Cross-Layer Automated Hardware Design for Accuracy-Configurable Approximate Computing

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwisssenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Tanfer Alan

Tag der mündlichen Prüfung: 22 Juli 2021

Referent:	Prof. Dr.-Ing. Jörg Henkel Karlsruher Institut für Technologie
Korreferent:	Prof. Dr. Andreas Gerstlauer The University of Texas at Austin
Korreferent:	Prof. Dr. rer. nat. Wolfgang Karl Karlsruher Institut für Technologie

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit — einschließlich Tabellen, Karten und Abbildungen — die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

_____

Tanfer Alan

Acknowledgements

First of all, I would like to share my sincere gratitude to my advisor Prof. Dr.-Ing. Jörg Henkel for accepting me to his research group, investing his time, efforts, and resources in me to become an independent scientist to the high standards he set.

I want to thank Prof. Dr. Andreas Gerstlauer and Dr.-Ing Lars Bauer for their continuous feedback, discussions, and questions that helped me to define my ideas in a concrete manner. Their contributions were immensely important for my development and the outcome of this dissertation. All this support has required a significant amount of their time and effort, to which I am very grateful.

I thank my co-examiner Prof. Dr. rer. nat. Wolfgang Karl for taking this role and the collaborative environment he provided with his research group. Additionally, I would like to thank Jun.-Prof. Dr. Pascal Friederich and Prof. Dr.-Ing. Rüdiger Dillmann for accepting the invitation for my oral defense; and Prof. Dr. Mehdi B. Tahoori and Prof. Dr. rer. nat. Peter Sanders for being members of the doctoral committee in my oral defense.

My colleagues at KIT - CES and the neighboring chairs CDNC and CAPP have contributed to an intellectually cultivating environment. Especially, I want to thank Dr.-Ing. Jorge Castro-Godinez, Dr.-Ing. Farzad Samie, Dr.-Ing. Dennis Gnad, and Dr.-Ing. Michael Bromberger for inspiring and challenging discussions.

It takes a lot of perseverance to do a Ph.D. Towards that, I would like to thank Prof. Dr.-Ing. Christian Hochberger for his guidance and support prior to starting my Ph.D. and my former colleagues at Intel Labs. Such good examples have given me the motivation to pursue this difficult yet very rewarding path.

Finally, I consider myself very lucky to have parents who have throughout my life given me unconditional support and continuous motivation towards reaching my full potential. Thank you!

Karlsruhe, July 2021 Tanfer Alan

Abstract

Approximate Computing trades off computation accuracy against performance or energy efficiency. It is a design paradigm that arose in the last decade as an answer to diminishing returns from Dennard's scaling and a shift in the prominent workloads. A range of modern workloads, categorized mainly as recognition, mining, and synthesis, features an inherent tolerance to approximations. Their characteristics, such as redundancies in their input data and robust-to-noise algorithms, allow them to produce outputs of acceptable quality, despite an approximation in some of their computations. Approximate Computing leverages the application tolerance by relaxing the exactness in computation towards primary design goals of increasing performance or improving energy efficiency. Existing techniques span across the abstraction layers of computer systems where cross-layer techniques are shown to offer a larger design space and yield higher savings. Currently, the majority of the existing work aims at meeting a single accuracy. The extent of approximation tolerance, however, significantly varies with a change in input characteristics and applications.

In this dissertation, methods and implementations are presented for cross-layer and automated design of accuracy-configurable Approximate Computing to maximally exploit the performance and energy benefits. In particular, this dissertation addresses the following challenges and introduces novel contributions:

