In reference to the accurate multiplier, the proposed approximate design fails to classify four points (out of 1000) correctly while excelling in classifying three other points (out of 1000). The ER for the accurate and approximate multipliers are 15.0% and 15.1% respectively. As before, we use $k = 6$ for the approximate multiplier.

Table 3.5 summarizes the design area and power consumption of each application when using both accurate multipliers and approximate multipliers proposed in this chapter. Here, we also report the area and power benefits achieved in reference to the accurate

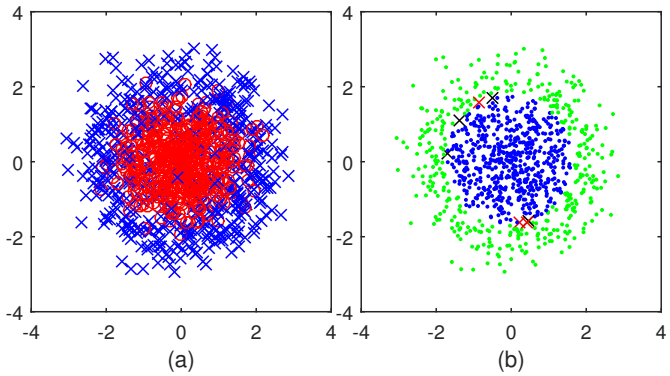


Figure 3.12: Classification Application. (a) The input data set of classes -1,1. (red=1); (b) The outputs of accurate and approximate multipliers. (dots:matching classification, crosses:mismatch. red:Additional detection, black:False alarm.)

Table 3.5: Design area and power savings for application implementations.

Application	Accurate Design		Approximate Design		Savings	
	area (um^2)	comb. power (mW)	area (um^2)	comb. power (mW)	area (%)	power (%)
Image Filtering	253982	15.55	186964	6.48	26.4	58.3
JPEG Compression	1862116	14.11	1357863	10.97	27.1	22.3
Perceptron Classifier	25022	2.24	19786	1.00	20.9	55.3

design. Note that we only report the total power for the combinational logic and the memory units are excluded from the analysis. Based on the application requirements we use 16×16, 32×32, and 16×32 input widths for image filtering, JPEG compression and perceptron classifier respectively. As expected, the achieved benefits strongly depend on the proportion of the total design dedicated to multipliers. As a result, power savings here range from 22%, in the case of JPEG compression algorithm, to over 50% for image filtering and the perceptron classifier.

Table 3.6: Accuracy results for the standalone divider for different values of k ($n = 16$).

	range				
	$k = 4$	$k = 6$	$k = 8$	$k = 10$	$k = 12$
Max. ED	64	44	26	13	6
Average Abs. Error %	13.57	6.37	3.08	1.42	0.59
Error Bias %	-1.78	-1.49	-0.93	-0.48	-0.23
Standard Deviation %	17.16	8.55	4.60	2.56	1.50

3.3.2 The Approximate Divider Results

Standalone Divider Results

In this Subsection, we evaluate the accuracy, power consumption, and design area characteristics of our approximate divider and compare these characteristics against those of an accurate divider. As before, first, in this subsection, we evaluate the approximate divider as an standalone divider design and report its behavior as a factor of k and n.

As discussed in Section 3.2.2, our divider is highly configurable. However, in the case of the divider, and as to maintain the $2/1$ ratio, k can range from 2 to n while assuming even numbers. As it was the case for the multiplier experiments, we first examine the impact of k on the divider performance. For this experiment, we fix n to $n = 16$. Table 3.6 summarizes the accuracy results with respect to the accurate design and for values of $k = 4$ to $k = 12$. As one would expect, the accuracy improves for all the evaluated accuracy metrics as we move to higher values of k.

Figure 3.13 shows the total power and design area savings offered for our approximate divider for different k values and in respect to an accurate divider. As expected a wide range of trade-offs are offered ranging from 29% to 90% in power savings with average absolute errors of 0.59% to 13%, respectively.

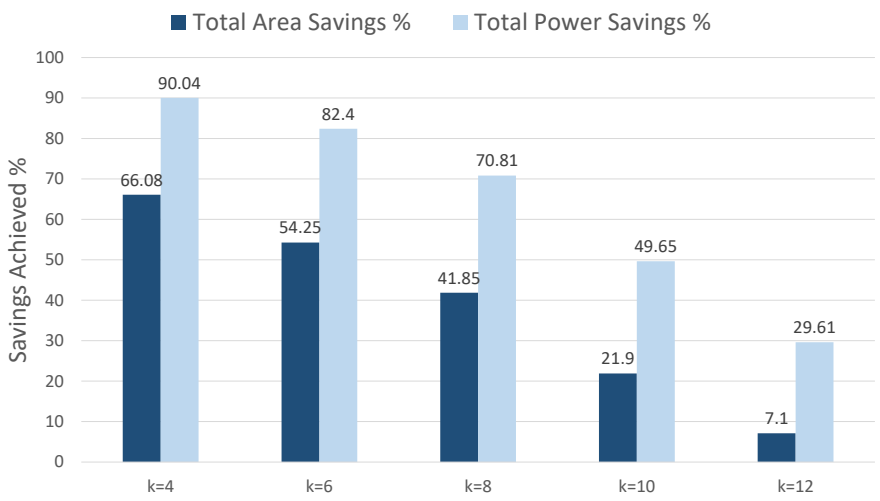


Figure 3.13: Area & power savings as a function of k.

Furthermore, we plot the error distance and the relative error distributions for a divider with $n = 16$ and $k = 8$. Figures 3.14(a) and 3.14(b) show the empirical error distributions. As illustrated in the figures, the biggest portion of the results are close to 0 in both distributions such that more than 99.5% of the relative error are within 15% error bound. The results in Figure 3.14(a) also show that the relative error is rather balanced around 0 resulting in lower error rates in applications where results of different divide operations are added together to generate the final result.

Next, we evaluate the performance of the approximate divider on different input sizes (n). We consider three input sizes, $16/8$, $24/12$, and $32/16$. The accuracy results are summarized in Table 3.7, and the design benefits are plotted in Figure 3.15. These results further support the benefits of our dynamic approach where the benefits only increase for higher divider widths while the errors stay within the same bounds.

Table 3.8 summarizes the accuracy performance, total power, design area, and the critical path delay values for both the accurate and approximate dividers. As an example, for $k = 8$, with 3.08% average absolute error, our proposed methodology can achieve

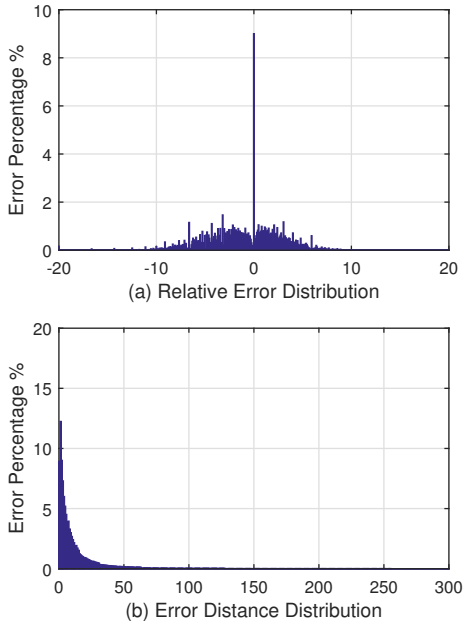


Figure 3.14: The error distributions, evaluated as a standalone divider.

Table 3.7: Accuracy for the standalone divider for different input sizes ($k = 8$).

	divider size		
	$n = 16$	$n = 24$	$n = 32$
Max ED	26	420	6772
Average Abs. Error %	3.03	3.09	3.09
Error Bias %	-0.97	-0.84	-0.84
Standard Deviation %	4.6	3.84	3.77

savings of more that 41% and 70% in design area and total power, respectively. The critical path delay is also improved by $1.77\times$ compared to an accurate divider.

Table 3.8: Design Comparison of the proposed divider to other published work.

Divider Design	Max. ED	Average Abs. Error	Error Bias	Area (um^2)	Power (mW)	Area Savings	Power Savings	Critical Path (ns)
Accurate	-	-	-	1354.32	90.58	-	-	8.39
Approximate	26	3.08	-0.93	787.58	26.44	41.85	70.81	4.75

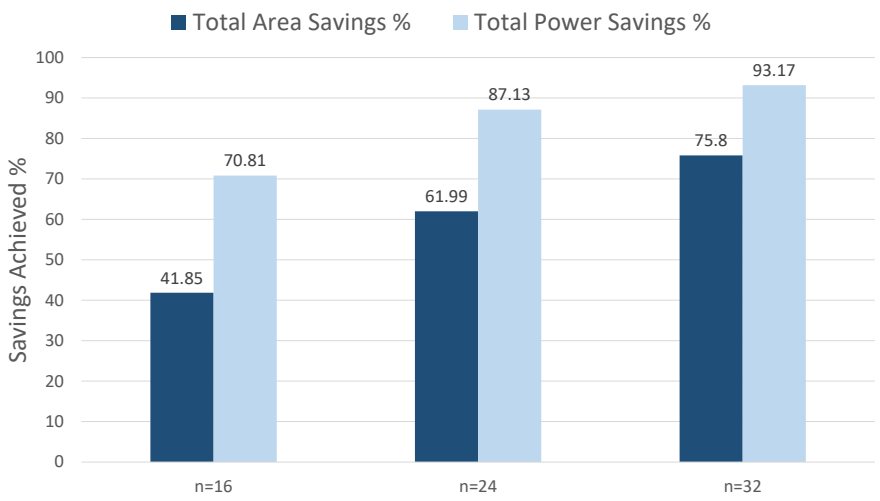


Figure 3.15: Area & power savings as a function of n.

Divider Application Results

In this subsection, we deploy the approximate divider based on our proposed methodology within three applications from the image processing domain. The evaluated applications are image change detection, JPEG compression and foreground extraction. We chose these applications as they have inherent tolerance to error and utilize division as part of their computational pipeline.

In change detection, two input images are compared and the output is an image of similar size highlighting the coordinates of changes. In a basic implementation of change detection, pixels of one images are divide by the respective pixels of the other image after pre-processing. A threshold is then used to classify each pixel to "changed" or "not-changed". We use images from a change detection dataset publicly available [92].

The results are visualized in Figure 3.16. For both sample sets, Figures 3.16.a and Figures 3.16.b show the input images, while Figures 3.16.c and Figures 3.16.d show the results for accurately and approximately ($k = 8$) computed results. As shown, the dif-

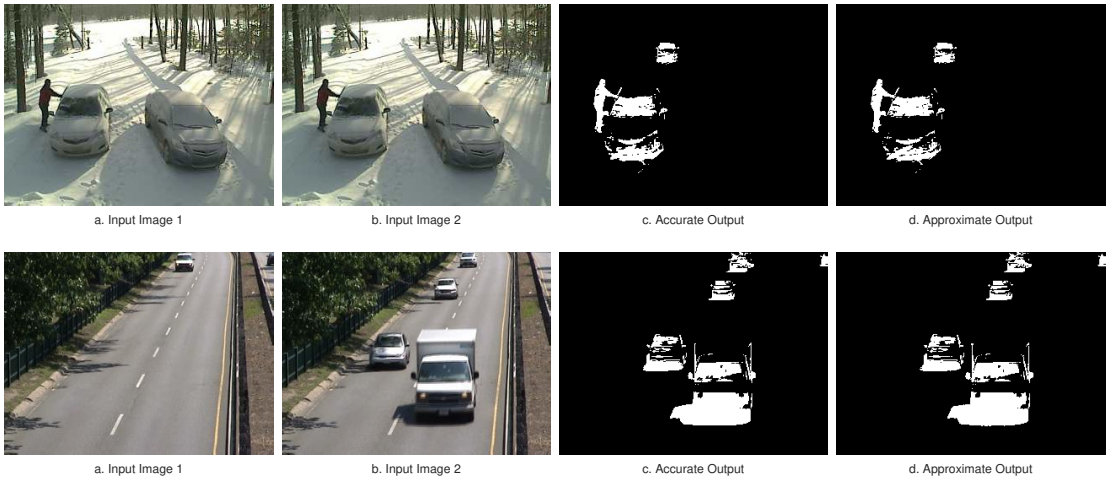


Figure 3.16: Change detection results for two sets of input images. (a) Input image 1; (b) Input image 2; (c) Detected using accurate divider; (d) Detected using approximate divider. PSNRs of 25.78 dB and 26.76 dB for driveway and highway input sets, respectively.

ference in the output image is not noticeable with PSNRs of 25.78 dB and 26.76 dB for driveway and highway input sets in reference to the accurately computed output image.

For our second application, we evaluate a JPEG compression algorithm. The proposed approximate divider replaces the accurate dividers used in the quantization step. More specifically, the output of the discrete cosine transform (DCT) is mapped to 16-bits while the quantization divisor is mapped to 8-bits. Figure 3.17 shows the JPEG output using

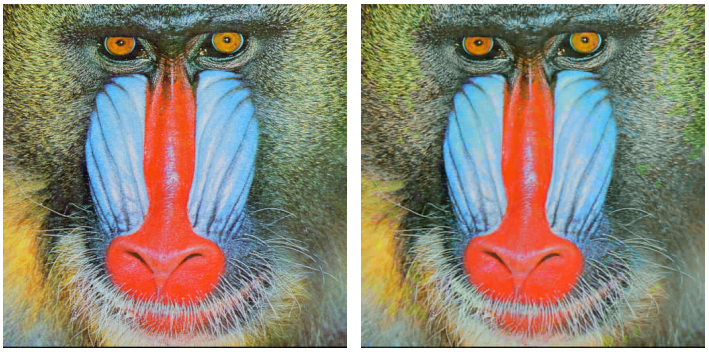


Figure 3.17: JPEG compression using accurate and approximate dividers. (a) Compressed image using accurate divider; (b) Compressed image using Approximate divider. $PSNR = 24.82$ dB.

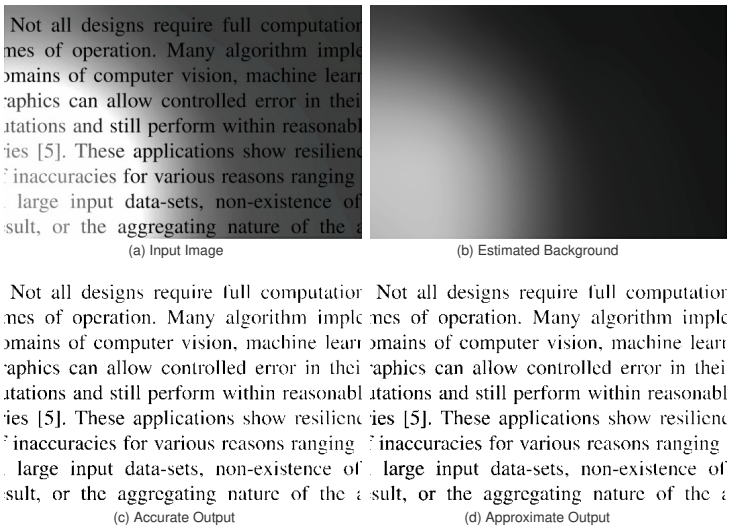


Figure 3.18: Foreground extraction using accurate and approximate dividers. (a) Input image; (b) Estimated background image; (c) Enhanced image using accurate divider; (d) Enhanced image using approximate divider. $PSNR = 23.96$ dB.

$k = 8$ for the approximate divider module. The similarity of the output images again highlights the sufficiency of the quality of service.

Finally, we evaluate our divider when utilized in a foreground extraction algorithm. Foreground extraction is commonly used as a pre-processing step for images with uneven illumination. In this application the input image is divided by an estimated background image effectively removing the background. The input images as well as the accurate and approximate ($k = 8$) results are showcased in Figure 3.18. Again, as observed, the approximate divider generates satisfactory results when compared to the accurate output.

Finally, the hardware metrics of all the applications for both the accurate and approximate counterparts are summarized in Table 3.9. Compared to the implementations utilizing accurate dividers, the designs with approximate counterparts offer significant savings in both design area and power consumption, ranging from 14% to 75%.

Table 3.9: Design area and power savings for application implementations.

Application	Accurate Design		Approximate Design		Savings	
	area (um^2)	comb. power (mW)	area (um^2)	comb. power (mW)	area (%)	power (%)
Change Detection	17361	1.69	10328	0.41	40.51	75.86
JPEG Compression	1291474	9.10	1102510	6.44	14.63	29.23
Foreground Extraction	9193	0.63	7511	0.23	18.30	64.02

3.4 Conclusions

In this chapter, we described our proposed dynamic arithmetic approximation scheme based on truncation, where the design automatically zooms in on the most important bits of each operand and does the computation accurately over a subset of each input. We further proposed to devise approximate arithmetic with zero-centered (unbiased) error distribution to further improve the accuracy. We explored and evaluated the performance of the approximate methodology on two resource heavy arithmetic, namely multiplier and divider. As demonstrated, the methodology delivers significant savings in hardware metrics such as design area and power consumption while introducing tolerable errors for both standalone and in application use.

