

A Digital System Design Methodology for Efficiency-Quality Tradeoffs Using Imprecise Hardware

A Dissertation

Presented to

the faculty of the School of Engineering and Applied Science
University of Virginia

In partial fulfillment

of the requirements for the degree of
Doctor of Philosophy in Computer Engineering

by

Jiawei Huang

May 2012

Approval Sheet

The dissertation is submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Engineering



Jiawei Huang (AUTHOR)

This dissertation has been read and approved by the examining committee:



Dr. John Lach (Advisor)



Dr. Gabriel Robins (Committee chair)



Dr. Scott Acton (Committee member)



Dr. Benton Calhoun (Committee member)

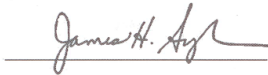


Dr. Joanne Dugan (Committee member)



Dr. Sudhanva Gurumurthi (Committee member)

Accepted for the School of Engineering and Applied Science:



Dean, School of Engineering and Applied Science

(May, 2012)

Not the quarry but the chase; not the laurel but the race.

— Gelett Burgess 1885

Research advisor
John Lach

Author
Jiawei Huang

A Digital System Design Methodology for Efficiency-Quality Tradeoffs Using Imprecise Hardware

Abstract

High power consumption has become a major barrier in the design of modern application-specific integrated circuits (ASICs). Recent studies have demonstrated the potential for significant power reduction in ICs by allowing errors to occur during computation. While most existing techniques for achieving this rely on voltage-overscaling (VOS), it is only one example in the vast design space of Imprecise Hardware (IHW), which is capable of converting relaxed quality requirements into higher implementation efficiency.

First we present the generalized concept of IHW, of which VOS is an example. A mathematical approach to IHW characterization is introduced to quantify error characteristics. We also present several novel IHW examples, including fidelity-compromising transformations at the algorithm level, imprecise adders and multipliers at the RTL level and the combined use of IHW and VOS.

Since IHW expands the design space of traditional HW by one dimension (quality), it is imperative to develop a fast and accurate quality evaluation method to efficiently explore

this space. We propose a static error estimation method that propagates the statistical distribution of data and errors through a network of arithmetic operations. It can be used to estimate the quality metrics of an IHW implementation without the need for simulation.

Finally, two methodologies for exploring efficiency-quality tradeoffs using IHW are presented. The first methodology is for applying fidelity-compromising transformations at the algorithmic level. The second methodology is for choosing IHW ALUs at the RTL level. They are fundamentally different from traditional circuit optimization methodologies in that they possess quality awareness and are capable of sacrificing quality for higher efficiency. Both methodologies can solve constraint-based and cost-function-based optimization problems. Experiments on real-world applications have shown that the proposed methodologies can achieve comparable results to exhaustive search but are orders of magnitude faster.

Acknowledgments

First of all, I would like to thank my adviser Prof. John Lach. It was under his years of teaching, training and guidance that I gradually became a more competent and independent researcher. During the lowest ebb of my research, when my conference submissions were rejected three times and I increasingly saw my project as hopeless, he allowed me to conduct an entirely different project of my choice for one semester. That semester witnessed my first publication and I returned to the original project with fully-regained confidence. After that I started to develop my own ideas around the original project and gradually formed the topic I wanted to pursue for my Ph.D.. In the meantime, I was awarded with an increasing number of conference publications as well as an increasing quality of those papers. During this period, John kept helping me distinguish my work from existing ones and focus my attention on the most original and significant contributions of my work. Without his help, I couldn't have completed this dissertation.

Secondly, I want to extend my gratitude to Prof. Gabriel Robins. To this day I still clearly remember our encounter on a flight from DC to Charlottesville. At the time I was yet to propose my Ph.D. topic and only had isolated ideas about what to pursue. Gabe listened to me talking about those ideas with absorption. When I was finished, he provided his vision about "Imprecise Hardware" and encouraged me to pursue it as my topic. Our conversation has since become a continuous source of inspiration to me.

Thirdly, I want to thank my other committee members Prof. Scott Acton, Prof. Benton Calhoun, Prof. Joanne Dugan, and Prof. Sudhanva Gurumurthi. Their constant interest and helpful feedback were important factors to maintain the quality of my work.

My colleagues and friends such as Michael Boyer, Jonathan Bolus, Shuai Che, Yu (Randolph) Yao, Jiajing Wang, Zhenyu (Jerry) Qi, Jiayuan Meng, Jeff Brantley, Shanshan

Chen and Curby Alexander have given me great support through the years both at work and at play. I really appreciate our friendship.

Finally, I want to thank my parents for their unconditional love. When I am down and weary, you are always there to comfort me and be on my side. I will always love you, mom and dad.

Contents

Title Page	i
Approval Sheet	ii
Abstract	iv
Acknowledgments	vi
Table of Contents	viii
List of Figures	xi
List of Tables	xiv
List of Acronyms	xvi
1 Introduction	1
1.1 Background and Motivation	1
1.2 Related Work on IHW	4
1.3 The Scope of This Dissertation	10
1.4 Major Contributions of This Dissertation	13
1.4.1 Design of Novel IHW	14
1.4.2 A Static Error Estimation Method	14
1.4.3 A Quality-Aware Efficiency Optimization Methodology	14
1.4.4 IHW Taxonomy	15
1.5 Organization	15
2 Imprecise Hardware (IHW)	17
2.1 Introduction	18
2.2 IHW Behavior	21
2.2.1 Based on I/O Mapping	21

2.2.2	Based on Error Characteristics	23
2.3	IHW Systems	26
2.4	Applications of IHW	28
2.5	IHW Taxonomy	30
2.6	Selected IHW Designs	31
2.6.1	Fidelity-Compromising Transformations [30]	32
2.6.2	Imprecise Adders and Multipliers	33
2.6.3	Implementing other elementary functions with CORDIC	45
2.6.4	Combining structural IHW and VOS	46
2.7	Conclusions	50
3	Static Error Estimation	51
3.1	Error as a Distribution	52
3.2	Interval Arithmetic and Affine Arithmetic	52
3.3	Modified Interval Arithmetic	55
3.4	Modified Affine Arithmetic	63
3.5	Error Propagation Model	66
3.6	Experimental Results	69
3.7	Conclusions	71
4	A Methodology for Efficiency-Quality Co-Optimization	73
4.1	For Fidelity-Compromising Transformations	73
4.1.1	Latency Evaluation in HDGs	78
4.1.2	Fidelity Evaluation in HDGs	80
4.1.3	Case Study on SRAD	80
4.2	For Imprecise Adders and Multipliers	87
4.2.1	Kernel-level Experimental Results	91
4.2.2	Application-level Experimental Results	97
4.3	Conclusions	101
5	Conclusion	102
5.1	Summary of Contributions	103

5.1.1	Imprecise Hardware	103
5.1.2	Static Error Estimation	104
5.1.3	A Methodology for Efficiency-Quality Optimization	104
5.2	Suggested Adoption of IHW	105
5.3	Future Work	106
5.3.1	Imprecise Hardware	106
5.3.2	Static Error Estimation	107
5.3.3	A Methodology for Efficiency-Quality Optimization	108
5.4	Concluding Remarks	109
A	Experimental Data	118
A.1	Raw data for building energy models	118
B	Publications related to this dissertation	124

List of Figures

2.1	Qualitative quality-efficiency curve of IHW. Both quality and efficiency metrics are normalized against the error-free operating point (quality=1, efficiency=1), which gives the highest quality and lowest efficiency.	19
2.2	Conceptual diagram of efficiency-quality tradeoff. IHW can convert relaxed quality requirements (e.g., error tolerance) into improved efficiency (e.g., power savings).	20
2.3	Probability Mass Function (PMF) examples.	24
2.4	IHW taxonomy.	30
2.5	16-bit Almost-Correct Adder structure with $K = 6$	36
2.6	Energy-delay space of ACA adder and KSA adder.	36
2.7	16-bit Modified Error-Tolerant Adder Type II structure with $BPB = 4, L = 3$	38
2.8	Energy-delay space of METAII adder and RCA adder.	39
2.9	Timing slack histograms of RCA and KSA.	40
2.10	Error frequency of various imprecise adders.	42
2.11	Error magnitude of various imprecise adders.	42
2.12	Comparison of energy-delay product (EDP) of CORDIC implemented with different IHW adders.	47
2.13	Structural IHW can be operated under deeper VOS than traditional HW, reducing power even further.	47

2.14	Quality-power curves of the IHW-VOS combination designs. The three numbers following the names of IHW adders refer to their design parameters. The format is $BPB-L-K$. When a parameter is not in use, it is set to zero.	49
3.1	Probability distribution of the sum of two uniformly-distributed variables. .	59
3.2	Estimated MIA of the sum of two independent uniform variables in $[-1, 1]$ using IA, MIA methods, and the theoretical MIA.	62
3.3	Error propagation model of an imprecise 2-operand operation.	66
3.4	Characterization of an IHW adder.	68
3.5	Error MIAs of DOT-PRODUCT and L2-NORM.	70
4.1	Hierarchical dependency graph.	74
4.2	Design flow with fidelity-compromising transformations.	76
4.3	Ultrasound images: (a) Perfect edges (b) Noisy image (c) SRAD output (d) Detected edges from SRAD output.	81
4.4	2D plots of various cost functions with two conditional transformation thresholds as variables. Representing cost function: (a) $latency/fidelity^{0.5}$ (b) $latency/fidelity$ (c) $latency/fidelity^2$. Letter “N” on the axes denotes the scenario of “never applying the transformation”, and Letter “A” means “always applying the transformation”. The circled squares denote the optimal design points that minimize the cost function.	85
4.5	Latency vs. fidelity curve of SRAD (with 90% fidelity point located). . . .	86
4.6	Fidelity vs. latency curve of SRAD.	87
4.7	Quality-constrained energy minimization flow.	89
4.8	Curve fitting for IHW energy models: (a) 64-bit ACA with variable K (b) 64-bit ACA-based multiplier with variable K (c) 64-bit METAIL with variable L ($BPB = 4, 8$) (d) 64-bit METAIL-based multiplier with variable L ($BPB = 4$) (e) 64-bit METAIL-based multiplier with variable L ($BPB = 8$) (f) 64-bit METAIL-based multiplier with variable L ($BPB = 16$).	93
4.9	Block diagram of our DOT-PRODUCT implementation.	94

4.10	Block diagram of our L2-NORM implementation.	94
4.11	Kernel-level energy-quality tradeoffs. Energy is calculated as the average energy consumed during one kernel operation. The design points for the precise designs are located far above all the imprecise design points, therefore are not shown on the plot. The energy of the precise design for the DOT-PRODUCT is 136.44pJ, and the energy of the precise design for the L2-NORM is 140.4pJ.	97
4.12	Application-level energy-quality tradeoffs. Energy is calculated as the average energy consumed during one kernel operation. The design points for the precise designs are located far to the right of all the imprecise design points, therefore are not shown on the plot. The energy of the precise design for the Leukocyte is 136.44pJ, and the energy of the precise designs for the K-means and SVM is 140.4pJ.	100

List of Tables

2.1	IHW examples by behavior.	23
2.2	IHW examples by error types.	25
2.3	An example of fidelity-compromising transformation based on mathematical approximation.	33
2.4	Probability of a random propagate chain exceeding K bits.	35
2.5	Average E/op and worst-case delay of 64-bit KSA and 64-bit ACA adders with $K = 16$	35
2.6	Average E/op and worst-case delay of 64-bit RCA and 64-bit METAIL adders with $BPB = 4$ and $L = 4$	39
2.7	Common imprecise adders and their error classifications.	43
2.8	Impact of anti-biasing on the error rate and magnitude of a METAIL adder.	44
2.9	Average energy per operation, critical path delay, and energy-delay product (EDP) of precise and imprecise ALUs.	45
2.10	Units tested under combined use of IHW and VOS (with critical path delays in brackets).	48
3.1	Percentage application runtime spent in computation kernels.	70
3.2	Speed and accuracy comparison between simulation and static estimation.	71
3.3	Overview of error estimation methods.	72
4.1	Principle component latencies in the SRAD algorithm [55].	82
4.2	Transformation library for SRAD case study.	83
4.3	Critical path latencies for different transformation combinations.	84

4.4	Tradeoff between quality of result and exploration effort for different neighborhood sizes.	85
4.5	Impact of transformations on system metrics (fidelity, latency, and cost function).	88
4.6	Kernel-level energy and quality (error rate) results.	95
4.7	Kernel-level energy and quality (mean error magnitude) results.	96
4.8	Number of designs points simulated.	100
A.1	Raw delay and power data of various adders for building energy models . .	118
A.2	Raw delay and power data of various multipliers for building energy models	120

List of Acronyms

AA affine arithmetic

ACA Almost-correct adder

ASIC application specific integrated circuit

ALU arithmetic logic unit

ANT Algorithmic noise-tolerance

BPB bits per block

BTWC better-than-worst-case

BTI bias temperature instability

CHW classic hardware

CMOS complementary metaloxidesemiconductor

CORDIC Coordinate Rotation Digital Computer

CDFG control-data flow graph

DSP digital signal processing

DVFS dynamic voltage-and-frequency scaling

DVS dynamic voltage scaling

EDP energy-delay product

ERSA	Error Resilient System Architecture
EPMF	error probability mass function
EIC	efficiency contribution
EAC	efficacy contribution
FID	fidelity
FPGA	field-programmable gate array
FSM	frequent small-magnitude (error)
GA	genetic algorithm
GUI	graphical user interface
HTI	hot carrier injection
HDG	Hierarchical Dependency Graph
IA	interval arithmetic
IHW	imprecise hardware
ITRS	International Technology Roadmap for Semiconductors
IC	integrated circuit
ILM	infrequent large-magnitude (error)
JPEG	Joint Photographic Experts Group
KSA	Kogge-stone adder
LAT	latency
LSB	least significant bit
MAA	Modified affine arithmetic

METAII Modified Type II Error-Tolerant Adder

MIA Modified interval arithmetic

MSB most significant bit

MSE mean squared error

MPEG Moving Picture Experts Group

NMR N-modular redundancy

PLAT path latency/delay

PVT process, voltage, and temperature

PCMOS Probabilistic CMOS

PMF probability mass function

PDF probability density function

PPG partial product generation

RCA ripple-carry adder

RTL register-transfer level

RMSE root mean squared error

SNA symbolic noise analysis

SNR signal-to-noise ratio

SNC Stochastic networked computation

SPICE Simulation Program with Integrated Circuit Emphasis

SQRT square root

SRAD Speckle Reducing Anisotropic Diffusion

SEU single event upset

SVM support vector machine

TH threshold

TDDB time dependent dielectric breakdown

UI user interface

ULP unit in the last place

VHDL very-high-speed integrated circuits hardware description language

VOS voltage-overscaling

Chapter 1

Introduction

1.1 Background and Motivation

According to the 2009 ITRS report [1], with fabrication technology scaling into the nanometer region, CMOS devices will inevitably become more susceptible to variations and errors. Process, voltage and temperature (PVT) variations, soft errors, and aging effects will have a more dramatic impact on circuit behavior, and the task of making a reliable circuit is more challenging than ever before. Meanwhile, device density grows as transistor sizes shrink. The power consumption of an individual transistor is not expected to decrease because of increasing leakage power, so the power density grows even faster. Indeed, there are two major challenges facing circuit designers in this deep-submicron era: energy efficiency and reliability [2]. The typical strategy to meet the increasingly stringent energy and reliability requirements is to introduce design margins and dynamic voltage and frequency scaling (DVFS), both of which incur additional cost and design complexity, sometimes prohibitive. Designers are seeking alternative methods to improve energy efficiency and

reliability without having to pay the price of additional die area or multiple power supplies.

To solve this dilemma, we should first look at the root cause of the problems. The merit of any IC design is ultimately determined by how well it meets the *functional* and *non-functional* requirements. Functional requirements stipulate the intended system behavior (what output should result from a certain input). For example, a circuit performing digital signal processing algorithms usually requires the signal-to-noise ratio (SNR) of the output signal be above a certain level and a circuit performing arithmetic operations may require some minimum precision. In these cases, SNR and precision are used to measure how well the system produces a desired output, so we denote them as *quality* metrics. Non-functional requirements specify the maximum resource consumption from implementing the function. For example, clock frequency, power consumption and die area are usually part of the non-functional requirements of any IC design. We use *efficiency* metrics to measure how well an implementation fulfills its non-functional requirements. Area, power and delay are all examples of efficiency metrics. Typically the quality metrics are extracted from the circuit output while efficiency metrics are associated with the actual circuit implementation. When following a traditional circuit design methodology, *functional correctness* must be guaranteed. That is, given an input vector I , there is only one acceptable output vector O . If the circuit under design produces a different output O' , the design is considered faulty and must be corrected. During the algorithm development stage, only those algorithms that meet the target quality requirement will be chosen. Once the algorithm becomes fixed, the hardware implementation must faithfully execute the specified functionality without compromise. Unfortunately, this stringent requirement hampers the optimization of the efficiency metrics and makes it difficult to meet non-functional requirements (such as power

consumption).

In recent years, a new design philosophy has emerged that relaxes the absolute correctness requirement to enable improvements in efficiency metrics. In other words, erroneous output O' is deemed acceptable provided that it leads to a more efficient implementation. Consider an N -bit adder. The quality metric is the Mean Squared Error (MSE) and the efficiency metric is the Energy-delay Product (EDP). The reliable adder with the lowest EDP is a Carry-Lookahead Adder (e.g., the Kogge-Stone Adder [KSA]), whose delay is proportional to $\log N$. However, this delay and the EDP can be further improved if we allow errors to occur. For example, voltage-overscaling (VOS) is a technique to operate a circuit at voltages below the critical operating voltage. The EDP of a voltage-overscaled KSA is even lower than a traditional KSA, but timing violations may cause errors in the output of the former. In general, however, the critical path is not triggered frequently and thus the majority of the output from the voltage-overscaled KSA will be correct. By adjusting the supply voltage, we can trade off the efficiency and quality of this adder. Later, in Section 2.6.2, we will see that the KSA does not provide the best efficiency-quality tradeoffs under VOS compared to other IHW adders. Similar efficiency-quality tradeoffs exist at various levels of granularity: from a transistor at the device level all the way to algorithmic tasks at the system level. *Imprecise Hardware (IHW)* is thus defined as **any circuit component designed to exhibit improved efficiency metrics relative to a traditional, precise circuit by sacrificing output quality.**

Many applications can benefit from IHW. For example, the aforementioned voltage-overscaled KSA is suitable for arithmetic computation in wireless sensors. Infrequent errors may be acceptable if information is gathered from multiple sensors and a majority

voting system can be used to filter out the erroneous results. And the improved energy efficiency of such an adder is highly desirable for extending the battery life of the sensors. Applications which process noisy data collected from the environment are also generally tolerant of computation errors. Section 2.4 gives an in-depth discussion of the extensive applications of IHW.

One might argue that IHW is not relevant to applications with no error tolerance, such as cryptographic algorithms, microprocessors and safety-critical applications. However, it must be noted that these applications must eventually be implemented on finite-precision hardware and therefore there is an intrinsic loss of accuracy just by representing a number in finite-precision. In general, any digital IC interacting with the analog world is error-tolerant due to the quantization error introduced at the analog-to-digital interface. Even for applications strictly intolerant of errors or impreciseness, there are still opportunities to correct errors at a later stage so that the overall design remains error-free. A processor is typically considered to be error-intolerant, but some of its components such as the branch predictor are error-tolerant. ALUs can also be error-tolerant while the processor is running multimedia applications. Therefore we believe that opportunities for correctness relaxation are ample, as long as the impact on quality can be quantitatively evaluated and guaranteed.

1.2 Related Work on IHW

One of the earliest pioneers in IHW was John Von Neumann, who studied the problem of reliable computation in the presence of unreliable components [3]. Unfortunately, his conclusion suggested that enormous complexity overhead via redundancy is required to achieve system-reliability higher than component-reliability [2]. Computation with un-

reliable components did not gain much attention for many years since technology scaling in the meantime was able to double circuit performance about every two years. Only in the last decade have circuit errors again raised interest, due to the slowing of technology scaling and the process variation problem associated with sub-90nm technologies. Under current technologies, it is very difficult to increase circuit performance without any side effects. The first side effect is increased power consumption. As computation is increasingly carried out on mobile platforms such as smartphones and tablets, power reduction has become an imperative task for circuit designers. The second side effect is reduced reliability. CMOS devices in advanced technologies are more susceptible to PVT variations and soft errors, because of their smaller feature sizes. The insertion of large design margins has become a norm in the VLSI design process in order to ensure correct operation of the circuit in the worst-case scenario. However, this method results in overdesign for normal conditions and causes significant waste of power and energy.

Existing Work on IHW

A number of power reduction techniques have been widely adopted in circuit design. One example is dynamic voltage scaling (DVS) [4, 5]. Based on the fact that the peak performance of a circuit is not continuously required, DVS operates a circuit at the lowest supply voltage that meets the timing constraints. However, the power savings offered by DVS are limited by the timing constraints of the circuit. DVS is also less effective in applications that constantly require high throughput, such as a DSP ASIC. Razor [6] is a design that pushes the limits of DVS. It uses aggressive DVS to dramatically reduce power while initially allowing timing errors. It then accompanies each flip-flop on the critical path

with a shadow register to detect timing errors due to VOS. The shadow register is clocked at a fixed delay behind the main flip-flop. Whenever a difference is detected between the main flip-flop content and the shadow register content, the shadow register's value is used to overwrite the value inside the main register. A Razor-based system ensures correct functionality while operating at a much lower voltage than a traditional performance-driven system, leading to lower power consumption. Although Razor still assumes that computation must always be performed correctly, it suggests that temporarily allowing timing errors can open up opportunities for power reduction. It is a very influential design which inspired many ensuing projects.

To reduce power further, the supply voltage must be lowered below the critical operating voltage (VOS), while leaving any timing errors uncorrected. A large number of IHW techniques use VOS to achieve power-quality tradeoffs. For example, algorithmic noise-tolerance (ANT) [7] supplements a voltage-overscaled DSP system with a reduced precision replica which limits the magnitude of the resulting error. The combined system consumes significantly less power with no impact on reliability, and a 20% increase in area. Stochastic processors [2, 8] and slack redistribution [9–11] suggest that carefully-designed processor modules can exhibit graceful accuracy-power tradeoff characteristics. Graceful degradation of the error rate under VOS is a desirable property to achieve the lowest possible power given a target error rate. Significance-driven computation [12, 13] applies VOS to functionally noncritical parts of an algorithm in order to save power. Mohapatra et al. [14] identify three computational kernels commonly used in multimedia, recognition and data mining applications and suggest a few design techniques to improve their reliability under VOS.

Structural simplification is another way of implementing IHW. A number of imprecise adders have been proposed using this technique. The Almost-correct adder [15] is an adder which speculates on the longest propagate sequence and simplifies the tree structure of a KSA based on this speculation. The METAII adder [16] divides the propagate sequence of a ripple-carry adder (RCA) into segments and selectively truncates the carry bits across segments in order to accelerate addition. The IMPACT adder [17] removes transistors from a mirror adder one by one while introducing minimal errors in the truth table. Instead of adjusting the operating condition (V_{dd}), these IHW designs use structural modification to achieve better efficiency than traditional HW. However, there is a lack of understanding of how one type of imprecise adder compares to another. When using imprecise adders in a system, the settings of such adders are usually determined in an ad hoc fashion, e.g., through trial and error. Furthermore, it is unknown how more complex arithmetic operations, such as multiplication, division, and square root can be implemented imprecisely.

At the architecture level, apart from the aforementioned ANT, Bau et al. [18, 19] propose using a combination of strict-reliability and relaxed-reliability cores to design low-cost, error-resilient systems. Soft NMR [20] uses several duplicates of the same computation with known independent error statistics and votes on the output using optimal estimation theory. BTWC (better-than-worst-case) design [21] converts a traditional computational task into a “performance-power-optimized core followed by a reliability checker” form. This results in better efficiency than worst-case designs in the typical case. The checker is used to ensure correctness in the worst case. If the typical case occurs frequently, the efficiency benefit will outweigh the overhead from the checker. However, many human-interacting applications do not need exactly correct numerical outputs, since

the human sensory system is inherently not perfect. Therefore, error correction can be safely removed if the error is bounded.

At the physical level, Probabilistic CMOS (PCMOS) [22–25] is a highly scaled CMOS device with intrinsic random behavior. For example, PVT variations and SEU (single error upset) events can cause the transistor’s output to assume a random distribution. Some probabilistic algorithms can benefit from a PCMOS implementation due to faster probabilistic calculations and lower power consumption. Jeong et al. [26] look into the impact of reducing the guardband of library models on output design quality. They find that circuit metric improvement can sometimes justify guardband reduction.

Limitations of Existing Work

The existing IHW components and techniques are not without limitations. First of all, with so many IHW components and techniques spanning multiple design abstraction levels, from high-level algorithmic blocks to low-level CMOS devices, each developed to benefit a specific class of applications, a circuit designer can be overwhelmed by the sheer number of choices. How can we compare one technique with another? If the goal is to design the lowest-power system subject to a quality requirement, how do we choose what IHW techniques and components to use and what parameter values to set? The *lack of a framework* for automated exploration of the efficiency-quality design space has become a hurdle to the usability and adoption of IHW. This dissertation is dedicated to providing such a framework.

Secondly, many existing IHW techniques use error-correction circuits to compensate for the errors introduced by IHW. But the correctness requirement in error-tolerant appli-

cations can be further relaxed, leaving errors *uncorrected*. For example, users are unlikely to notice small or rare degradations in multimedia quality, and computation errors often do not affect the results of recognition or data mining analyses. Therefore it is possible to eliminate the high-energy circuit structures within certain traditional hardware to realize IHW.

Thirdly, VOS and structural IHW are two very different types of IHW techniques. Currently VOS is still the dominant IHW technique used in ASIC designs while the use of structural IHW has been neglected. It is possible that structural-IHW-based systems can achieve better efficiency than VOS-based systems at the same quality level. Combining VOS and structural IHW is another way to improve the efficiency-quality tradeoffs.

Finally, the traditional method to evaluate the quality of an IHW-based system is repeated random testing (i.e., Monte Carlo simulations). With increasing system complexity, Monte Carlo simulations are becoming less practical due to their long runtime and unreliable corner-covering capability (i.e., rare error events could be missed if not simulated with enough samples). A faster and more accurate method to estimate system quality is needed.

Problem statement

Many IHW techniques have been proposed to improve the efficiency of a circuit at a small cost of output quality, but the choice of IHW techniques and their settings is made in an ad hoc fashion without considering other alternative implementations. Therefore, the final design is likely to be suboptimal. A methodology is needed to effectively explore the design space created by IHW and produce optimized designs.