- A main Approximate Computing category in hardware is to scale either voltage or frequency beyond the safe limits for power or performance benefits, respectively. The rationale is that timing errors would be gradual and for an initial range tolerable. This scaling enables a fine-grain accuracy-configurability by varying the timing error occurrence. However, conventional synthesis tools aim at meeting a single delay for all paths within the circuit. Subsequently, with voltage or frequency scaling, either all paths succeed, or a large number of paths fail simultaneously, with a steep increase in error rate and magnitude. This dissertation presents an automated method for minimizing path delays by individually constraining the primary outputs of combinational circuits. As a result, it reduces the number of failing paths and makes the timing errors significantly more gradual, and also rarer and smaller on average. Additionally, it reveals that delays can be significantly reduced towards the least significant bit and which allows operating at a higher frequency when small operands are computed.
- Precision scaling, i.e., reducing the representation of data and its accuracy is widely used in multiple abstraction layers in Approximate Computing. Reducing data precision also reduces the transistor toggles, and therefore the dynamic power consumption. Application and architecture level precision scaling results in using only LSBs of the circuit. Arithmetic circuits often have less complexity and logic depth in LSBs compared to MSBs. To take advantage of this circuit property, a delay-altering synthesis methodology is proposed. The method finds

energy-optimal delay values under configurable precision usage and assigns them to primary outputs used for different precisions. Thereby, it enables dynamic frequency-precision scalable circuits for energy efficiency.

- Within the hardware architecture, it is possible to instantiate multiple units with the same functionality with different fixed approximation levels, where each block benefits from having fewer transistors and also synthesis relaxations. These blocks can be selected dynamically and thus allow to configure the accuracy during runtime. Instantiating such approximate blocks can be a lower dynamic power but higher area and leakage cost alternative to the current state-of-the-art gating mechanisms which switch off a group of paths in the circuit to reduce the toggling activity. Jointly, instantiating multiple blocks and gating mechanisms produce a large design space of accuracy-configurable hardware, where energy-optimal solutions require a cross-layer search in architecture and circuit levels. To that end, an approximate hardware synthesis methodology is proposed with joint optimizations in architecture and circuit for dynamic accuracy scaling, and thereby it enables energy vs. area trade-offs.

Kurzfassung

Approximate Computing wägt ab zwischen der Rechengenauigkeit und der Leistung oder Energieeffizienz. Es ist ein Design-Paradigma, das im letzten Jahrzehnt als Antwort auf die sinkenden Verbesserungen durch Dennards Skalierung und eine Verschiebung der wichtigen Anwendungen entstanden ist. Eine Reihe moderner Anwendungen, die hauptsächlich als Erkennung, Mining und Synthese kategorisiert werden, weisen eine inhärente Toleranz gegenüber Approximationen auf. Ihre Eigenschaften, wie Redundanzen in ihren Eingabedaten und Algorithmen, die robust gegen Rauschen sind, ermöglichen es ihnen, trotz einer Approximation in einigen ihrer Berechnungen Ergebnisse von akzeptabler Qualität zu erzeugen. Approximate Computing nutzt die Anwendungstoleranz, indem die Genauigkeit der Berechnung in Richtung der primären Entwurfsziele der Leistungssteigerung oder Verbesserung der Energieeffizienz gelockert wird. Bestehende Techniken erstrecken sich über die Abstraktionsschichten von Computersystemen, wobei gezeigt wird, dass Cross-Layer-Techniken einen größeren Entwurfsraum bieten und höhere Einsparungen erzielen. Derzeit zielt der Großteil der vorhandenen Arbeiten darauf ab, eine einzige Genauigkeit zu erreichen. Das Ausmaß der Approximationstoleranz variiert jedoch erheblich mit einer Änderung der Eingabeeigenschaften und -anwendungen.

In dieser Dissertation werden Methoden und Implementierungen für das schichtübergreifende und automatisierte Design von genau konfigurierbarem Approximate Computing vorgestellt, um die Leistung und die Energievorteile maximal zu nutzen. Diese Dissertation befasst sich insbesondere mit folgenden Herausforderungen und stellt neue Beiträge vor:

- Eine Hauptkategorie von Approximate Computing in der Hardware besteht darin, entweder die Spannung oder die Frequenz über die sicheren Grenzen hinaus zu skalieren, um Leistungsverbrauchs- bzw. Rechenleistungsvorteile zu erhalten. Das Grundprinzip ist, dass Zeitsteuerungsfehler allmählich eintreten und für einen Anfangsbereich tolerierbar sein können. Diese Skalierung ermöglicht eine feinkörnige Genauigkeitskonfigurierbarkeit durch Variieren des Auftretens von Zeitsteuerungsfehlern. Herkömmliche Synthesewerkzeuge zielen jedoch darauf ab, eine einzige Verzögerungszeit für alle Pfade innerhalb der Schaltung zu erreichen. Anschließend sind bei der Spannungs- oder Frequenzskalierung entweder alle Pfade erfolgreich oder eine große Anzahl von Pfaden fällt gleichzeitig aus, wobei die Fehlerrate und -größe stark ansteigen. Diese Dissertation stellt eine automatisierte Methode zur Minimierung von Pfadverzögerungen vor, indem die primären Ausgänge von kombinatorischen Schaltungen individuell eingeschränkt werden. Infolgedessen wird die Anzahl der fehlerhaften Pfade verringert und die Zeitsteuerungsfehler werden wesentlich abgestufter und im Durchschnitt auch seltener und kleiner. Darüber hinaus zeigt sich, dass Verzögerungen in Richtung des niedrigstwertigen Bits erheblich reduziert werden können und dass bei Berechnungen mit kleinen Operanden mit einer höheren Frequenz gearbeitet werden kann.

- Präzisionsskalierung, d.h. Reduzieren der Darstellung von Daten und ihrer Genauigkeit, wird in Approximate Computing häufig in mehreren Abstraktionsschichten verwendet. Durch Verringern der Datengenauigkeit werden auch die Transistorumschaltungen und damit der dynamische Stromverbrauch verringert. Die Präzisionsskalierung auf Anwendungs- und Architekturebene führt dazu, dass nur LSBs der Schaltung verwendet werden. Arithmetische Schaltungen weisen in LSBs im Vergleich zu MSBs häufig eine geringere Komplexität und Logiktiefe auf. Um diese Schaltungseigenschaft auszunutzen, wird eine verzögerungsverändernde Synthesemethode vorgeschlagen. Das Verfahren ermittelt energieoptimale Verzögerungswerte unter konfigurierbarer Präzisionsnutzung und weist sie primären Ausgängen zu, die für verschiedene Präzisionen verwendet werden. Dadurch werden dynamische, frequenzpräzise skalierbare Schaltungen für Energieeffizienz ermöglicht.
- Innerhalb der Hardwarearchitektur ist es möglich, mehrere Einheiten mit derselben Funktionalität mit unterschiedlichen festen Approximationspegeln zu instanziieren, wobei jeder Block von weniger Transistoren und auch Syntheserelaxationen profitiert. Diese Blöcke können dynamisch ausgewählt werden und ermöglichen so die Konfiguration der Genauigkeit zur Laufzeit. Das Instanziieren solcher approximierter Blöcke kann eine geringere dynamische Leistungs-, aber höhere Flächen- und Leckstromkostenalternative zu den gegenwärtigen Gating-Mechanismen nach dem Stand der Technik sein, die eine Gruppe von Pfaden in der Schaltung abschalten, um die Umschaltaktivität zu verringern. Durch das gemeinsame Instanziieren mehrerer Blöcke und Gating-Mechanismen entsteht ein großer Entwurfsraum für genau konfigurierbare Hardware, bei dem energieoptimale Lösungen eine schichtübergreifende Suche in Architektur und Schaltungsebenen erfordern. Zu diesem Zweck wird eine approximierte Hardwaresynthesemethode mit gemeinsamen Optimierungen in Architektur und Schaltung für die dynamische Genauigkeitsskalierung vorgeschlagen, die einen Kompromiss zwischen Energie und Fläche ermöglicht.