Chapter 4

Approximate Synthesis using Boolean Matrix Factorization

4.1 Introduction

This chapter seeks to devise a new direction for approximate Boolean-level circuit synthesis. Our inspiration comes from Non-Negative Matrix Factorization (NNMF), which is a factorization technique that factors a non-negative matrix into two non-negative matrices [49]. The non-negativity constraints on the factorization arise in physical domains, such as computer vision and document clustering [94]. Recent advances in the mathematical community extends NNMF techniques to Boolean matrices, where matrix operations are carried in $GF(2)$ (Galois field of two elements), such that multiplications are performed using logical AND, and additions are performed using logical OR (for Boolean semi-ring implementations) and logical XOR (for Boolean field implementations) [58, 59]. The use of BMF as a technique for logic synthesis is a new direction in the

field, and we show that it provides a solid foundation for approximate logic synthesis. We summarize our contributions as follows.

- We propose a novel methodology for BMF-based Logic Approximate SYnthesiS (BLASYS) that is based on solid mathematical foundations, where Boolean Matrix Factorization (BMF) is used to generate approximate circuits with controllable trade-off between accuracy and circuit complexity.
- We modify existing BMF algorithms to incorporate the ability to work with different quality-of-results (QoR) functions, instead of the standard L_2 norm. Specifically, we introduce a weighted QoR function, as is the case in the binary representation.
- To scale the factorization method to large circuits, we propose a circuit decomposition method to break down a given circuit into manageable subcircuits with limited number of inputs and outputs. We propose a design-space exploration heuristic to order the subcircuits and identify a good sequence for generating their approximate variants. Our technique results in a very smooth trade-off between accuracy and circuit complexity.
- We implement our approach and test it on a number of application circuits that are typically used in approximate computing domains. We show that our approach is able to trade-off accuracy with circuit area and power consumption as evaluated by an industry-strength synthesis tool.

The organization of this chapter is as follows. We discuss the details of our proposed method in Section 4.2, where we describe the basic idea of using BMF algorithms to approximate logic circuits, and then show how to scale our proposed method to larger circuits. We provide comprehensive results of our method's performance together with a

comparison against a previous technique in Section 4.3. Finally, we summarize our main contributions and findings in Section 4.4.

4.2 Proposed Methodology

Non-negative matrix factorization (NNMF) is a factorization technique where a $k \times m$ matrix $\mathbf{M}$ is factored into two non-negative matrices: a $k \times f$ matrix $\mathbf{B}$, and an $f \times m$ matrix $\mathbf{C}$, such that $\mathbf{M} \approx \mathbf{BC}$ [49]. The non-negativity constraints on the factorization enables the utilization of the factorization algorithm in physical domains, such as computer vision and document clustering. NNMF essentially compresses the data in an approximate manner depending on the *factorization degree* (f) [94]. In the mathematical statistics community, the *factorization degree* determines the number of "feature" that are computed [58]. Therefore, f, clearly represents a trade-off between quality of factorization and storage amount. Recently, NNMF has been extended to Boolean matrices where elements of all matrices are restricted to Boolean values. In this case, multiplications can be performed using logical AND, and additions are performed using logical OR (for Boolean semi-ring implementations) and logical XOR (for Boolean field implementations) [58, 59]. Formally, the BMF problem can be formulated as,

$$\begin{aligned} \arg\min_{\mathbf{B},\mathbf{C}} |\mathbf{M} \oplus (\mathbf{B} \circ \mathbf{C})| &= \sum_{i,j} |m_{i,j} - (\mathbf{B} \circ \mathbf{C})_{i,j}|, \\ \ni \forall_{i,j} m_{i,j}, b_{i,j}, c_{i,j} &\in \{0, 1\} \end{aligned} \tag{4.1}$$

where $\oplus$ denotes the XOR operation, and $\mathbf{B} \circ \mathbf{C}$ denotes the Boolean matrix multiplication. BMF in an NP-Hard problem and current solutions are based on heuristics [58].

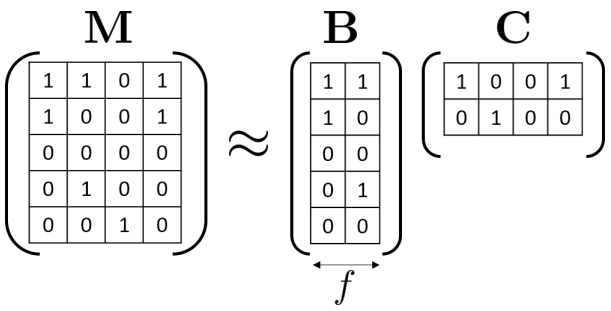


Figure 4.1: Boolean NNMF example.

A classic example of such an application is document characterization as utilized in knowledge discovery applications. More specifically, matrix $\mathbf{M}$ can represent a document-word relation, where each column of the matrix corresponds to a document and each row corresponds to a word in the dictionary. In such a representation, a 1 in index i, j of matrix $\mathbf{M}$ illustrates the presence of word i in document j. The BMF algorithm, therefore, divides the document-word matrix to a word-category matrix, matrix $\mathbf{B}$, where words are effectively mapped to categories, and a category-document matrix, matrix $\mathbf{C}$, where elements indicate the connection between the categories and the document. One benefit of the BMF is the reduction in storage requirements while maintaining the more critical information. Next, we will discuss how this methodology can be adopted for approximate circuit synthesis. Figure 4.1 provides an example of NNMF over GF(2).

4.2.1 Circuit Approximation using BMF

In our proposed approach, a multi-output logic circuit with k inputs and m outputs is first evaluated to generate its truth table. The truth table, represented by $\mathbf{M}$, is then given as input to a BMF algorithm together with the target factorization degree $1 \leq f < m$, to produce the two factor matrices $\mathbf{B}$ and $\mathbf{C}$. Matrix $\mathbf{B}$ is then given as the input truth table to a logic synthesis tool to generate a k input, f output circuit, which we refer to as the

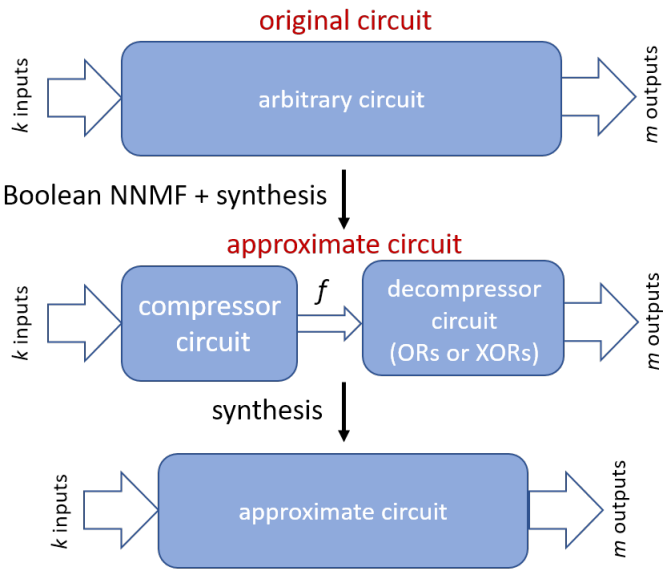


Figure 4.2: Generating approximate circuits using BMF.

compressor circuit. Note that the compressor matrix is simply the truth table of a circuit with the same number of inputs as the original circuit but with fewer (f to be exact) output signals. Therefore, it can easily be mapped to logic. These f outputs from the compressor circuit are then combined by the *decompressor circuit* according to the $\mathbf{C}$ matrix using a network of OR gates (for Boolean semi-ring implementations) or XOR gates (for Boolean field implementations), to generate the approximate m outputs. More specifically, a 1 in the (i, j) index of the decompressor matrix represents the existence of the f_i intermediate signal in the j-th output, effectively mapping each one to a OR (or XOR) gate. Using this methodology, any arbitrary circuit can be approximated by forcing the circuit to compress as much information as possible in f intermediate signals and then decompress them using simple OR (or XOR) gates. Figure 4.2 illustrates the proposed approach.

Figure 4.3 provides an illustrative example of a 4-input, 4-output arbitrary logic circuit. First, we present the original circuit with its truth table, and we synthesize it with Synopsys Design Compiler (DC) using 65 nm library. We then provide approximate variants for the circuit with $f = 1$, $f = 2$, and $f = 3$. We computed the truth tables for the compressor and

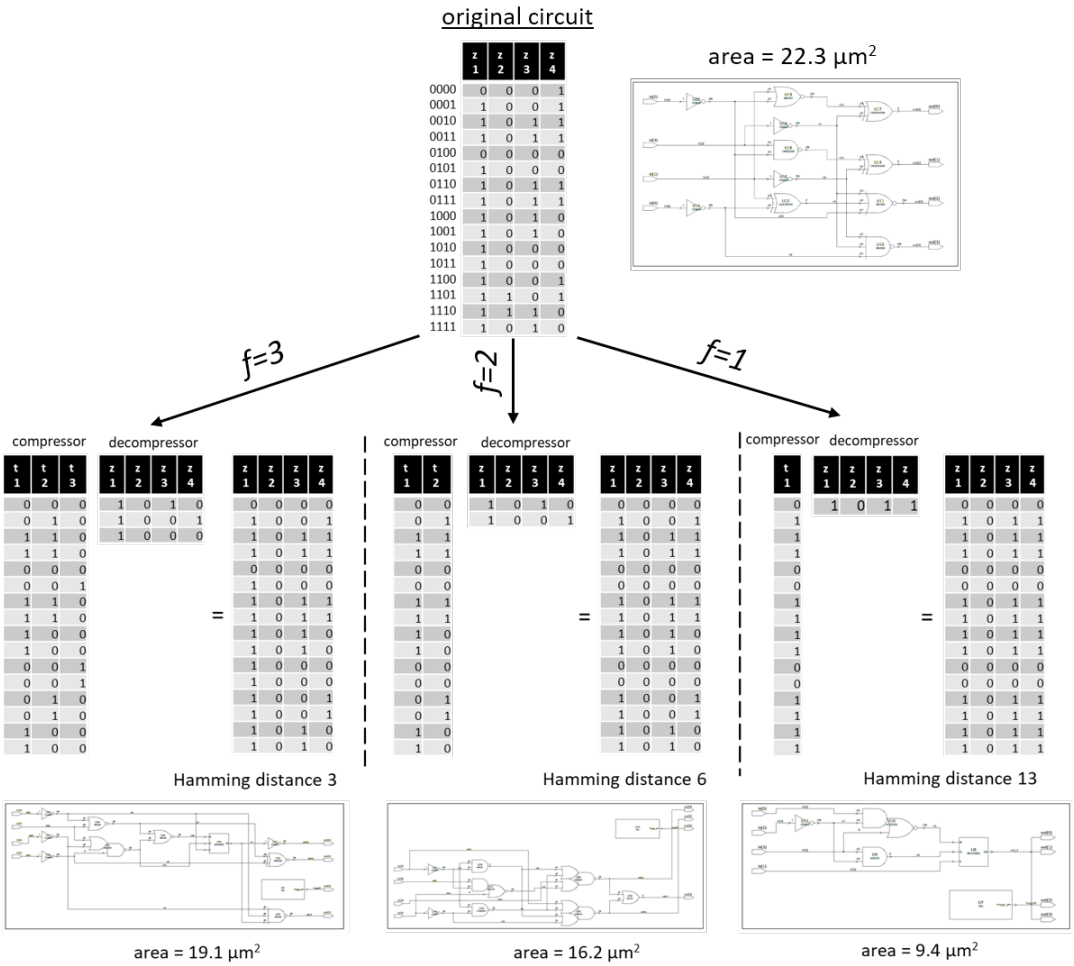


Figure 4.3: Results of proposed approximation method with various f on a simple circuit for illustration purposes. Circuits are synthesized using Synopsys DC using 65 nm technology library. A semi-ring implementation is used for Boolean NNMF.

decompressor using the ASSO NNMF algorithm [58, 59]. We provide both the quality of results as measured by the Hamming distance between the truth table of the original circuit and the approximate circuit as well as the design area reported by DC. For instance, when $f = 3$, we reduce the area of the circuit by 14.3%, while compromising the quality of results (QoR) by only 4.6% since the Hamming distance between the original and the approximate truth tables is equal to 3; that is out of the 64 entries in the truth table, only 3 entries flipped in the approximate circuit. With $f = 2$ and $f = 1$, we can reduce the area by 27.3% and 57.8% while compromising the QoR by 9.3% and 20.3% respectively.

Our approach leads to a new paradigm for creating approximate logic circuits in a controlled fashion that are based on solid mathematical foundations. There are two main challenges:

1. NNMF algorithms use the L_2 norm to measure the quality of factorization. For Boolean matrices, L_2 translates to Hamming distance. In addition, we need to identify methods to factorize for other QoR metrics that are relevant for approximate applications.

2. The basic idea is limited in scalability since the complexity of generating the truth table grows exponentially as a function of the number of circuit's inputs. Thus, we need to create factorization methods that can scale up for large circuits.

4.2.2 Factorization for Arbitrary QoR

The goal of the BMF algorithm is to minimize $||\mathbf{M}-\mathbf{BC}||_2$, which translates to Hamming distance in $GF(2)$. However, not all applications or circuits necessarily use this metric to assess QoR. For instance, in the case of circuit design, if an m bit signal is to be interpreted as an m bit number, Hamming distance is not really an accurate representation of the inaccuracies as mismatches in different bit indices contribute differently to the actual error.

To take into account the non-equal nature of bit significance, we propose to modify the NNMF algorithms in the literature to account for bit indices. More specifically, instead of minimizing $||\mathbf{M}-\mathbf{BC}||_2$, we propose minimizing $||(\mathbf{M}-\mathbf{BC})\mathbf{w}||_2$, where $\mathbf{w}$ is a constant weight vector. For example, if numerical difference is the target QoR, then the $\mathbf{w}$ vector will be based on powers-of-two (e.g., 8, 4, 2, 1) therefore reflecting the fact that different bit positions lead to different numerical weights. In this work, we modify the ASSO [59]

algorithm as such to penalize mismatches on the higher bit locations more than on the less significant bits. In Section 4.3.1, we demonstrate how such weighting scheme can improve the results compared to the uniformly weighted standard BMF algorithms.

4.2.3 Scaling Up for Large Circuits

Calculating the BMF is limited by computational complexity as one needs to generate the truth table for every possible input and state combination. Furthermore, BMF is a NP-hard problem, and most algorithms in the literature are heuristics [49, 58, 59]. We propose a simple approach to scale BMF calculations for larger circuits. The basic idea is to decompose a large circuit into a number of subcircuits each with a maximum of k inputs and m outputs as afforded by the runtime of the factorization algorithm and memory requirements. Note that this approach is reminiscent *but yet fundamentally different* than FPGA mapping algorithms, where the goal is to map a circuit into logic elements, each with limited number of inputs [14]. Our motivation for decomposition is different because (1) we are mapping to subcircuits purely to address computational complexity, and (2) we apply the BMF on the truth tables of the subcircuits, and then we synthesize the factored circuits into *any* target ASIC or FPGA technology. Instead of using classical k-cut algorithms, e.g. [14], we propose to use $k \times m$-cut algorithms (e.g., KL algorithm [55]) to identify subcircuits with a maximum input of k and maximum output of m. Note that k and m are design choices mostly determined by the runtime and memory budgets.