This dissertation is devoted to **the development of a methodology for trading off ef-**

efficiency and quality in ASIC designs using IHW arithmetic units. Such a methodology first requires a library of IHW arithmetic units. They must exhibit higher efficiencies than their precise counterparts, and their quality must be controllable by design parameters. Second, we need a way to rapidly compare the quality of two IHW implementations without simulation. Statically propagating the error distribution from the input to the output is one potential solution. Finally, the library, the efficiency estimator and the quality estimator must be tied into a unified framework. This framework should leverage an existing or new optimization algorithm to solve various optimization problems. Signal processing and data mining applications are good candidates to evaluate the effectiveness of our methodology. We expect the designs produced by our methodology to have significantly higher efficiencies than those of traditional precise designs. The quality of the IHW designs should be at an acceptable level from a system perspective. And the time to obtain such designs should be much shorter than exhaustively searching the design space.

1.3 The Scope of This Dissertation

Implementation Platform: ASIC. There are many implementation platforms on which to perform computation. In this work, we target Application-Specific Integrated Circuits (ASICs). An ASIC is a custom circuit that implements a single algorithm or application. It is the preferred implementation to achieve high performance or low power at the cost of limited design flexibility. Another popular implementation platform is a general-purpose processor. This platform provides good flexibility because of the programmability of the software running on top of the processor, but usually is less efficient in terms of power and performance relative to an ASIC. Most of the IHW techniques mentioned in

this dissertation are only applicable to ASICs. One exception is fidelity-compromising transformations (Section 2.6.1). This technique is widely adopted in both ASIC and software development processes when an algorithm designer fine-tunes an algorithm to balance quality metrics with efficiency metrics. An approximate algorithm running on a processor is simply another way of performing imprecise computation with the aim of improving efficiency. However, the delay and power analysis of software running on a processor is fundamentally different from the analysis of an ASIC. The analysis is further complicated by the compilation process and the presence of operating systems and other programs running on the processor. Due to our inadequate knowledge in computer architecture, we do not consider software approaches to IHW, but they are still valuable to algorithms that are traditionally implemented in software.

Types of HW components: arithmetic units. Arithmetic Logic Units (ALUs) such as adders and multipliers are the basic building blocks of datapaths in many error-tolerant applications. For example, an FIR filter is primarily composed of repeated multiply-accumulate (MAC) operations [27]. In image compression, Discrete Cosine Transform (DCT) and Inverse Discrete Cosine Transform (IDCT) are integral components of the JPEG image compression standard [28]. Both DCT and IDCT also consist of MAC operations. In video compression, Motion Estimation (ME) accounts for 70% of the total power consumption of an MPEG encoder [29]. The basic computation kernel of ME is Sum of Absolute Differences (SAD) [29], which is purely composed of adders and subtractors. Since ALUs are the dominant factor in the timing and power of data-intensive systems, we focus on imprecise implementation of ALUs to maximize the efficiency impact. Control logic units such as comparators and multiplexers are usually used for decision making in an algorithm

and the correct functionality of such units is crucial to obtaining meaningful outputs. We do not consider imprecise implementation of such circuits. Sequential logic circuits (such as latches and registers) and memory elements (such as SRAM, DRAM and NVRAM) are other classes of circuit components outside our scope. As a matter of fact, memory elements are becoming a major power sink of many digital systems, and they are also one of the most unreliable circuit structures under PVT variation. Most research in memory design is a balancing act among reliability, performance, area, and power. It is possible to slightly sacrifice the reliability of a memory element to gain improvements in efficiency.

IHW Behavior: deterministic. In Section 2.2 we classify IHW into three categories: deterministic IHW, spatially-varying IHW, and temporally-varying IHW. The focus of this dissertation is on *deterministic IHW*, whose error is a deterministic function of the input. For deterministic IHW, certain input patterns will always produce error-free outputs while others will always produce incorrect outputs. For our target applications, the error from a single computation is insignificant; what matters most are the statistical error properties of the system under representative inputs. The error distribution of deterministic IHW should not be confused with that of temporally-varying IHW. The former is really an error histogram when the input is drawn from a dataset. If the input is a single value, the error also becomes a single value. The latter is a real distribution—even when the input is a single value, the output is still a random variable. Spatially varying IHW can be analyzed in a manner similar to deterministic IHW, while errors from temporally varying IHW can be modeled as stochastic processes.

Target Application: errors can be tolerated; error correction not necessary. If errors resulting from IHW must be corrected to maintain application quality, the efficiency

benefit gained by an imprecise implementation might be offset by the overhead from the correction circuit. Therefore we are particularly interested in applications or algorithms that can tolerate errors and do not require error correction. Most DSP algorithms, multimedia applications, and classification and data mining algorithms fall into this category.

1.4 Major Contributions of This Dissertation

This dissertation is devoted to **the development of a methodology for trading off efficiency and quality in ASIC designs using IHW arithmetic units. It can be used to produce more efficient designs for error-tolerant applications than traditional IC design methodologies.** The methodology incorporates a library of parameterized IHW components and routines to evaluate efficiency and quality metrics. The quality evaluation routine is simulation-free and achieves higher speed than and comparable accuracy to Monte Carlo simulations. The following is list of essential components that must be developed or identified to fulfill this task:

- A library of parameterized IHW components and techniques that exhibit superior efficiency metrics than their traditional counterparts.
- Fast and accurate estimation methods for both efficiency and quality metrics.
- A framework that explores the IHW design space and generates a final design according to user-specified efficiency and quality requirements.
- A number of target applications and algorithms to inform the development and verify the effectiveness of the methodology.

1.4.1 Design of Novel IHW

At the algorithm level, we propose fidelity-compromising transformations to realize performance-quality tradeoffs. At the RTL level, we design imprecise multipliers based on imprecise adders and improve the error characteristics of IHW adders and multipliers. We also suggest using CORDIC to implement a wide variety of elementary functions imprecisely. Finally, we present a hybrid IHW combining two IHW techniques. It exhibits superior efficiency-quality tradeoff characteristics than each individual IHW technique alone.

1.4.2 A Static Error Estimation Method

Quality metrics are traditionally measured by running Monte Carlo simulations and comparing the imprecise output with the precise output, but when the design space is large this is a time-consuming process. Assuming IHW is an error source in the system, chapter 3 presents a novel error estimation method which is almost simulation-free. It leverages statistical analysis to propagate the error distribution from the input to the output, provided that we know in advance the input data distribution and the error characteristics of the IHW components. This static method is orders of magnitude faster than Monte Carlo simulations.

1.4.3 A Quality-Aware Efficiency Optimization Methodology

The introduction of IHW expands the design space with several degrees of freedom: the freedom to choose different IHW components and techniques for implementation; the freedom to choose parameters that define the IHW components and techniques, and finally the freedom to connect those IHW components to each other in different ways. Such a complex

design space cannot be efficiently explored without a systematic methodology. This dissertation presents a methodology designed to solve many versions of the efficiency-quality optimization problem. It can solve constraint-based optimization problem (e.g., energy minimization subject to a minimum quality constraint); it can solve a cost-function-based problem (e.g., minimization of the energy-delay-error-rate product); and it can generate Pareto optimal curves. The methodology has been encapsulated into a released tool.

1.4.4 IHW Taxonomy

To better understand the relationships of various IHW techniques and components, we build an IHW taxonomy. Existing IHW techniques and components are classified according to where in the design hierarchy they are implemented, their error behavior, and their system-level impact.

1.5 Organization

This dissertation is divided into 5 chapters. Following the motivation and background provided in this chapter, Chapter 2 introduces the concept of IHW and the working mechanisms of various IHW designs. An IHW taxonomy is developed to classify all IHW components and techniques. We elaborate on the design of two IHW examples: fidelity-compromising transformations and imprecise arithmetic units (i.e., adders and multipliers) and their error characteristics. In Chapter 3, we present the static error estimation method. Mathematical models of error distributions are first introduced and rules for error propagation are then derived. We compare its speed and accuracy with classical Monte Carlo sim-

ulations and show its advantage. The efficiency-quality tradeoff methodology is described in Chapter 4. Case studies on medical imaging, classification and arithmetic calculation are presented to demonstrate the usage and effectiveness of the proposed methodology. In the final chapter, we summarize the contributions as well as the limitations of this study. We also highlight potential areas of further research.

Chapter 2

Imprecise Hardware (IHW)

Contrary to traditional hardware which always produces correct results, IHW only produces correct results probabilistically or conditionally, as determined by its structure, parameters, environment, and input patterns. IHW takes advantage of relaxed quality requirements to improve efficiency beyond the level achievable by traditional hardware. Before we use it to implement real applications, it is important to understand how IHW achieves efficiency-quality tradeoffs and how various IHW instances differ from or resemble each other. In this chapter, we study the principles of IHW and build an IHW taxonomy. We also give an in-depth analysis of two selected IHW designs—fidelity-compromising transformations and IHW arithmetic units, both of which will be used in the case studies in Chapter 4.

2.1 Introduction

Imprecise hardware is a natural product of computation with limited resources. When HW resources such as power, area and computation time do not allow for sufficiently precise computation, the simplest strategy is to report failure, but sometimes it is better to attempt to compute an inexact result which is still acceptable—ideally any increase of HW resources should contribute to the generation of a higher-quality result. Nevertheless, traditional HW is not designed in this manner; once the operating frequency exceeds or the supply voltage drops below the normal level, a large percentage of results will be incorrect and the output quality becomes unacceptable. Admittedly, many applications fit into this category which distinguish only between correct and incorrect results: processor modules such as the program counter and the writeback logic cannot tolerate even the slightest error. However, there is another large class of applications where correctness is not a boolean value, but can be measured with a continuous *quality* metric (soft-correctness). The exact definition of quality is application dependent. For example, signal-to-noise ratio (SNR) can be used as a quality metric for many DSP circuits, and root mean square error (RMSE) can be adopted for arithmetic logic blocks. A lower-quality but efficient implementation may be more valuable than a high-quality but inefficient implementation. IHW is unique in its capability to adjust efficiency and quality simultaneously, making it an ideal building block for implementing soft-correctness applications.

Traditional correctness-centric hardware has a convex efficiency-quality curve (Figure 2.1), which does not allow efficiency-quality tradeoffs. The quality metric remains high at low efficiency regions. Once we start to improve the efficiency metric beyond a certain critical threshold, quality drops sharply. IHW, however, exhibits a concave curve

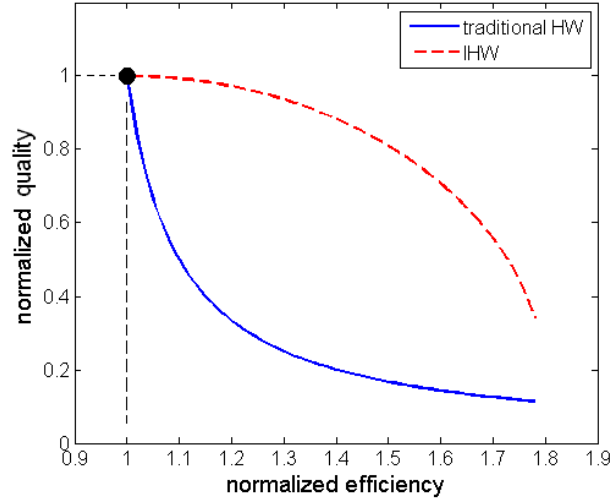


Figure 2.1: Qualitative quality-efficiency curve of IHW. Both quality and efficiency metrics are normalized against the error-free operating point (quality=1, efficiency=1), which gives the highest quality and lowest efficiency.

and can achieve orders-of-magnitude improvement in efficiency with a small but non-zero impact on system quality. Examples include voltage overscaling (VOS) to achieve higher performance and lower power, simpler approximations to complex math functions, and deeply scaled CMOS devices with probabilistic behavior. Compared to their traditional counterparts, their quality degradation occurs more gradually as we increase efficiency, causing complete failure at a much higher implementation efficiency.

IHW has a broader meaning than imprecise arithmetic units such as adders or multipliers. It is a generalized circuit implementation that carries out its intended operation with the possibility of error. Examples of IHW include arithmetic units, control units, and memory units. IHW can be as large as a processor core that is prone to error during program execution or as small as a logic gate which exhibits probabilistic behavior due to PVT variation. They all share a similar trait: improved HW efficiency as a result of allowing the presence of errors.

IHW guarantees an efficiency improvement, provided that there is room for quality

relaxation. During the efficiency optimization stage, all metrics are bounded by a multi-dimensional *Pareto surface* defining the set of optimal design points that provide correct functionality. Points on the Pareto surface (called *Pareto-optimal points*) have the following property: any improvement on one metric will necessarily deteriorate some other metrics. Since the Pareto surface is determined by a circuit's functional requirements, adding an extra metric of quality to the design space adds another dimension to the optimal Pareto surface, the cross-sections of which reveal different tradeoff surfaces in the original design space. IHW is capable of translating quality slack into efficiency improvement (Figure 2.2), so it guarantees that new cross-sections strictly dominate the old one and thus efficiency can be improved beyond previous bounds.

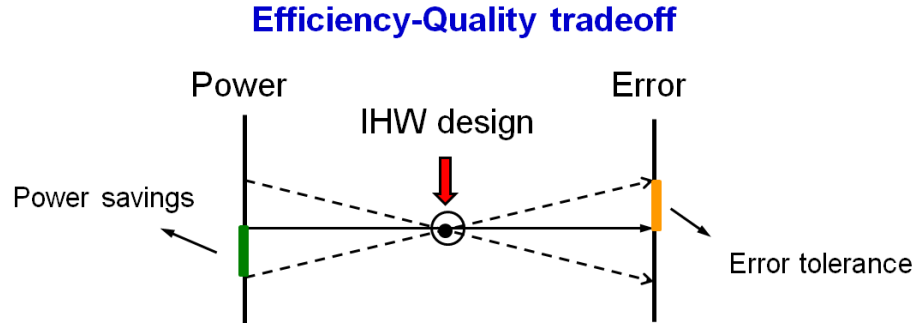


Figure 2.2: Conceptual diagram of efficiency-quality tradeoff. IHW can convert relaxed quality requirements (e.g., error tolerance) into improved efficiency (e.g., power savings).

The magnitude of the efficiency-quality tradeoff from IHW can be *controlled* through design-time and run-time parameters. For example, by changing the V_{dd} of VOS techniques or the RTL structural parameters of IHW adders and multipliers, we can shift the balance between high-quality-low-efficiency and low-quality-high-efficiency designs. If we can further derive or model the relationship between efficiency and quality and those parameters, it is possible to quantitatively perform certain optimization tasks.

Applications with stringent correctness requirements can benefit from IHW as well.

Better-than-worst-case designs (BTWC) [21] indicate that any system can be implemented as a performance-optimized core plus a correctness checker. Since the task of ensuring correctness is delegated to the checker, the performance-optimized core is free to adopt any IHW design. The overall system is error-free while the efficiency of the IHW and checker combined may still be higher than that of a traditional correctness-driven design. Razor [6] is a perfect example of error-free design with IHW.

2.2 IHW Behavior

2.2.1 Based on I/O Mapping

We can classify IHW into several categories based on the method of mapping the input to the output.

- (i) *Classical hardware (CHW)* maps any specific input to its corresponding output faithfully.

$$CHW : I \Rightarrow O$$

O is a deterministic function of I and the mapping CHW remains the same temporally and spatially; i.e., different copies of the same hardware always have the same mapping and every copy always has the same mapping at any given time.

- (ii) *Deterministic IHW* relaxes the CHW in terms of the mapping function.

$$IHW_{deterministic} : I \Rightarrow O'$$

Here O' is not equal to O at all input values, but the mapping remains stable across time and space. The output error (given by $O' - O$) is also a deterministic function of

the input I —certain inputs will always produce incorrect outputs while other inputs will always produce error-free outputs. For example, trimming the propagate chain of an adder will cause an output error only if the input triggers a longer propagate path. Deterministic IHW is *incorrect by design*. Producing O' usually requires less resources than producing O .

- (iii) *Spatial IHW* not only has a different mapping than the CHW, but different copies of the same circuit have different I/O mappings. These mappings are all temporally stable. Spatial IHW shares the features of both deterministic and probabilistic IHW (see below). Given the limited knowledge about process variation at design time, spatial IHW exhibits probabilistic behavior because process variation is treated as a random variable at design time. At runtime, however, each copy of spatial IHW is deterministic. So a design strategy for spatial IHW combines the strategies for both deterministic and probabilistic IHW.

$$IHW_{spatial1} : I \Rightarrow O'_1$$

$$IHW_{spatial2} : I \Rightarrow O'_2$$

$$IHW_{spatial3} : I \Rightarrow O'_3$$

The main source of spatial difference of these circuits is process variation. Modern design methods insert significant margins to ensure that a large percentage of fabricated circuits produce O . These margins can be reduced or eliminated if most O' are acceptable.

- (iv) *Probabilistic IHW* has the most chaotic behavior of all IHW. For a given probabilistic IHW, it can produce different outputs every time it evaluates a given input. The

output is really a random process controlled by the input vectors and the surrounding environment.

$$IHW_{probabilistic1} : I \Rightarrow O'_1 \text{ at } t_1, O'_2 \text{ at } t_2, O'_3 \text{ at } t_3, \dots$$

$$IHW_{probabilistic2} : I \Rightarrow O''_1 \text{ at } t_1, O''_2 \text{ at } t_2, O''_3 \text{ at } t_3, \dots$$

$$IHW_{probabilistic3} : I \Rightarrow O'''_1 \text{ at } t_1, O'''_2 \text{ at } t_2, O'''_3 \text{ at } t_3, \dots$$

The main sources of the temporal difference of these circuits are temperature and voltage variations, soft errors, and aging effects. As with process variation, considerable design margins can be reclaimed if most of the O' , O'' and O''' are acceptable.

Throughout this dissertation, we will only focus our discussion on deterministic IHW.

Table 2.1 lists examples of each category of IHW.

Table 2.1: IHW examples by behavior.

Deterministic	Spatial	Probabilistic
round-up multiplier, approximation, VOS adder	reducing process variation guardband	probabilistic CMOS [22], reducing VT variation, aging, soft error guardband

2.2.2 Based on Error Characteristics

IHW can also be classified based on the characteristics of the errors it produces. If an IHW design is fed a large number of random inputs, statistically the output error histogram will resemble a discrete random distribution. This distribution can be quantified by its *error magnitude* and *error probability*. The error magnitude is defined as the difference between

the actual and the accurate output ($O' - O$), and the error probability is the probability of getting an error of a particular magnitude. We can use the *Probability Mass Function (PMF)* to describe the distribution of the error. It can be visualized as a bar chart on the frequency vs. magnitude plane, as shown in Figure 2.3.

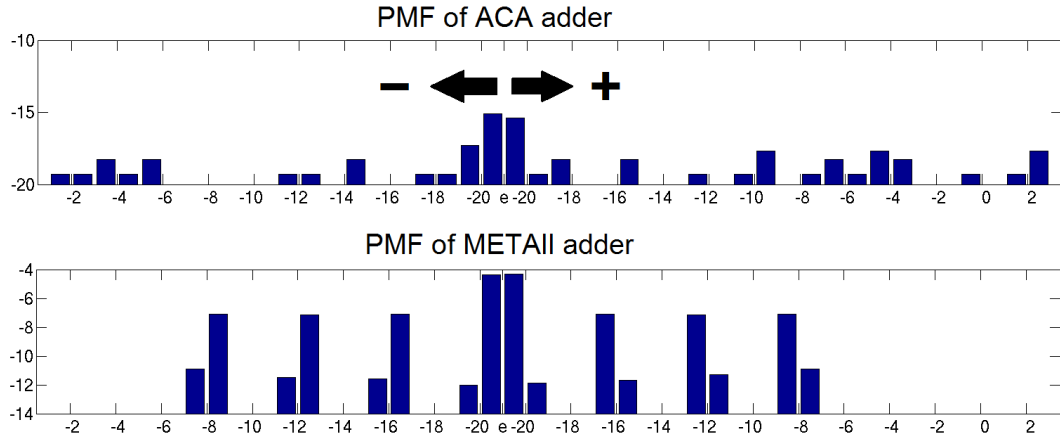


Figure 2.3: Probability Mass Function (PMF) examples.

The PMF is a type of histogram. Each bar of a PMF indicates the percentage of the error falling in a particular magnitude range. The location of the bar on the x-axis indicates the magnitude range of the error and the height indicates its frequency of occurrence. The taller a bar is, the more frequent the error occurs. Both the x-axis and y-axis are base 2 logarithmic scaled in order to cover a wider frequency-magnitude range. To give an example, a bar bounded by marker -8 and -7 with a height -10 means the frequency of observing the error between magnitude 2^{-8} and 2^{-7} is 2^{-10} . The e symbol in the middle of the x-axis represents zero error; thus bars to the left have negative error magnitudes and those to the right have positive. The sum of the heights of all of the bars in a PMF is equal to the total error probability (P_e); therefore the probability of no error is implicitly obtained by $1 - P_e$. Within each bar, the error is assumed to be equally likely to take on any value

in that magnitude range. Since the PMF provides higher resolution near zero magnitude, a distribution not centered at zero should have its mean subtracted from it to achieve zero-mean so that it can be represented with higher accuracy. The mean will participate in the PMF propagation (Chapter 3) as a separate constant.

Figure 2.3 shows two different types of error distributions. Errors in the upper PMF have a wide magnitude range, but their frequency of occurrence is small; the lower PMF has smaller error magnitude but those errors occur more frequently. We denote the former type of error *Infrequent-large Magnitude (ILM)* error and the latter *Frequent-small Magnitude (FSM)* error. The type of errors produced by an IHW component is dependent on its structures (Table 2.2).

Table 2.2: IHW examples by error types.

IHW producing ILM errors	IHW producing FSM errors
ACA adder and multiplier	METAII adder and multiplier
voltage-overscaled circuits	reduced-precision circuits
guardband reduced circuits [26]	fidelity-compromising transformations [30]

There is no widely accepted rule on which type of error will lead to higher output quality because different applications and even different operations within one application have different sensitivities to error frequency and error magnitude. For example, if multiplication is implemented with repeated addition, FSM errors may be more detrimental than ILM errors because error magnitude tends to accumulate with repeated operations. ILM errors, however, may not produce a single error during the entire multiplication if the error frequency is sufficiently low. In another example, many image processing algorithms require

iterating over a loop until a condition is met, such as convergence testing. If addition is needed for the condition evaluation, imprecise adders with FSM errors are the preferred choice. A big error in the condition evaluation can cause a false early exit of the loop, significantly affecting the result quality. FSM errors are much safer because they are less likely to change the decision made by the condition evaluation. Many massively parallel applications are equally sensitive to error frequency and error magnitude. The key is that errors must be considered within the context of the application. Therefore it is necessary to develop a method to quantitatively estimate the error distribution of an IHW-based system. Error PMFs of IHW provide a mathematical representation that makes quantitative analysis possible. A static error propagation method will be proposed in Chapter 3.

2.3 IHW Systems

While a single IHW component or technique can effectively trade off efficiency and quality, most applications and algorithms are so complex that the benefits from a single tradeoff point have little impact at the system level. Also, using one type of IHW usually has obvious drawbacks (such as unbounded error magnitude or error frequency) that limit its use. Constructing a system of various types of IHW can effectively solve these two problems, although at the cost of some area and power overhead. Careful strategies are needed for applying IHW components and techniques to a system so as not to accumulate errors excessively. Several possible interconnections of IHW are listed below:

- *ANT* [7]: VOS-circuit in parallel with a reduced precision circuit (combining “infrequent large magnitude error” and “frequent small magnitude error”, we get “infre-

quent small magnitude error”).

- *Soft NMR* [20]: Several duplicates of the same computation with known independent error statistics that vote on the output using optimal estimation theory.
- *ERSA* [18, 19]: One reliable core paired with many unreliable cores, for RMS applications.
- *BTWC (better-than-worst-case)* design [21]: performance and power optimized core followed by a reliability checker. BTWC designs have better efficiency than worst-case designs when the typical case is encountered. A checker is used to ensure correctness when the worst case is encountered. If the typical case occurs frequently, the whole system is more efficient than the worst-case design.
- *Stochastic Networked Computation (SNC)* [31]: multiple sensor outputs fused to generate the final corrected output.
- *reverse BTWC*: error condition checker followed by a reliable core in parallel with an IHW core. The checker first detects if the input can induce large errors if run on the IHW core. If the condition is true, it feeds the input to the IHW core and turns off the reliable core to save power. If the condition is positive, it feeds the input to the reliable core and turns off the IHW core to maintain quality. If the IHW core is executed most of the time, the average power consumption of the whole system will be close to that of the IHW core.

Notice that BTWC and reverse BTWC both appear error free to the outside, so they can be used to implement error-intolerant applications. Actually, the error-free situation

is merely a special case of a generic IHW system. The error distribution of the system is a function of IHW parameters and certain parameter settings happen to entirely eliminate errors to produce an error-free system.

2.4 Applications of IHW

We have identified several groups of target applications that can benefit from IHW implementations:

Signal processing, multimedia, and communication. Data being processed by these applications are intrinsically contaminated with noise, so they usually incorporate a noise removal algorithm. Furthermore, the output from these applications is usually for consumption by the human sensory system, which is known to be error tolerant. That means that absolute correctness is not required for these applications. In recent years, more and more multimedia applications are running on mobile devices such as smartphones and tablets. Wearable medical devices are also equipped with signal processing capabilities to help diagnose patients' conditions. Employing IHW in these scenarios can significantly extend battery life with unnoticeable impact on quality of service.

Statistical analysis, data mining, clustering, and classification. These algorithms extract global knowledge from a large dataset. Since the calculation on a single data point contributes a negligible amount to the final output, imprecise implementations are not penalized if errors occur only for a small fraction of the data points, or if consecutive errors cancel each other out so that no significant error accumulation is present in the final output. In other words, isolated errors are acceptable as long as the final *Error Probability Mass Function (EPMF)* has a desirable distribution. Although these algorithms typically run on

server machines with sufficient power, IHW can significantly reduce the runtime of these algorithms by slightly sacrificing the result quality.

Nonfunctional and performance-enhancing units [32] within any application. Microprocessors are generally not tolerant of any error because they are meant to run a wide variety of applications, including those with stringent correctness requirements. However, within a microprocessor, there are many structures designed just for performance enhancement purposes; a fault in those units will only cause performance degradation. For example, an error in a branch predictor has no impact on the correctness of the processor. The only error it causes is a misprediction, which merely results in wasted clock cycles. These performance-enhancing structures are present in ASIC designs as well. Many image recognition algorithms begin with an optional noise removal step which helps the later recognition step. Implementing these structures with IHW improves their efficiency while reduces the efficiency of the following stages.

Probabilistic and stochastic applications. These algorithms all involve a probabilistic step to generate random results. They can benefit from probabilistic CMOS [22], which exhibits probabilistic behavior at the logic level.