Contents

Acknowledgements . . . iii

Abstract . . . v

Kurzfassung . . . vii

1 Introduction . . . 1
- 1.1 Approximate Computing . . . 2
- 1.2 Accuracy-Configurable Approximations . . . 3
- 1.3 Dissertation Contribution . . . 3

2 Background . . . 5
- 2.1 Research at KIT - Chair for Embedded Systems . . . 6
- 2.2 Approximate Computing Across the Stack . . . 8
- 2.3 Dynamic Accuracy Reconfiguration . . . 12
  - 2.3.1 Accuracy-Configurable System . . . 12
  - 2.3.2 Case Study: Edge Detection in Video Frames . . . 13
  - 2.3.3 Case Study: DNN Inference . . . 15
  - 2.3.4 Accuracy-Configurable Hardware . . . 17
- 2.4 Logic Synthesis . . . 19
  - 2.4.1 Synthesis of Approximate Units . . . 22
  - 2.4.2 Impact of Critical Path Length on Circuit Trade-offs . . . 23
- 2.5 Summary . . . 26

3 Synthesis for Graceful Timing Violations . . . 27
- 3.1 SlackHammer: Preliminaries and Approach . . . 28
  - 3.1.1 Traditional Logic Synthesis . . . 29
  - 3.1.2 Non-Critical Path Optimization . . . 30
  - 3.1.3 Synthesis for Graceful Errors . . . 30
  - 3.1.4 Path Analysis in Isolation . . . 31
  - 3.1.5 Constraining Path Delays . . . 31
- 3.2 Design Methodology . . . 32
- 3.3 Experimental Methodology . . . 35
- 3.4 Results . . . 36

3.4.1 Accuracy-Frequency Trade-Off . . . 37
3.4.2 Delay Distribution Comparison . . . 39
3.4.3 Cross-Layer Effectiveness . . . 40
3.5 Summary . . . 40

4 Synthesis of Frequency-Precision Scalable Circuits . . . 43
4.1 Logic Synthesis with Multiple Delay Constraints . . . 44
4.1.1 Delay Variations in Circuit Topology . . . 44
4.1.2 Exploiting Delay Variations for Energy Gains . . . 45
4.1.3 Dynamic Frequency-Precision Scaling System . . . 46
4.2 Design Methodology . . . 46
4.2.1 Energy Optimization with a Throughput Target . . . 46
4.2.2 Leverage & Distance . . . 47
4.2.3 Design Space Exploration . . . 48
4.3 Experiments . . . 50
4.3.1 Evaluation of DSE Iterations . . . 50
4.3.2 Circuit Level Trade-Offs . . . 51
4.3.3 Energy vs. Leverage . . . 52
4.4 Summary . . . 53

5 Architecture and Circuit Co-Synthesis . . . 55
5.1 Background . . . 56
5.2 Accuracy-Configurable Hardware Architecture . . . 57
5.2.1 Gating Groups of Paths in Circuit . . . 57
5.2.2 Instantiating Approximate Circuits with Different Accuracies . . . 57
5.2.3 Cross-Layer Design Approach . . . 59
5.2.4 Runtime Accuracy Management . . . 60
5.3 Exploration Methodology . . . 60
5.4 Experiments and Results . . . 64
5.4.1 Design Space Exploration . . . 66
5.4.2 Comparison of Pareto-Optimal Solutions . . . 68
5.4.3 Analysis of Integration and Control Overhead . . . 68
5.4.4 Area vs. Energy Trade-offs . . . 69
5.4.5 Energy Cost of Runtime Accuracy Reconfiguration . . . 70
5.4.6 Input Dependency of Cross-Layer Design Space . . . 71
5.4.7 Leakage Energy Analysis and Technology Independence . . . 73
5.5 Summary . . . 74

6 Conclusion . . . 75

Bibliography . . . 77

List of Publications . . . 87

List of Figures . . . 89

List of Tables . . . 91

1 Introduction

Computer hardware is evolving with changes in workloads and confronted physical limitations. Ever-increasing computational demands under thermal dissipation constraints and limited battery capacities require us to reduce energy consumption for both high-performance and low-power systems. Previous shifts in the workloads have led to the introduction of alternative computing platforms to central processing units (CPUs), such as graphics processing units (GPUs) and accelerators to address the changing characteristics. Similarly, modern applications have different characteristics than before: an inherent resilience to a varying degree of inaccuracies, i.e., errors in their computations.