Decomposing a large circuit into subcircuits of size $k \times m$ requires changing the way we evaluate the QoR. In particular one cannot evaluate the QoR of a subcircuit in isolation from the rest of the circuit, since a small error in the output of the subcircuit can propagate leading to larger errors. Thus, instead of evaluating the QoR of an original subcircuit against its approximate version, we have to evaluate QoR of the entire circuit $Cir(s_i \rightarrow$

Algorithm 1: BLASYS: Boolean Level Approximate Circuit Synthesis

Input : Accurate Circuit $ACir$, Error Threshold
Output: Approximate Circuit Cir

```
1  subcircuits=Decompose input circuit using k × m decomposition
2  // Factorization profiling Phase
3  for each subcircuit s_i with m_i ≤ m outputs do
4      M=Construct truth table of s_i
5      // profile for every possible factorization degree
6      for f=1 to m_i-1 do
7          [B, C] = BMF(M, f)
8          T_{s_i,f}=Construct truth table of BC
9      end
10 end
11 // Circuit Space Exploration Phase
12 Cir=ACir;
13 Let f_i = m_i for all subcircuits s_i
14 while QoR(Cir) < threshold do
15     for each subcircuit s_i with f_i > 1 do
16         Cir'=Cir(s_i → T_{s_i,f_i-1})
17         Δerr_i = QoR(Cir') − QoR(Cir)
18     end
19     b = arg min_i(Δerr_i)
20     Cir = Cir(s_b → T_{s_b,f_b-1})
21     f_b = f_b − 1
22 end
23 Cir=Synthesize Best new Design
24 return Cir
```

T_{s_i,f_i}), where $Cir(s_i \rightarrow T_{s_i,f_i})$ represents the approximate circuit created by substituting an accurate subcircuit, s_i, with its approximate version, T_{s_i,f_i}, using a f_i factorization degree. As evaluating the entire circuit for all possible inputs is infeasible, we use Monte Carlo sampling to estimate the QoR of the approximate version of the entire circuit.

In addition, the order of processing the subcircuits and the target factorization degree for each subcircuit is an important consideration. We devise Algorithm 1 to gradually approximate the circuit as guided by circuit accuracy. After identifying the subcircuits (Line 1), the first stage of the algorithm (lines 3-10) calculates the *potential* approximate versions for each subcircuit under various factorization degrees. The next stage (lines 12-22) seeks to explore the space of potential approximate subcircuits to identify a good approximation order. Lines 15-18 assess the reduction in accuracy of the entire circuit if the degree of factorization of each subcircuit is decremented. The subcircuit that leads

to the smallest error is then chosen (line 19), and its more approximated version is then substituted in the main circuit (lines 20-21). The process is then repeated until the error is above the set threshold or all subcircuits are approximated to the highest degree possible.

4.3 Experimental Results

In this section we evaluate our proposed BMF based approximation methodology. Similar to previous work [88, 70], we consider a number of arithmetic circuits (adder and multiplier) and a number of application circuits that are amenable for approximate computing such as a multiply-accumulate circuit (MAC), a butterfly network (BUT), a sum of absolute differences (SAD) circuit and finite impulse response (FIR) circuit. Table 4.1 summarizes the characteristics of the evaluated applications. Here we also give the number of input and output pins, and the design metrics of the accurate design. To evaluate design area and power consumption, we use Synopsys design compiler with an industrial 65 nm technology library in typical processing corner.

For all our experiments, as discussed in 4.2.3, we first decompose each circuit to $k \times m$-cut subcircuits and then perform factorization. In our experiments we chose both $k = 10$ and $m = 10$. These numbers are simply chosen as they provide a balanced trade-off between truth table complexity and number of subcircuits. We use the modified ASSO algorithm for BMF [58, 59]. Further, for each subcircuit we perform a sweep on the factorization threshold in order to get the best accuracy. In order to evaluate the accuracy on the evaluated applications, we use a Monte Carlo simulation using one million randomly

Table 4.1: The list of benchmarks evaluated using the proposed NNMF methodology.

Name	Function	I/O	Accurate Design Metrics		
			Area (um^2)	Power (uW)	Delay (ns)
Adder32	32-bit Adder	64/33	320.8	81.1	3.23
Mult8	8-bit Multiplier	16/16	1731.6	263.5	2.03
BUT	Butterfly Structure	16/18	297.4	80.6	1.79
MAC	Multiply and Accumulate with 32-bit Accumulator	48/33	6013.1	470.5	2.36
SAD	sum of absolute difference	48/33	1446.5	195.1	2.43
FIR	4-Tap FIR Filter	64/16	8568.0	466.3	1.56

generated input test cases. We define average relative error as,

$$\text{Average Relative Error} = \frac{1}{N}\sum_{i=1}^{N}\frac{|R_i - R_i'|}{R_i}, \tag{4.2}$$

and average absolute error as,

$$\text{Average Absolute Error} = \frac{1}{N}\sum_{i=1}^{N}|R_i - R_i'|, \tag{4.3}$$

where N is the size of the test case, and R and R' are accurate and approximate results respectively.

Next, in the first subsection we show the impact of enabling arbitrary QoR functions, when compared to standard L_2 metric used in Boolean matrix factorization. In the second subsection, we show the trade-offs and Pareto Frontiers offered by our methodology for our applications. We also compare the results of our work to previous work.

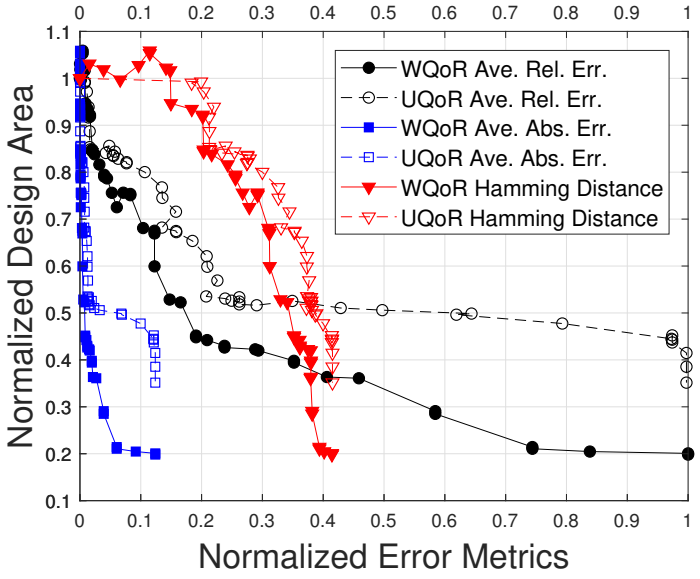


Figure 4.4: Comparison of the trade-offs offered using the proposed weighted QoR vs. the original factorization algorithm.

4.3.1 Evaluation of QoR Impact

As described in Section 4.2, we modify the Boolean NNMF factorization algorithm, ASSO in this case, to enable weighted cost functions, where a bit error on higher bit indices results in a higher penalty compared to disparities on the lower significance bits.

Figure 4.4 shows the accuracy vs. design area trade-offs offered for the approximate Mult8 design when comparing a factorization algorithm using standard L_2 QoR with uniform bit weighting against the proposed weighted QoR. We provide the trends in average relative error, normalized average absolute error, and the normalized Hamming distance. The results obtained from the weighted QoR (WQoR) are shown with solid lines while the dashed lines show the results for the original uniform algorithm (UQoR).

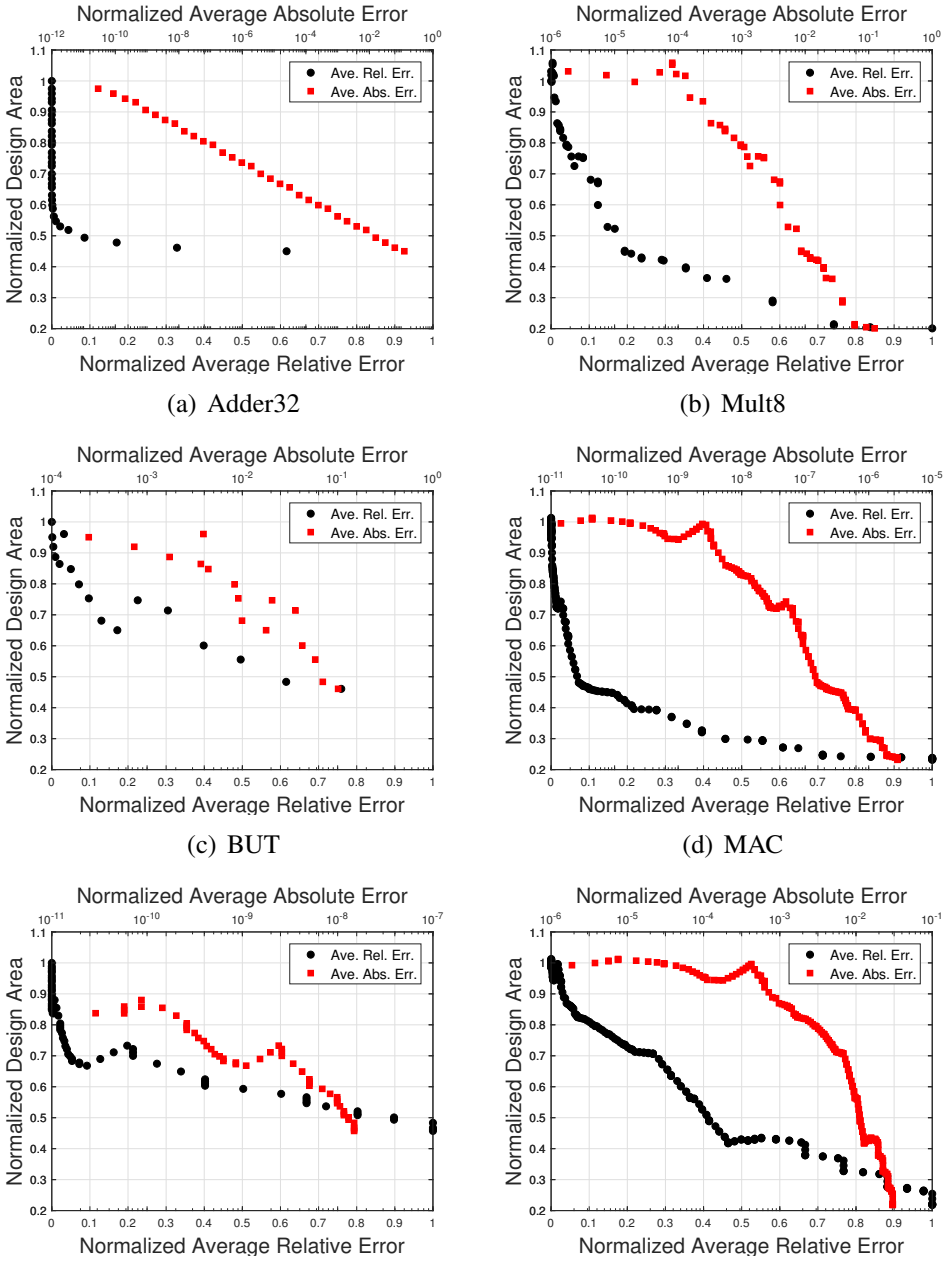
As shown in the figure, compared to the original algorithm, the weighted scheme provides consistent benefits in accuracy for the same design complexity for all three accuracy metrics. This result confirms the benefit of modifying the BMF algorithm to differentiate

among inaccuracies in different indices. Furthermore, this figure highlights the necessity of an algorithm guiding the approximation process in the right direction as suboptimal points are commonly encountered. Next, we evaluate the trade-offs offered for all of our application circuits using our heuristic design space exploration and compare our results against SALSA [88].

4.3.2 Application Results

As previously described, for each application, first the circuit is decomposed into subcircuits with reduced number of inputs and outputs. Then, for each subcircuit and various values of f, each subcircuit is approximated and the approximate characteristics are stored. Next, the heuristic proposed in Algorithm 1 iteratively approximates the subcircuits while assessing the impact on the whole circuit.

Figure 4.5 shows the trade-offs offered by BLASYS for each of our six benchmarks. In our experiments as the inner workings of accuracy among different blocks is more difficult to model, we simulate the whole circuit while modeling the design metric. More specifically, for design space exploration purposes, we assume the design metric, e.g. design area or power, of the large circuit is the sum of design metrics of the $k \times m$-cut subcircuits. For our experiments in order to simplify our design metric model, we use design area as it has less variation compared to power consumption when assembling the subcircuits into the larger circuit. Furthermore, our design area model is only a function of the subcircuit blocks being approximated, while registers and control paths are not considered. We plot the normalized combinational design area utilization as a function of average relative error (black plot and using the bottom x axis) and average normalized absolute error (red plot and using the top x axis). In the case of average absolute error, we normalize the values to the highest output possible. Further, to better show the trend, the



(a) Adder32 (b) Mult8

(c) BUT (d) MAC

(e) SAD (f) FIR

Figure 4.5: The trade-offs offered for each application. (a) Adder32, (b) Mult8, (c) BUT, (d) MAC, (e) SAD, and (f) FIR.

average absolute error is plotted in log scale.

As shown in the figure, the proposed methodology enables the designer to choose among a wide range of fine-grain trade-offs. Intuitively, our design space exploration heuristic aims to find the lowest error possible for a specific degree of approximation where the degree of approximation is incremented by one in each generation. This insight explains the smooth trend of trade-offs for larger circuits while the smaller circuits can change in performance significantly in one iteration. Furthermore, note that while reducing the number of intermediate signals (f) generally reduces the complexity of the circuit, there are cases where the number of literals in the logic representation for one output can increase. This phenomenon, explains the temporary increases in design area observed in some of the trends.

The overall runtime of the algorithm is dominated by the accuracy simulation of the intermediate points. Therefore, the runtime is dictated by the Monte Carlo sample size, the threshold set for accuracy, and the tool chain utilized. For example, in our experiments and in the case of the Adder32, the simulation takes about 11 Seconds (using 1 million samples) for each design point, while the BMF algorithm for all the subcircuits takes 0.35 seconds. The runtimes are reported for a workstation with core-i7 clocked at 4GHz and with 16GB of memory.

Table 4.2 summarizes all the design metrics of our 6 testcases and for two accuracy thresholds as synthesized at the end of the design space exploration. As shown in the table, significant reductions in design metrics are possible while insignificant errors are introduced to the circuit. Based on the application, benefits of approximately 8%-47% can be achieved for average relative errors of 5%.

We also compare our proposed methodology against the previous work SALSA [88].

Table 4.2: The hardware characteristics of the approximate testcases for two accuracy thresholds, namely 5% and 25%.

Design	Accuracy Thershold	Area Savings (%)	Power Savings (%)	Delay Reduction (%)
Adder32	5%	44.78	63.79	12.07
	25%	48.15	69.35	17.03
Mult8	5%	28.77	26.87	12.32
	25%	63.18	68.93	41.38
BUT	5%	7.87	11.25	2.23
	25%	26.39	35.14	5.03
MAC	5%	47.55	55.58	64.41
	25%	65.86	75.13	69.07
SAD	5%	32.80	41.47	69.14
	25%	38.08	55.07	77.37
FIR	5%	19.52	22.26	12.18
	25%	34.00	33.84	16.03

Table 4.3: The design area savings at error thresholds 5% and 25% for the applications evaluated with comparison to SALSA [88].

	Threshold 5% Area Savings (%)		Threshold 25% Area Savings (%)	
	BLASYS	SALSA	BLASYS	SALSA
Adder32	44.9	20.5	48.2	23.2
Mult8	28.8	1.8	63.2	8.9
BUT	7.9	5.0	26.4	24.7
MAC	47.6	1.7	65.9	8.2
SAD	32.8	3.3	38.1	15.8
FIR	19.5	3.2	34.0	15.8

Table 4.3 compares the results obtained using BLASYS against SALSA for given thresholds of 5% and 25%. As it can be seen from the table, in all cases, BLASYS delivers significant improvements in design area. We attribute the benefits to BLASYS' ability to approximate multiple outputs, up to m outputs, simultaneously, whereas SALSA approximates each output bit individually.

4.4 Conclusions

In this chapter we proposed a new direction for approximate logic synthesis using Boolean matrix factorization. Our proposed methodology, BLASYS, leads to a systematic approach to trade-off accuracy with circuit complexity. To scale our approach into larger circuits, we proposed a circuit decomposition heuristic together with a processing order for the subcircuits. Our algorithm results in a very smooth way to trade-off the complexity of entire large circuits with accuracy. We also investigated ways to incorporate different QoR metrics into the circuit factorization algorithm. Our experimental results show solid improvements over state-of-the-art techniques. On applications explored, BLASYS offers benefits ranging from 11% to 63% in power savings with an average error of 5% while offering up to 75% in power savings for a more relaxed threshold of 25%.

Chapter 5

Accuracy-Energy Trade-offs in Deep Neural Networks

5.1 Introduction

In this chapter, we discuss one of case studies considering application of approximate computing techniques on complex machine learning applications, more specifically energy-efficient deep learning. Deep neural networks (DNN) have provided state-of-the-art results in many different applications specifically related to computer vision and machine learning. One dominant feature of neural networks is their high demand in terms of memory and computational power thereby limiting solutions based on these networks to high power GPUs and data centers. In addition, such high demands have led to the investigation of low power application-specific integrated circuit (ASIC) accelerators where designers are free to assign dedicated resources to increase the throughput. However, memory accesses and data transfer overheads play an important part in the total computation time

and energy. When using accelerators, as a solution to data transfer overheads, specialized buffers have been introduced, thereby isolating the data transfer from the computation and enabling the memory subsystem to load the new data while the computation core is processing the previously loaded data.