For *error-intolerant applications*, a mapping $I \Rightarrow O'$ different from $I \Rightarrow O$ is not acceptable, but we can construct a correction circuit with a functional mapping $O' \Rightarrow O$. The correction circuit cleans up the errors in the faulty circuit so that their combined function is equivalent to the original ($(I \Rightarrow O') + (O' \Rightarrow O) \equiv I \Rightarrow O$). To have an overall gain, the efficiency of the IHW circuit combined with the correction circuit needs to be higher than that of the tradition precise circuit.

- (IHW + correction) efficiency > classical hardware efficiency

In summary, an application can benefit from IHW if it has one of the following properties: (i) output errors are tolerated up to certain magnitude and frequency levels; (ii) the system quality is not affected by individual output error, but rather is determined by aggregate output statistics; (iii) it contains performance-enhancing units; (iv) it involves a probabilistic step; or (v) it can be converted to a more efficient “IHW plus correction” style.

2.5 IHW Taxonomy

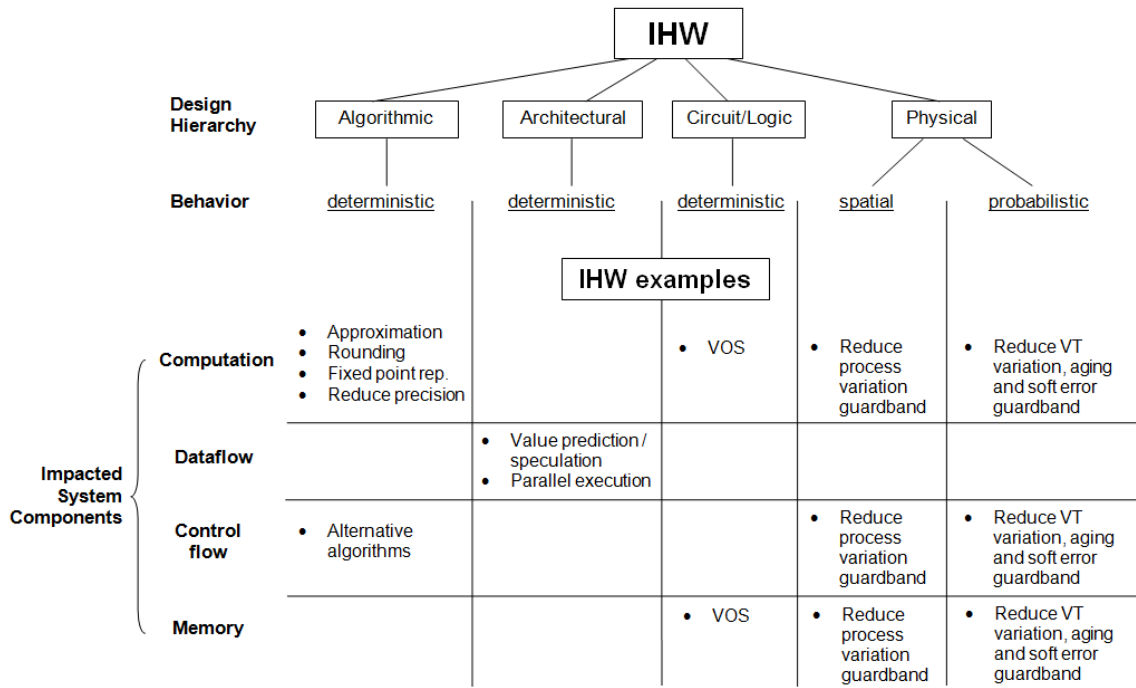


Figure 2.4: IHW taxonomy.

At the advent of any new field of research, a taxonomy is an effective tool to study the explored and unexplored space in detail. For IHW, it is useful for two reasons. First, it groups similar IHW components and techniques together. This allows synthesis algo-

rithms to be developed for each class of IHW, rather than each individual technique and component. Second, it exposes open areas for future research. Figure 2.4 shows a tree-like taxonomy of all known IHW instances. They are first distinguished by whether their behavior is deterministic, spatial or probabilistic (refer to Section 2.2). Then they are further divided according to the design abstraction level where errors are introduced. Deterministic IHW can be constructed at the algorithmic, architectural, and circuit level; while the only known spatial and probabilistic IHW is realized through device physics. The last branch of the taxonomy represents the four principle constituents of a digital system: computation, dataflow, control flow, and memory. IHW components and techniques are classified by the digital system component being impacted. Many error detection and correction techniques have been designed for each category.

2.6 Selected IHW Designs

In this Section, three examples of IHW are introduced: fidelity-compromising transformations, imprecise adders, and imprecise multipliers. Fidelity-compromising transformations are novel algorithmic-level techniques that we propose. At the logic level, we will review two imprecise adders in the literature, improve their error distributions, and use them to build imprecise multipliers. These imprecise adders and multipliers will be used in the case studies in Chapter 4.

2.6.1 Fidelity-Compromising Transformations [30]

Fidelity-compromising transformations are designed to replace an implementation of an algorithm routine (e.g., a function) with a simpler approximation. For example, when variable x is in the range of $(-1, 1)$, replacing $\frac{1}{1+x}$ with $1 - x$ reduces computation time, because division usually takes longer to compute than addition, but the result is not completely accurate. Fidelity-compromising transformations can be parameterized by a threshold to control the level of inaccuracy being introduced. They can be derived from two major sources. The general source is mathematical approximation theory. A more application-specific source is an alternative algorithm designed to solve the same problem in a different way (such as Roberts edge detector [33] versus Canny edge detector [34]). Table 2.3 shows an example of a fidelity-compromising transformation based on a mathematical approximation. This transformation has two parameters: p_1 is the order of the approximation, and p_2 is the threshold that controls how often the transformation is applied. The precise operation is replaced with the approximation whenever the condition is met. The last two rows of the table show the change in latency and fidelity as a result of the transformation. The latency is calculated using the data from Table 4.1. The change in fidelity is calculated as the percentage difference between the approximate value and the precise value.

Taylor series representations are popular mathematical approximations:

$$\begin{aligned}
 (1 \pm x)^m &\approx 1 \pm mx \pm \frac{m(m-1)}{2}x^2 + \dots \\
 \sin x &\approx x - \frac{x^3}{6} + \frac{x^5}{120} + \dots \\
 \log(1+x) &\approx x - \frac{x^2}{2} + \frac{x^3}{3} + \dots
 \end{aligned}$$

Table 2.3: An example of fidelity-compromising transformation based on mathematical approximation.

precise operation	$\frac{1}{1+x}$			
approximation	1	$1 - x$	$1 - x + x^2$	$1 - x + x^2 - x^3$
p_1	0	1	2	3
p_2	A			
condition	$ x < A, A \in (0, 1)$			
$\Delta\text{latency (control step)}$	-11	-10	-9	-8
$\Delta\text{fidelity}\%$	x	$-x^2$	x^3	$-x^4$

The efficiency-quality tradeoff can be controlled by the number of terms in the approximation as well as the threshold of x for applying approximation. According to Taylor's theorem, the remainder of a Taylor series approximation is bounded by the order of the approximation (i.e., the number of terms). Therefore, fidelity-compromising transformations produce FSM-type errors—the error magnitude can be made arbitrarily small with more approximation terms, but the approximated value is *almost* always different from the precise value. In fact the approximated value is equal to the precise value only at special data points.

2.6.2 Imprecise Adders and Multipliers

Among all arithmetic logic units (ALUs), adders and multipliers are used most extensively in the data paths of multimedia and data mining applications. It is also relatively easy to establish quality metrics for ALUs. Imprecise implementations of adders and multipliers have the most direct impact on system efficiency and output quality. Here we present

two imprecise adder designs from the literature. They are based on two types of traditional adders and exhibit distinctive error characteristics. We then modify the adders to improve their error distributions. Finally, we use them to build imprecise multipliers.

Almost-Correct Adder (ACA) [15]

The ACA is a modified version of the traditional logarithmic-delay adders, such as the Kogge-Stone Adder (KSA). ACA leverages the fact that for random inputs, the vast majority of the actual critical paths are much shorter than the worst-case critical path. Table 2.4 gives the probability of two random 64-bit inputs (A and B) triggering a critical path longer than K . Even with K much smaller than 64, the probability of a critical path violation is quite small and that probability decreases rapidly with larger K . ACA then uses a tree structure to compute the *propagate* and *generate* signals similar to KSA but assuming the longest run of *propagate* never exceeds K ; i.e., Sum_i is computed using only $A_i \cdots A_{i-K+1}$ and $B_i \cdots B_{i-K+1}$. Its worst case delay is proportional to $\log K$. ACA's structure is essentially a trimmed KSA tree which grows short of the full height. A smaller tree translates to lower delay, smaller area, and less energy per addition. Figure 2.5 shows the structure of a 16-bit ACA with $K = 6$. The definitions of the operators are as follows:

$$\text{P, G generation: } G = A_i B_i$$

$$P = A_i \oplus B_i$$

$$\text{dot operator: } (G, P) \cdot (G', P') = (G + PG', PP')$$

$$\text{sum generation: } C_i = G_i$$

$$S_i = P_i \oplus C_{i-1}$$

Each *dot operator* produces a *propagate* and a *generate* bit. The culminating generate bits (the carries) are produced in the last level, and these bits are XOR'd with the initial *propagate* from *P, G generation* to produce the sum bits.

Table 2.4: Probability of a random propagate chain exceeding K bits.

K	12	16	24	32
Probability	0.0024	1.22×10^{-4}	2.4×10^{-7}	$< 2.06 \times 10^{-10}$

Compared to a KSA which requires 4 levels and 49 dot operators, ACA only has 3 levels and 41 dot operators. We synthesized both KSA and ACA in a 130nm technology to obtain their critical path delays and performed SPICE simulations of 1,000 random additions to obtain their average energy per operation. The energy and delay data are shown in Table 2.5. As can be seen in the table, while the delay of the ACA adder is only slightly smaller than KSA due to the logarithmic effect, its energy per operation is almost 22% lower than KSA due to the simplified logic structures. During the simulation, the supply voltage is fixed at a nominal level (1.2V) for both adders. But since ACA has a lower delay than KSA, it can potentially reduce Energy per operation further via voltage scaling.

Table 2.5: Average E/op and worst-case delay of 64-bit KSA and 64-bit ACA adders with $K = 16$.

Adder	Average E/op (pJ)	Worst-case Delay (ns)
KSA64	8.47	0.8
ACA64 (K=16)	6.58	0.7

Errors occur in ACA when its assumption of maximum propagate length is violated, i.e., when the inputs trigger a propagate chain longer than K . For example, when A and

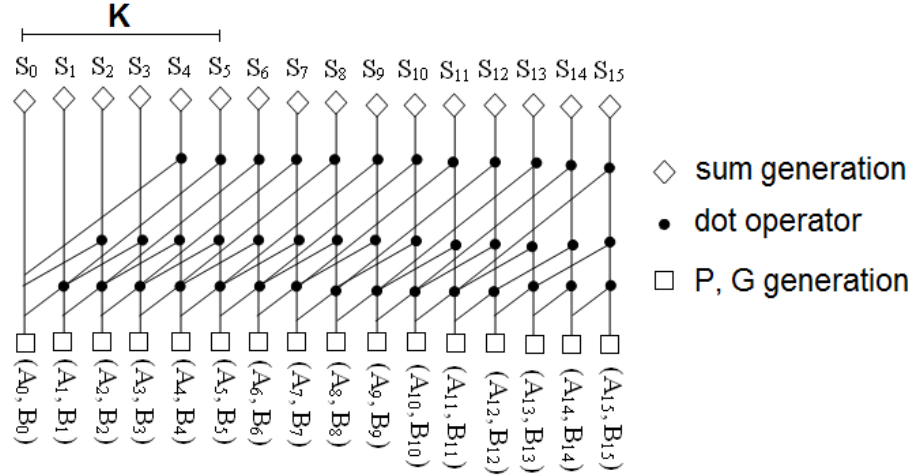
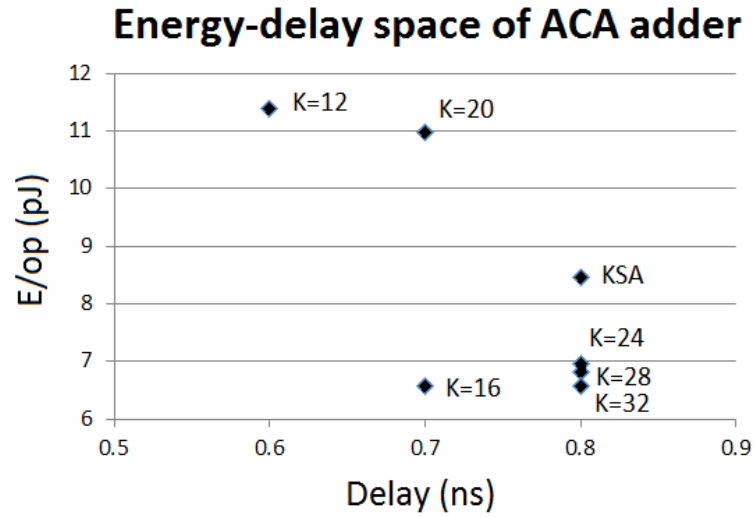
Figure 2.5: 16-bit Almost-Correct Adder structure with $K = 6$.

Figure 2.6: Energy-delay space of ACA adder and KSA adder.

B are exactly complementary, the propagate chain will span the full adder's length. To produce the correct Sum_i , all the bits from both inputs will be needed, but ACA makes a speculation and approximates it with the propagate chain from bit i down to $i - K + 1$ with the carry-in set to constant 0. When the speculation is wrong, a large error will appear in Sum_i . The largest error occurs when bit i is the MSB. Since the error magnitude of ACA can potentially be very large, but its error frequency decreases exponentially with parameter K , it belongs to the ILM error category. Certain applications are quite tolerant of ILM errors. For example, in image and video codec applications, ILM errors can manifest as single pixel errors which are usually not appreciable by human vision. However, ILM errors can cause catastrophic failure in other applications. Fortunately, ILM errors can be easily detected by inspecting the first few MSBs of the result.

Energy-quality tradeoffs can be achieved by tuning the design parameter K . Setting K equal to the adder width reduces the adder to KSA, which is a precise adder. Changing K can also directly affect the delay and energy of an adder. Figure 2.6 shows the energy-delay space of 64-bit ACA adders with different K settings. The delay refers to the worst-case critical path delay and the energy refers to the average energy per addition. The adders are synthesized to their respective critical path delays. The design point of a 64-bit KSA is also shown for comparison.

Modified Error-tolerant Type II (METAII) adder [16]

The METAII is another type of imprecise adder based on the Ripple-Carry Adder (RCA). RCA has a simple linear propagate chain (Figure 2.7). METAII works by partitioning the propagate chain into segments of variable widths. The carry bits across two

segments are truncated to zero. In order to provide better protection for higher bits, segments are wider (include more bits) on the MSB side than on the LSB side. METAII has two parameters: *BPB* (bits per block) and *L* (the number of blocks used for generating the MSB). A block refers to the smallest segment, which is usually located at the LSB. The maximum error is limited by the longest segment width (given by $BPB \times L$). However, carry generation across blocks is common; therefore errors occur quite frequently in METAII. Errors of METAII belong to the FSM error category because their magnitudes are bounded by the design parameters and are usually small compared to ILM errors.

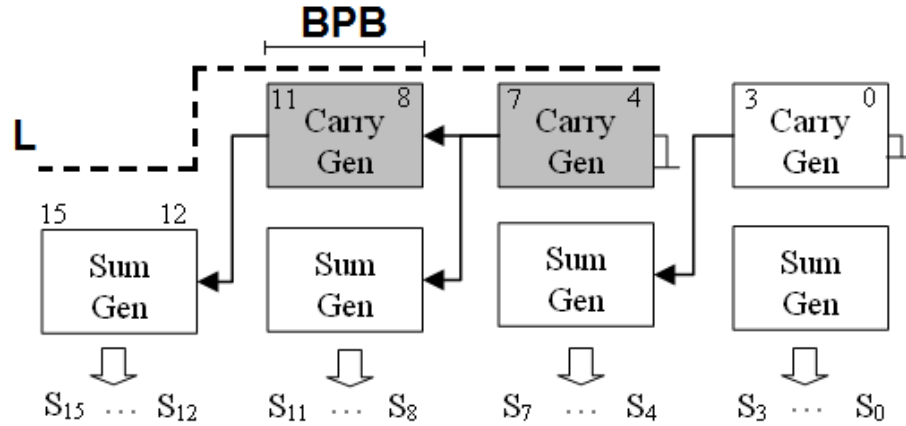


Figure 2.7: 16-bit Modified Error-Tolerant Adder Type II structure with $BPB = 4$, $L = 3$.

Table 2.6 compares the average energy per operation and worst-case delay of a RCA adder and a METAII adder. METAII exhibits much lower delay and energy consumption than RCA. We can achieve energy-quality and energy-delay tradeoffs by changing the design parameters *BPB* and *L*. Figure 2.8 shows the energy-delay space of a 64-bit METAII adder obtained by changing *L* while *BPB* is fixed at 4. The design point of a 64-bit RCA is also shown for comparison. Similar to the ACA adder, the delay is defined as the worst-case critical path delay and the energy is the average energy per addition. The adders are synthesized to their respective critical path delays.

Table 2.6: Average E/op and worst-case delay of 64-bit RCA and 64-bit METAIL adders with $BPB = 4$ and $L = 4$.

Adder	Average E/op (pJ)	Worst-case Delay (ns)
RCA64	5.48	5.3
METAIL (BPB=4, L=4)	3.07	0.8

Energy-delay space of METAIL adder (BPB=4)

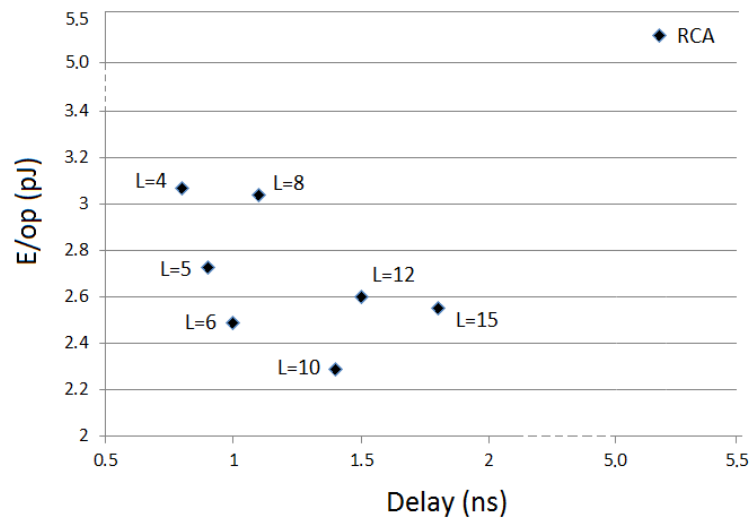


Figure 2.8: Energy-delay space of METAIL adder and RCA adder.

Voltage-overscaled adder

Voltage and frequency scaling is a useful run-time technique for power reduction. It has been shown that RCA based circuits can continue to function at a clock period well below their critical path delays [8]. Scaling past the critical path delay using VOS results in a gradual increase in error rate, rather than a complete system failure. If the incurred error rate is within an acceptable level, VOS can be regarded as an efficiency-quality tradeoff technique. This run-time technique produces ILM errors, because the errors only occur

when the worst-case delay is triggered, which is a rare event. But once an error occurs, it typically affects the MSB first, resulting in a large magnitude error.

Not all circuits are well suited for VOS. For speed-optimized circuits such as KSAs, most of their timing paths have timing slacks very close to the critical path so that they can operate at high-speed. If we plot the timing slacks of all timing paths on a histogram, we will see a very narrow band of timing slacks just above zero (Figure 2.9). Therefore even if the voltage is lowered by a small amount, a large number of timing paths will be affected, resulting in intolerable errors. On the other hand, RCA has a much wider timing histogram due to its linear structure. As a result, when we lower V_{dd} , only a small number of paths will fail to meet their setup time and the result may still be sufficiently accurate.

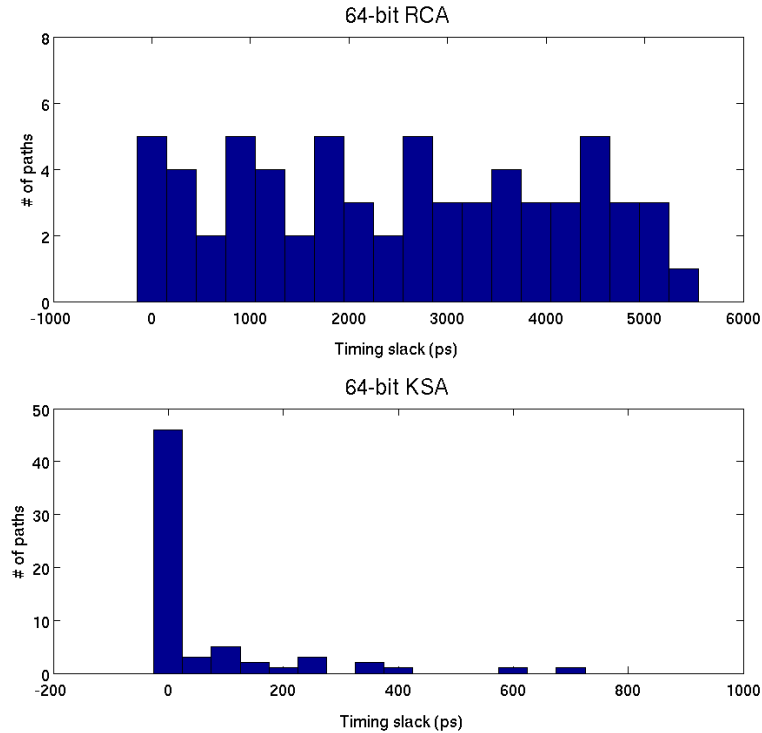


Figure 2.9: Timing slack histograms of RCA and KSA.

Reduced-precision adder

The numerical precision of a system is commonly used to adjust the efficiency-quality balance of that system. When we reduce the precision of a system (e.g., from 32-bit to 24-bit), we are effectively truncating the lower bits of the output. A reduced-precision adder is an adder that truncates the lower bits of the inputs in order to process higher-precision data. For example, a 24-bit adder used to process 32-bit data can still function by truncating the bottom 8 bits of the data. Therefore the errors resulting from a reduced-precision adder bear FSM characteristics: higher bits are correct while lower bits are truncated.

Analysis of imprecise adders

We pick the four aforementioned IHW adders and list them in Table 2.7. Figure 2.10 and Figure 2.11 show the error characteristics of those adders. All four types of adders are tested with 2 million random inputs drawn from a uniform distribution on the interval $(-0.5, +0.5)$. The number format is 2 bits (including the sign bit) before the decimal point and 30 bits after. Delay and error data are collected by tuning the appropriate parameters of the adder (Table 2.7). The delays are normalized to the delay of each adder in its precise form (i.e., no errors). In Figure 2.10, for the ILM error adders (ACA and VOS adder), error frequency drops exponentially as adder delay increases (approaching the delay of the precise adder), while the worst observed error magnitude remains nearly constant (Figure 2.11). The abrupt drop of ACA's error magnitude near delay = 1 is due to the extremely low error frequency; two million inputs are insufficient to induce a single error. For FSM error adders (METAII and reduced-precision adder), both error frequency and error magnitude decrease with increasing delay, but their error frequency decreases much more slowly

(linearly) than ILM error adders. A special case is the reduced-precision adder, whose error frequency remains constant, because it is almost always incorrect.

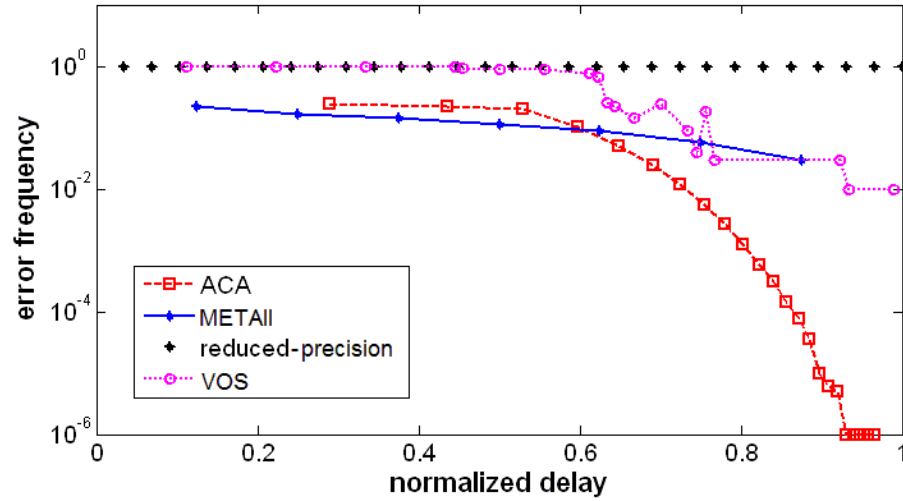


Figure 2.10: Error frequency of various imprecise adders.

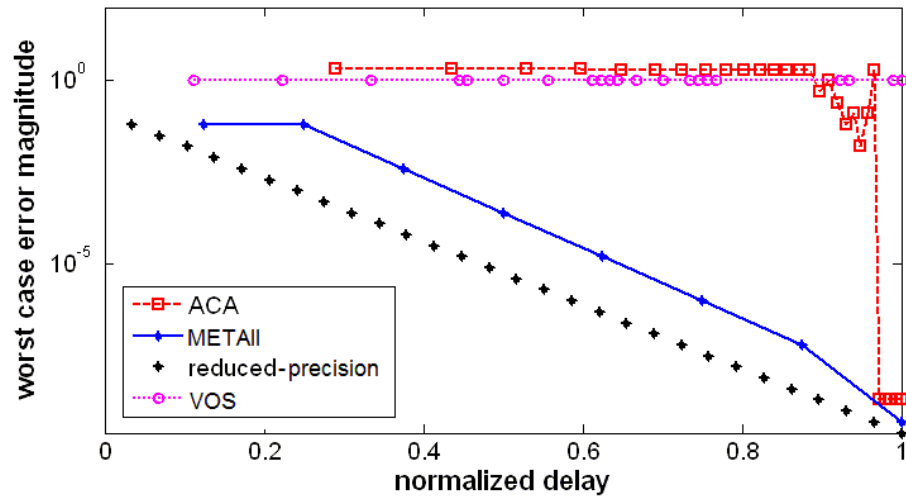


Figure 2.11: Error magnitude of various imprecise adders.

Notice that the error frequency and the worst case magnitude can be derived mathematically or extracted experimentally from Monte Carlo simulations. The error statistics

Table 2.7: Common imprecise adders and their error classifications.

Adder	Parameters	Error Classification
ACA	K	ILM
METAI	BPB, L	FSM
reduced-precision adder	data width (W)	FSM
VOS adder	V_{dd}	ILM

in Figure 2.10 and 2.11 are obtained from Monte Carlo simulations. They do not represent the theoretical bounds on error frequency or error magnitude, as some errors are so rare that they may not occur in a simulation with a small number of random samples. The theoretical bounds, however, can be derived using classical range analysis methods, such as Interval Arithmetic [35] and Affine Arithmetic [36]. Development of efficient and accurate error estimation method will be presented in Chapter 3.

Improving imprecise adders

The original ACA and METAI designs do exhibit a weakness. For simplicity, both designs use a constant zero as the carry-in at the cut-off point of the critical path, but this leads to negatively-biased errors because zero is an underestimation of the carry-in bit. Similarly, constant one will produce positively-biased errors. One possible improvement is to take the carry-in from the bit immediately before the propagate chain. For ACA, that means the propagate chain formed by $A_i \cdots A_{i-K+1}$ and $B_i \cdots B_{i-K+1}$ will take A_{i-K} (or B_{i-K}) as the carry-in. For METAI, it means the carry bit across blocks will be taken from the highest bit in the previous block. If the inputs are random during the computation,

those carry-in bits have a equal probability of being zero or one. This will eventually produce an unbiased error distribution in the sum. Table 2.8 is obtained from simulating the summation of 20 numbers randomly drawn from $[-0.5, 0.5]$ using the METAII adder ($BPB = 8, L = 4$). This anti-biasing technique notably reduces both the error rate and the mean error magnitude of the adder.

Table 2.8: Impact of anti-biasing on the error rate and magnitude of a METAII adder.

Metrics	without anti-biasing	with anti-biasing
Error rate	12.3%	6.9%
Mean error magnitude	6.3×10^{-8}	3.3×10^{-8}

Imprecise multipliers

Despite the lack of imprecise multipliers in the literature, it is possible to build imprecise multipliers based on imprecise adders. A typical multiplier consists of three stages: partial product generation, partial product accumulation, and a final stage adder [37]. The idea of building an imprecise multiplier is simple: replace the final stage adder with an imprecise adder. Therefore the ACA and METAII adders will yield corresponding ACA and METAII multipliers. For the other two stages, we adopt the popular simple partial product generation (shifted versions of the multiplicand without recoding) [37] and Wallace-tree partial product accumulator (3:2 compressor tree) [38]. These choices will influence the actual energy of the produced multiplier but they do not affect the ability to perform energy-quality tradeoffs.

Table 2.9 compares the average energy per operation (E/op), critical path delays, and

energy-delay products of various precise and imprecise adders and multipliers. They are all synthesized to their respective critical path delays. Average energy is obtained by performing 1,000 random additions for adders and 100 random multiplications for multipliers. As can be seen in the table, imprecise ALUs consume significantly less energy than their precise counterparts due to their simplified logic structures. At runtime this translates to less switching activity and lower leakage. Imprecise ALUs also have lower delays. Therefore, they can achieve even greater energy savings through voltage scaling (data not shown in the table).

Table 2.9: Average energy per operation, critical path delay, and energy-delay product (EDP) of precise and imprecise ALUs.

ALU	E/op (pJ)	Delay (ns)	EDP (pJ·ns)
KSA64	8.47	0.8	6.776
ACA64 (K=16)	6.58	0.7	4.606
RCA64	5.48	5.3	29.044
METAII (BPB=4, L=4)	3.07	0.8	2.456
MULT64_KSA	413.18	2.6	1,074.268
MULT64_ACA (K=32)	401.57	2.5	1,003.925
MULT64_RCA	174.8	11.1	1,940.28
MULT64_METAII (BPB=4, L=4)	384.96	2.3	885.408

2.6.3 Implementing other elementary functions with CORDIC

CORDIC (Coordinate Rotation Digital Computer) [39] is an algorithm for computing many elementary functions with repeated shifts and additions. Realizable functions include

multiplication, division, square root, trigonometric, logarithmic, and exponential functions. It is widely used in portable electronics due to its small hardware footprint. By replacing the adder inside a CORDIC core with an imprecise adder, CORDIC can effectively serve as an imprecise arithmetic core in larger systems. Since the adder in a CORDIC system accounts for the vast majority of the computation, this replacement also has the biggest impact on quality and energy. We implemented the *sine* and *sqrt* functions with IHW-based CORDIC algorithm and achieved an 11%~19% energy-delay product (EDP) reduction when the output precision was relaxed from 32-bit to 24-bit [40]. Figure 2.12 compares the energy-delay product of CORDIC implementation using different IHW adders. We observed that simply reducing the precision of a CORDIC system actually yields lower energy at the same quality level, compared to replacing the precise adder with an ACA or METAIL adder. The reason is that the number of iteration performed by a CORDIC system is proportional to the system precision, and energy is linearly related to iteration count. Therefore a lower-precision CORDIC has a significant energy advantage over higher-precision ones, including other IHW-based CORDIC implementations. Also, the converging-operands property of the CORDIC algorithm causes the ACA adder to produce large-magnitude errors [40]. One potential solution is to forcibly reduce the number of iterations of an ACA-based CORDIC system so that large-magnitude errors are not introduced.

2.6.4 Combining structural IHW and VOS

While structural IHW offers some power savings over traditional HW, the savings are quite modest (around 10%) compared to those obtained from VOS (around 30%). It is expected that if VOS techniques are applied to structural IHW components, the power sav-

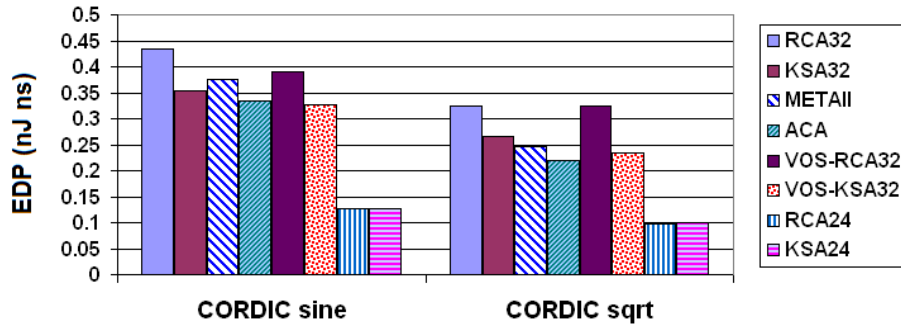


Figure 2.12: Comparison of energy-delay product (EDP) of CORDIC implemented with different IHW adders.

ings could increase significantly. Structural IHW has shorter critical paths than traditional HW, thus allowing VOS to be performed at lower voltages without causing serious errors (Figure 2.13). This section will demonstrate the benefits of the combined use of IHW and VOS on adders and multipliers.

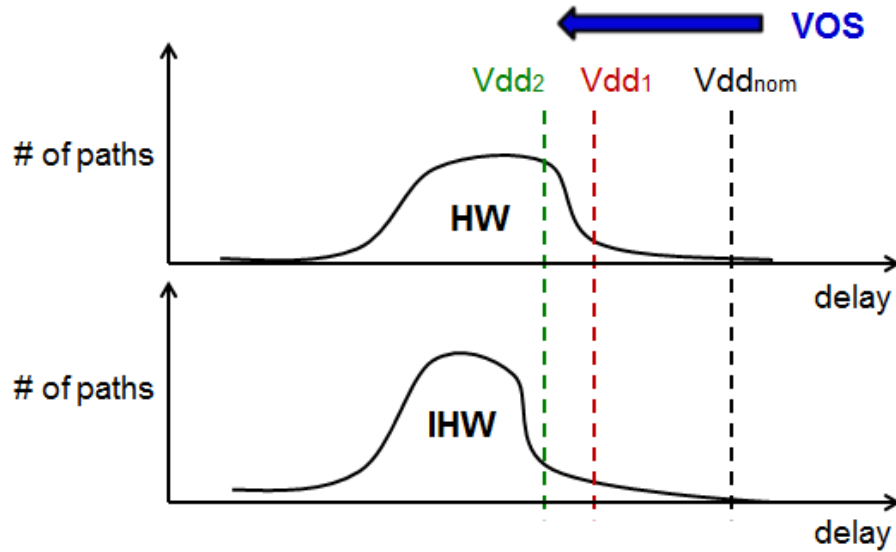


Figure 2.13: Structural IHW can be operated under deeper VOS than traditional HW, reducing power even further.

We tested various types of precise and imprecise arithmetic circuits under VOS (Table 2.10) and recorded their power consumption and result quality. To measure power, we first

Table 2.10: Units tested under combined use of IHW and VOS (with critical path delays in brackets).

Adder	Adder Delay (ns)	Multiplier	Multiplier Delay (ns)
KSA64	0.8	MUL-KSA	2.6
ACA64	0.7	MUL-ACA	2.5
RCA64	5.3	MUL-RCA	11.1
METAII64	0.8	MUL-METAII	2.3

synthesize each circuit according to its critical path delay under nominal V_{dd} (1.2V). Then SPICE simulation is performed on the netlist over a range of V_{dd} values (0.45V~1.2V) with random inputs. In each imprecise and precise hardware pair, since the IHW usually has smaller delay than the precise HW, it is simulated at the frequency that matches the precise HW to take advantage of the voltage-frequency scaling effect. Power is calculated as the product of the average current and V_{dd} . The error rate is calculated by comparing the actual circuit outputs with the accurate outputs, and we take $(1 - \text{error rate})$ as the quality metric, although it can be substituted with other expressions. Figure 2.14 compares the quality-power (Q-P) curves of IHW+VOS designs and simple VOS designs (applying VOS directly on precise HW).

Each data point is obtained at a different V_{dd} , starting from nominal V_{dd} on the right to near-threshold V_{dd} on the left. As can be seen, simple VOS designs are superior to the IHW-VOS combination at high-power, high-quality region. This is because the IHW-VOS combination cannot produce error-free outputs at nominal V_{dd} while precise HW can. But as V_{dd} is lowered (to meet tighter power requirements), the quality of simple VOS designs starts to deteriorate earlier than the combination designs. The shaded area in each graph is

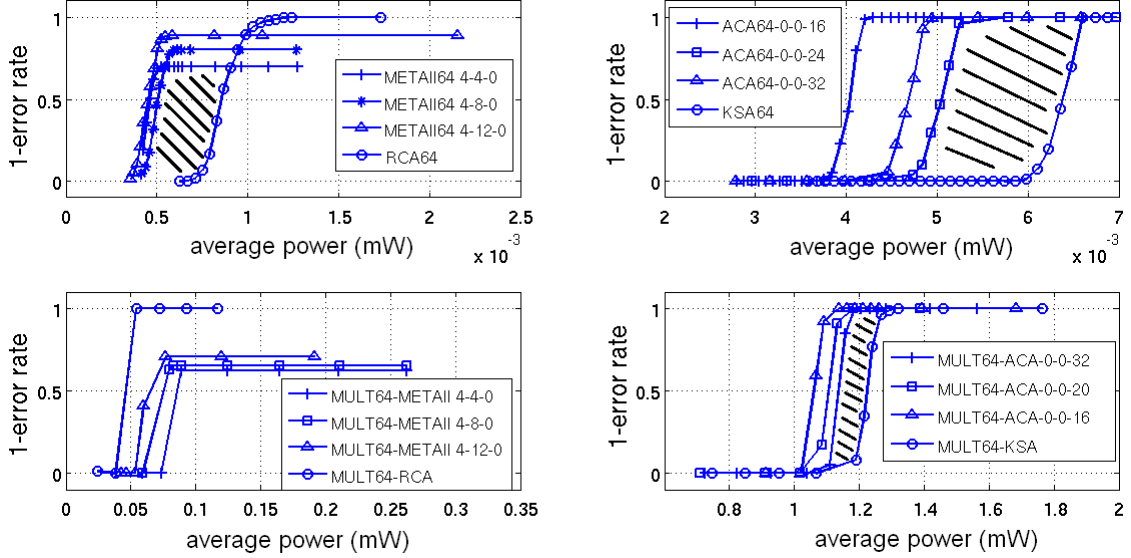


Figure 2.14: Quality-power curves of the IHW-VOS combination designs. The three numbers following the names of IHW adders refer to their design parameters. The format is $BPB-L-K$. When a parameter is not in use, it is set to zero.

where the combination outperforms simple VOS (i.e., consuming less power at the same quality level). For example, METAI64 consumes up to 46% less power than RCA64 at the iso-quality level of error rate = 69%. With the exception of the METAI-based multiplier (MULT64_METAI), the IHW-VOS combination has a clear advantage over VOS in the low-power, low-quality region. This is due to the fact that IHW has a smaller delay which allows VOS to scale to an even lower V_{dd} .

The optimal Q-P curve of an arbitrary IHW design is thus the *supremum*¹ of the two curves. It is a piecewise function—it follows the simple VOS curve at high power regions and switches to the curve of the IHW-VOS combination when the power constraint is below a certain knee point. That knee point (where the combination curve and the pure VOS curve intersect) is influenced by the structural parameters of IHW (Figure 2.14). By changing

¹The supremum of curve A and curve B is the lowest possible curve that is above both A and B. More formally, the supremum (*sup*) of two functions $f(x)$ and $g(x)$ is defined as $\sup(f(x), g(x)) = \max(f(x), g(x))$.

those parameters, designers can fine tune the balance between quality and efficiency.

2.7 Conclusions

In this chapter, we introduced the concept of IHW and studied the behavior of various IHW designs. Most IHW designs produce one of two types of errors: infrequent-large magnitude errors or frequent-small magnitude errors. We focused on two types of IHW designs, fidelity-compromising transformations and imprecise adders and multipliers. They represent IHW at the algorithmic level and the circuit level and exhibit both ILM and FSM errors. They will be used in the case studies in Chapter 4.

Chapter 3

Static Error Estimation

The benefit of IHW compared to conventional HW is determined by (1) the increased efficiency offered by the imprecise computation, and (2) the amount of error introduced in the output. Many CAD tools can evaluate common efficiency metrics of an integrated circuit, such as area, power and performance. However, choices are scarce when it comes to quality evaluation i.e., determining how much the imprecise output differs from the precise output. Typically a system with IHW components is simulated with random inputs in order to obtain an output profile, which is then compared to the output profile of a precise implementation to produce the quality metric. There is a fundamental drawback to this approach: the simulation time grows exponentially with data width and computation length. For example, a length-10 DOT-PRODUCT¹ with 32-bit numbers would require $32^{20} \approx 1.3 \times 10^{30}$ different input vectors to cover the entire input space. Due to the relaxed accuracy requirement, the IHW design space is much larger than the traditional precise hardware design space, thus fast error estimation is crucial to the exploration of

¹ $DOT - PRODUCT(X, Y) = \sum_{i=1}^n x_i \cdot y_i$

the space. This chapter presents two methods for simulation-free analytic modeling of the statistical error distributions and error bounds introduced by IHW. These methods are based on Interval Arithmetic and Affine Arithmetic and are modified to handle the characteristics of IHW errors. Compared to simulations, they provide orders-of-magnitude speedup and comparable accuracy.

3.1 Error as a Distribution

The IHW components described in Section 2.6 all belong to the class of deterministic IHW. Although in deterministic IHW the output error is a deterministic function of the input, many applications are concerned not with individual error values, but with the global error statistics (e.g., error rate) under typical workloads. For this purpose, such errors can be modeled as a random signal added to the output. This error signal follows a certain distribution, and can be considered independent of the input because we are only concerned with error statistics. Section 2.2.2 has already presented a model (PMF) to represent the distribution of discrete errors. Errors in a floating-point system can be similarly characterized by a probability density function (PDF). In addition to errors, PMFs and PDFs can also model the distribution of any data during computation.

3.2 Interval Arithmetic and Affine Arithmetic

Two classic methods of estimating variable ranges during numerical computations are *Interval Arithmetic (IA)* [35] and *Affine Arithmetic (AA)* [36]. IA uses a single interval $[x] = [x_l, x_r]$ to represent any variable. This range is guaranteed to contain the extreme

values of that variable during computation. The basic operation of IA is simply a mapping from the input interval(s) to the output interval: $[z] = f([x])$ or $[z] = f([x], [y])$. For example, addition and multiplication between IA forms should obey the following rules:

$$\begin{aligned} [x_{l1}, x_{r1}] + [x_{l2}, x_{r2}] &= [x_{l1} + x_{l2}, x_{r1} + x_{r2}] \\ [x_{l1}, x_{r1}] \cdot [x_{l2}, x_{r2}] &= [\min(x_{l1}x_{l2}, x_{l1}x_{r2}, x_{r1}x_{l2}, x_{r1}x_{r2}), \\ &\quad \max(x_{l1}x_{l2}, x_{l1}x_{r2}, x_{r1}x_{l2}, x_{r1}x_{r2})] \end{aligned} \quad (3.1)$$

These operations are easy to compute and are pessimistic (the actual range may be smaller). For example, assume $x, y \in [-1, 1]$ and we want to compute the range of $A + B$ using IA, where $A = 2x + y$, and $B = x - 2y$:

$$\begin{aligned} [A] &= [2x + y] = [-3, 3] \\ [B] &= [x - 2y] = [-3, 3] \\ [A + B] &= [-3, 3] + [-3, 3] = [-6, 6] \end{aligned}$$

But the actual range should be $[A + B] = [2x + y + x - 2y] = [3x - y] = [-4, 4]$. Since IA does not consider correlations between variables, the computed range can be larger than the actual range if variable correlation is present, causing overestimation.

AA uses an *affine form*: $\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \dots + x_n\varepsilon_n$, where x_0 is the central (mean) value of the distribution, ε_i are independent uniformly-distributed variables in the range $[-1, 1]$ and $x_1 \dots x_n$ are coefficients. AA improves on the accuracy of IA by considering the first-order correlation of error signals. By sharing error symbols (ε_i) between affine forms, AA allows the compounding and cancellation of common error symbols so that the first-order variable correlation is properly handled. AA generally produces smaller ranges than IA but requires more complex computations. In the previous example, if AA

method is chosen instead of IA, the result would be $A + B = (2x + y) + (x - 2y) = (2\varepsilon_1 + \varepsilon_2) + (\varepsilon_1 - 2\varepsilon_2) = 3\varepsilon_1 - \varepsilon_2 = [-4, 4]$, which is accurate.

Multiplication between AA forms is calculated as follows. If

$$\hat{x} = x_0 + \sum_{i=1}^n x_i \varepsilon_i, \quad \hat{y} = y_0 + \sum_{i=1}^n y_i \varepsilon_i$$

then

$$\hat{z} = \hat{x}\hat{y} = x_0 y_0 + \sum_{i=1}^n (x_0 y_i + y_0 x_i) \varepsilon_i + \left(\sum_{i=1}^n |x_i| \sum_{i=1}^n |y_i| \right) \varepsilon_k \quad (3.2)$$

where ε_k is a newly introduced error symbol.

Among AA operations, only linear operations (e.g., addition, subtraction, and scaling by a constant) are *affine operations* which result in a perfect affine form. Multiplication and division are *non-affine operations* whose results cannot be exactly represented in an affine form. Approximations must be performed to convert the result into a closest affine form, but overestimation and underestimation can occur. In certain situations, the range estimates of AA can be worse than those of IA. For example, for two variables A and B uniformly distributed in the interval of $[-1, 0]$ and $[0, 1]$ respectively, their product in affine form is represented by the form $(-0.5 + 0.5\varepsilon_1)(0.5 + 0.5\varepsilon_2) = -0.25 + 0.25\varepsilon_1 - 0.25\varepsilon_2 + 0.25\varepsilon_3$ which spans the range $[-1, 0.5]$, but the actual range of the product is $[-1, 0]$. Kolev [41] proposed a modification to the AA multiplication rule. If

$$\hat{x} = x_0 + \sum_{i=1}^n x_i \varepsilon_i, \quad \hat{y} = y_0 + \sum_{i=1}^n y_i \varepsilon_i$$

then

$$\hat{z} = \hat{x}\hat{y} = z_0 + \sum_{i=1}^{n+1} z_i \varepsilon_i,$$

where

$$\begin{aligned}
 c &= 0.5 \sum_{i=1}^n x_i y_i \\
 z_0 &= x_0 y_0 + c \\
 z_i &= x_0 y_i + y_0 x_i \quad (i = 1 \cdots n) \\
 z_{n+1} &= \left(\sum_{i=1}^n |x_i| \sum_{i=1}^n |y_i| \right) - |c|
 \end{aligned} \tag{3.3}$$

This modification leads to smaller overestimation compared to the original method (Eq. 3.2). Take the same example above $(-0.5 + 0.5\varepsilon_1)(0.5 + 0.5\varepsilon_2)$, using Kolev's method the result will be $-0.125 + 0.125\varepsilon_3$, which translates to $[-0.25, 0]$. While it is still not the accurate range $[-1, 0]$, its overestimation is much smaller compared to $[-1, 0.5]$.

However, both IA and AA forms are only capable of representing *symmetric* distributions. Highly asymmetric distributions, such as the errors produced by IHW (Figure 2.3), are not representable by either IA or AA forms. In order to solve this problem, we propose some extensions to IA and AA. Instead of representing the entire distribution with a single interval or affine form, we use *multiple* intervals or affine forms.

3.3 Modified Interval Arithmetic

Modified Interval Arithmetic (MIA) [42] extends IA by using multiple intervals to represent a distribution to enhance accuracy. MIA can be easily mapped to the PMF: each PMF bar corresponds to one interval in MIA form. Thus, the MIA representation of a variable with a distribution between $[-2^p, 2^q]$ can be expressed as:

$$MIA_X(n) = \left\{ \begin{array}{l} P(-2^{p-n+1} \leq X \leq -2^{p-n}), \text{ if } 1 \leq n \leq p - \varepsilon \\ P(-2^\varepsilon \leq X \leq 0), \text{ if } n = p - \varepsilon + 1 \\ P(0 \leq X \leq 2^\varepsilon), \text{ if } n = p - \varepsilon + 2 \\ P(2^{2\varepsilon+n-p-3} \leq X \leq 2^{2\varepsilon+n-p-2}), \text{ if } p - \varepsilon + 3 \leq n \leq p + q - 2\varepsilon + 2 \end{array} \right\},$$

where $n \in [1, p + q - 2\varepsilon + 2]$, $p, q, \varepsilon \in \mathbb{Z}$

In the above notation, 2^ε represents the maximum resolution of this particular MIA form because any implementation of the MIA form can only contain a finite number of terms. There are a total of $p + q - 2\varepsilon + 2$ terms in the above MIA form. In practice 2^ε can be set to an error magnitude level that has a negligible effect on output quality. Unit in the last place (ULP) is usually a good choice for 2^ε . The upper bound 2^q and the lower bound -2^p should be set to match the maximum and minimum value in the number system of choice, e.g., a 32-bit signed 2's complement number with 7-bit integer part and 24-bit fractional part (denoted as 1_7_24) should have $p = q = 7$ and $\varepsilon = -24$.

When we use an MIA form to represent an error distribution, the total error probability is given by $\sum MIA(n)$.

For operations between two MIA forms MIA_a and MIA_b , we need to perform pairwise IA operation from both MIAs: every interval from the MIA_a must perform that operation with every interval from MIA_b . This is based on the assumption that the two input MIAs are uncorrelated. Variables a and b can be independently chosen from intervals of their respective MIA and the joint probability should be the product of the probabilities of both intervals. Based on the rules of IA operations (Eq. 3.1), each resulting IA interval will be wider than both input IA intervals and may not align exactly to the 2^ε MIA bound-

aries. Therefore an IA-to-MIA conversion will be needed. The process is described below (Algorithm 1): every MIA interval fully or partially contained in the IA interval will get the probability of d_i/d where d_i is the width of the overlapping interval and d is the width of the IA interval.

Algorithm 1 IA-to-MIA conversion: $IA_x \rightarrow MIA_x$

 $IA_x = [x_l, x_r]$
for $MIA_x(i) = [x_{i,l}, x_{i,r}] \in MIA_x$, where $x_{i,l}, x_{i,r}$ are integer powers of 2 **do**
if $x_{i,l} \leq x_l$ and $x_l \leq x_{i,r} \leq x_r$ **then**

$$P(i) = \frac{x_{i,r} - x_l}{x_r - x_l}$$

else if $x_l \leq x_{i,l} \leq x_{i,r} \leq x_r$ **then**

$$P(i) = \frac{x_{i,r} - x_{i,l}}{x_r - x_l}$$

else if $x_l \leq x_{i,l} \leq x_r$ and $x_{i,r} \leq x_r$ **then**

$$P(i) = \frac{x_r - x_{i,l}}{x_r - x_l}$$

else

$$P(i) = 0$$

end if
end for

After conversion, a new MIA form is produced for every pair of IA intervals in the original operands. These intermediate MIA forms are merged to form the final result MIA. While merging, the probabilities of the same intervals will be summed to produce the final probability. The following pseudo-code (Algorithm 2) describes the calculation of $MIA_c = MIA_a + MIA_b$. MIA multiplication is similar to MIA addition—the only difference is the replacement of pair-wise IA additions with pair-wise IA multiplications.

Algorithm 2 Calculate $MIA_c \leftarrow MIA_a + MIA_b$

```

for  $MIA_a(i) \in MIA_a$  do
    for  $MIA_b(j) \in MIA_b$  do
         $MIA_{c,i,j} \leftarrow MIA_a(i) + MIA_b(j)$  (IA addition)
    end for
end for

for all  $n$  do
     $MIA_c(n) \leftarrow 0$ 
    for all  $i$  do
        for all  $j$  do
             $MIA_c(n) \leftarrow MIA_c(n) + MIA_{c,i,j}(n)$  (MIA merging)
        end for
    end for
end for

```

When performing pair-wise IA operations, we suggest making some modifications to the original rules to improve modeling accuracy. These modifications are based on the fact that the sum or product of two uniformly-distributed variables is usually *not* a uniformly-distributed variable. For example, the sum of two uniform intervals on $[-2, -1]$ and $[0.5, 1]$ has a trapezoid-shaped distribution (Figure 3.1).

Using a single $[-1.5, 0]$ interval to describe the result leads to loss of information. A more accurate way is to calculate the areas between any two consecutive 2^k markers and convert that area, which is also a probability, to a new MIA term. For example, if the above trapezoid-shaped distribution is to be represented in MIA with a maximum resolution of

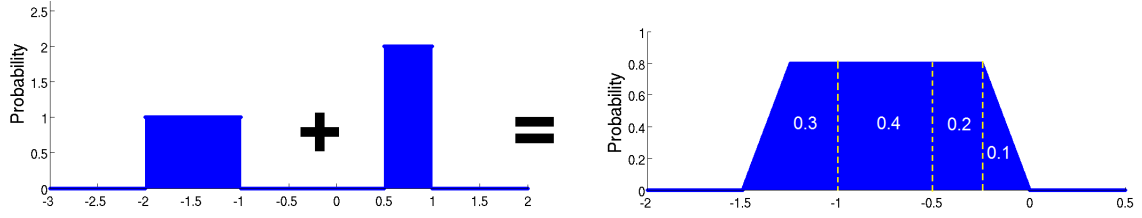


Figure 3.1: Probability distribution of the sum of two uniformly-distributed variables.