Neural networks show inherent resilience to small and insignificant errors within their calculations, therefore making such networks ideal candidates for approximate computing techniques. In addition, the training nature of deep learning applications, where errors can be mitigated (or reduced) by relearning and fine tuning the parameters, further fits the approximate computing paradigm. In this light, techniques proposed by approximate computing, such as approximate arithmetic, are an attractive option to lower the power consumption and design complexity in neural networks accelerators. Here we pursue a unified approach to evaluation of deep learning applications using different bit precisions. We have previously published our work in [30]. More specifically, we summarize our contributions as follows.

- We perform a detailed evaluation of a broad range of networks precisions, from binary weights to single precision floating-points, as well as several points in between.
- We utilize learning techniques to improve the lost accuracy by taking advantage of the training process to increase the accuracy.
- We evaluate our designs for both accuracy and hardware specific metrics, such as design area, power consumption, and delay, and demonstrate the results on a Pareto Frontier, enabling better evaluation of the available trade-offs.
- Exploiting the benefits of lower precisions, we propose increasing the network size to compensate for accuracy degradation. Our results showcase low precision networks capable of achieving equivalent accuracy compared to smaller floating-point

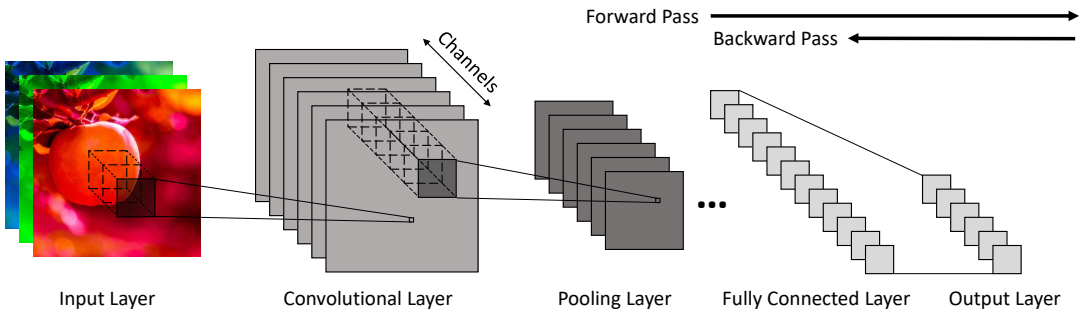


Figure 5.1: The structure of a typical deep neural networks.

networks while offering significant improvements in energy consumption and design area.

The rest of the chapter is organized as follows. In Section 5.2, we briefly summarize the basics of neural networks as well as providing a short review of related work. Then, in Section 5.3, we describe various precisions and network training techniques used in our evaluations and argue for increasing network size to recoup accuracy loss in lower precision networks. The results from our evaluations are provided in Section 5.4. Finally, in Section 5.5, we summarize our finding and the contributions of this chapter.

5.2 Background and Previous Work

Typically, deep neural networks are organized in layers where each layer is only connected to the layers immediately before and after it. Each layer gets its input from the previous layer and feeds it to the next layer after some layer-specific processing. Figure 5.1 shows the general structure and connectivity of the layers. As shown in the figure, each layer consists of several channels. Deep Neural networks, in general, consist of a combination of three main layer types: convolutional layers, pooling layers, and fully connected layers.

In typical neural networks the dominant portion of the computation is performed in the convolution layers and fully connected layers, while pooling layers simply down-sample the data. More specifically, channels in convolutional and fully connected layers are comprised of neuron units where each neuron performs a weighted sum of its inputs before feeding the result to a nonlinearity function. The intermediate values between layers are called feature maps, as they each abstract some structure in the input image. From a data perspective, neural networks operate on two main set of parameters: input data and intermediate feature maps, and network parameters (or weights). Since inputs and feature maps are treated similarly by the network, similar precisions are used for their representation. However, numerical precision of the network parameters can be changed independently of the input precision.

While the input data is assumed to be given for each network, the flexibility of neural networks arises from their ability to adapt their response to a specific input by training the network parameters. More specifically, use of neural networks comprises two phases, a training process during which the network parameters are learned, and a test phase which performs the inference and classification of the test data. In the training phase, neural networks usually utilize a backpropagation algorithm during which the classification error is propagated backwards using partial gradients. Network parameters are then updated using stochastic gradient descent. After training and in the test phase, the learned network is utilized in the forward phase to classify the test data. As discussed later, the main complexity of using lower precision in these networks arises due to the learning process.

The high demand of DNNs, in terms of complexity and energy consumption, has shifted attention to low-power accelerators. Many works have proposed implementing neural networks on FPGAs [25, 23], or as an ASIC accelerator [40, 83]. In all these works, different precisions have been utilized with little or no justification for the chosen bit-width.

Chen *et al.* proposed Eyeriss, a spatial architecture along with a dataflow aimed at minimizing the movement energy overhead using data reuse [11]. For their implementation, a 16-bit fixed-point precision is utilized. Sankaradas *et al.* empirically determine an acceptable precision for their application [75] and reduce the precision to 16-bit fixed-point for inputs and intermediate values while maintaining 20-bit precision for weights. A FPGA-based accelerator is proposed by Zhang *et al.*, where single precision floating-point arithmetic has been utilized [98]. While this work offers a brief comparison between resources required for floating-point and fixed-point arithmetic logic in FPGAs, no discussion of accuracy is provided. Chakradhar *et al.* proposed a configurable co-processor where input and output values are represented using 16 bits while intermediate values use 48 bits [7].

Many works have successfully integrated techniques commonly used in approximate computing to lower the computation and energy demands of neural networks. A feed-forward neural network is proposed by Kung *et al.*, where approximations are introduced to lower-impact synapses [43]. Venkataramani *et al.* proposed an approximate design where error-resilient neurons are replaced with lower-precision neurons and an incremental training process is used to compensate for some of the added error [89]. However, no specifications for the bit precision range used in the experiments are provided. Tann *et al.* proposed an incremental training process during which most of the network can be turned off to save power [82]. The neurons are then turned on during run-time if deemed necessary for correct classification. In this work, 32-bit floating-point representation was used.

As demonstrated by Chen *et al.* [10] and Tann *et al.* [82], the dominant portion of power and energy of hardware neural network accelerators is consumed in the memory subsystem, limiting the scope of arithmetic approximation. While many accelerators have been proposed using different bit-precisions, most of these studies have been ad-hoc and

give little to no explanation for choosing the specific precision. In this light, one particularly effective solution is reducing the bit-width required to represent the data. While the use of limited precision in neural networks has been proposed before [51, 15, 71], there exists no comprehensive exploration of their effect on energy consumption and computation time in reference to network accuracy. A recent publication by Gysel *et al.* provides an analysis of precision on network accuracy; however, the design parameters are not evaluated [28]. Our objective is to precisely quantify the effect of each numerical precision or quantization on all aspects of the networks focusing specially on hardware metrics.

5.3 Methodology

Here, first in Section 5.3.1, we discuss the range of precisions and quantizations considered in our evaluation. We also briefly discuss the network training techniques used to minimize the accuracy degradation due to the limited precision. Finally, in Section 5.3.2 we propose two expanded network architectures to compensate for the accuracy drop.

5.3.1 Evaluated Precisions and Train-Time Techniques

We consider a broad range of numerical precisions and quantizations, from 32-bit floating-point arithmetic to binary nets, as well as several precision points in between. We summarize them below:

Floating-Point Arithmetic

This is the most commonly used precision as it generates the state-of-the-art results in accuracy. However, floating-point arithmetic requires complicated circuitry for the computational logic such as adders and multipliers as well as large bit-width, necessitating ample memory usage. As a result, this precision is not suitable for low-power and embedded devices.

Fixed-Point Arithmetic

Fix-point arithmetic is less computationally demanding as it simplifies the logic by fixing the location of the radix point. This arithmetic also provides the flexibility of a wide range of accuracy-power trade-offs by changing the number of bits used in the representation. In this work, we evaluate 4-, 8-, 16- and 32-bit precisions. To improve accuracy, we allow a different radix point location between data and parameters [28]. However, we refrain from evaluating bit precisions that are not powers of 2 since they result in inefficient memory usage that might nullify the benefits.

Power-of-Two Quantization

Multipliers are the most demanding computational unit for neural networks. As proposed by Lin [51], limiting the weights to be in the form of 2^i, enables the network to replace expensive, frequent, and power-hungry multiplications with much smaller and less complex shifts. In our evaluations, we consider power of two quantization of the weights while representing the inputs with 16-bit fixed-point arithmetic.

Binary Representation

Recent work suggests that neural networks can generate acceptable results using just 1-bit weight representation [16]. While work by Courbariaux suggests binarizing activation between network layers, it does not binarize the input layer [15]. For this reason, our accelerator would still need to support multi-bit inputs. Thus, we evaluate the binary net using one bit for weights, while using 16-bit fixed-point representation for the inputs and feature maps.

Hardware Accelerator: For our experiments, we adopt a tile-based hardware accelerator similar to DianNao [10]. We implement 16 neuron processing units each with 16 synapses. Figure 5.2 shows our hardware implementation. As illustrated in the figure, three separate memory subsystems are used to store the intermediate values and outputs and buffer the inputs and weights. These subsystems are comprised of an SRAM buffer array, a DMA, and control logic responsible for ensuring that the data is loaded into the buffers and made available to the neural functional unit (NFU) at the appropriate clock cycle without additional latency. The NFU pipelines the computation into three stages, weight blocks (WB), adder tree, and non-linearity function. As shown in Figure 5.2, the weight blocks will be modified to accommodate for different precisions and quantizations as needed. In the case of binary precision, we merge the first two pipeline stages, effectively leading to a two stage NFU, in order to reduce the runtime. Furthermore, the size of all buffers and the control logic are modified according to the precision.

Training Time Techniques: We include a training phase in our experiments to enable the network to determine appropriate weights and adapt to the lower precision. Training processes, in nature, require high precision in order to converge to a good minima as the increments made to the parameters can be extremely small. On the other hand, if the network is made aware of its inference restrictions (in our case, the limited precision),

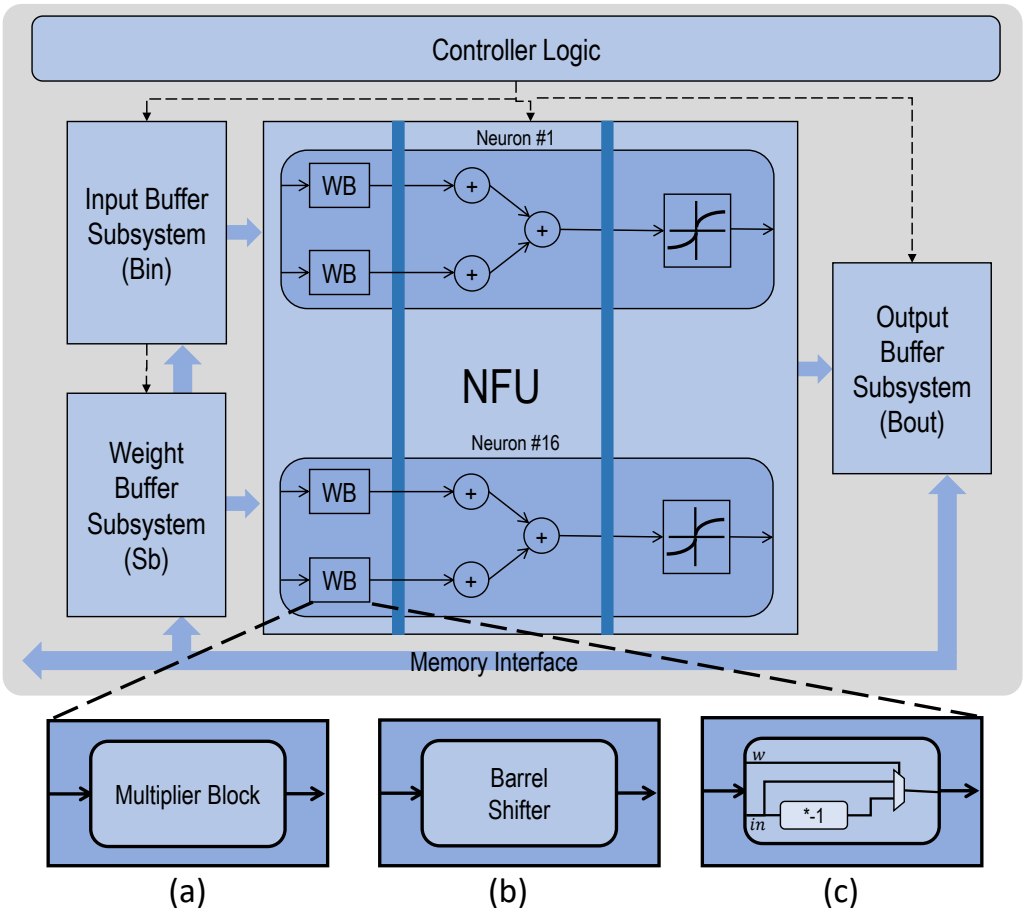


Figure 5.2: The hardware model used for our experiments. The first stage (WB) has different variants for (a) floating-point and fixed-point arithmetic, (b) powers of two quantization, and (c) binary network.

the training process can potentially compensate for some of the errors by fine-tuning the parameters and therefore improve the accuracy at no extra cost.

While the effects of reduced precision are analytically complicated to formulate as part of the training process [26], intuitive techniques can be utilized to improve the test phase accuracy. One approach proposed in [82] is to utilize a set of full precision weights, trained independently, as the starting point of a re-training process, in which the weights and inputs are restricted to the specified precision. This approach assumes that by using lower precisions, close to optimal performance can be obtained if a local search is performed around the optimal set of parameters as learned with full precision.

A second approach for improving the accuracy is to utilize weights with different precisions in different parts of the training process, as proposed by Courbariaux *et al.* [16]. They solve the zero-gradient issue by keeping two sets of weights: one in full precision and one in the selected lower precision. The network is then trained using the full precision values during backward propagation and parameter updates, while approximating and using low precision values for forward passes. This approach allows for the accumulation of small gradient updates to eventually cause incremental updates in the lower precision.

In our approach, we train all of the low precision networks using a combination of the first and second approaches. We initialize the parameters for lower precision training from the floating point counterpart. Once initialized, we train by keeping two sets of weights.

5.3.2 Expanded Network Architectures

While significant savings in power, area, and computation time can be achieved using lower precisions, even a small degradation in accuracy can prohibit their use in many applications. However, we observe that, due to the nature of neural networks, the benefits obtainable by using lower precisions are disproportionately larger than the resulting accuracy degradation. This opens a new and intriguing dimension, where the accuracy can be boosted by increasing the number of computations while still consuming less energy. We therefore propose increasing the number of operations by increasing the network size, as needed to maintain accuracy while spending significantly less for each operation.

In this light, in Section 6.4.2, we showcase two significantly larger networks and demonstrate that even by significantly increasing the size of the network, low precision can still result in improvements in energy consumption while eliminating the accuracy degradation. We discuss the specifications of the two larger networks in Section 6.4.2.

Table 5.1: Benchmark Networks Architecture Descriptions.

MNIST LeNet [47]	SVHN ConvNet [77]	CIFAR-10 ALEX [41]
28×28×1	32×32×3	32×32×3
conv 5×5×20	conv 5×5×16	conv 5×5×32
maxpool 2×2	maxpool 2×2	maxpool 3×3
conv 5×5×50	conv 7×7×512	conv 5×5×32
maxpool 2×2	maxpool 2×2	avgpool 3×3
innerproduct 500	innerproduct 20	conv 5×5×64
innerproduct 10	innerproduct 10	avgpool 3×3
		innerproduct 10

5.4 Experimental Results

5.4.1 Experimental Setup

We evaluate our designs both in terms of accuracy and design metrics (i.e., power, energy, memory requirements, design area). To measure accuracy, we adopt Ristretto [28], a Caffe-based framework [37] extended to simulate fixed-point operation. We modify Ristretto to accommodate our techniques, as needed. In different experiments, we ensure that all design parameters except for the bit precision are the same. This is critical to ensure the isolation of the effects of bit precision from any other factor.

We compile our designs using Synopsys Design Compiler using a 65 nm industry strength technology node library. We use a 250 MHz clock frequency and synthesize in nominal processing corner. We design our accelerator to have a zero timing slack for the full-precision accurate design. We confirm the functionality of our hardware implementation with extensive simulations. As before, we ensure that all other network parameters, including the frequency, are kept constant across different precision experiments.