0.25, then

$$MIA_{sum}(x) = \begin{cases} 0.3, & -2 \leq x \leq -1 \\ 0.4, & -1 \leq x \leq -0.5 \\ 0.2, & -0.5 \leq x \leq -0.25 \\ 0.1, & -0.25 \leq x \leq 0 \end{cases}$$

A more compact notation is

$$MIA_{sum}(x) = [0.3 \quad 0.4 \quad 0.2 \quad 0.1 \quad \varepsilon]$$

where ε denotes the position of zero. The key is to first derive the output distribution as a function of the input distribution and then convert it to MIA format. Since format conversion is a lossy process, it is more accurate to delay the conversion for as long as possible.

Obtaining the relationship between input and output distributions is a well-studied problem in probability theory. Assuming both inputs are independently distributed, the results of the four basic precise arithmetic operations (addition, subtraction, multiplication and

division) are listed below:

$$\textbf{Precise addition: } Z = A + B \quad (3.4)$$

$$PMF_Z = PMF_A * PMF_B \text{ (convolution)}$$

$$\textbf{Precise subtraction: } Z = A - B \quad (3.5)$$

$$PMF_Z = PMF_A \text{ xcorr } PMF_B \text{ (cross-correlation)}$$

$$\textbf{Precise multiplication: } Z = A \times B \quad (3.6)$$

$$PMF_Z = PMF_A ** PMF_B \text{ (multiplication-correlation)}$$

$$\textbf{Precise division: } Z = A/B \quad (3.7)$$

$$PMF_Z = PMF_A // PMF_B \text{ (division-correlation)}$$

$$\text{Define multiplication-convolution: } f * g(t) \equiv \int_{-\infty}^{\infty} f(\tau) g\left(\frac{t}{\tau}\right) d\tau$$

$$\text{Define division-convolution: } f // g(t) \equiv \int_{-\infty}^{\infty} f(\tau) g(t\tau) d\tau$$

Consider the addition operation as an example. In digital signal processing, *convolution* is defined as

$$f * g(t) \equiv \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

Here $f(x)$ and $g(x)$ are PMFs of two independent variables to be added. The PMF of their sum $h = f + g$ therefore has the following property:

$$h(a + b) = f(a)g(b)$$

The convolution between f and g precisely aggregates all possible cases where both inputs sum to the same value. To confirm this result, we notice that the convolution of two square

waves of equal widths results in a triangle wave, and the distribution of the sum of two standard uniform random variables also has a triangle distribution. Eq. 3.4 simply means that *the probability distribution of the sum of two independent random variables is the convolution of their individual distributions* [43].

These rules allow us to perform MIA operations with higher accuracy. For example, assuming the maximum resolution (2^ϵ) is 2^{-4} , we want to estimate the sum of two independent uniformly-distributed variables (a and b), both in $[-1, 1]$. First, their MIAs can be represented as:

$$MIA_a(x) = MIA_b(x) = [0.25 \quad 0.125 \quad 0.0625 \quad 0.03125 \quad 0.03125 \quad \epsilon \\ 0.03125 \quad 0.03125 \quad 0.0625 \quad 0.125 \quad 0.25]$$

Using the MIA addition rules described above, we can obtain the sum distribution as

$$MIA_{a+b, MIA}(x) = [0.1250 \quad 0.1594 \quad 0.0961 \quad 0.0556 \quad 0.0318 \quad 0.0321 \quad \epsilon \\ 0.0321 \quad 0.0318 \quad 0.0556 \quad 0.0961 \quad 0.1594 \quad 0.1250]$$

while the theoretical (triangle) distribution is

$$MIA_{a+b, theoretical}(x) = [0.1250 \quad 0.1563 \quad 0.1016 \quad 0.0566 \quad 0.0295 \quad 0.0305 \quad \epsilon \\ 0.0305 \quad 0.0295 \quad 0.0566 \quad 0.1016 \quad 0.1563 \quad 0.1250]$$

But a simple IA addition will produce a uniform interval on $[-2, 2]$ which translates to

$$MIA_{a+b, IA}(x) = [0.2500 \quad 0.1250 \quad 0.0625 \quad 0.0313 \quad 0.0156 \quad 0.0156 \quad \epsilon \\ 0.0156 \quad 0.0156 \quad 0.0313 \quad 0.0625 \quad 0.1250 \quad 0.2500]$$

A plot of these MIAs on a PMF bar chart (Figure 3.2) shows that the MIA-based estimation is much more accurate than IA-based estimation. This higher accuracy is due to

two reasons. First, MIA is a finer-grained representation of a distribution than IA. The shape of a non-uniform distribution is better preserved with MIA. Second, the pair-wise IA operations also use accurate rules derived from probability theory to preserve the shape of the final distribution.

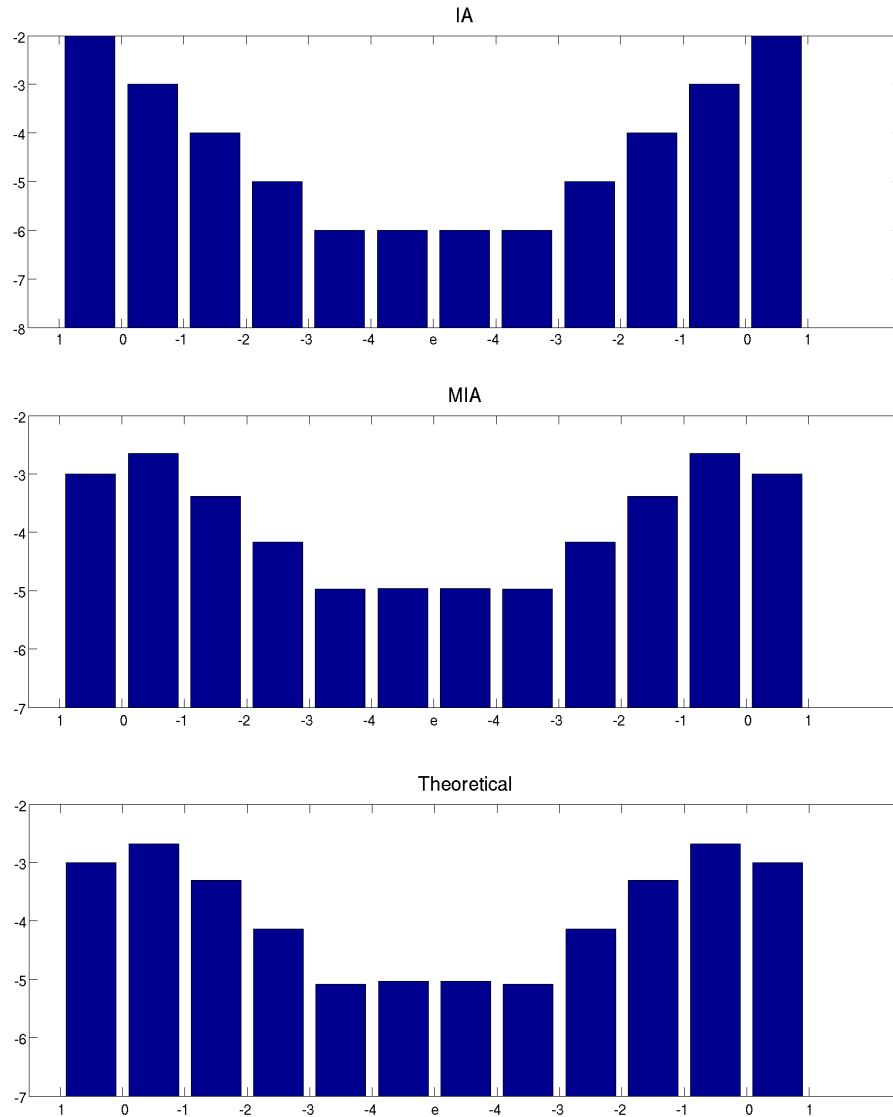


Figure 3.2: Estimated MIA of the sum of two independent uniform variables in $[-1, 1]$ using IA, MIA methods, and the theoretical MIA.

3.4 Modified Affine Arithmetic

Similar to MIA, Modified Affine Arithmetic (MAA) extends AA by using multiple AA forms to represent a given distribution. The mathematical representation of MAA form is:

$$MAA_X = \begin{cases} p_1 : x_{1,0} + x_{1,1}\alpha_0 + x_{1,2}\beta_0 + \dots \\ p_2 : x_{2,0} + x_{2,1}\alpha_1 + x_{2,2}\beta_1 + \dots \\ p_3 : x_{3,0} + x_{3,1}\alpha_2 + x_{3,2}\beta_2 + \dots \\ \dots \end{cases}$$

where $x_{i,0}$ are central values, $x_{i,j}$ are scaling coefficients and α, β, \dots are error symbols. Each affine form occurs with probability p_i .

When two MAA forms operate together, every AA form of the first operates with every AA form of the second. In order to preserve the property of $\sum p_i = 1$ for every MAA form, we introduce the concept of an *exclusive set*: a group of error symbols that originate from the same distribution. Any variable which appears for the first time will produce a new exclusive set. Any two different symbols from the same exclusive set are mutually exclusive, and they are called *conflicting* symbols. Conflicting symbols are denoted by the same symbol with unique subscripts, for example, $\alpha_0, \alpha_1, \alpha_2, \dots$ belong to one exclusive set and $\beta_0, \beta_1, \beta_2, \dots$ belong to another. Through repeated and cross computations, these symbols will occur in many derived MAA forms. When two MAA forms operate, only those affine forms with no conflicting symbols are allowed to operate with each other. The reason is that symbols in the same exclusive set are originally an integral part of the same distribution (denoted as D). If a certain affine form of a derived variable contains a symbol in the exclusive set, it means this interval is the result of taking a value in that original part

of D . Having two conflicting symbols in the same affine form is thus an impossible event.

For example, consider two MAA forms:

$$MAA_A = \begin{cases} 0.5 : 1 + 0.25\varepsilon_1 \\ 0.5 : -1 + 0.5\varepsilon_2 \end{cases}$$

$$MAA_B = \begin{cases} 0.8 : 0.5 + 0.5\alpha_1 \\ 0.2 : -0.25 + \alpha_2 \end{cases}$$

We want to calculate the MAA for $(A+B)(A-B)$. First we express $A+B$ and $A-B$ in MAA form:

$$MAA_{A+B} = \begin{cases} 0.4 : 1.5 + 0.25\varepsilon_1 + 0.5\alpha_1 & (a) \\ 0.1 : 0.75 + 0.25\varepsilon_1 + \alpha_2 & (b) \\ 0.4 : -0.5 + 0.5\varepsilon_2 + 0.5\alpha_1 & (c) \\ 0.1 : -1.25 + 0.5\varepsilon_2 + \alpha_2 & (d) \end{cases} \quad (3.8)$$

$$MAA_{A-B} = \begin{cases} 0.4 : 0.5 + 0.25\varepsilon_1 - 0.5\alpha_1 & (a) \\ 0.1 : 1.25 + 0.25\varepsilon_1 - \alpha_2 & (b) \\ 0.4 : -1.5 + 0.5\varepsilon_2 - 0.5\alpha_1 & (c) \\ 0.1 : -0.75 + 0.5\varepsilon_2 - \alpha_2 & (d) \end{cases} \quad (3.9)$$

The multiplication procedure is as follows:

$$MAA_{(A+B)(A-B)} = \begin{cases} 0.4 : 3.8(a) \times 3.9(a) \\ 0.1 : 3.8(b) \times 3.9(b) \\ 0.4 : 3.8(c) \times 3.9(c) \\ 0.1 : 3.8(d) \times 3.9(d) \end{cases} \quad (3.10)$$

$3.8(a) \times 3.9(b)$ is not allowed because it contains conflicting symbols α_1 and α_2 . Similarly $3.8(a) \times 3.9(c)$ and $3.8(a) \times 3.9(d)$ are not allowed because they contain conflicting symbols ε_1 and ε_2 . Since conflicting symbols can cause impossible events, their probabilities are instead added to the remaining non-conflicting operations. For example, between $3.8(a)$ and $3.9(a-d)$, only $3.8(a) \times 3.9(a)$ is permitted, so it has the probability $0.4 \times 1 = 0.4$, rather than $0.4 \times 0.4 = 0.16$, because the probabilities of those prohibited operations are transferred to valid operations. This is because there are correlated variables in this operation. The four AA forms of MIA_{A+B} are not independent of the four AA forms of MIA_{A-B} . The choice of an AA form in MIA_{A+B} will influence the probability of the four AA forms of MIA_{A-B} . In this case, the choice of a specific AA form of MIA_{A+B} will uniquely determine the choice of an AA form within MIA_{A-B} .

MAA has a practical issue of *storage explosion*. When we want to operate two MAA forms each containing N and M AA forms, in general the resulting MAA form will contain $N \times M$ AA forms. The exponential growth of required storage limits the practical value of the MAA technique, but this problem can be addressed in several ways. First, we can use a larger scaling factor between neighboring intervals while constructing the MAA. For example, choosing 4 instead of 2 as the scaling factor can effectively reduce the number of AA forms by half. Second, AA forms with low frequency (small p values) or low magnitude (small coefficients) can be merged into their neighboring AA forms. Both techniques carry the side effect of reduced estimation accuracy.

3.5 Error Propagation Model

In the previous section we discussed the operations of PMF under error-free conditions (with precise hardware). This section presents a primitive model for error propagation across a single IHW operation. The goal is to derive the output PMF using the input PMF and the IHW parameters.

The first step is to use a common data structure (PMF_d, PMF_e) to represent the distribution of any data during an imprecise operation. PMF_d is the error-free data PMF obtained assuming all operators are precise while PMF_e is the pure error MIA introduced by imprecise operators. The sum of PMF_d and PMF_e gives the actual (error-present) data PMF. The PMF can be realized in either MIA form or MAA form.

Next it is necessary to build a model to obtain the output $(PMF_{d_out}, PMF_{e_out})$ from the input (PMF_{d_in}, PMF_{e_in}) , see Figure 3.3. The imprecise operator (marked with $*$) will also introduce PMF_{e_op} , which can be regarded as additive noise to the system.

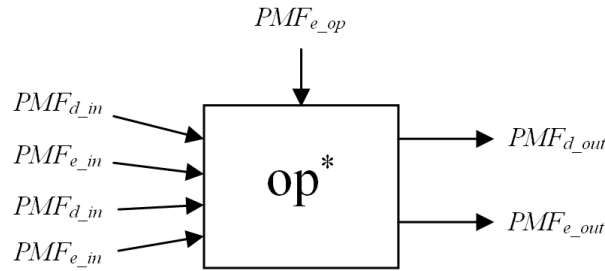


Figure 3.3: Error propagation model of an imprecise 2-operand operation.

The principal task of error propagation is to derive the relationships between these quantities for common operations (ADD, SUB, MUL, and DIV). More complex operations can be studied by decomposing them into the four basic operations. Assuming we use MIA

as the PMF implementation, the derivation results are as follows:

$$\underline{\text{ADD}} : MIA_{d.out} = MIA_{d.in1} + MIA_{d.in2}$$

$$\begin{aligned} MIA_{e.out} &= (MIA_{d.in1} + MIA_{e.in1}) + (MIA_{d.in2} + MIA_{e.in2}) \\ &\quad + MIA_{e.add} - MIA_{d.out} \\ &= MIA_{e.in1} + MIA_{e.in2} + MIA_{e.add} \end{aligned}$$

$$\underline{\text{MUL}} : MIA_{d.out} = MIA_{d.in1} \times MIA_{d.in2}$$

$$\begin{aligned} MIA_{e.out} &= (MIA_{d.in1} + MIA_{e.in1}) \times (MIA_{d.in2} + MIA_{e.in2}) \\ &\quad + MIA_{e.mul} - MIA_{d.out} \\ &= MIA_{d.in1} \times MIA_{e.in2} + MIA_{d.in2} \times MIA_{e.in1} \\ &\quad + MIA_{e.in1} \times MIA_{e.in2} + MIA_{e.mul} \end{aligned} \tag{3.11}$$

$$\underline{\text{SQUARE}} : MIA_{d.out} = MIA_{d.in}^2$$

$$\begin{aligned} MIA_{e.out} &= (MIA_{d.in} + MIA_{e.in})^2 + MIA_{e.squared} - MIA_{d.out} \\ &= MIA_{e.in}^2 + 2MIA_{d.in} \times MIA_{e.in} + MIA_{e.squared} \end{aligned}$$

In Eq. 3.11, operations between MIAs should follow the rules set forth in Section 3.3. Notice that SQUARE is separated from MUL because it cannot be obtained by simple MIA addition and multiplication. Even if X and Y have the same distribution, the distribution of X^2 will be quite different from that of $X \cdot Y$. The modeling of SQUARE relies on characterization (see the next paragraph).

$MIA_{e.add}$, $MIA_{e.mul}$ and $MIA_{e.square}$ are attributes of the imprecise operator determined by the design parameters. They can be obtained by simulation. The process of obtaining $MIA_{e.op}$ through simulation is called *characterization* of IHW. To characterize ADD (Figure 3.4), we randomly draw data from intervals of both inputs' MIAs (i.e.,

draw first operand from $[2^i, 2^{i+1}]$ and draw second operand from $[2^j, 2^{j+1}]$ and perform the imprecise operation. Simulation is made possible by creating functional models of the imprecise adders and multipliers written in Matlab. The MIA_d and MIA_e of the result are then stored into a matrix at index (i, j) . When the whole matrix is populated, we can later use it to quickly retrieve MIA_{e_op} during MIA propagation. All binary (two-input) operators can be characterized in this way. For unary operator SQUARE, the result is stored in a vector instead of a matrix and we produce two vectors for SQUARE: one for storing the error (MIA_{e_square}) and the other for storing the squared data ($MIA_{d_in}^2$ and $MIA_{e_in}^2$). IHW can be characterized *a priori* and each IHW configuration (a unique setting of BPB , L , K , and V_{dd}) needs to be characterized only once. The characterization data can then be reused many times for different input workloads.

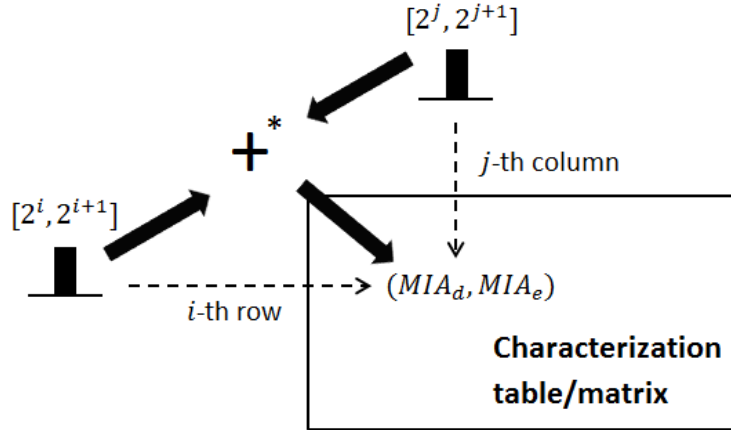


Figure 3.4: Characterization of an IHW adder.

In summary, static MIA propagation follows three steps:

- (1) Construct the characterization vector or matrix by simulating the IHW with inputs drawn from various $[\pm 2^i, \pm 2^{i+1}]$ intervals.

- (2) During propagation, use the input MIAs to look up the characterization vector or matrix to obtain $MIA_{e_{op}}$.
- (3) Apply rules in Eq. 3.11 to obtain output MIA.

Step 2 and step 3 may need to be repeated because the output MIA normally becomes the input MIA in the next round of computation. The final MIA_d and MIA_e accurately describe the data and error distribution of the kernel output and they can be used to evaluate output quality. Common quality metrics such as “error rate” and “mean error magnitude” are computed as follows:

$$\begin{aligned}\text{error rate} &= \int MIA_e(x) \\ \text{mean error magnitude} &= \int |x \cdot MIA_e(x)|\end{aligned}$$

Static MIA propagation is much faster than Monte Carlo simulations because no actual computation is performed. It is the distributions of the data (in the form of MIA) that are being propagated, rather than the actual data.

3.6 Experimental Results

For the purpose of evaluating the proposed error estimation methods, we selected two kernel functions common in multimedia, recognition and mining applications [14]. Table 3.1 summarizes the two kernels and three applications that use these kernels. The errors from imprecise implementations of the two kernels will be evaluated using both the proposed static methods and Monte Carlo simulations, after which the results will be compared. The applications will be used to evaluate the efficiency-quality co-optimization framework in Chapter 4.

Table 3.1: Percentage application runtime spent in computation kernels.

Kernel	Application	Runtime%
$DOT - PRODUCT(X, Y) = \sum_{i=1}^n x_i \cdot y_i$	Leukocyte Tracker [44]	22%
$L2 - NORM(X, Y) = \sum_{i=1}^n (x_i - y_i)^2$	SVM	98%
	K-means	49%

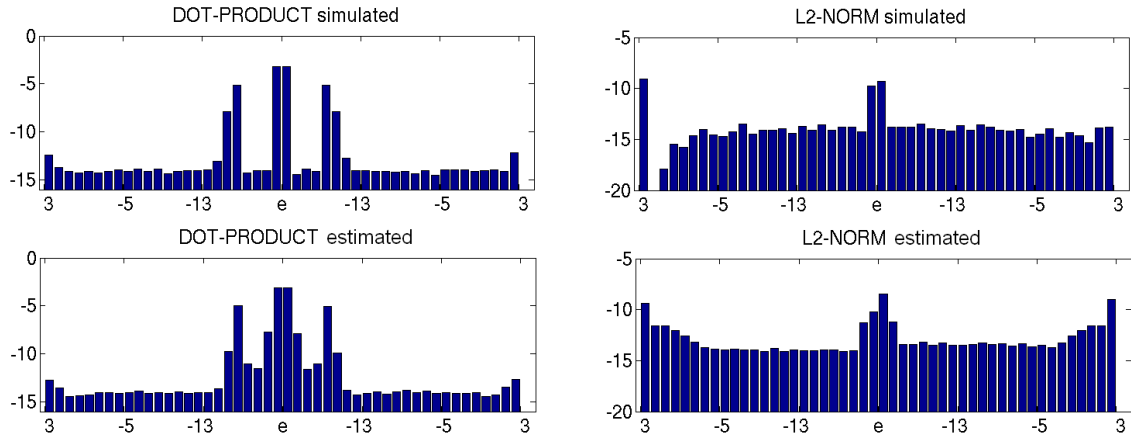


Figure 3.5: Error MIAs of DOT-PRODUCT and L2-NORM.

Figure 3.5 shows the final error MIAs after performing a size-25 DOT-PRODUCT and a size-49 L2-NORM using both Monte Carlo simulations and static estimation. DOT-PRODUCT contains an ACA adder with $K=16$ and an METAII multiplier with $BPB=8$ and $L=4$; L2-NORM contains an ACA adder with $K=16$ and an ACA multiplier with $K=24$. Table 3.2 compares the speed and accuracy of simulated and estimated error MIAs. All experiments are run on a dual-core Xeon 2.4GHz with 32GB memory. The simulation size is 500,000 and is used as the basis of comparison. As seen in the table, the speed improvement is dramatic and the simulated and estimated error distributions are very close. For

example, a Hellinger distance² of 0.05 is comparable to 1 million random samples from two uniform distributions between $[-1, 1]$ generated by the Mersenne Twister algorithm [46]. The Hellinger distance is calculated by integrating the difference of two distribution functions over the entire distribution range; therefore small-magnitude errors are not weighted as heavily as large-magnitude errors. However, from a system perspective, large-magnitude errors *should* be weighted more heavily because they have a greater impact on the output quality. Thus we believe the Hellinger distance is an accurate measure of similarity between two distributions functions. If one is more interested in comparing the small-magnitude range of two distributions, he or she just needs to confine the integration interval to the region of interest. For example, integrating over a narrow region around zero will reveal the difference in small-magnitude range more clearly than the original Hellinger distance.

Table 3.2: Speed and accuracy comparison between simulation and static estimation.

Kernel	Simulation time	Estimation time	Hellinger distance
DOT-PRODUCT	565 hr	13 s	0.07
L2-NORM	620 hr	6 s	0.04

3.7 Conclusions

In order to safely deploy IHW in a system, we need to quantitatively evaluate the errors induced by IHW implementations. This chapter proposes two analytic error estimation

²A statistical measure of similarity between two distributions—smaller values indicate higher similarity [45]

methods that offer significant speedup over simulation with reasonable estimation accuracy.

The pros and cons of the related methods are summarized in Table 3.3.

Table 3.3: Overview of error estimation methods.

Method	Simulation	MIA	MAA
Pros	Statistically accurate	Fastest, reasonably accurate	Fast if low complexity, tight bounds
Cons	Slow, bounds too tight	Loose bounds if variables correlate	Storage explosion if high complexity

Chapter 4

A Methodology for Efficiency-Quality Co-Optimization

What circuit and system designers are ultimately interested in is how to use IHW to obtain a design that better meets their design objectives. In this chapter we will present two different methodologies, one at the algorithmic level using fidelity-compromising transformations, the other at the RTL level using structural IHW adders and multipliers and variable V_{dd} . Both methodologies can solve constraint-based and cost function-based optimization problems. These optimization methodologies are fundamentally different from traditional circuit optimization methodologies in that output quality is taken into account.

4.1 For Fidelity-Compromising Transformations

As discussed in Section 2.6.1, fidelity-compromising transformation can be extremely powerful in reducing the latency of a design, but a methodology is needed to use them to

achieve design objectives, similar to a synthesis algorithm for applying delay and power optimizations. This section introduces a design optimization methodology through HDG manipulations that include fidelity-compromising transformations. A *hierarchical dependency*

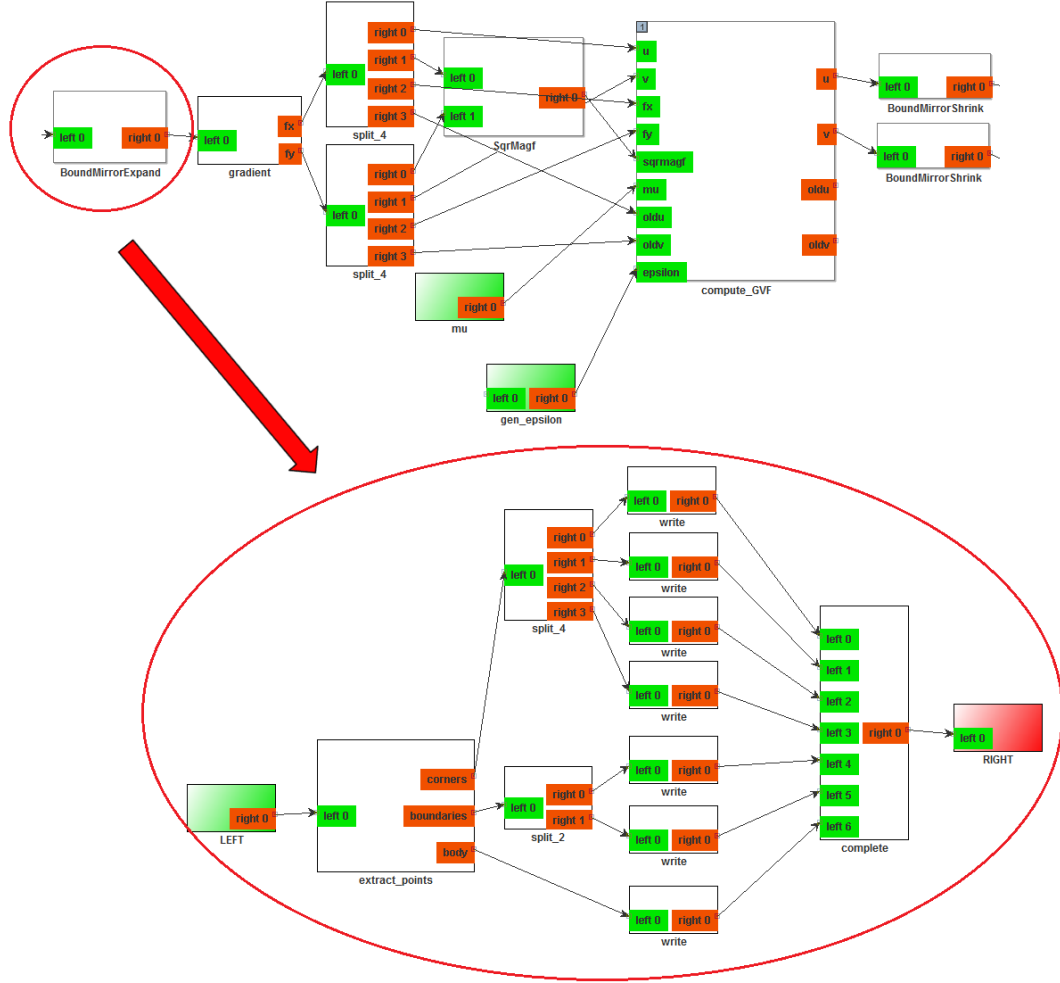


Figure 4.1: Hierarchical dependency graph.

dependency graph (HDG) [30] (Figure 4.1) is a data structure used to represent both the algorithm and its hardware implementation, enabling bi-directional communication between the algorithm and hardware designers during design space exploration. It shares some similarities with traditional task graphs [47–49] but is structured hierarchically to manage design com-

plexity. At the top level of the graph, each node represents an algorithmic task (e.g., edge detection, noise removal, etc.). Subsequent levels are similar, with nodes representing sub-tasks, all the way down to the most basic operations, such as addition and multiplication. At the bottom level, circuit designers can annotate every functional block with its HW metrics such as power, delay, and area. These metrics are automatically aggregated at higher levels. At the algorithm level, system designers can experiment with alternative algorithms and find the best tradeoff between quality and efficiency. We implemented HDG in a tool called *ColSpace*. It comes with some common library tasks and blocks pre-loaded to ease HDG development, and the library can be extended by adding custom components. The tool is available for download [50] - the current version supports only latency and throughput as efficiency metrics, with area and power to be added soon.

Suppose an image processing algorithm is already provided in HDG form, and we want to minimize latency metric under a minimum fidelity constraint. The problem can be stated as follows. Given the following:

1. an HDG describing an algorithm and its lowest latency implementation and a runnable source code,
2. a library containing applicable fidelity-compromising transformations (in the same format as Table 2.3), $T_n : (\Delta FID_n, \Delta LAT_n, p_1, p_2, \dots, p_{H_n})$ where H_n denotes the number of thresholds that can be tuned for transformation T_n ,
3. a latency evaluation routine that gives total latency LAT_{total} and a fidelity evaluation routine that gives global fidelity FID_{total} , and
4. minimum acceptable fidelity FID_{min} .

the goal of the design space exploration algorithm is to determine:

1. which transformations to apply,
2. the proper settings for each transformation, and
3. the order in which to apply those transformations so as to minimize LAT_{total} .

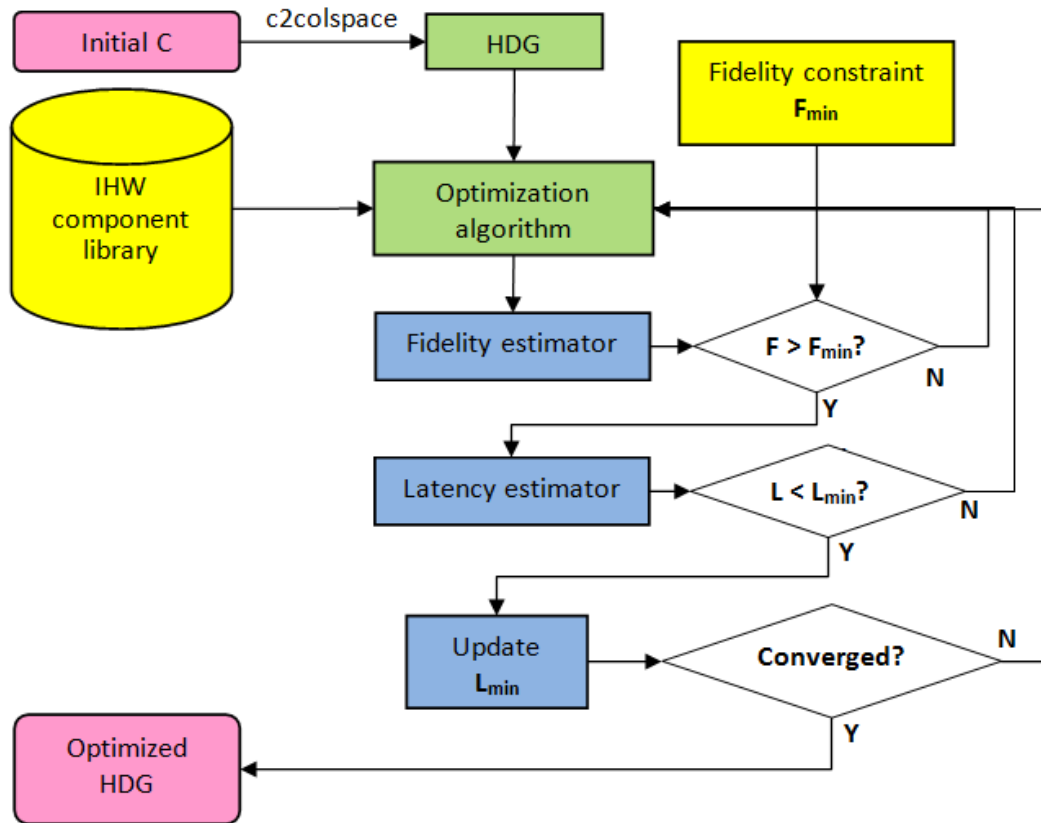


Figure 4.2: Design flow with fidelity-compromising transformations.

Figure 4.2 shows the procedure to solve this problem. To obtain the initial lowest latency implementation, an HDG is set up to represent the dependency information of the algorithm. If the original system is implemented in C, we provide a tool *c2colspace* to

convert the C source into an HDG project. Then an optimization algorithm is chosen to solve the problem. The algorithm will call a fidelity estimator and a latency estimator to determine whether a particular design point is a good candidate to direct the search. Design points are chosen by picking IHW designs from the component library. Also, the user is expected to provide a target constraint. Here a greedy algorithm is presented to showcase a typical search procedure:

- For every fidelity-compromising transformation T_n , adjust its threshold $P_n = (p_{n,1}, p_{n,2}, \dots, p_{n,H_n})$ so that the global fidelity $FID_{total} = FID_{min}$.
- While P_n is set, calculate resultant global latency LAT_{total} using profiling information.
- Find the transformation T_k that gives minimum latency, which will be returned as LAT_{min} .

If multiple threshold settings P_n can result in $FID_{total} = FID_{min}$, all of them will be tried for calculation. Only the lowest LAT_{total} is recorded.

For cases where the objective is to minimize a global cost function, such as $\frac{LAT_{total}}{FID_{total}}$ (again, a greedy algorithm is used for demonstration), the procedure is as follows:

- For every fidelity-compromising transformation T_n , adjust its threshold $P_n = (p_{n,1}, p_{n,2}, \dots, p_{n,H_n})$ so as to minimize $d\frac{LAT_{total}}{FID_{total}}$. If calculating $d\frac{LAT_{total}}{FID_{total}}$ is hard, we can apply *l'Hôpital's rule* using $d(LAT_{total})$ and $d(FID_{total})$.
- Find the transformation T_k with minimum $d\frac{LAT_{total}}{FID_{total}}$. If that value is still positive, end algorithm, return current P settings.

- Apply T_k with corresponding P_k .
- Re-evaluate FID_{total} and LAT_{total} for every transformation.

The derivatives can be calculated against a discretized domain (to be demonstrated in Section 4.1.3). As long as the step size chosen is small enough that the cost function is monotonic in this range, global optimal point can still be properly located.

There are two ways to reduce the simulation time, but both can cause inaccuracies. One is to apply only uncorrelated transformations. Transformations that are uncorrelated can be applied separately and their combined effect on LAT and FID will be the same as applying them simultaneously. Another way is to calculate a local disturbance error of the fidelity from the transformation and propagate that error across the entire design to obtain a global error estimate. Symbolic Noise Analysis (SNA) [51] can accomplish this task without simulation.

4.1.1 Latency Evaluation in HDGs

There are two types of latencies in an HDG. First, every node in the HDG is assigned a delay value (LAT). This value is defined in the library for basic nodes such as adders and multipliers; it reflects the relative speed of execution of every functional unit at the logic level. Its unit is control step, or c-step, which is a multiple of the clock period. Second, a *path delay* ($PLAT$) is the accumulative delay along a data path formed by nodes and edges. The path is specified by a pair of nodes (N_{start}, N_{finish}) , where N_{start} and N_{finish} are the beginning and end nodes of the path, respectively. The latency of a composite node

N (a node with sub-structures) is computed as:

$$LAT_N = \max (PLAT(N_{input}, N_{output})) \quad (4.1)$$

which is the length of the critical path from N's inputs to its outputs.

Fidelity-compromising transformations allow a node to have multiple LAT values corresponding to different alternative implementations. If the pertinent transformation is controlled by a threshold, then each alternative implementation receives a percentage of execution. The aggregate latency for that node is given by:

$$LAT_{total} = \sum_n LAT_n W_n \quad (4.2)$$

where W_n is the percentage of execution of transformation n . For correlated transformations, a node with multiple alternative designs cannot be substituted by an aggregate node. Each combination of alternative designs for all correlated transformations must be considered. For instance, if transformation T_k has A_k alternative designs, there are a total of $M = \prod_k A_k$ execution paths, each with latency LAT_m and corresponding execution percentage W_m . The aggregate latency for these nodes is given by:

$$LAT_{total} = \sum_{m=1}^M LAT_m W_m \quad (4.3)$$

It can be easily shown that this formula reduces to simple aggregate node formula 4.2 when all transformations are uncorrelated:

$$LAT_{T_1 T_2} = LAT_{T_1} + LAT_{T_2}$$

$$W_{T_1 T_2} = W_{T_1} W_{T_2}$$

For nodes involving control flows, since the number of iterations of a loop may not be known at design-time, the application can be profiled to determine the mean or worst-case

number of iterations. The number of iterations can also be modeled as a variable. In the latter case, a latency analysis will prompt the user to enter a value for each variable in the HDG (which can again be estimated via profiling). In the case of a branch, both paths are represented, with the longer path defining the latency.

4.1.2 Fidelity Evaluation in HDGs

In most applications, fidelity can only be obtained by simulating an actual implementation of the algorithm and evaluating its output. A pure software approach is usually preferred due to its ease of development. Fidelity-compromising transformations will be implemented in the target programming language. Wherever one or more fidelity-compromising transformations are applicable, a switch statement is inserted so that a simulator can easily run the simulation with different transformations turned on and off. For transformations with thresholds, a profiling counter should be kept to record the percentage of executions along each path.

The fidelity evaluation routine can be automated by including replacement code segments as part of the transformation library. Whenever a transformation is applied in the HDG, the corresponding code segment will be looked up in the library and enabled in the source code.

4.1.3 Case Study on SRAD

SRAD (Speckle Reducing Anisotropic Diffusion) [52] is a moderately complicated image processing algorithm tailored to ultrasonic and radar imaging applications. It aims to remove multiplicative noise in imagery. The images in Figure 4.3 illustrate the SRAD

process. The fidelity metric compares the reconstructed image (d) to the original (a). Real-time processing requires a minimum frame-per-second throughput [53], thus defining a frame latency constraint. The Matlab implementation of SRAD is significantly slower than real-time [53], so latency reduction is the focus of this algorithm-implementation co-optimization.

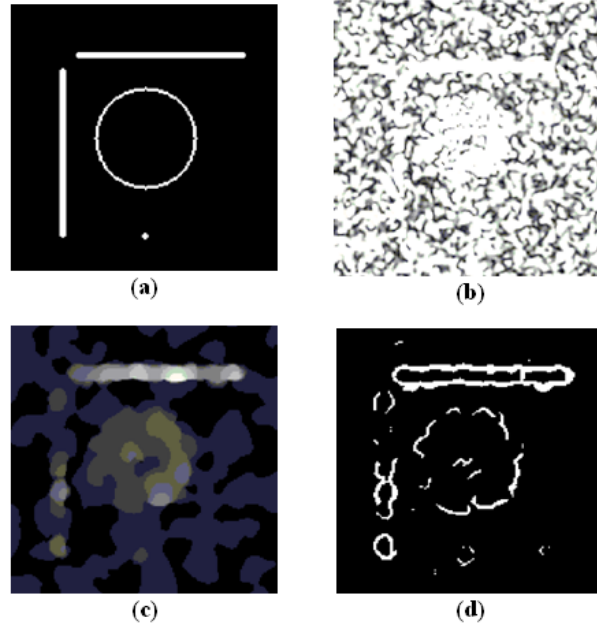


Figure 4.3: Ultrasound images: (a) Perfect edges (b) Noisy image (c) SRAD output (d) Detected edges from SRAD output.

The initial HDG based on the Matlab code is annotated with the component-level latency values listed in Table 4.1. The latency of the HDG is then lowered by duplicating hardware units to create parallelism and applying fidelity-preserving transformations. Fidelity-preserving transformations are standard logic synthesis optimizations, including dead code elimination, loop unrolling and term rewriting [54]. The resulting HDG has 9 levels in the hierarchy and consists of a total of 950 nodes. Given this initial architecture, each pixel requires 48 clock cycles to finish one iteration of its update workload, and each

pixel in the frame is processed serially. An improved Jacobian update removes inter-pixel dependency so that all pixels can be processed in parallel. The latency of processing one frame is $100 \text{ iterations/frame} \times 48 \text{ cycles/iteration} = 4,800 \text{ cycles/frame}$, or a throughput of 208K frames/second (fps) with a circuit running at 1GHz. Such a high throughput can be difficult to achieve due to the tremendous cost of building a circuit that supports such parallelism. While massive parallelism is expensive, we can still meet the real-time constraint by reducing the latency of processing each pixel. Applying fidelity-compromising transformations is an effective method.

Table 4.1: Principle component latencies in the SRAD algorithm [55].

Principle Component	Latency (unit=1 clk cycles)
Addition/Subtraction	1
Multiplication/Squaring(^2)	1
Division/Inversion	10
Square Root	10
Shift	1
Memory Access	1

Table 4.2 contains relevant entries from the fidelity-compromising transformation library. Both entries are mathematical approximations with thresholds.

Now consider how these two transformations are obtained. As discussed in the previous section, the ColSpace tool automatically finds the critical path and bottlenecks, which then become targets for additional fidelity-compromising transformations. For example, the bottleneck of the SRAD algorithm comes down to two DIVs and one SQRT opera-

Table 4.2: Transformation library for SRAD case study.

Transformation	(a)	(b)
Before	$\frac{1}{(1 + \frac{1}{4}x)^2}$	$\frac{1}{1 + y}$
After	$1 - \frac{x}{2}$	$1 - y$
p1 (threshold)	$TH(x)$	$TH(y)$
Condition	$ x < TH(x), TH(x) \in (0, 2)$	$ y < TH(y), TH(y) \in (0, 1)$
Δlat	11	10
Δfid	$\frac{3TH(x)^2}{16} + \frac{TH(x)^3}{32}$	$TH(y)^2$

tion. Then appropriate transformations can be easily developed to specifically attack the bottleneck operations. Transformations (a) and (b) have been developed to replace the two expensive DIVs. Both transformations are applied when x or y is smaller than their respective threshold. Since the values of x and y are not known until runtime, the transformations can be made conditional based on a pre-determined thresholds ($TH(x)$ and $TH(y)$). The transformation is enabled only when the variable x or y drops below the corresponding threshold.

During actual runtime, a transformation can be either applied (Yes) or not applied (No) based on whether or not the threshold condition is met. Therefore there are a total of $2^2 = 4$ execution paths for each candidate transformation. The latency evaluation routine considers each path that could be taken through the conditional transformations, as shown in Table 4.3. The percentage of times each of these eight paths is taken is threshold-dependent and is determined by profiled executions.

Fidelity is evaluated in terms of *Pratts figure of merit* [56], which is a value between 0

Table 4.3: Critical path latencies for different transformation combinations.

Case	A	B	C	D
Apply(a)	No	Yes	No	Yes
Apply(b)	No	No	Yes	Yes
Latency	4.8K	3.7K	3.8K	2.7K

(worst) and 1 (best), generated by comparing the SRAD output with the output of a perfect edge detector. The original SRAD algorithm, free of any alterations, produces an average fidelity of 0.3678. Parameterizing the transformations using thresholds enables quantitative analysis. Figure 4.4 shows threshold exploration results for transformations (a) and (b) using three different cost functions: $\frac{LAT}{FID}$, $\frac{LAT}{\sqrt{FID}}$, and $\frac{LAT}{FID^2}$. The two axes denote discretized threshold settings, and each square denotes a certain combination of $TH(x)$ and $TH(y)$, with the two extremes being “never apply the transformation” (denoted by N) and “always apply the transformation” (denoted by A). So the upper right corner corresponds to the scenario in which both transformations are always applied while the lower left corner corresponds to the scenario in which both transformations are never applied. A lighter color signifies a higher cost function value, so darker colors are better design points. The derivative is calculated by taking the difference between adjacent squares. The plots shown are the average results from running ten different ultrasound images.

Since there is a possibility for greedy algorithms to converge to a local minimum on a 2D design space, we allow the greedy algorithms to use different neighborhood sizes during the search process. The neighborhood size determines the region within which $d(\frac{LAT}{FID})$ will be evaluated. The one with the lowest cost function in the neighborhood will become

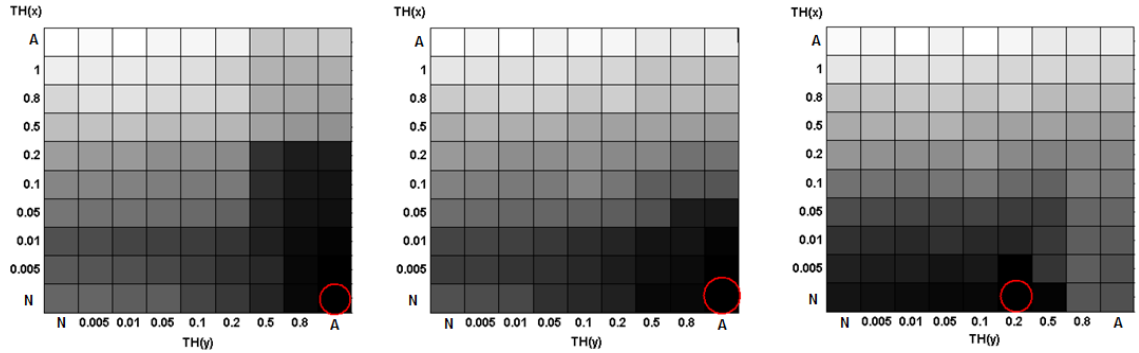


Figure 4.4: 2D plots of various cost functions with two conditional transformation thresholds as variables. Representing cost function: (a) $\text{latency}/\text{fidelity}^{0.5}$ (b) $\text{latency}/\text{fidelity}$ (c) $\text{latency}/\text{fidelity}^2$. Letter “N” on the axes denotes the scenario of “never applying the transformation”, and Letter “A” means “always applying the transformation”. The circled squares denote the optimal design points that minimize the cost function.

the design point in the next iteration. Larger neighborhoods are more likely to escape local minima, but they require more processing time. Table 4.4 compares results using different neighborhood sizes (i.e., number of squares away from the initial point).

Table 4.4: Tradeoff between quality of result and exploration effort for different neighborhood sizes.

Neighborhood Size	Average Space Explored	Average Result Quality (compared to optimal point)
1	20%	98.15%
2	36.11%	100%
3	42.00%	100%
4	49.78%	100%

The tradeoff between the two transformations is more subtle. Observation shows that the optimal design point (the circled square) never apply transformation (a). This suggests that the system is more sensitive to transformation (a) than transformation (b). Thus, a

good strategy is to apply transformation (a) conservatively (with a small threshold) and transformation (b) aggressively (with a larger threshold). Furthermore, the location of the optimal design point can change based on the definition of the cost function.

This cost function-based approach can also be used to solve a constraint-based problem, where one metric is optimized while another is subject to a constraint (such as the real-time requirements of SRAD). The latency vs. fidelity and fidelity vs. latency Pareto curves in Figure 4.5 and Figure 4.6 enable a designer to identify the design points where one metric is constrained and the other is minimized. These curves are derived from the mesh grid plot by searching the entire design space using the proposed algorithm and finding all points with equal fidelity (to construct Figure 4.5) or equal latency (to construct Figure 4.6). Then from this group of points, the one with lowest latency or highest fidelity is chosen.

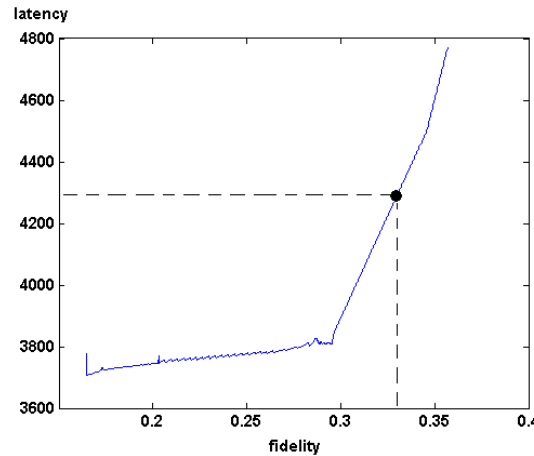


Figure 4.5: Latency vs. fidelity curve of SRAD (with 90% fidelity point located).

Assume that our constraint is to allow no more than 10% degradation of fidelity, we will first calculate 90% best fidelity = 0.3310. We then look up the latency vs. fidelity plot to find the corresponding lowest latency, which is 4305, a 10.31% latency reduction from

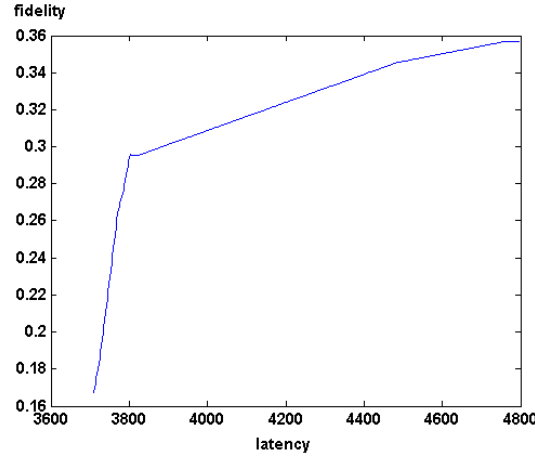


Figure 4.6: Fidelity vs. latency curve of SRAD.

the original design. In addition to meeting metric constraints, these curves also allow designers to identify breaking points at which big savings in system metrics can be achieved. For example, fidelity=0.3 and latency=3,800 are turning points where the slopes change abruptly.

Table 4.5 is a summary of improvements resulting from fidelity-compromising transformations (based on the optimal points selected in Figure 4.4). It is clear that the benefit of applying fidelity-compromising transformations is highly dependent on the type of cost function. If the cost function puts greater weight on latency ($\frac{latency}{\sqrt{fidelity}}$), our methodology can lower the cost function by up to 13%. As the emphasis shifts to fidelity ($\frac{latency}{fidelity^2}$), the reduction on cost function that can be achieved is significantly smaller.

4.2 For Imprecise Adders and Multipliers

In this section, we will take E/op as the efficiency metric and present an efficiency-quality optimization framework at the RTL level. At the RTL level, efficiency metrics such

Table 4.5: Impact of transformations on system metrics (fidelity, latency, and cost function).

	Δfid	Δlat	$\Delta cost$
$\frac{latency}{\sqrt{fidelity}}$	-16.73%	-20.74%	-12.99%
$\frac{latency}{fidelity}$	-16.73%	-20.74%	-4.36%
$\frac{latency}{fidelity^2}$	+0.05%	-0.72%	-0.82%

as power, energy and delay can be more accurately estimated via logic synthesis and SPICE simulations. This methodology will primarily focus on imprecise ALUs, i.e., adders and multipliers, but is extensible to other logic units. Figure 4.7 shows the proposed design flow with IHW. In the beginning, the designer implements the initial target algorithm in RTL. ALUs in the RTL are identified and can potentially be replaced with IHW components from the library. The optimization algorithm requires two metric estimation routines: one for efficiency (energy estimator) and one for quality (error estimator). If it is a constraint-based optimization problem, a quality or efficiency constraint is expected from the user. The optimization is an iterative process, each iteration of which involves selecting a subset of IHW components from the library to apply to the RTL and evaluating the subsequent efficiency and quality metrics. The final output is an optimized RTL with suggested settings for IHW components. In this section, we will focus on quality-energy optimization problems. Other efficiency metrics such power, area, and delay can use the same flow with a different efficiency estimator.