Benchmarks: We consider three well-recognized neural network architectures utilized with three different datasets, MNIST [48] using the LeNet [47] architecture, SVHN using

Table 5.2: ALEX Larger Network Architecture Descriptions.

CIFAR-10	
ALEX+	ALEX++
32×32×3	32×32×3
conv 5×5×64	conv 3×3×64
maxpool 3×3	maxpool 2×2
conv 5×5×64	conv 3×3×128
avgpool 3×3	maxpool 2×2
conv 5×5×128	conv 3×3×256
avgpool 3×3	maxpool 2×2
innerproduct 10	innerproduct 512
	innerproduct 10

CONVnet [77], and CIFAR-10 [41] using the network described by Alex Krizhevsky [41] (here we refer to this network as ALEX). For all cases, we randomly select 10% of each classification category from the original test set as our validation set. To showcase the benefits from increasing the network size while using lower precision, we evaluate two networks as summarized in Table 5.2. Here, we focus on CIFAR-10 since MNIST and SVHN do not provide a large range in accuracy differences between various precisions and quantizations. As summarized in Table 5.2, we evaluate two larger variations of the ALEX network: (1) ALEX+, where the number of channels in each convolutional layer is doubled, and (2) ALEX++, where the number of channels is doubled when the feature size is halved [79]. As shown in Section 5.4.2, this methodology results in significant improvements in accuracy while still delivering significant savings in energy.

5.4.2 Results

Figure 5.3 shows the breakdown of power and area for the accelerator in the cases investigated. Values shown as (w, in) represent the number of bits required for representing weight and input values, respectively. Note, that these graphs do not reflect the power consumption of the main memory. As shown in the figure, the majority of the resources,

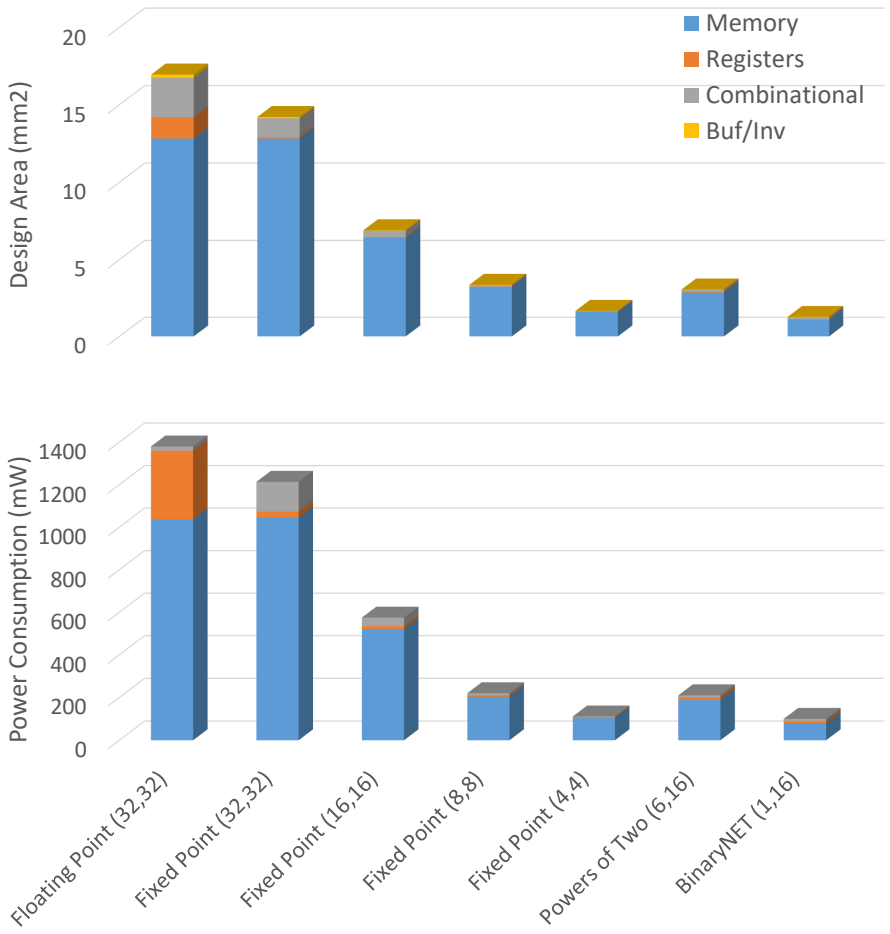


Figure 5.3: The breakdown of design area and power consumption using different precisions.

both in power and design area, are utilized in the memory buffers necessary for seamless operation of the computational logic. To be more specific, in our experiments, the buffers consume between 75%-93% of the total accelerator power, while using 76%-96% of the total design area. These values highlight the necessity of approximation approaches targeting the memory footprint.

Table 5.3 summarizes the design metrics of the accelerator for each of the numerical precisions considered. In order to maintain a fair comparison, we keep all the other parameters, such as the frequency, number of hardware neurons, etc., constant among different precisions. Changing the frequency or the accelerator parameters (other than precision)

Table 5.3: Design metrics of the evaluated numerical precisions and quantizations.

Precision (w, in)	Design Area (mm^2)	Power Cons. (mW)	Area Saving $(\%)$	Power Saving $(\%)$
Floating-Point (32,32)	16.74	1379.60	0	0
Fixed-Point (32,32)	14.13	1213.40	15.56	12.05
Fixed-Point (16,16)	6.88	574.75	58.92	58.34
Fixed-Point (8,8)	3.36	219.87	79.94	84.06
Fixed-Point (4,4)	1.66	111.17	90.07	91.94
Powers of Two (6,16)	3.05	209.91	81.78	84.78
Binary Net (1,16)	1.21	95.36	92.73	93.08

adds another dimension to the design space exploration which is out of the scope of our work.

We evaluate the accuracy of the networks, as well as energy requirements for processing each image for each of our benchmarks. Table 5.4 summarizes the results for MNIST and SVHN datasets. We were able to achieve little to no accuracy drop for all but one of the network precisions in the MNIST classification. In the case of SVHN, however, while keeping the network architecture constant, the 4-bit fixed-point and binary representations failed to converge. For SVHN dataset, for instance in the case of powers of two network, we are able to achieve more than 84% energy saving with an accuracy drop of approximately 2%. Note that as we keep the frequency constant the processing time per image changes very marginally among different precisions. Additional runtime savings can be achieved by increasing the frequency or changing the accelerator specification which is not explored in this work.

The reduction in precision also reduced the required memory capacity for network parameters, as well as the input data. We quantify our memory requirements for all the network architectures using different bit precisions. In our experiments, for the full-precision design, network parameters require approximately 1650KB, and 2150KB, and 350KB of memory for LeNet, CONVnet, and ALEX, respectively. Since there is a direct correla-

Table 5.4: The Accuracy, per image inference energy, and the energy savings achievable using each of the evaluated precisions. For each dataset, energy savings are in reference to the full-precision implementation.

	MNIST			SVHN		
Precision (w, in)	Class. Acc. (%)	Energy (uJ)	Energy Sav. (%)	Class. Acc. (%)	Energy (uJ)	Energy Sav. (%)
Floating-Point (32,32)	99.20	60.74	0	86.77	754.18	0
Fixed-Point (32,32)	99.22	52.93	12.86	86.78	663.01	12.09
Fixed-Point (16,16)	99.21	24.60	59.50	86.77	314.05	58.36
Fixed-Point (8,8)	99.22	8.86	85.41	84.03	120.14	84.07
Fixed-Point (4,4)	95.76	4.31	92.90	NA	NA	NA
Powers of Two (6,16)	99.14	8.42	86.13	84.85	114.70	84.79
Binary Net (1,16)	99.40	3.56	94.13	19.57	52.11	93.09

tion between bit precision and network memory requirements, the memory footprint of each network reduces from 2× to 32× for different bit precisions. Note, we do not utilize any of recent parameter encoding and compression techniques, and such techniques are orthogonal to our work.

As discussed in Section 5.3.2, we propose that a portion of the benefits from using low precision arithmetic can be exploited to boost the accuracy to match that of the floating point network while spending some portion of the energy savings by increasing the network size. Here, we showcase the benefits from our proposed methodology on CIFAR-10 dataset. The summary of the performances for the ALEX as well as the two larger networks (ALEX+ and ALEX++) is provided in Table 5.5. Here, we do not report the results for fixed-point (32,32) for ALEX+ and ALEX++ as its energy saving is not competitive compared to other precisions. Also, the fixed-point (4,4) fails to converge for all three networks on CIFAR-10 and the respective rows have been removed from the table. Furthermore, we find that the accuracy for fixed-point++ (8,8) is lower in comparison to the other networks with the same precision. We observe that for this network, there is a significant difference in the range of parameter and feature map values and as a result, 8 bits fails to capture the necessary range of the numbers.

Table 5.5: Network performance for different precision on CIFAR-10 dataset and using ALEX, ALEX+, and ALEX++. Energy savings are in reference to the ALEX full-precision implementation.

	CIFAR-10		
Precision (w, in)	Class. Acc. (%)	Energy (uJ)	Energy Sav. (%)
Floating-Point (32,32)	81.22	335.68	0
Fixed-Point (32,32)	79.71	293.90	12.45
Fixed-Point (16,16)	79.77	136.61	59.30
Fixed-Point+ (16,16)	81.86	491.32	1.5× More
Fixed-Point++ (16,16)	82.26	628.17	1.9× More
Fixed-Point (8,8)	77.99	49.22	85.34
Fixed-Point+ (8,8)	78.71	177.02	47.27
Fixed-Point++ (8,8)	75.03	226.32	32.59
Powers of Two (6,16)	77.03	46.77	86.07
Powers of Two+ (6,16)	77.34	168.21	49.89
Powers of Two++ (6,16)	81.26	215.05	35.93
Binary Net (1,16)	74.84	19.79	94.10
Binary Net+ (1,16)	77.91	71.18	78.80
Binary Net++ (1,16)	80.52	91.00	72.89

As shown in the table, lower precision networks can outperform the baseline design in accuracy while still delivering savings in terms of energy. The parameter memory requirements for the full-precision networks are roughly 350KB, 1250KB, and 9400KB for ALEX, ALEX+, and ALEX++ respectively. As discussed previously, the memory footprint reduces linearly with parameter precision when reducing the precision.

The available trade-offs in terms of accuracy and energy using different precisions and expanded networks are plotted in Figure 5.4 for the CIFAR-10 testbench. The figure highlights the previous argument that a wide range of power and energy savings are possible using different precisions while maintaining acceptable accuracy. Further, when operating in low precision/quantization, a portion of the obtained energy benefits can be re-appropriated to recoup the lost accuracy by increasing the network size. As shown in the Figure 5.4, this methodology can eliminate the accuracy drop (for example in the case of Power of Two++ (6,16)) while still delivering energy savings of 35.93%. The figure highlights that larger networks with lower precision can dominate the full-precision

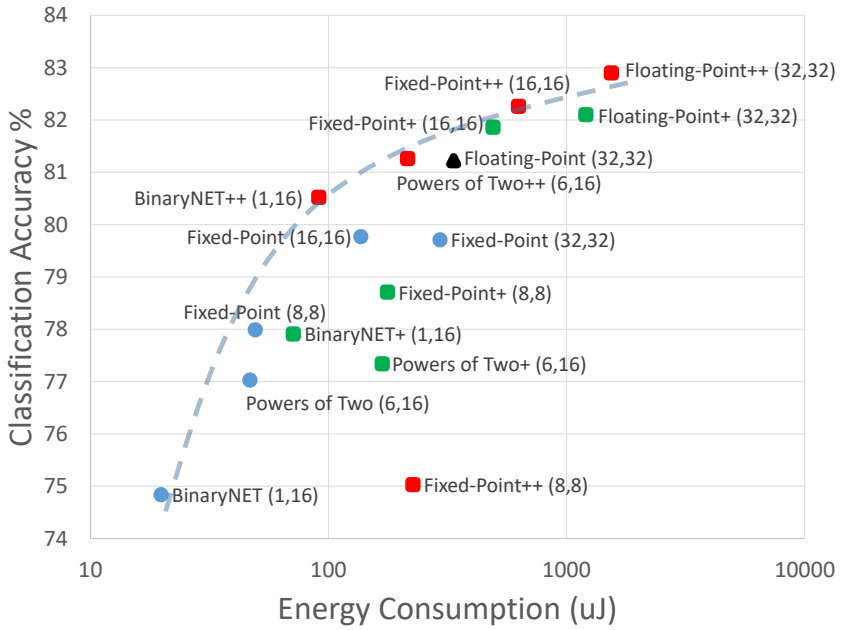


Figure 5.4: The Pareto Frontier plot of the evaluated design point for CIFAR-10 testcase. The x axis is plotted in logarithmic scale to cover the energy range of all the designs. Here, the black point indicates the initial full-precision design, the blue points indicate the lower precision points, while the red and green points show the results from the larger networks.

baseline design in both accuracy and energy requirements.

5.5 Conclusions

In this chapter, we perform an analysis of numerical precisions and quantizations in neural networks. We evaluate a broad range of numerical approximations in terms of accuracy, as well as design metrics such as area, power consumption, and energy requirements. We study floating-point arithmetic, different precisions of fixed-point arithmetic, quantizations of weights to be of powers of two, and finally binary nets where the weights are limited to one bit values. We also demonstrate that lower-precision, larger networks can be utilized which outperform the smaller full-precision counterparts in both energy and accuracy.

Chapter 6

Accuracy-Energy Trade-offs in Iris Recognition

6.1 Introduction

In this chapter, we provide our second case study for end-to-end approximate systems. We observe that biometric security is another ideal candidate field for the application of approximate computing techniques. Biometric security applications include: finger printing, iris scanners and face identifications to name a few. These applications are ideal for two main reasons: (1) the biometrics are data rich, (2) they operate on noisy inputs as captured from a sensor, and (3) the difference in biometric signatures of different individuals are large, and as a result signatures from the same individual are considered equivalent even if there are minor differences in them. For instance, the industry standards for iris scanning, e.g., ISO/IEC 19794-6, consider an iris encoding, which is represented by 2048 bits, as high quality even if the quality drops by 25% compared to the ideal case, because there

is 1 in 13 billion chance to have a Hamming distance less than 25% between the irises of two different individuals [17].

In this work, we propose an end-to-end biometric security system with an approximate SW/HW pipeline. Using an iris recognition application we showcase how approximate computing methodologies can be effectively implemented on an end-to-end system that is widely deployed. This Chapter has previously been published in [33]. More specifically, this chapter makes the following contributions.

- We propose biometric security systems as a novel direction where approximate computing techniques can be readily applied, and we showcase the benefits of such approximations on an iris scanning system, which is a prime method for biometric identification.
- We develop an end-to-end accurate system with approximate SW/HW processing pipelining for iris scanning systems. The complex pipeline consists of four major algorithms, where we identified in total eight knobs to trade-off accuracy of intermediate computations with runtime and energy consumption. We show that while controlled inaccuracies are added in the pipeline, the end encoding outcomes follow the guidelines set in the industry for guaranteed security.
- To explore the design space of the approximate knobs, we utilize a design exploration methodology enabling us to navigate the non-convex design space toward optimal points.
- We fully implement our methodology on an FPGA-based SoC using a camera with infrared sensor as input. We evaluate the performance of our system using both standard datasets and live feeds directly from the camera. We demonstrate that significant benefits can be achieved on an accurate end-to-end system using approx-

imate pipeline. We report benefits of up to 48× in runtime while maintaining 100% accuracy on the datasets and live feed.

The rest of this chapter is organized as follows. In Section 6.2, we provide a background of iris recognition and its pipeline. In Section 6.3, we describe our SW/HW proposed methodology, our approximation knobs and our design space exploration methodology. We discuss our experimental setup, as well as our experimental results in Section 6.4. Finally, Section 6.5 summarizes our results and the contribution of this chapter.

6.2 Background

Employment of iris recognition systems as a biometric identification system has grown exponentially in the security application domain. These systems can be utilized for identification purposes, where one instance is compared against the whole database, or for verification purposes, where one instance is compared against one reference. While many other biometric features sets have been proposed for identification purposes, iris recognition provides a wide gap between the inter-class and intra-class. In addition, compared to other biometric features such as person's face, which vary over time or due to different poses, the iris is independent of age or external factors [17].

Figure 6.1 shows the flowchart of commercial iris recognition systems. These systems use a pipeline consisting of four major components that takes input images from a camera and produce as output the 2048 bit encoding of the iris.

1. At the front-end, a *camera* with an infrared sensitive sensor captures multiple frames of an iris illuminated with infrared LEDs as given in Figure 6.1.a.

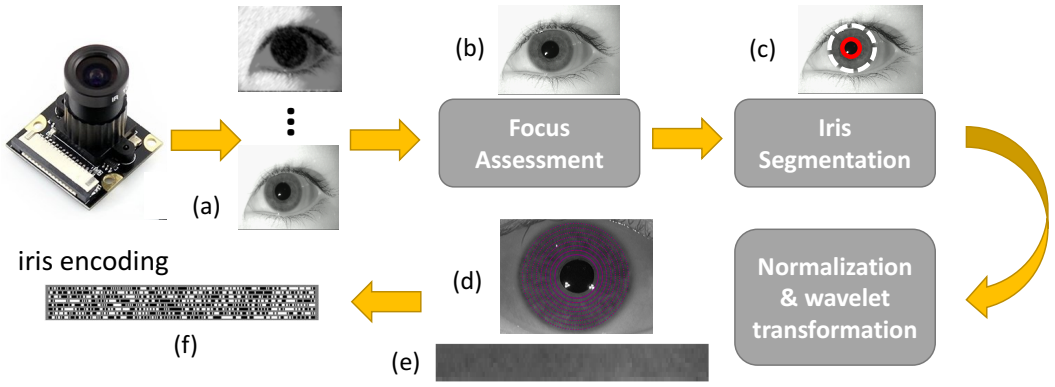


Figure 6.1: The components of an iris recognition system.

2. The *focus assessment* stage assesses the focus of the captured frames and picks the sharpest image for subsequent processing as illustrated in Figure 6.1.b. The degree of focus for each frame is computed using a convolutional kernel that calculates the energy of the images as described in [17].

3. Next, the iris image is *segmented* and the center points for the iris and the pupil as well as their radii are computed as illustrated in Figure 6.1.c. Here, solutions based on integrodifferential algorithm [17] and circular hough transform (CHT) [84] have been proposed. In this work, the integrodifferential algorithm is utilized. In integrodifferential algorithm, the whole image is first scanned for candidate pixels, and each candidate pixel is then evaluated using a circular differential methodology while the radius is changed from $Rmin$ to $Rmax$. Next, the candidate pixel with the maximum value is passed to a fine-grain search where a small window around the candidate is evaluated in a similar manner, resulting in the iris coordinates. Finally, and in a similar step, a small window around the iris center point is investigated for best match for pupil.

4. The output of the segmentation algorithm is then fed to the *normalization* algorithm where the iris pixels are subsampled and organized in a Cartesian coordination system. This is achieved by simply spacing the angular resolution and the radial resolution equally, based on the segmentation results. Figure 6.1.d and 6.1.e show the

subsampling process and the resulting 2-D output of the normalization respectively. Finally, the normalized pixels are *encoded* into 2048 bits using a 2-D Gabor demodulation [17] as shown in Figure 6.1.f. These 2048 bits form the signature of the iris.

6.3 Methodology

Our goal is to *minimize the response time* to the user from the time of image capture to the encoding of the iris, under the *constraints* that (1) the encoding is accurate by industry standards, and (2) the resultant SoC can fit within the given resources of our logic device.

We describe next our novel methodologies for approximate computing for iris scanning that achieve our goal. First, in Subsection 6.3.1, we explain our methodology for the partitioning of the pipeline flow between the hardware accelerators and the soft processor. Next, in Subsection 6.3.2, we summarize the approximation knobs that we have identified and explored in this work. Finally, Subsection 6.3.3 discusses the methodologies used to explore the design space created by the approximate knobs. Here, we discuss our novel methodology for approximations where a RNN is utilized to find the optimal knob settings. In this section, we also briefly discuss our methodology for computing the runtime and the accuracy of each design point.

6.3.1 Proposed SW/HW Partitioning

As an initial step, we profile the pipeline to measure the runtime of its different algorithmic components. To minimize the runtime, we synthesize the most computational

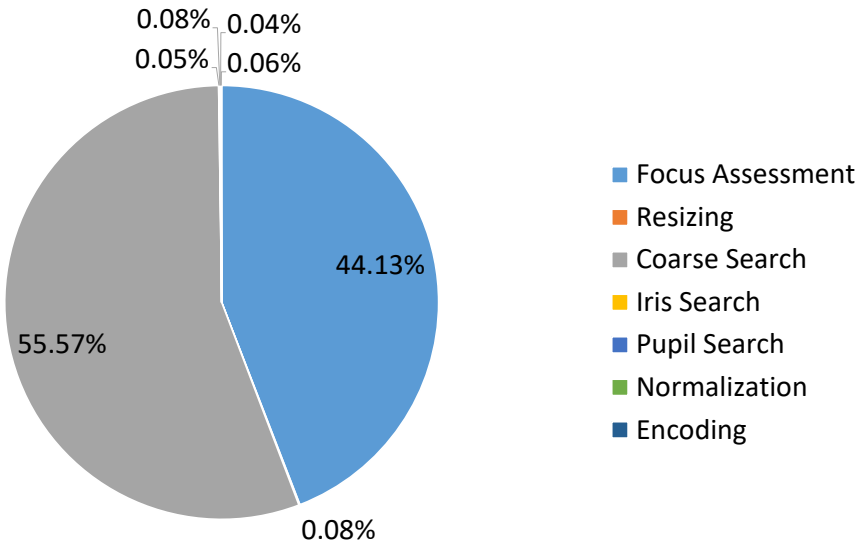


Figure 6.2: Percentage of runtime used by various algorithms in the iris scanning pipeline.

intensive modules into hardware accelerators. The pie chart in Figure 6.2 summarizes the runtime profiling results when running the flow in software. As shown in the graph, the overwhelming majority of the runtime is spent in the focus assessment and the segmentation components, while the encoding component takes less than 1% of the total runtime. Therefore, we choose to map these two components into custom hardware accelerators. Here, for the focus assessment, we deploy a full-fledged accelerator with complex buffering to take advantage of data locality. On the other hand, for the iris segmentation, we leave the control sequence in software and move the computationally intensive integrodifferential computation to a hardware accelerator. More specifically, candidate points are located in software and then passed to the accelerator, along with $Rmin$ and $Rmax$, for the computation of the maximum integrodifferential value.

Next, we provide some details on the accelerators implemented, based on the profiling, and show the final system.

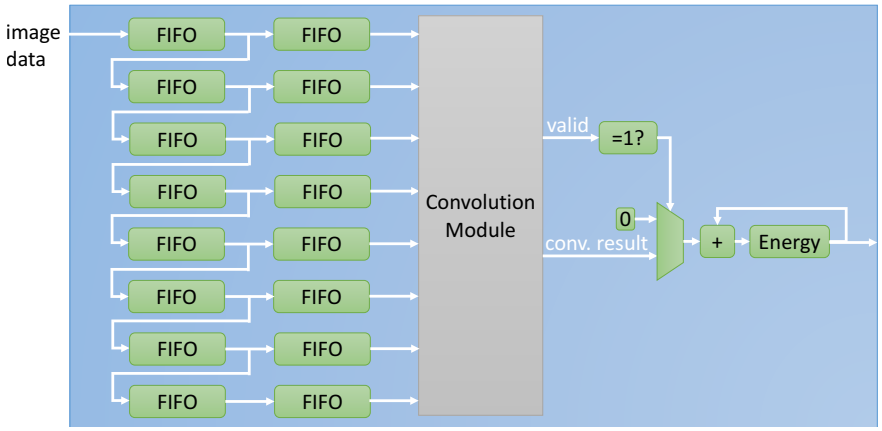


Figure 6.3: Architecture of the Convolution Accelerator used in the Focus Assessment Unit.

Focus Assessment Accelerator

The main operation in the focus assessment component is the energy computation, which is essentially a convolutional operation of the input image with a fixed kernel. Figure 6.3 shows the architecture of our hardware module for an 8×8 filter size. For each image, the data is fetched from a DDR3 memory and fed into a series of FIFO to maintain the consistency between the rows of the frame. At every clock cycle, the column of FIFO that receives the data from the data port is updated with a new pixel data, and the convolution module produces one output. For this reason, it requires an interface that exploits the second column of FIFO that feeds the module every clock cycle. The energy computed by the accelerator is 64 bits wide.

Integro-differential Accelerator

A fine-grain profiling of the pipeline flow algorithm timing shows that the runtime required for the segmentation part overwhelmingly dominates the total runtime. More specifically, the computations of the line integral consume more than 55% of the total runtime as shown

in Figure 6.2. For this reason, we develop a hardware accelerator for fast calculation of the Integro-differential operator. For HW/SW division of the flow, the software running on MicroBlaze scans and detects possible candidates for center points as suggested by Dougman *et al.* [17]. More specifically for each pixel in the image, the pixel is assumed to be candidate if it is a local minima with intensity value lower than a predefined threshold. If a pixel is deemed a candidate, its coordinates are passed to the accelerator and the start signal is asserted. The hardware accelerator directly interfaces the memory in order to speed up the movement of the data.

Figure 6.4 shows the structure for the accelerator implementation. The hardware unit consists of two main components, namely curve integral unit and radius iterator, and a gaussian filter unit as an intermediate component. For each input candidate center point, the radius iterator iterates through all the radii from $rmin$ to $rmax$, which are predefined for irises and pupils. The unit then passes the radius signal r to the curve integral unit together with required control signals. The curve integral unit then generates the line integral of the curve for a given center coordinates and radius. This component has direct access to the on board DDR3 memory component and can initiate memory requests as needed. The results from this unit are passed to the Gaussian filter unit where a shift register is used to the differential operations between two consecutive radii. The differential results are then smoothed using a 1D Gaussian filter as described by Dougman *et al.* [17]. The smoothed differential results are then passed to the Radius Iterator which stores and finds the maximum differential and the corresponding radius. The radius with the maximum differential results is then produced as output of the accelerator.

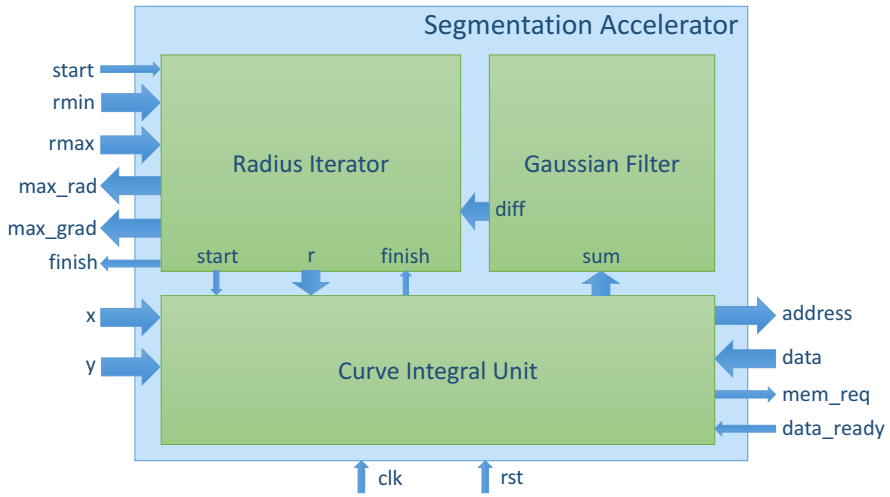


Figure 6.4: Architecture of the Accelerator Unit used in the Integro-differential Operator.

System Overview

Figure 6.5 shows the integration of various system components. Here a Microblaze soft-core processor along with multiple accelerators and interface units are implemented on the FPGA fabrics. The software and control portions of the iris recognition pipeline are executed on the processor while other computational demanding parts run on the accelerators. The processor communicates different commands to accelerator and interface units through an AXI-4 Lite interconnect. Data and instruction caches for the processor are implemented using the block RAM resources on the FPGA. For data read-write, the accelerators and interface components communicate with the on-board DDR3 memory via the AXI-4 interconnect and a memory controller block.

With HW/SW pipeline fully determined, we explore next the possible approximation "knobs" of the system, whether in software or hardware. The runtime speedups and the hardware resource utilization are then reported in Section 6.4.2.

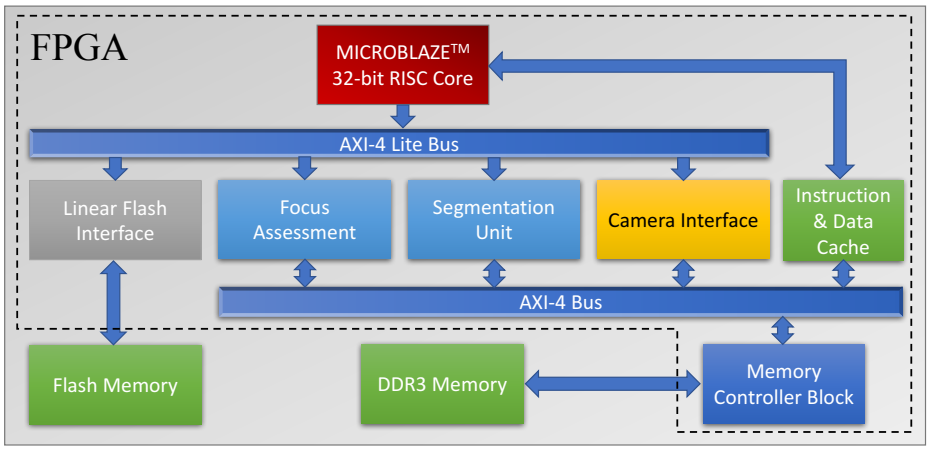


Figure 6.5: Overview of the FPGA SoCs and on board memory.

6.3.2 Proposed Approximation Knobs

We propose eight approximation *knobs* in our SoC iris scanning pipeline, where changing these knobs effectively trades accuracy for runtime benefits. Therefore, we refrain from introducing knobs that do not offer runtime benefits.

- Focus assessment: As the energy of each frame is computed as a convolution of a kernel filter with the image, one obvious accuracy trade-off is the *kernel size* of the filter. Furthermore, instead of computing the focus assessment on entire frames, we can only compute the energy for a subset of the image; i.e., a region of interest (*ROI*), to further reduce the runtime.

- Iris segmentation: We identify six more accuracy knobs in iris segmentation stage. Here, *Npoints* in the number of points used to compute the differential in each circle; *Scale* is the resizing factor used to reduce the segmentation image resolution; *Thresh* represents the threshold beyond which a point is considered to be dark enough to be a candidate; *Rmin* and *Rmax* define the range of radii for which the integration is performed; and *Search Window Size* gives the size of the window around which the local iris and pupil searches are performed.

Table 6.1: The list of approximation knobs evaluated in the design space exploration. Values in brackets show the possible values.

Pipeline Accelerator	Approximation Knobs [List of values]	Real. in
Focus Assessment	Kernel Size $[8, 6, 4]$	HW
	RoI $[1, 0.78, 0.50, 0.33, 0.20]$	SW
Iris Segmentation	Npoints $[600, 400, 200, 150, 100, 75, 50]$	SW
	Scale $[1, 0.85, 0.75, 0.50, 0.25, 0.20]$	SW
	Thresh $[102, 90, 77, 64, 51, 35, 26]$	SW
	Rmin $[45, 55, 65, 75, 85, 95, 100]$	HW
	Rmax $[180, 170, 160, 150, 140, 130, 120]$	HW
	Search Window Size $[11 \times 11, 7 \times 7, 3 \times 3]$	SW

- Normalization and Encoding: Lastly, in this step, as the design knobs providing accuracy vs. runtime trade-offs also affect the signature specification, in order to stay consistent, we refrain from introducing any knobs.

Table 6.1 summarizes the design knobs evaluated in our design space exploration as well as their possible value sets. Here, we also list the component in which each of these knobs are realized based on our SW/HW partitioning. The proposed design knobs result in approximate design space of $648,270$ design corners. Clearly a brute force exploration of the design space in not possible and a design space methodology is required for effective exploration. Section 6.3.3, shows our work in addressing this issue.

6.3.3 Design Space Exploration Methodology

Before we can explore the design space to identify the best settings, we need to quantify the accuracy and runtime of different sets of design knobs.

Table 6.2: The formulation used to model the runtime behavior as a function of the design knobs. Note that as we do not modify any knobs in the encoding components, we consider its runtime as a constant.