The quality-energy optimization problem can be formulated in many different ways, such as $energy^a/quality^b$ cost minimization or quality maximization subject to an energy

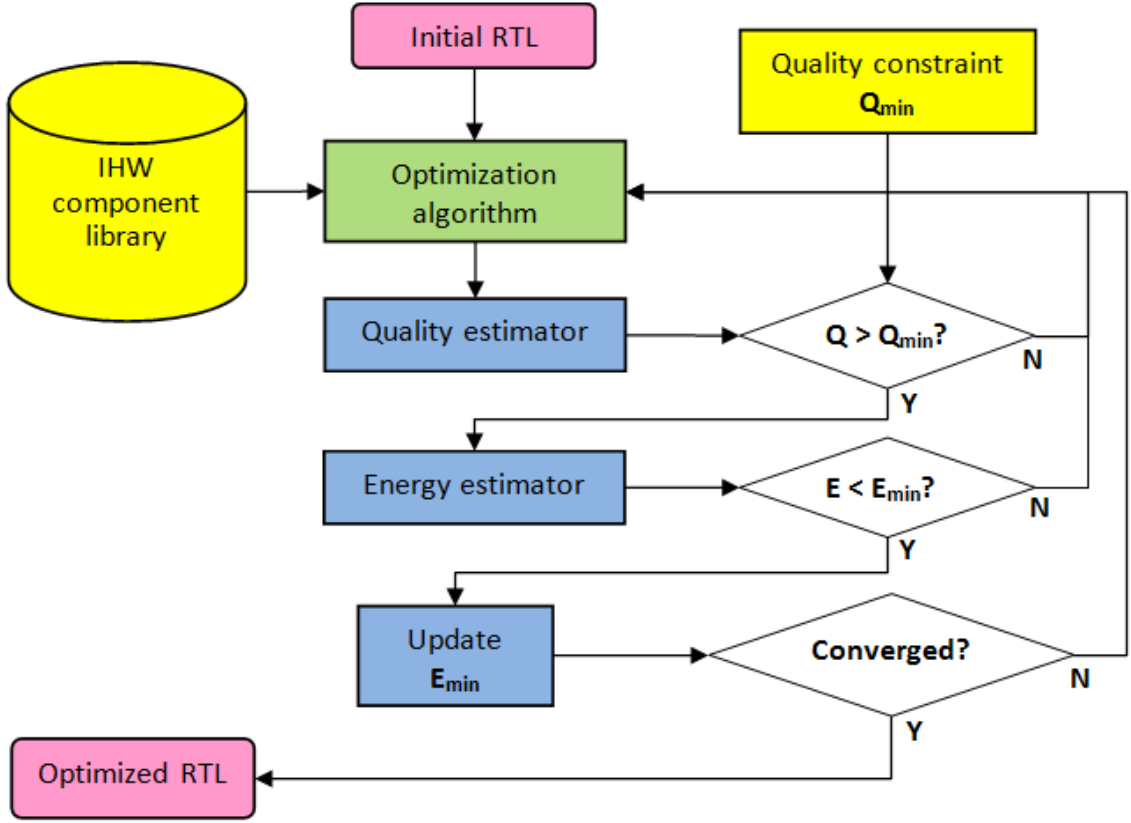


Figure 4.7: Quality-constrained energy minimization flow.

constraint. Our focus is on solving the quality-constrained energy minimization problem:

$$\textbf{minimize: } E(x_0, x_1, \dots, x_n)$$

$$\textbf{subject to: } Q(x_0, x_1, \dots, x_n) \geq Q_0$$

E denotes the energy consumed performing a kernel computation; Q denotes the result quality. x_0, x_1, \dots, x_n are circuit structural parameters such as BPB , L and K or circuit operating conditions such as V_{dd} and frequency. Suppose an IHW system consists of one adder and one multiplier, both of which are restricted to 64-bit. The x vector for this system

is as follows:

$$[add_{mode} \quad BPB_{add} \quad L_{add} \quad K_{add} \quad mul_{mode} \quad BPB_{mul} \quad L_{mul} \quad K_{mul}]$$

add_{mode} and mul_{mode} are integers representing IHW type: 0=KSA, 1=ACA, 2=METAII, 3=RCA. There are certain restrictions on each parameter, such as the adder width (64) must be divisible by BPB and $BPB \times L$ cannot exceed 64. Energy savings also diminish once K_{add} exceeds 32 or K_{mul} exceeds 64 due to the trimmed tree having the same height as the original tree. Parameters will be swept in their valid ranges only. Including precise (KSA and ACA) designs, there are a total of 39 adder designs and 101 multiplier designs to choose from.

Since all the parameters must be integers, this is an integer programming problem. Matlab offers a genetic algorithm function (*ga*) to solve these types of problems. It requires two routines to calculate E and Q respectively. For energy calculation, parameterized RTL models were developed for ACA and METAII adders and multipliers and the RTL for KSA and RCA was obtained online [57]. We then synthesized the models into netlists using Cadence RC Compiler in ST 130nm CMOS technology and simulated 1,000 random additions and 100 random multiplications using Cadence Ultrasim. Energy per operation can be extracted from the simulation waveforms. An energy model is subsequently built using curve-fitting to extrapolate to the entire parameter space. For simplicity, the energy consumed in the control logic is ignored and the sum of ALU energies is used to represent the energy of the kernel.

For calculation of quality, MIA propagation was implemented in C++ as an extension to the *libaffa* project [58]. The workload is written into a text file with each line in the

following format:

```
MUL  METAII  8  4  0  4  60  -1  1  -1  1
```

This specifies the operator's parameters (METAII multiplier with $BPB=8$, $L=4$), input format (4_60) and input data ranges $([-1, 1])$. A program parses this file and the characterization vector and matrix files, performs the MIA propagation, and writes the output data and error MIA into a result file. A final Matlab script extracts the error rate and mean error magnitude from the result file.

4.2.1 Kernel-level Experimental Results

Before running the optimization algorithm, we first need to build energy models for all IHW components in the library as well as precise ALUs. Table A.1 and A.2 in the Appendix lists the raw data used to build the models. The same type of adders and multipliers are synthesized to the longest delay in each class using Cadence RC Compiler. We then use Cadence's Ultrasim to perform SPICE simulations on them to extract average power. The average is taken over 1,000 additions for adders and 100 multiplications for multipliers, due to longer runtime. The product of average power and delay gives the average energy per operation.

Next we fit curves to the obtained data points. The following models are adopted for ACA and METAII-based adders and multipliers:

$$E_{ACA} = A_1 \lceil \log_2 K \rceil + A_2 (K - 2^{\lceil \log_2 K \rceil}) + A_3 K + A_4$$

$$E_{METAII} = B_1 \cdot BPB + B_2 2^L + B_3 \cdot BPB \cdot L + B_4$$

The basic reasoning is as follows: the E/op of the ACA adder should be roughly proportional to the size of the adder tree, thus it should be a function of both the tree height ($\lceil \log_2 K \rceil$) and the tree span (K) as well as the last level breadth ($K - 2^{\lceil \log_2 K \rceil}$). The E/op of the METAI adder should be mostly determined by $BPB \times L$, but should also be affected by BPB and L individually. The fitting results are shown in Figure 4.8. Even though the fitting results are not very good for METAI-based ALUs, the relative error is still quite small ($<4\%$).

The two kernels from Table 3.1 consist only of additions and multiplications. Figure 4.9 and 4.10 show the direct implementation of both kernels. DOT-PRODUCT requires one adder and one multiplier, thus its x vector is:

$$[add_{mode} \quad BPB_{add} \quad L_{add} \quad K_{add} \quad mul_{mode} \quad BPB_{mul} \quad L_{mul} \quad K_{mul}]$$

L2-NORM needs an additional subtractor, which can be implemented with an adder. The squaring operation is implemented with a multiplier. Its x vector is:

$$[sub_{mode} \quad BPB_{sub} \quad L_{sub} \quad K_{sub} \quad add_{mode} \quad BPB_{add} \quad L_{add} \quad K_{add} \\ mul_{mode} \quad BPB_{mul} \quad L_{mul} \quad K_{mul}]$$

The energy for both kernels can thus be calculated as:

$$E_{DOTPRODUCT} = E_{ADD} + E_{MUL}$$

$$E_{L2NORM} = E_{SUB} + E_{ADD} + E_{MUL}$$

where the subtractor can be implemented with an adder.

DOT-PRODUCT needs 1 adder and 1 multiplier, forming a space of 8 variables and 3939 design points. L2-NORM needs 2 adders and 1 multiplier, forming a space of 12 variables and 154K points.

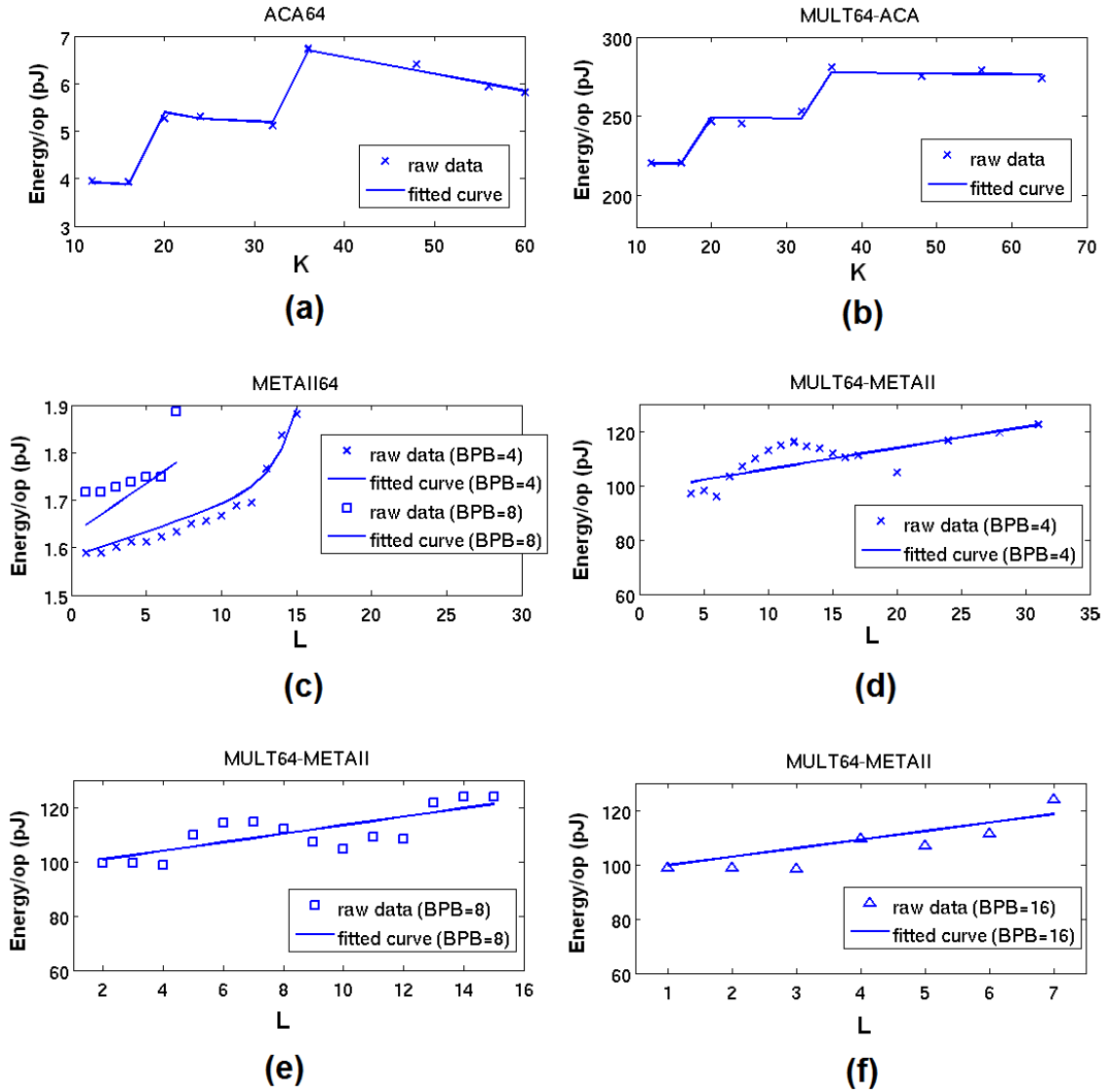


Figure 4.8: Curve fitting for IHW energy models: (a) 64-bit ACA with variable K (b) 64-bit ACA-based multiplier with variable K (c) 64-bit METAII with variable L ($BPB = 4, 8$) (d) 64-bit METAII-based multiplier with variable L ($BPB = 4$) (e) 64-bit METAII-based multiplier with variable L ($BPB = 8$) (f) 64-bit METAII-based multiplier with variable L ($BPB = 16$).

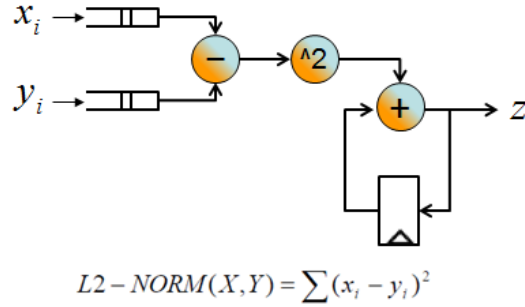


Figure 4.9: Block diagram of our DOT-PRODUCT implementation.

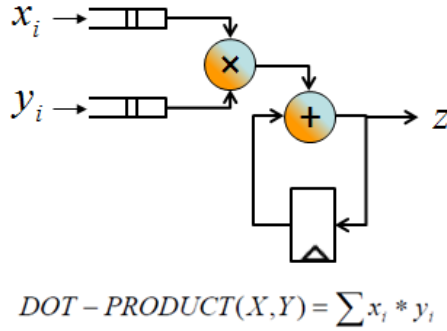


Figure 4.10: Block diagram of our L2-NORM implementation.

The methodology is tested on the two kernels with the following setup: size-8 DOT-PRODUCT with inputs in $[-1, 1]$ and size-10 L2-NORM with inputs in $[-0.25, 0.25]$. Their sizes and dynamic ranges are based on the actual computation and data range profiled while running their corresponding applications. Two quality metrics are evaluated: *error rate* and *mean error magnitude*. By setting the quality constraint at different values between $[2^{-10}, 2^{-1}]$, the optimizer is able to produce optimized designs shown in Table 4.6 and Table 4.7. The energy-quality tradeoff curves formed by imprecise designs are shown in Figure 4.11. As a comparison, we also show curves obtained by running an exhaustive search on all possible design points. In all four figures, the optimizer curves

follow the exhaustive-search curves with a maximum deviation of 2%. Both kernels enjoy a region of about 10% energy reduction with graceful quality degradation. All the curves are significantly lower than the lowest energy achievable by precise designs (136.44pJ for DOT-PRODUCT and 140.4pJ for L2-NORM).

Table 4.6: Kernel-level energy and quality (error rate) results.

Kernel	Optimized Design	Error Rate	Energy per op (pJ)
DOT-PRODUCT	METAIL_8_2_METAIL_8_2	0.107	102.18
	METAIL_8_4_METAIL_8_2	0.095057	102.24
	METAIL_8_7_METAIL_16_2	0.0068907	104.8
	METAIL_16_2_METAIL_16_2	0.00023205	104.79
	METAIL_16_2_METAIL_32_2	5.41E-05	109.73
	RCA_RCA (precise)	0	136.44
L2-NORM	METAIL_8_3_METAIL_8_2_METAIL_8_2	0.21904	103.87
	METAIL_8_2_METAIL_8_2_METAIL_8_2	0.23454	103.84
	METAIL_8_4_METAIL_8_3_METAIL_8_2	0.19698	103.93
	METAIL_16_3_METAIL_16_2_METAIL_8_2	0.093217	104.18
	RCA_METAIL_8_7_METAIL_16_2	0.0083431	108.76
	RCA_METAIL_16_2_METAIL_16_2	0.0002702	108.75
	METAIL_16_3_RCA_METAIL_32_2	7.85E-05	113.75
	RCA_RCA_RCA (precise)	0	140.4

Table 4.7: Kernel-level energy and quality (mean error magnitude) results.

Kernel	Optimized Design	Mean Error Magnitude	Energy per op (pJ)
DOT-PRODUCT	METAIL_4_4_METAIL_8_2	0.00038244	102.14
	METAIL_4_5_METAIL_8_2	5.76E-05	102.16
	METAIL_4_6_METAIL_16_2	1.70E-06	104.64
	METAIL_8_4_METAIL_8_4	1.67E-07	105.33
	METAIL_8_4_METAIL_8_5	3.78E-08	106.88
	METAIL_8_5_METAIL_16_3	9.74E-09	107.79
	METAIL_16_3_METAIL_16_3	6.41E-11	107.68
	RCA_RCA (precise)	0	136.44
L2-NORM	METAIL_4_4_METAIL_4_4_METAIL_8_2	0.0006583	103.76
	METAIL_4_5_METAIL_4_5_METAIL_8_2	7.82E-05	103.79
	METAIL_4_6_METAIL_4_6_METAIL_16_2	3.46E-06	106.3
	METAIL_8_3_METAIL_8_4_METAIL_8_4	9.63E-07	107.03
	METAIL_8_4_METAIL_8_5_METAIL_8_5	9.85E-08	108.63
	METAIL_16_2_METAIL_8_7_METAIL_16_3	7.09E-09	109.54
	METAIL_16_3_METAIL_16_3_METAIL_16_3	1.53E-10	109.64
	RCA_RCA_RCA (precise)	0	140.4

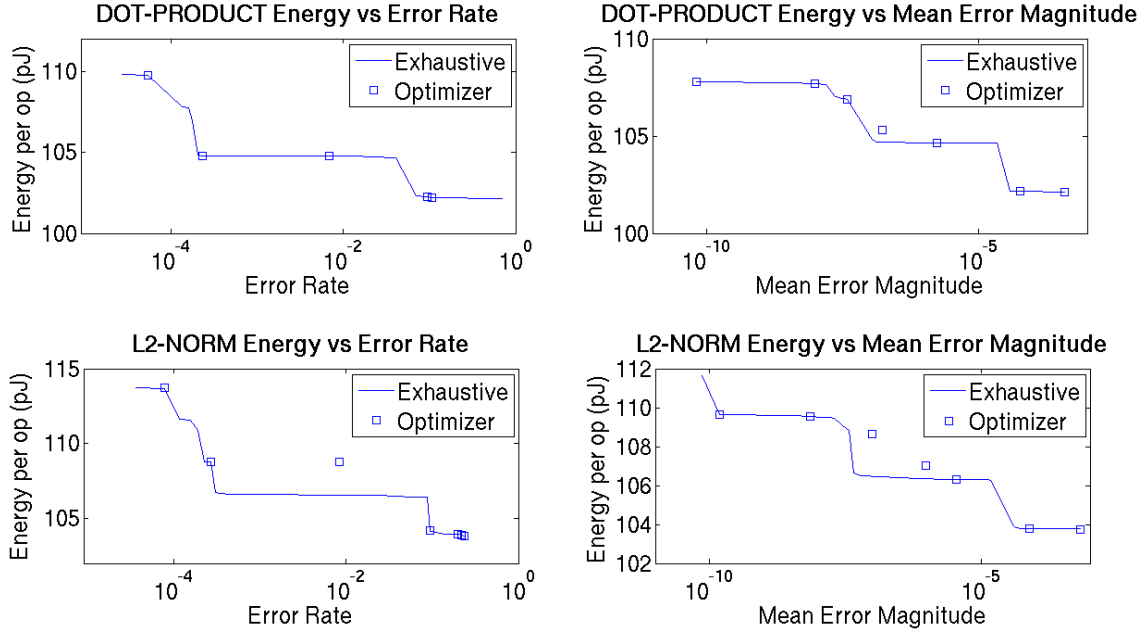


Figure 4.11: Kernel-level energy-quality tradeoffs. Energy is calculated as the average energy consumed during one kernel operation. The design points for the precise designs are located far above all the imprecise design points, therefore are not shown on the plot. The energy of the precise design for the DOT-PRODUCT is 136.44pJ, and the energy of the precise design for the L2-NORM is 140.4pJ.

4.2.2 Application-level Experimental Results

Since the application-level quality can only be obtained through simulation, it is difficult to extend the kernel-level methodology to the application level. Simulating the application with IHW is usually 2–3 orders of magnitude slower than with traditional hardware, because the host machine cannot use a single ALU instruction to perform an imprecise operation. However, kernel-level solutions can facilitate the application-level exploration process. The first step is to solve the kernel-level problem multiple times using static analysis, each time with a different quality constraint. Then, assuming application-level quality is a monotonic function of kernel-level quality, the application can be simulated using only the points identified during the kernel-level exploration. If enough application quality and

parameter pairs are collected, we can build an application-level quality model and apply the same genetic algorithm (GA) to obtain the minimum-energy point, given an application-level quality requirement. This section presents experimental results at the application level assisted by kernel-level exploration. The goal of these experiments is to demonstrate the energy-quality behavior of different applications under IHW implementation and the benefit of the proposed methodology.

The three applications chosen to evaluate the proposed methodology are shown in Table 3.1. Leukocyte Tracker [44] implements an object tracking algorithm, in which an important step is to compute the sum of gradients on the 8 neighboring pixels. SVM is a classification algorithm that consists of a training stage and a prediction stage. The training stage involves computing the Euclidean distance of two data points (called radial basis function) in order to map them into a higher dimensional space. K-means is a data clustering algorithm; the basic operation is calculating the distance between two data points. The Euclidean distance is commonly used. Both K-means and SVM use the L2-NORM kernel and Leukocyte Tracker uses the DOT-PRODUCT kernel. In each application, the corresponding kernel represents a significant percentage of the runtime (Table 3.1). The source code for Leukocyte and K-means is obtained from the Rodinia benchmark suite [59] and SVM from *libsvm* [60]. All benchmarks provide sample input data. In Leukocyte Tracker we tracked 36 cells in 5 frames; in SVM we attempted to classify 683 breast cancer data points with 10 features into 2 classes; in K-means, we tried to cluster 100 data points with 34 features into 5 clusters.

Quality metrics for the three applications are defined as follows. For Leukocyte, the center locations of the centers of the tracked cells are compared with the locations returned

by the precise implementation. The *average cell-center deviation* serves as a good negative quality metric. *Classification accuracy* is a well-established quality metric for SVM. Finally for K-means, *mean centroid distance* [14] is used.

Before simulation, the programs are first profiled to determine the dynamic range of data during kernel computation. If the dynamic range is greater than the characterized data range, it is necessary to perform scaling on the input and output data. Certain applications, such as SVM and Leukocyte, already incorporate data normalization into their algorithm, so no scaling is necessary. Then, the design points returned during kernel-level optimization are used to rewrite the kernel portions of the three applications using those imprecise designs.

The final application-level energy-quality tradeoff curves are shown in Figure 4.12. Since running a SPICE simulation of the entire application to obtain its energy is prohibitively slow, the kernel's energy is used to represent the entire application's energy. Among the three applications, Leukocyte has a smooth quality-energy transition region. At its lowest-energy point (102.24pJ), the mean deviation from precise outputs is merely 0.1 pixels. Its energy is 25% lower than that of the precise design (136.44pJ). For K-means, the mean centroid distance remains unchanged (1,429.22) above the 103.8pJ energy point (26% reduction over precise design's 140.4pJ). Any design below that energy point fails to converge during simulation. A similar situation is observed in SVM where the critical energy point is 103.76pJ. Such convergence failures can be attributed to the lack of *watch-dog* logic in the K-means and SVM algorithms. In the Leukocyte Tracker, the iteration is guaranteed to finish if the iteration count exceeds a certain threshold. However, K-means and SVM do not have such safeguarding mechanism, and the convergence is purely based

on the relative stability of the obtained solution. If the IHW causes substantial fluctuations in the solution between consecutive iterations, those algorithms may never converge and iterate indefinitely.

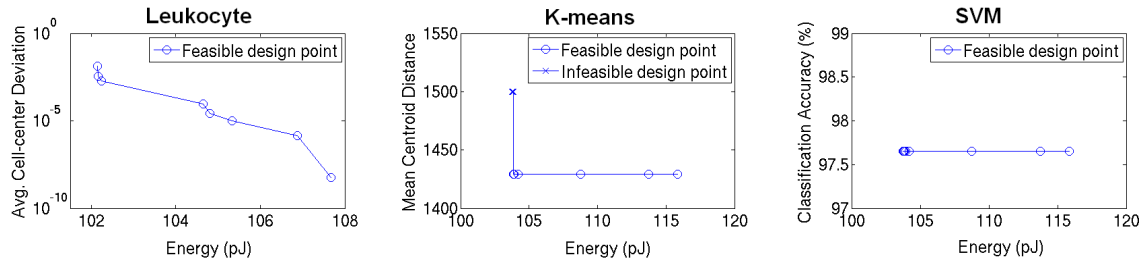


Figure 4.12: Application-level energy-quality tradeoffs. Energy is calculated as the average energy consumed during one kernel operation. The design points for the precise designs are located far to the right of all the imprecise design points, therefore are not shown on the plot. The energy of the precise design for the Leukocyte is 136.44pJ, and the energy of the precise designs for the K-means and SVM is 140.4pJ.

Table 4.8: Number of designs points simulated.

Search method	Leukocyte Tracker	SVM	K-means
Exhaustive search	3,939	153,621	153,621
GA (app-level)	887	1,343	1,343
Proposed methodology	15	17	17

Table 4.8 compares the number of design points that need to be simulated in order to generate the application-level energy-quality tradeoff curves in Figure 4.12. Exhaustive search simulates all the design points once, while applying GA at the application-level simulates only a subset. The proposed methodology simulates the least number of design points because it only chooses those points on the kernel-level optimal energy-quality curve.

4.3 Conclusions

In this chapter, we proposed two quality-aware optimization methodologies on the algorithmic level and RTL level. Both methodologies employ a similar framework: a central optimization algorithm facilitated by an efficiency estimator, a quality estimator and a library of IHW components. The input to the flow is an initial design and the output is an optimized design. Depending on whether the objective is to optimize a cost function or to optimize one metric with other metrics subject to constraints, the user shall supply the definition of the cost function or the metric constraints.

Since the algorithmic IHW (fidelity-compromising transformations) and RTL IHW (imprecise adders and multipliers) are orthogonal IHW techniques, both methodologies described here can potentially be merged into a unified methodology. The resultant IHW library will contain IHW components on both levels and the initial design shall be described in HDG with RTL code associated with the bottom level ALUs. Merging the algorithmic and RTL flows is a promising future project.

Chapter 5

Conclusion

In this dissertation, we introduced the concept of IHW, explored existing IHW components and techniques, and presented some new IHW designs such as fidelity-compromising transformations, IHW adders, and IHW multipliers. We proposed a novel static error estimation method to facilitate the evaluation of output quality as a result of IHW implementation. It is then incorporated into a methodology for exploring the efficiency-quality tradeoffs using IHW. This methodology is able to explore different IHW designs and solve both cost-function based and constraint-based optimization problems.

During the process of conducting this research, empirical and experimental data strongly suggest the existence of an efficiency-quality tradeoff space of many error-tolerant applications. There are many techniques to explore this space, ranging from algorithmic-level techniques to logic and circuit-level techniques. Since there is no consensus on which technique is the most effective in achieving the best efficiency-quality tradeoffs, all of the potential techniques must be considered. This necessitates exploration methodologies that compare and determine the best technique for a particular application and efficiency and

quality requirements. For those methodologies to be effective, we need to know the relationship between IHW parameters and efficiency and quality. While many CAD tools can be used to derive the relationship between circuit parameters and efficiency metrics, quality is typically measured by running Monte Carlo simulations. Static error estimation is an attractive alternative to simulation due to its fast speed and good coverage of rare cases.