Pipeline Component	Runtime Model
Focus Assessment	$\propto RoI^2.KernelSize^2$
Iris Segmentation	$= R_{Coarse} + R_{Iris} + R_{Pupil}$
–Coarse Search	$\propto Scale^3.Thresh.Npoints.(Rmax - Rmin)$
–Iris Local Search	$\propto WindowSize^2.Npoints.(Rmax - Rmin)$
–Pupil Local Search	$\propto WindowSize^2.Npoints.r_{Iris}$
Total	$= R_{FocusAssessment} + R_{Resize} +$ $R_{Segmentation} + R_{Encoding}$

Accuracy and Runtime Measurements and Modeling

As evaluating all of the corners on hardware is infeasible, we simulate and formulate the accuracy and the runtime respectively. Since the accuracy performance of one component in the pipeline greatly affects the other components and the final results, we have to estimate the accuracy of a set of knobs using the entire flow through simulation. Thus, to compute the accuracy for each set of design knobs, we run a SW/HW co-simulation of the entire flow. Since such co-simulation can consume significant amount of time, we describe in the experimental results section techniques to speed it up. Unlike accuracy, the runtime of the pipeline flow can be readily decomposed. To that end, we mathematically model the runtime based on the input design knobs and our understanding of the complexity of the algorithm. A summary of our runtime models is shown in Table 6.2. With the runtime formulated, we profiled some training sets of design knobs to quantify the coefficients. We then verified our formulation on another set of knobs settings demonstrating a runtime modeling error of less than 5%. Note that this runtime merely guides the design space exploration and will not translate into inaccuracies.

Reinforcement Learning Based Design Space Exploration

As a powerful machine learning method, reinforcement learning enables an autonomous agent to make good decisions in its environment through trial-and-error using reward functions. As the agent learns, it starts to tune in on the best set of actions that maximizes its expected reward. Recently this approach has been used to determine the appropriate architectures for general deep neural network classifiers with promising performance [99]. This ability to learn and navigate through complex environment is a perfect fit for our problem. Our multi-objective function, which minimizes runtime and meets encoding error rate requirement, is non-convex. In addition, the input design space contains many dimensions, which makes it hard for traditional exploration methods such as gradient descent. Using reinforcement learning, we direct the agent, an RNN, to learn the best approximations for SW/HW knobs in the exponential design space. Unlike traditional feed-forward neural networks, RNNs have the ability to produce arbitrary-length output sequence. In this case, we encode SW/HW knobs to form a "sentence" [99]. The RNN is then used to predict the best sentence to optimize the systems metrics. During the learning process, the agent seeks to change its predicted sentence to maximize its reward function.

The words in the predicted sentence corresponds to approximations for the knobs. Using an accuracy co-simulation flow and runtime models, we then evaluate the impact of the approximations as shown in Figure 6.6. More details on the RNN training process and the reward function are available in our previously published work [33]. Algorithm 2 describes the overall design space exploration methodology.

After initialization, in an iterative manner, new values for the approximation knobs are proposed by the design space exploration methodology. Specifically, for each iteration, first the RNN is used to suggest a new set of approximation knob values. Once the knobs values are chosen, we compute the runtime using our models and the error rates through

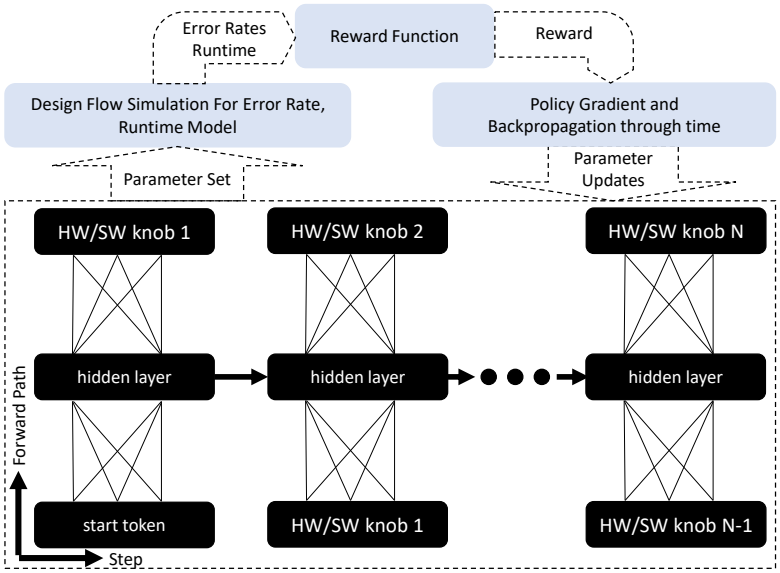


Figure 6.6: Design Space Exploration with Reinforcement Learning using RNN.

simulation. These characteristics are then utilized to update the RNN [33]. In lines 10-13, if the maximum error is lower than the threshold and the design offers the best runtime observed so far, current knobs values are stored as the output of algorithm.

While RNN can efficiently explores the breadth of the design space, it can takes significant amount of time to zoom in on local optimal point. To further improve the runtime benefits, we propose to perform a local search (LS) step using the best result obtained from the RNN. Here in one iteration, we change each parameter one step as long as it does not violate the accuracy.

6.4 Experimental Results

In this section we evaluate our methodology empirically, considering both runtime and accuracy performance. We compare our proposed methodology against a greedy and a local search based heuristics (similar approach to methodology proposed in recent work [69])

Algorithm 2: Design Space Exploration Methodology

```
// Design_Knobs:  List of tunable design knobs with
   their possible approximations.
// Approx_Knobs:  The approximations for design knobs.
```

Input : Design_Knobs
Output: Approx_Knobs

```
Initialize(RNN);
for iteration = 1 to N do
    In = START_TOKEN;
    knobs = RNN(In);
    rt_current = GetRuntime(knobs);
    err_rate = SimulateErr(knobs);
    avg_err = Average(err_rate);
    max_err = Max(err_rate);
    // Update RNN parameters [33]
    Update(RNN,rt_current, rt_best, avg_err, max_err);
    if max_err < threshold AND runtime < best_runtime then
        rt_best = rt_current;
        Approx_Knobs = knobs;
    end
end
return Approx_Knobs
```

and report its performance.

6.4.1 Experimental Setup

For our experiments we use a Spartan6 Xilinx development board interfaced to a 5 MP Videology camera with infrared optical filter and infrared LEDs for illumination. This 24B5.05USB3 camera features a unique 10 bit digital output port, which allows a direct interface to the raw image sensor pixel data. We also use an Agilent 34410A multimeter to monitor the current and measure power consumption accordingly. Figure 6.7 shows our camera and FPGA setup. We compile and synthesize all our results on the FPGA board and confirm correct functionality. While we test and run all our designs on the FPGA; for DSE, we co-simulate the accuracy of the SW/HW system and model the runtime. To

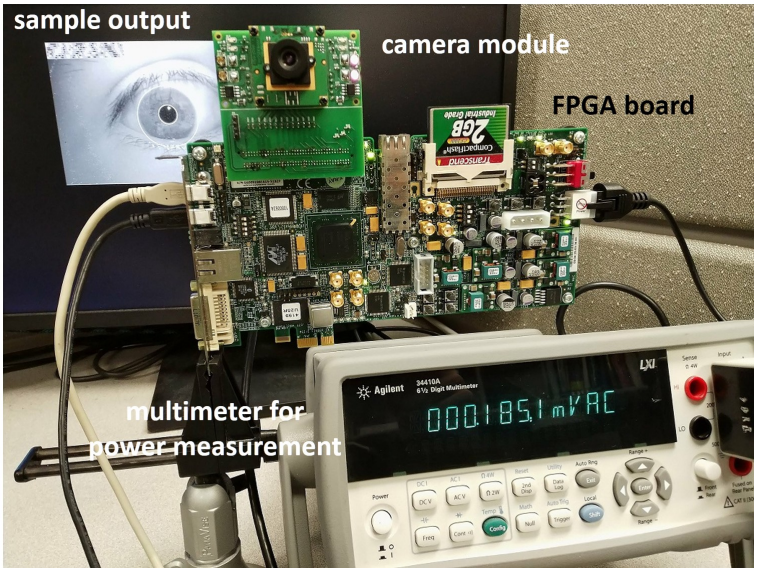


Figure 6.7: Our camera and FPGA board Setup.

speed up the computationally demanding co-simulation, we use Verilator [81] to compile the Verilog-based hardware accelerators into C-based simulators, and then use gcc to compile all the components in software. For runtime, we use the methodology described in Section 6.3 based on training samples of runtime from the actual board. To validate our performance and to compare against industry standards, we use two sources: (1) images from MMU open source dataset, and (2) live feeds captured from our camera system. Since MMU provides still images, we bypass the focus assessment module for their evaluation. For images captured from camera, we explore the complete flow where the energy of 10 frames are evaluated before the sharpest image is passed to the rest of the pipeline.

In order to assess accuracy, we cross validate the signature of each image in the dataset, using the approximate knob settings, against all of the signatures of the same eye in the repository when computed with full quality. To ensure 100% accuracy in the design, if at any point the maximum hamming distance error of any two images from the same eye goes above 0.35% the design is discarded. Using this threshold ensures a false positive rate of 1 in 133,000 [17]. Next section provides our results.

Table 6.3: The components chosen for hardware acceleration, the corresponding speedups, and the hardware utilization of each accelerator.

Pipeline Component	Speedup	HW utilization (%)
Focus Assessment	1234×	25.71
Iris Segmentation	6.8×	13.24
System		15.42

6.4.2 Results and Discussion

We first evaluate the benefits achieved from mapping part of the computing pipeline into custom hardware accelerators. As discussed in Section 6.3.1 and as commonly practiced, we map the algorithms that have the highest contribution to the total runtime of the iris scanning pipeline to hardware. Thus, we manually translate and map the focus assessment and the segmentation components into hardware accelerators, and leave the remaining parts of the flow to run as software on the MicroBlaze processor. After mapping these two components, the total logic utilization of our device reaches about 60%. Table 6.3 shows the speedups when focus assessment and segmentation are mapped to hardware accelerators, together with the logic utilization. These results are merely the benefits from hardware acceleration and use no approximations.

Next, we evaluate the performance of our proposed DSE methodology. Figure 6.8 shows the design points evaluate using our proposed RNN methodology. Here, the baseline point shows the average error of the SW/HW design without approximation. Using our reinforcement learning based DSE method, our design can achieve up to $42\times$ speedup while still maintaining the standard accuracy limits. Using the proposed RNN+LS method, the algorithm achieves $48\times$ in speedup. We stress that these speedups are due to approximate SW/HW processing and they are on top of the savings achieve due to the HW acceleration.

We also compare our proposed method against a greedy and a local search heuristics.

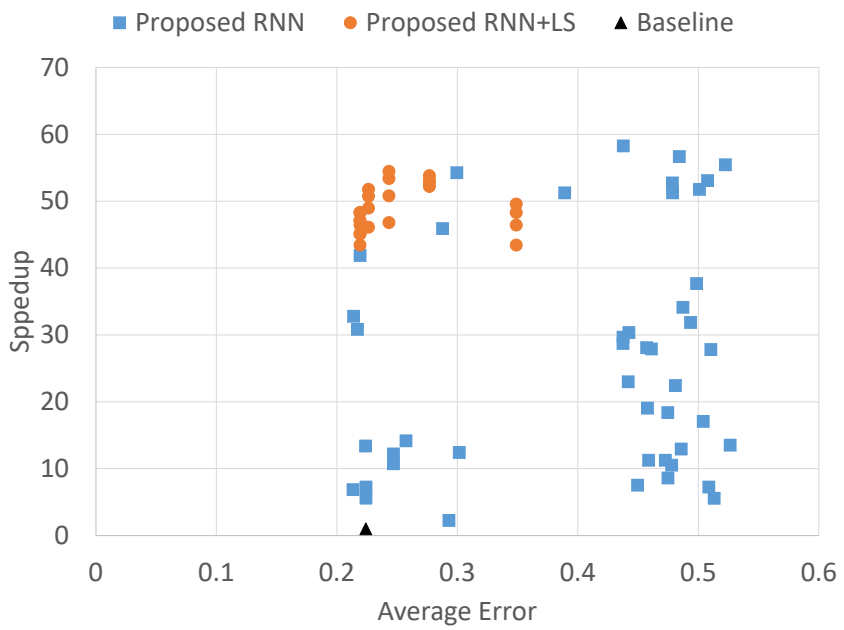


Figure 6.8: The design space explored using the proposed RNN, as well as RNN+LS.

In the greedy approach, starting from the beginning of the pipeline we approximate each design knob as much as possible before moving to the next one. The process is continued until all knobs have been visited. On the other hand, for the local search approach, we iteratively change each quality knob individually by one in order to generate a new design point. The design point with the highest *gradient*, where the gradient is defined by the ratio of runtime improvement over quality degradation, is chosen as the parent for the next iteration. Table 6.4 summarizes the trade-offs in speedups and effort offered by each design space methodology. Further, as shown in the table, the proposed method results in the highest speedup while requiring significantly less effort when compared to the gradient descent. On the other hand comparing to the greedy, higher speedups are provided. Note that here, the number of design points evaluated reported in the table directly indicates the efficiency of the DSE algorithm.

Finally, we evaluate the output design of the our methodology on the FPGA board to verify the runtime and the accuracy performance. We also compare the result of our methodology with pure software and software/hardware codesign methods. While for

Table 6.4: The comparison of the results using the proposed method ad compared against pure gradient descent and greedy methods.

Design	Speedup	Ave. Error (%)	Design Points Evaluated
RNN+LS	$48\times$	21.92	63
Greedy	$37\times$	21.44	21
Local Search	$42\times$	21.38	132

Table 6.5: The hardware characteristics of the end-to-end system.

Design	Runtime (s)	HW (%) Utilization	Energy (kJ)	Memory (MB)
Pure SW	2419.6	15.42	47.90	0.69
SW/HW	198.3	54.37	3.94	2.20
Approx. SW/HW	6.4	46.42	0.12	0.89

the previous experiments, we relied on the MMU benchmark dataset, for this experiment we use our camera system to capture iris images and run the complete flow. Here, we highlight the immense benefits of exploring the hardware/software codesign domain in conjugation with the approximate design knobs exploration. Table 6.5 summarizes the results. As shown in the table, significant benefits are achieved in terms of both runtime, and the total energy. Compared to a pure SW implementation, our approximate SW/HW pipeline is able to achieve a speedup of $378\times$ while meeting the industry standard accuracy requirements.

6.5 Conclusions

In this chapter, we identified biometric security as a potential application domain for approximate computing, and we illustrated this potential through a comprehensive case study on iris scanning system. We devised a SW/HW flow that processes input images from a live camera to produce the final encoding of the iris. Our pipeline flow consists of four major algorithms, where we identified eight design knobs that can trade-off design met-

rics with accuracy as measured by the Hamming distance of the final iris encoding to the golden encoding. We explored the approximate design space offered by these knobs and identified optimal points. Using a comprehensive SoC implementation in a FPGA-based system that receives inputs from a camera sensor with infrared optics, we showed that we can minimize runtime, which in an FPGA system directly maps to energy consumption, by 48$\times$ compared to exact SW/HW while fitting in the given FPGA resources and meeting the target accuracies of the iris encoding as set by industry standards.

Chapter 7

Summary of Dissertation and Potential Future Extensions

In this thesis, we proposed new techniques in different areas of hardware approximate computing paradigm. We extensively evaluated our results using simulations and synthesis using industry strength standard cell libraries. We further evaluated application specific approximations for two different case studies, namely deep neural networks, and iris recognition systems. In this chapter, we summarize the main contributions of this thesis, highlight the key results, and discuss the potential extension directions to this work.

7.1 Summary of Results

First, in Chapter 3, we discussed in detail, our proposed arithmetic approximation scheme based on unbiased dynamic truncation. Our methodology provides two significant advantages over previous work. First, our dynamic technique guarantees a tight maximum error

bound by always selecting the most important bits of each operand. Second, the resulting unbiased error distribution ensures the occurrence of both underestimation and overestimation errors, therefore allowing for reductions in errors through errors canceling each other out. Our methodology also benefits from a design time tunable parameters where a smooth accuracy-power trade-off is available based on the application quality requirements.

We explored and evaluated the performance of the approximate methodology on two resource heavy arithmetic, namely multiplier and divider. For each operation, we explore the design space both as a factor of the approximation factor, and the operator size. We show significant benefits for both operations where benefits of up to 71.45% in power savings are available for an average absolute error of 1.47% for the approximate multiplier. Similarly, in the case of the approximate divider our scheme delivers up to 70.81% in power benefits with an average absolute error of 3.08%. To further highlight the benefits, we evaluated our design using real world applications showing considerable benefits with insignificant reductions in QoR.

Next, in Chapter 4, and in approximate synthesis domain, we proposed and evaluated a new technique where approximations can be introduced in the truth table of any arbitrary logic using Boolean matrix factorization. Our approach, BLASYS, benefits from a *approximation knob* where different degrees of approximation can be exploited. We also investigate the benefits of expanding the QoR metrics as used in factorization algorithms to account for binary representation. BLASYS, therefore, provides a systemic approach to trade accuracy for benefits in design complexity. We also provided a heuristic optimization algorithm, based on local search, to navigate the design space of larger circuits. BLASYS offers benefits of up to 47.55% for an error threshold of 5%, showing significant improvements over the previous work.

In chapters 5 and 6 we investigated the applicability of the approximate computing concepts and techniques to real world applications by evaluating two case studies, namely deep neural networks, and iris recognition. First, in Chapter 5, we performed an numerical analysis evaluating a broad range of precisions and quantizations ranging from floating point to fixed point, powers-of-two and binary networks. We explored these precisions considering both the accuracy and the hardware metrics such as design area, power demands, and energy requirements. In this chapter, we also demonstrated that slightly larger networks utilizing lower-precision offer significantly more energy efficient models and outperform the full-precision counterparts in both accuracy and hardware cost. In our experiments and on CIFAR-10 dataset, we observe benefits of up to 36% in in energy benefits while outperforming the original full-precision model in classification accuracy.

Finally, Chapter 6 described the approximate methodologies introduced as part of an iris recognition pipeline. The main motive behind this case study is to showcase applications where due to the security nature of the application, accuracy does not appear as an expendable metric. However, through this comprehensive case study we highlight the applicability of approximate computing paradigm in biometric security applications without compromising the target accuracies required by the industry. Furthermore, we implement our techniques as part of an end-to-end SW/HW co-designed pipeline, where image are captured from an infrared camera and processed all the way to the final encoding result. In our pipeline, and for each component, we identify *design knobs* where the accuracy can be traded-off for complexity reduction. In order to explore, the design space of such knobs, we devised an exploration methodology based on reinforcement learning. We showed runtime benefits of up to 48$\times$ while remaining within the industry standards for acceptable accuracy.

7.2 Potential Research Extensions

In this thesis, we explored different approaches to approximate computing paradigm, as well as evaluating two complex computing systems as case studies. The ideas put forth in this work can be expanded in the following ways.

Since the publication of our results in approximate arithmetic design, many work have investigated different approaches [1, 53, 76, 97] as well as expanded on our work [4, 38]. However, one less explored implementation is the idea of implementing the same dynamic unbiased approach for other complex arithmetic systems, such as square root, etc. We also described our approach to design of approximate synthesis methodologies. The proposed approach opens many doors for investigation in logic synthesis. Future work can include improved techniques for BMF including literal aware approximations, direct incorporation of the QoR metric into the numerical optimization of the factorization algorithm, improved $k \times m$ circuit decomposition, a design space exploration for the cut size, and improved design space heuristics for fewer design point evaluations.

Similarly for our case studies, and in the case of neural networks, we plan on analytically investigating the correlations between network and datasets and their behavior in lower precision thereby effectively predicting the optimal lower accuracy and hardware metrics. Further, we plan to develop architectures which support multiple radix point locations between layers. Our exploration can also be expanded to other network topologies and architectures. Alternatively, precision requirements in training process can also be a potential addition to this work.

Finally, and in the case of the iris recognition, exploration of different biometric systems appears a compelling extension to the current work. Such biometric systems, include 3D facial recognition, and fingerprint recognition where similar approximate techniques

can readily be applied in the future work. BMF can also be applied to approximate the underlying accelerators of the proposed biometric security systems.

Bibliography

[1] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram. Dual-quality 4:2 compressors for utilizing in dynamic accuracy configurable multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(4):1352–1361, April 2017.

[2] Z. Babić, A. Avramović, and P. Bulić. An iterative logarithmic multiplier. *Microprocessors & Microsystems*, 35(1):23–33, 2011.

[3] Woongki Baek and Trishul M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '10, pages 198–209, New York, NY, USA, 2010. ACM.

[4] Benjamin Barrois. *Methods to Evaluate Accuracy-Energy Trade-Off in Operator-Level Approximate Computing*. Theses, Université de Rennes 1, December 2017.

[5] Vincent Camus, Jeremy Schlachter, and Christian Enz. A low-power carry cut-back approximate adder with fixed-point implementation and floating-point precision. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 127:1–127:6. ACM, 2016.

[6] S. T. Chakradhar and A. Raghunathan. Best-effort computing: Re-thinking parallel software and hardware. In *Design Automation Conference*, pages 865–870, June 2010.

[7] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. A dynamically configurable coprocessor for convolutional neural networks. In *ISCA*, pages 247–257, 2010.

[8] L. Chen, J. Han, W. Liu, and F. Lombardi. On the design of approximate restoring dividers for error-tolerant applications. *IEEE Transactions on Computers*, 65(8):2522–2533, Aug 2016.

[9] Linbin Chen, Jie Han, Weiqiang Liu, and Fabrizio Lombardi. Design of approximate unsigned integer non-restoring divider for inexact computing. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, pages 51–56, New York, NY, USA, 2015. ACM.

[10] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ASPLOS*, pages 269–284, 2014.

[11] Y. H. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ISCA*, pages 367–379, 2016.

[12] V.K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S.T. Chakradhar. Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency. In *47th ACM/IEEE Design Automation Conference (DAC)*, pages 555–560, 2010.

[13] Don Clark. Intel rechisels the tablet on moores law. https://blogs.wsj.com/digits/2015/07/16/intel-rechisels-the-tablet-on-moores-law/.

[14] Jason Cong and Yuzheng Ding. Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 13:1–12, 1994.

[15] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[16] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR*, abs/1511.00363, 2015.

[17] J. Daugman. How iris recognition works. In *IEEE Transactions on Circuits and Systems for Video Technology*, 2004.

[18] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, Oct 1974.

[19] K. Dev, S. Reda, I. Paul, W. Huang, and W. Burleson. Workload-aware power gating design and run-time management for massively parallel gpgpus. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 242–247, July 2016.

[20] Kapil Dev. New techniques for power-efficient cpu-gpu processors. *PhD Thesis, Brown University*, 2017.

[21] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Architecture support for disciplined approximate programming. *SIGPLAN Not.*, 47(4):301–312, March 2012.

[22] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 449–460, Washington, DC, USA, 2012. IEEE Computer Society.

[23] C. Farabet, C. Poulet, and Y. LeCun. An fpga-based stream processor for embedded real-time vision with convolutional networks. In *ICCV Workshops*, pages 878–885, 2009.

[24] J. George, B. Marr, B. E. S. Akgul, and K. V. Palem. Probabilistic arithmetic and energy efficient embedded signal processing. In *ACM Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 158–168, 2006.

[25] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello. A 240 g-ops/s mobile coprocessor for deep neural networks. In *CVPRW*, pages 696–701, 2014.

[26] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2013.

[27] Vaibhav Gupta, Debabrata Mohapatra, Sang Phill Park, Anand Raghunathan, and Kaushik Roy. Impact: Imprecise adders for low-power approximate computing. In *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design*, ISLPED '11, pages 409–414, Piscataway, NJ, USA, 2011. IEEE Press.

[28] Philipp Gysel. Ristretto: Hardware-oriented approximation of convolutional neural networks. *CoRR*, abs/1605.06402, 2016.

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

[30] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda. Understanding the impact of precision quantization on the accuracy and energy of neural networks.

In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1474–1479, March 2017.

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

[32] S. Hashemi, R. I. Bahar, and S. Reda. A low-power dynamic divider for approximate applications. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2016.

[33] S. Hashemi, H. Tann, F. Buttafuoco, and S. Reda. Approximate computing for biometrie security systems: A case study on iris scanning. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 319–324, March 2018.

[34] J. Hu and W. Qian. A new approximate adder with low relative error and correct sign calculation. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1449–1454, March 2015.

[35] M. Imani, A. Rahimi, and T. S. Rosing. Resistive configurable associative memory for approximate computing. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1327–1332, March 2016.

[36] Mohsen Imani, Daniel Peroni, and Tajana Rosing. Cfpu: Configurable floating point multiplier for energy-efficient computing. In *Design Automation Conference*, pages 76:1–76:6. ACM, 2017.

[37] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[38] Honglan Jiang, Cong Liu, Leibo Liu, Fabrizio Lombardi, and Jie Han. A review, classification, and comparative evaluation of approximate arithmetic circuits. *J. Emerg. Technol. Comput. Syst.*, 13(4):60:1–60:34, August 2017.

[39] A. B. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *DAC Design Automation Conference 2012*, pages 820–825, June 2012.

[40] J. Y. Kim, M. Kim, S. Lee, J. Oh, K. Kim, and H. J. Yoo. A 201.4 gops 496 mw real-time multi-object recognition processor with bio-inspired neural perception engine. *IEEE Journal of Solid-State Circuits*, 45(1):32–45, 2010.

[41] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[42] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *24th International Conference on VLSI Design*, pages 346–351, 2011.

[43] J. Kung, D. Kim, and S. Mukhopadhyay. A power-aware digital feedforward neural network platform with backpropagation driven approximate synapses. In *ISLPED*, pages 85–90, 2015.

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

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

[46] Mark S.K. Lau, Keck-Voon Ling, and Yun-Chung Chu. Energy-aware probabilistic multiplier: Design and analysis. In *ACM Proceedings of the International Confer-*

ence on Compilers, Architecture, and Synthesis for Embedded Systems, pages 281–290, 2009.

[47] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, 1998.

[48] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, 1998.

[49] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[50] S. Lee, L. K. John, and A. Gerstaluer. High-level synthesis of approximate hardware under joint precision and voltage scaling. In *Design, Automation and Test in Europe*, 2017.

[51] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. *CoRR*, abs/1510.03009, 2015.

[52] Cong Liu, Jie Han, and Fabrizio Lombardi. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, pages 95:1–95:4, 2014.

[53] 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 Transactions on Computers*, 66(8):1435–1441, Aug 2017.

[54] H.R. Mahdiani, A. Ahmadi, S.M. Fakhraie, and C. Lucas. Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(4):850–862, 2010.

[55] Osvaldo Martinello, Renato Perez Ribas Felipe Marque, and Andr?? Reis. Kl-cuts: A new approach for logic synthesis targeting multiple output blocks. In *Design Automation Test in Europe*, pages 777–782, 2010.

[56] Jiayuan Meng, S. Chakradhar, and A. Raghunathan. Best-effort parallel execution framework for recognition and mining applications. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–12, May 2009.

[57] Jin Miao, Andreas Gerstlauer, and Michael Orshansky. Approximate logic synthesis under general error magnitude and frequency constraints. In *Proceedings of the International Conference on Computer-Aided Design*, pages 779–786, 2013.

[58] P. Miettinen and J. Vreeken. Model order selection for boolean matrix factorization. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 51–59, 2011.

[59] P. Miettinen and J. Vreeken. Mdl4bmf: Minimum description length for boolean matrix factorization. *ACM Transactions on Knowledge Discovery from Data*, 8(4):18:1–31, 2014.

[60] John N. Mitchell. Computer multiplication and division using binary logarithms. *IRE Transactions on Electronic Computers*, EC-11(4):512–517, 1962.

[61] G. E. Moore. Cramming More Components onto Integrated Circuits. In *Electronics*, pages 114–117, 1965.

[62] S. Narayanamoorthy, H.A. Moghaddam, Zhenhong Liu, Taejoon Park, and Nam Sung Kim. Energy-efficient approximate multiplication for digital signal processing and classification applications. *IEEE Transactions on Very Large Scale Integration Systems*, 23(6):1180–1184, 2015.

[63] S. Hamid Nawab, Alan V. Oppenheim, Anantha P. Chandrakasan, Joseph M. Winograd, and Jeffrey T. Ludwig. Approximate signal processing. *Journal of VLSI signal processing systems for signal, image and video technology*, 15(1):177–200, Jan 1997.

[64] K. Nepal, S. Hashemi, C. Tann, R. I. Bahar, and S. Reda. Automated high-level generation of low-power approximate computing circuits. In *IEEE Transactions on Emerging Topics in Computing*, 2016.

[65] K. Nepal, Yueting Li, R.I. Bahar, and S. Reda. Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, pages 1–6, 2014.

[66] Online. http://www.itrs2.net/.

[67] K.V. Palem. Energy aware computing through probabilistic switching: a study of limits. *IEEE Transactions on Computers*, 54(9):1123–1137, 2005.

[68] Arnab Raha, Hrishikesh Jayakumar, Soubhagya Sutar, and Vijay Raghunathan. Quality-aware data allocation in approximate dram. In *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '15, pages 89–98, Piscataway, NJ, USA, 2015. IEEE Press.

[69] Arnab Raha and Vijay Raghunathan. Towards full-system energy-accuracy tradeoffs: A case study of an approximate smart camera system. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, pages 74:1–74:6, New York, NY, USA, 2017. ACM.

[70] Ashish Ranjan, Arnab Raha, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. Aslan: Synthesis of approximate sequential circuits. In *Design, Automation & Test in Europe Conference*, pages 1–6, 2014.

[71] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016.

[72] Pooja Roy, Rajarshi Ray, Chundong Wang, and Weng Fai Wong. Asac: Automatic sensitivity analysis for approximate computing. In *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, LCTES '14, pages 95–104, New York, NY, USA, 2014. ACM.

[73] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: Approximate data types for safe and general low-power computation. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 164–174, New York, NY, USA, 2011. ACM.

[74] Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. Approximate storage in solid-state memories. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 25–36, New York, NY, USA, 2013. ACM.

[75] Murugan Sankaradas, Venkata Jakkula, Srihari Cadambi, Srimat Chakradhar, Igor Durdanovic, Eric Cosatto, and Hans Peter Graf. A massively parallel coprocessor for convolutional neural networks. In *ASAP*, pages 53–60, 2009.

[76] J. Schlachter, V. Camus, K. V. Palem, and C. Enz. Design and applications of approximate circuits by gate-level pruning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(5):1694–1702, May 2017.

[77] Pierre Sermanet, Sandhya Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. In *ICPR*, pages 3288–3291, 2012.

[78] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 124–134, New York, NY, USA, 2011. ACM.

[79] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recog- nition. *ICLR*, abs/1607.05418, 2015.

[80] Amit Sinha, Alice Wang, and Anantha P. Chandrakasan. Algorithmic transforms for efficient energy scalable computation. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, ISLPED '00, pages 31–36, New York, NY, USA, 2000. ACM.

[81] Wilson Snyder. Verilator, the fastest free verilog hdl simulator [online]. https://www.veripool.org/wiki/verilator.

[82] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda. Runtime configurable deep neural networks for energy-accuracy trade-off. In *CODES+ISSS*, pages 1–10, 2016.

[83] Olivier Temam. A defect-tolerant accelerator for emerging high-performance applications. In *ISCA*, pages 356–367, 2012.

[84] Qi-Chuan Tian, Quan Pan, Yong-Mei Cheng, and Quan-Xue Gao. Fast algorithm and application of hough transform in iris segmentation. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, volume 7, pages 3977–3980 vol.7, Aug 2004.

[85] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi. Truncapp: A truncation-based approximate divider for energy efficient dsp applications. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1635–1638, March 2017.

[86] Zdenek Vasicek and Lukas Sekanina. Evolutionary design of complex approximate combinational circuits. *Genetic Programming and Evolvable Machines*, 17(2):169–192, Jun 2016.

[87] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Quality programmable vector processors for approximate computing. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, Dec 2013.

[88] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: Systematic logic synthesis of approximate circuits. In *49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 796–801, 2012.

[89] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. Axnn: Energy-efficient neuromorphic systems using approximate computing. In *ISLPED*, pages 27–32, 2014.

[90] Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Design, Automation and Test in Europe*, pages 1367–1372, 2013.

[91] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. Macaco: Modeling and analysis of circuits for approximate computing. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 667–673, 2011.

[92] Yi Wang, P.-M. Jodoin, F. Porikli, J. Konrad, Y. Benezeth, and P. Ishwar. Cdnet 2014: An expanded change detection benchmark dataset. In *Computer Vision and Pattern Recognition Workshops*, pages 393–400, 2014.

[93] L. Wu and C. C. Jong. A curve fitting approach for non-iterative divider design with accuracy and performance trade-off. In *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, pages 1–4, June 2015.

[94] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273, 2003.

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

[96] R. Zendegani, M. Kamal, A. Fayyazi, A. Afzali-Kusha, S. Safari, and M. Pedram. Seerad: A high speed yet energy-efficient rounding-based approximate divider. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1481–1484, March 2016.

[97] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi. Design-efficient approximate multiplication circuits through partial product perforation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(10):3105–3117, Oct 2016.

[98] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *FPGA*, pages 161–170, 2015.

[99] B. Zoph and Q. Le. Neural architecture search with reinforcement learning. *International Conference on Learning Representations*, 2017.