The best places to apply IHW techniques are the datapaths of an algorithm because they are implemented with energy-intensive ALU circuits and their quality impact is predictable (compared to control logics). We also found that structural (such as ACA and METAIL adders) and operational (such as VOS) IHW techniques are orthogonal and can be combined in a design to significantly boost the energy savings. Finally, IHW are not limited to error-tolerant applications. By placing a wrapper around IHW components, we can make them appear error-free and can be used in any application to take advantage of the benefits of IHW.

The following is a list of major contributions, organized in the order of the chapters they are related to. Then a similar list of potential future work is provided. We briefly summarize this dissertation in the concluding remarks section.

5.1 Summary of Contributions

5.1.1 Imprecise Hardware

1. **IHW classification:** We classified IHW techniques based on their I/O mappings and error characteristics.
2. **IHW error characterization:** We proposed using Probability Mass Function as the

mathematical representation of errors produced by IHW.

3. **IHW application:** We identified classes of applications that can benefit from IHW implementation.
4. **Novel IHW designs:** We suggested fidelity-compromising transformation as an algorithmic IHW example. On the RTL level, we improved the implementation of ACA and METAI adders and developed IHW multipliers. We proposed using the CORDIC algorithm to develop new IHW arithmetic circuits.
5. **Combined use of IHW and VOS:** We experimented with applying VOS on top of structural IHW adders and IHW multipliers and achieved greater energy reduction than VOS or IHW alone.

5.1.2 Static Error Estimation

1. **MIA and MAA:** We extended classical Interval Arithmetic and Affine Arithmetic methods to support asymmetric distributions and provide improved accuracy.
2. **Static error propagation:** We developed a simulation-free error propagation model that can be used to estimate the quality of any pure-arithmetic computation kernels implemented with IHW.

5.1.3 A Methodology for Efficiency-Quality Optimization

1. **Methodology for fidelity-compromising transformation:** We developed a methodology for reducing the latency of an algorithm by replacing long-latency computations with faster approximate computations.

2. **Methodology for structurally imprecise ALUs:** We developed a methodology for improving the efficiency of an algorithm by replacing traditional ALUs with imprecise ALUs.
3. **Functional models of imprecise adders and multipliers:** These functional models are written in C++, thus can be easily plugged into any application for fast functional simulation of an IHW implementation.

5.2 Suggested Adoption of IHW

The most straightforward way to adopt the IHW components and methodologies proposed in this work follows this procedure. First we need to identify the algorithm and application to be implemented. We need to understand the algorithm and application to the level where we know what portions of the computation are error-tolerant. In addition, the computation identified in the previous step must heavily utilize arithmetic operations. Then we can apply the methodologies on the kernel level and finally extended to the application level to obtain the energy-quality tradeoff curves. The cost of running the methodologies mostly comes from two processes. The first process is IHW characterization for each imprecise operator we want to use as a candidate implementation. This process can be bypassed if the characterization is performed before hand on a large cluster and the results are saved in files. The second process is the application-level simulation with IHW which is usually an order of magnitude slower than simulation with precise HW. Our kernel-assisted methodology (Section 4.2.2) can help reduce the number of simulation runs at the application level.

For error-intolerant applications, we can still employ IHW by leading it with a condition checker. This condition checker should be matched to the IHW, and it will detect all the conditions that will trigger an error in the IHW output. When the error condition is detected, the computation will be forwarded to a precise HW, otherwise IHW will be used. Another approach is to develop IHW which also produces a correctness bit to indicate whether the result is error-free. This correctness bit can be used by a subsequent precise HW or SW for necessary re-computation.

5.3 Future Work

Imprecise Hardware is an emerging field with many unanswered questions and this dissertation is merely a small step toward solving those problems. As with any scholarly work, this work has its limitations which should be addressed by me or other prospective researchers (I appeal to you) in the future. Here I will document the ideas for future work that occurred to me while conducting this research in the hope that they can serve as a source of inspiration to future researchers.

5.3.1 Imprecise Hardware

1. **Runtime reconfigurable IHW:** We can develop IHW that dynamically adjusts its efficiency-quality tradeoff characteristics at runtime to adapt to changing system requirements.
2. **Repurposing IHW for error-intolerant applications:** There are two ways we can present an IHW component as error-free: “IHW + correction circuit” and “detection

circuit + IHW”. The correction circuit in the first form detects erroneous outputs from IHW and corrects those outputs. It is important to keep the overhead of the correction circuit low while still presenting an error-free functionality. In the second form, the detection circuit should detect the conditions which might cause an error in the following IHW (such as long propagate sequence triggered by the inputs). Similarly, the detection circuit has to be low-cost.

3. **Imprecise architecture:** Processors are typically not error-tolerant, but compilers can detect error-tolerant workloads and issue corresponding *imprecise instructions* that can be implemented with IHW. Development of such compilers and supporting architecture has the potential to influence a much broader audience.
4. **Floating-point IHW:** The IHW ALUs described in this work are all fixed-point. To represent a dynamic range of representable values, a floating-point unit is needed. Since a floating-point adder will always contain a fixed-point adder, the easiest way to construct an IHW FPU is simply to replace the fixed-point adder with an IHW adder, although the accuracy impact must be studied.

5.3.2 Static Error Estimation

1. **Bounding peak error:** Even though many applications can tolerate errors in a statistical sense, they may be very sensitive to worst-case errors. For example, a rare but large magnitude error can destabilize the entire algorithm and cause the application to fail unexpectedly. Although MIA and MAA can roughly estimate the shape of the error distribution, we need stronger guarantees for the peak error, such as maximum error magnitude and its frequency of occurrence.

2. **Application vulnerability factor:** Static error estimation can speed up the quality evaluation on the kernel level, but the bottleneck still lies in the evaluation of application-level quality since simulation is still required. In order to promote the static error estimation one level higher to the application level, we need to know how a kernel-level error will impact application-level quality. Mukherjee et al. introduced the concept of application vulnerability factor [61], which is the probability that a fault in a processor structure will result in a visible error in the final output of a program. If we can develop a similar concept to facilitate the PMF propagation at the application level, then the quality evaluation time can be significantly reduced.
3. **Improving MAA:** Although Modified Affine Arithmetic (MAA) can accurately model correlated variables in an expression, it suffers from storage explosion problem. It would be much more useful if this problem could be avoided or mitigated to some extent.
4. **Modeling control logic:** Most applications contain more than just dataflow operators. Control logic such as MUX must also be considered while propagating data and error distributions.

5.3.3 A Methodology for Efficiency-Quality Optimization

1. **Minimum energy theory:** If we consider arithmetic operations to be a special case of transmitting information (e.g., obtaining the sum signal of two input signals), there must exist a theoretical minimum energy cost to accurately obtain that information. Mathematically deriving the function $E_{min} = f_{op}(quality)$ could be as important as Shannon's Theorem in information theory.

2. **Including V_{dd} in the RTL flow:** Due to the long runtime of SPICE simulations, we were unable to complete the IHW characterization under various V_{dd} . Once it is complete, the methodology will have one more variable and could potentially achieve better results.
3. **Merging the algorithmic and RTL flows:** Currently the algorithmic flow to apply fidelity-compromising transformations and the RTL flow to select imprecise ALUs are separated. Combining both flows can simultaneously consider IHW techniques on both design levels, and can lead to a better final design. To enable such a cross-level design flow, we need to design generic IHW library components which have parameterized efficiency and quality models. These components should be able to represent both fidelity-compromising transformations and imprecise ALUs.

5.4 Concluding Remarks

Traditional hardware is an inefficient building block for error-tolerant applications: it is not designed to take advantage of the relaxed quality requirements and further improve its efficiency. IHW is an attractive alternative that gracefully trades off quality for improved efficiency. In this work we reviewed the existing IHW techniques and components, summarized them based on their characteristics and proposed several novel IHW techniques and components. In addition, we developed two quality-aware circuit optimization methodologies using IHW components. A key component of the methodology, a static error estimation method, is proposed to enable rapid and accurate evaluation of the quality impact of an IHW implementation. The methodology can be used to solve cost-function based

optimization problems (e.g., energy-delay product minimization) or constraint-based optimization problems (e.g., energy minimization subject to a minimum quality constraint). Experiments suggest that the utilization of IHW can reduce energy per operation for up to 10% in computation kernels and up to 26% in certain applications without significant impact on output quality.

IHW is still in the early stages of study. Not only do we need *more* IHW techniques and components in the inventory, but we should continue to refine the methodology that utilizes IHW to design more efficient circuits. The best is yet to come!

Bibliography

- [1] ITRS, “International technology roadmap for semiconductors 2009 edition,” ITRS, Tech. Rep., 2009.
- [2] N. R. Shanbhag, R. A. Abdallah, R. Kumar, and D. L. Jones, “Stochastic computation,” in *47th ACM/IEEE Design Automation Conference (DAC), 2010*, jun. 2010, pp. 859–864.
- [3] J. von Neumann, “Probabilistic logics and synthesis of reliable organisms from unreliable components,” in *Automata Studies*, C. Shannon and J. McCarthy, Eds. Princeton University Press, 1956, pp. 43–98.
- [4] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, “A dynamic voltage scaled microprocessor system,” in *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International*, 2000, pp. 294–295, 466.
- [5] V. Gutnik and A. Chandrakasan, “Embedded power supply for low-power dsp,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 5, no. 4, pp. 425–435, dec. 1997.
- [6] D. E. et al., “Razor: A low-power pipeline based on circuit-level timing speculation,” in *Proc. 36th IEEE/ACM Intl. Symp. on Microarchitecture*, 2003, pp. 7–18.
- [7] B. Shim, S. Sridhara, and N. Shanbhag, “Reliable low-power digital signal processing via reduced precision redundancy,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 5, pp. 497–510, may. 2004.

- [8] S. Narayanan, J. Sartori, R. Kumar, and D. Jones, “Scalable stochastic processors,” in *Design, Automation and Test in Europe (DATE)*, March 2010.
- [9] A. Kahng, S. Kang, R. Kumar, and J. Sartori, “Slack redistribution for graceful degradation under voltage overscaling,” in *15th Asia and South Pacific Design Automation Conference (ASP-DAC), 2010*, 18-21 2010, pp. 825 –831.
- [10] A. Kahng, S. Kang, R. Kumar, and J. Sartori, “Designing a processor from the ground up to allow voltage/reliability tradeoffs,” in *IEEE 16th International Symposium on High Performance Computer Architecture (HPCA), 2010*, 9-14 2010, pp. 1 –11.
- [11] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, “Recovery-driven design: A power minimization methodology for error-tolerant processor modules,” in *47th ACM/IEEE Design Automation Conference (DAC), 2010*, 13-18 2010, pp. 825 –830.
- [12] D. Mohapatra, G. Karakonstantis, and K. Roy, “Significance driven computation: a voltage-scalable, variation-aware, quality-tuning motion estimator,” in *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, ser. ISLPED ’09, 2009, pp. 195–200.
- [13] G. Karakonstantis, N. Banerjee, K. Roy, and C. Chakrabarti, “Design methodology to trade off power, output quality and error resiliency: application to color interpolation filtering,” in *IEEE/ACM International Conference on Computer-Aided Design, 2007. (ICCAD 2007)*, 4-8 2007, pp. 199 –204.
- [14] D. Mohapatra, V. Chippa, A. Raghunathan, and K. Roy, “Design of voltage-scalable meta-functions for approximate computing,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, march 2011, pp. 1 –6.
- [15] A. Verma, P. Brisk, and P. Ienne, “Variable latency speculative addition: A new paradigm for arithmetic circuit design,” in *Design, Automation and Test in Europe, 2008. (DATE ’08)*, mar. 2008, pp. 1250 –1255.
- [16] N. Zhu, W. L. Goh, and K. S. Yeo, “An enhanced low-power high-speed adder for

- error-tolerant application,” in *Proceedings of the 2009 12th International Symposium on Integrated Circuits, (ISIC '09)*, dec. 2009, pp. 69–72.
- [17] V. Gupta, D. Mohapatra, S. Park, A. Raghunathan, and K. Roy, “Impact: Imprecise adders for low-power approximate computing,” in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, aug. 2011, pp. 409–414.
- [18] J. Bau, R. Hankins, Q. Jacobson, S. Mitra, B. Saha, and A. A. Tabatabai., “Error resilient system architecture (ersa) for probabilistic applications,” in *Workshop on System Effects of Logic Soft Errors (SELSE)*, April 2007.
- [19] L. Leem, H. Cho, J. Bau, Q. Jacobson, and S. Mitra, “Ersa: Error resilient system architecture for probabilistic applications,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, mar. 2010, pp. 1560–1565.
- [20] E. Kim, R. Abdallah, and N. Shanbhag, “Soft nmr: Exploiting statistics for energy-efficiency,” in *International Symposium on System-on-Chip, 2009. (SOC 2009)*, oct. 2009, pp. 052–055.
- [21] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge, “Opportunities and challenges for better than worst-case design,” in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference (ASP-DAC)*, vol. 1, jan. 2005, pp. I/2 – I/7 Vol. 1.
- [22] B. E. S. Akgul, L. N. Chakrapani, P. Korkmaz, and K. V. Palem, “Probabilistic cmos technology: A survey and future directions,” in *Proceedings of 2006 IFIP International Conference on Very Large Scale Integration*, October 2006, pp. 1–6, 16–18.
- [23] L. Chakrapani, B. Akgul, S. Cheemalavagu, P. Korkmaz, K. Palem, and B. Seshasayee, “Ultra-efficient (embedded) soc architectures based on probabilistic cmos (pcmos) technology,” in *Proceedings of Design, Automation and Test in Europe (DATE '06.)*, vol. 1, mar. 2006, pp. 1–6.
- [24] P. Korkmaz, B. Akgul, and K. Palem, “Ultra-low energy computing with noise: energy performance probability,” in *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, vol. 00, mar. 2006, p. 6 pp.

- [25] L. N. Chakrapani, P. Korkmaz, B. E. S. Akgul, and K. V. Palem, "Probabilistic system-on-a-chip architectures," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 1–28, 2007.
- [26] K. Jeong, A. Kahng, and K. Samadi, "Quantified impacts of guardband reduction on design process outcomes," in *Quality Electronic Design, 2008. ISQED 2008. 9th International Symposium on*, march 2008, pp. 790–797.
- [27] A. Oppenheim, *Discrete-time signal processing*. Upper Saddle River, N.J: Prentice Hall, 1999.
- [28] G. K. Wallace, "The jpeg still picture compression standard," *Commun. ACM*, vol. 34, no. 4, pp. 30–44, Apr. 1991.
- [29] P. Kuhn, *Algorithms, complexity analysis, and VLSI architectures for MPEG-4 motion estimation*. Dordrecht Boston: Kluwer, 1999.
- [30] J. Huang and J. Lach, "Colspace: Towards algorithm/implementation co-optimization," in *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, oct. 2009, pp. 404–411.
- [31] G. V. Varatkar, S. Narayanan, N. R. Shanbhag, and D. L. Jones, "Stochastic networked computation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, no. 99, pp. 1–1, 2009.
- [32] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2003, p. 29.
- [33] L. Roberts, *Machine Perception of Three-Dimensional Solids*. Massachusetts Institute of Technology, Lexington Lincoln Lab, 1963.
- [34] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, Jun. 1986.

- [35] R. E. Moore, *Interval Analysis*. Prentice-Hall, 1966.
- [36] J. Stolfi and L. H. D. Figueiredo, “Self-validated numerical methods and applications,” Brazilian Mathematics Colloquium, 1997.
- [37] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. San Francisco, CA: Morgan Kaufmann Publishers, 2004.
- [38] C. S. Wallace, “A suggestion for a fast multiplier,” *Electronic Computers, IEEE Transactions on*, vol. EC-13, no. 1, pp. 14–17, feb. 1964.
- [39] J. S. Walther, “A unified algorithm for elementary functions,” in *Proceedings of the May 18-20, 1971, Spring Joint Computer Conference*, ser. AFIPS ’71 (Spring). New York, NY, USA: ACM, 1971, pp. 379–385.
- [40] J. Huang and J. Lach, “Exploring the fidelity-efficiency design space using imprecise arithmetic,” in *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, jan. 2011, pp. 579–584.
- [41] L. V. Kolev, “An improved interval linearization for solving nonlinear problems,” *Numerical Algorithms*, vol. 37, pp. 213–224, 2004.
- [42] G. R. J. Huang and J. Lach, “Analytic error modeling for imprecise arithmetic circuits,” in *7th IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE)*, March 2011.
- [43] C. M. Grinstead and L. J. Snell, *Grinstead and Snell’s Introduction to Probability*, version dated 4 july 2006 ed. American Mathematical Society, 2006. [Online]. Available: http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.html
- [44] N. Ray and S. Acton, “Motion gradient vector flow: an external force for tracking rolling leukocytes with shape and size constrained active contours,” *Medical Imaging, IEEE Transactions on*, vol. 23, no. 12, pp. 1466–1478, dec. 2004.

- [45] M. Hazewinkel, *Hellinger distance*. Springer, 2001. [Online]. Available: <http://www.encyclopediaofmath.org/index.php/H/h046890>
- [46] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998.
- [47] K. Vallerio and N. Jha, "Task graph extraction for embedded system synthesis," in *VLSI Design, 2003. Proceedings. 16th International Conference on*, jan. 2003, pp. 480 – 486.
- [48] T. Stefanov, C. Zissulescu, A. Turjan, B. Kienhuis, and E. Deprette, "System design using khan process networks: the compaan/laura approach," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 1, feb. 2004, pp. 340 – 345 Vol.1.
- [49] F. Balmas, "Displaying dependence graphs: a hierarchical approach," in *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*, 2001, pp. 261 –270.
- [50] J. Huang. (2012) Colspace. [Online]. Available: <http://www.cs.virginia.edu/~jh3wn/downloads.html>
- [51] A. Ahmadi and M. Zwolinski, "Symbolic noise analysis approach to computational hardware optimization," in *Design Automation Conference (DAC)*. IEEE, June 2008, pp. 391–396.
- [52] Y. Yu and S. Acton, "Speckle reducing anisotropic diffusion," *IEEE Transactions on Image Processing*, vol. 11, no. 11, pp. 1260 – 1270, Nov. 2002.
- [53] W. Wu, S. Acton, and J. Lack, "Real-time processing of ultrasound images with speckle reducing anisotropic diffusion," in *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on*, 29 2006–nov. 1 2006, pp. 1458 –1464.
- [54] S. Gupta, R. Gupta, and N. Dutt, *SPARK: a parallelizing approach to the high-level synthesis of digital circuits*. Kluwer Academic Publishers, 2004, no. v. 1.

- [55] P. Soderquist and M. Leeser, "Division and square root: choosing the right implementation," *Micro, IEEE*, vol. 17, no. 4, pp. 56–66, jul/aug 1997.
- [56] A. Pinho and L. Almeida, "Figures of merit for quality assessment of binary edge maps," in *Image Processing, 1996. Proceedings., International Conference on*, vol. 3, sep 1996, pp. 591–594 vol.3.
- [57] T. University. (2012) Arithmetic module generator based on arith. [Online]. Available: <http://www.aoki.ecei.tohoku.ac.jp/arith/mg/index.html>
- [58] G. Project. (2012) C++ affine arithmetic library. [Online]. Available: <http://savannah.nongnu.org/projects/libaffa>
- [59] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, oct. 2009, pp. 44–54.
- [60] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [61] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, dec. 2003, pp. 29–40.

Appendix A

Experimental Data

A.1 Raw data for building energy models

Table A.1: Raw delay and power data of various adders for building energy models

Unit	Delay constraint (ns)	Simulated Average Power (mW)	Energy / op (pJ)
KSA64	0.9	6.12	5.51
ACA64 (K=12)		4.39	3.95
ACA64 (K=16)		4.36	3.92
ACA64 (K=20)		5.85	5.27
ACA64 (K=24)		5.89	5.30
ACA64 (K=32)		5.69	5.12

continued on the next page

Table A.1 – continued from previous page

Unit	Delay constraint (ns)	Simulated Average Power (mW)	Energy / op (pJ)
ACA64 (K=36)		7.48	6.73
ACA64 (K=48)		7.12	6.41
ACA64 (K=56)		6.59	5.93
ACA64 (K=60)		6.46	5.81
RCA64	5.5	0.72	3.96
METAI64 (BPB=4,L=1)		0.289	1.5895
METAI64 (BPB=4,L=2)		0.289	1.5895
METAI64 (BPB=4,L=3)		0.291	1.6005
METAI64 (BPB=4,L=4)		0.293	1.6115
METAI64 (BPB=4,L=5)		0.293	1.6115
METAI64 (BPB=4,L=6)		0.295	1.6225
METAI64 (BPB=4,L=7)		0.297	1.6335
METAI64 (BPB=4,L=8)		0.3	1.65
METAI64 (BPB=4,L=9)		0.301	1.6555
METAI64 (BPB=4,L=10)		0.303	1.6665
METAI64 (BPB=4,L=11)		0.307	1.6885
METAI64 (BPB=4,L=12)		0.308	1.694
METAI64 (BPB=4,L=13)		0.321	1.7655
METAI64 (BPB=4,L=14)		0.334	1.837

continued on the next page

Table A.1 – continued from previous page

Unit	Delay constraint (ns)	Simulated Average Power (mW)	Energy / op (pJ)
METAI64 (BPB=4,L=15)		0.342	1.881
METAI64 (BPB=8,L=1)		0.312	1.716
METAI64 (BPB=8,L=2)		0.312	1.716
METAI64 (BPB=8,L=3)		0.314	1.727
METAI64 (BPB=8,L=4)		0.316	1.738
METAI64 (BPB=8,L=5)		0.318	1.749
METAI64 (BPB=8,L=6)		0.318	1.749
METAI64 (BPB=8,L=7)		0.343	1.8865
METAI64 (BPB=16,L=1)		0.320	1.76
METAI64 (BPB=16,L=2)		0.320	1.76
METAI64 (BPB=16,L=3)		0.328	1.804

Table A.2: Raw delay and power data of various multipliers
for building energy models

Unit	Delay constraint (ns)	Simulated Average Power (mW)	Energy / op (pJ)
MULT64_KSA		97.55	273.14

continued on the next page

Table A.2 – continued from previous page

Unit	Delay constraint (ns)	Simulated Average Power (mW)	Energy / op (pJ)
MULT64_ACA (K=12)	2.8	78.68	220.304
MULT64_ACA (K=16)		78.86	220.808
MULT64_ACA (K=20)		88.15	246.82
MULT64_ACA (K=24)		87.63	245.364
MULT64_ACA (K=32)		90.35	252.98
MULT64_ACA (K=36)		100.24	280.672
MULT64_ACA (K=48)		98.25	275.1
MULT64_ACA (K=56)		99.70	279.16
MULT64_ACA (K=64)		97.71	273.588
MULT64_RCA		11.04	132.48
MULT64_METAI (BPB=4,L=4)		8.08	96.96
MULT64_METAI (BPB=4,L=5)		8.17	98.04
MULT64_METAI (BPB=4,L=6)		7.98	95.76
MULT64_METAI (BPB=4,L=7)		8.61	103.32
MULT64_METAI (BPB=4,L=8)		8.91	106.92
MULT64_METAI (BPB=4,L=9)		9.15	109.8
MULT64_METAI (BPB=4,L=10)		9.42	113.04
MULT64_METAI (BPB=4,L=11)		9.57	114.84
MULT64_METAI (BPB=4,L=12)		9.67	116.04

continued on the next page

Table A.2 – continued from previous page

Unit	Delay constraint (ns)	Simulated Average Power (mW)	Energy / op (pJ)
MULT64_METAI (BPB=4,L=13)	12.0	9.54	114.48
MULT64_METAI (BPB=4,L=14)		9.46	113.52
MULT64_METAI (BPB=4,L=15)		9.33	111.96
MULT64_METAI (BPB=4,L=16)		9.20	110.4
MULT64_METAI (BPB=4,L=17)		9.25	111
MULT64_METAI (BPB=4,L=20)		8.72	104.64
MULT64_METAI (BPB=4,L=24)		9.72	116.64
MULT64_METAI (BPB=4,L=28)		9.94	119.28
MULT64_METAI (BPB=4,L=31)		10.21	122.52
MULT64_METAI (BPB=8,L=2)		8.29	99.48
MULT64_METAI (BPB=8,L=3)		8.29	99.48
MULT64_METAI (BPB=8,L=4)		8.23	98.76
MULT64_METAI (BPB=8,L=5)		9.15	109.8
MULT64_METAI (BPB=8,L=6)		9.51	114.12
MULT64_METAI (BPB=8,L=7)		9.54	114.48
MULT64_METAI (BPB=8,L=8)		9.33	111.96
MULT64_METAI (BPB=8,L=9)		8.95	107.4
MULT64_METAI (BPB=8,L=10)		8.72	104.64
MULT64_METAI (BPB=8,L=11)		9.10	109.2

continued on the next page

Table A.2 – continued from previous page

Unit	Delay constraint (ns)	Simulated Average Power (mW)	Energy / op (pJ)
MULT64_METAI (BPB=8,L=12)		9.02	108.24
MULT64_METAI (BPB=8,L=13)		10.15	121.8
MULT64_METAI (BPB=8,L=14)		10.31	123.72
MULT64_METAI (BPB=8,L=15)		10.31	123.72
MULT64_METAI (BPB=16,L=1)		8.23	98.76
MULT64_METAI (BPB=16,L=2)		8.23	98.76
MULT64_METAI (BPB=16,L=3)		8.20	98.4
MULT64_METAI (BPB=16,L=4)		9.11	109.32
MULT64_METAI (BPB=16,L=5)		8.89	106.68
MULT64_METAI (BPB=16,L=6)		9.27	111.24
MULT64_METAI (BPB=16,L=7)		10.33	123.96
MULT64_METAI (BPB=32,L=1)		8.82	105.84
MULT64_METAI (BPB=32,L=2)		8.82	105.84
MULT64_METAI (BPB=32,L=3)		9.89	118.68

Appendix B

Publications related to this dissertation

The following is the list of publications related to this thesis work.

1. **J. Huang** and J. Lach, "ColSpace: Towards Algorithm/Implementation Co-Optimization," International Conference on Computer Design (ICCD), 2009
2. **J. Huang** and J. Lach, "Exploring the Fidelity-Efficiency Design Space using Inexact Arithmetic", Asia and South Pacific Design Automation Conference (ASP-DAC), 2011
3. **J. Huang**, G. Robins and J. Lach, "Analytic Error Modeling for Imprecise Arithmetic Circuits", Silicon Errors in Logic - System Effects (SELSE), 2011
4. **J. Huang**, G. Robins and J. Lach, "A Methodology for Energy-Quality Tradeoffs Using Imprecise Hardware", Design Automation Conference (DAC), 2012
5. **J. Huang**, G. Robins and J. Lach, "A Circuit Design Methodology for Efficiency-Quality Tradeoffs Using Imprecise Hardware", IEEE Transactions on Computer-Aided Design (TCAD), 2012 (submitted for review)