Workloads of today differ from the past. If we look at the history of computers, the earlier examples were designed to solve complex differential equations for workloads of military, scientific, and financial domains [117]. Later, with the popularization of graphical user interface (GUI), a large fraction of the operations became graphics-related, which led to the invention of GPUs. Today, a classification of modern workloads is recognition (R), mining (M), and synthesis (S) [24]. The set of RMS workloads include graphics, gaming, media-mining (e.g., image classification), unstructured information management (e.g., internet / semantic search), financial analytics, among others. These workloads have characteristics such as redundancies in their input data, robust-to-noise algorithms, or being used in applications that only need to produce an acceptable output rather than a unique golden result [23]. Thus, they possess an inherent error resilience, producing outputs of acceptable quality, despite an error in some of their computations.

At the same time, physical limits in technology scaling have given us diminishing returns from newer technologies. There is a deceleration in the scaling of transistor dimensions which has in the past led to a continuous increase in performance and energy efficiency and addressed increasingly more demanding workloads [30, 91]. Over the last decade ,this scaling has led to reliability challenges. Moreover, in advanced technology nodes, supply and threshold voltages have scaled down at a lesser factor than transistor dimensions. This has led to trends, in which we have less energy budget than the area budget in our chips. With decreasing gains from technology node scaling, alternative and orthogonal ways to improve efficiency without sacrificing generality have become very desirable. For instance, at the circuit level reliability is traditionally ensured by providing sufficient timing guardbands. However, alternative approaches have emerged to reduce the increasingly high safety margins due to process and runtime variations in advanced technology nodes [44]. As an alternative, the idea of recovery-based computing depends on detecting and recovering from timing errors, but which has high recovery overheads [34]. On the other hand, Approximate Computing relaxes the correct execution strictness for application domains that possess intrinsic error resilience.

1.1 Approximate Computing

Approximate Computing leverages the application error resilience by relaxing exactness in computation towards primary design goals: improving the performance and energy efficiency [23]. As previously introduced, several modern and prominent application domains such as machine learning, big data processing, semantic search, gaming & multimedia, the datapath computation accuracy is dispensible to some degree [23, 24, 26, 99, 109, 114, 115]. This inherent application tolerance to inaccuracies can be attributed to iteratively or statistically self-correcting algorithms, lack of a single golden answer, and limits of human visual perception. Hence, an accuracy compromise can be made through intentionally introducing approximations, without a significant compromise on application quality. The impact of approximations on energy facilitates solving larger problems at both ends of the computing spectrum [26]. It has become an enabling factor for workloads such as multimedia, gaming, and object recognition with a limited energy budget as in mobile devices [75]. Also, it is a recent and major driver of machine learning at both embedded edge devices [19, 116, 119] and high-performance servers [39, 50, 58] through precision scaling, i.e., quantization in machine learning terms.

The application space of Approximate Computing is bound with computation where accuracy trade-offs can be made. To give a counterexample, an approximation-sourced difference in state machines can lead to a wrong control flow rather than an approximation. Therefore, a notion of *disciplined* approximations is required to isolate the tolerant part from the sensitive and bound the error rate and magnitude to the tolerable degree. To achieve that, a significant amount of work is proposed across the abstraction layers, such as searching the approximation tolerant parts and degree of tolerance in the algorithm, marking the tolerant code in software, quality checks in runtime system, approximate-aware instruction set architecture (ISA) extensions to utilize underlying hardware, architecture-level modifications, functional approximations, and circuit-level timing speculations by voltage or delay starvation of transistors. A detailed overview can be found in Section 2.2

In hardware, the focus of this dissertation, functional approximations and circuit-level timing speculation make up the two main disjoint categories of Approximate Computing. Several previous efforts utilized both hardware categories together with software in a cross-layer fashion and reported a larger design space and more favorable trade-offs than techniques from any single category [23, 27, 120].

Traditionally, a large class of research explored Approximate Computing at the hardware level targeting a single accuracy in manual [124, 132] and automated design [21, 73, 86, 110] of functionally approximate circuits. The hardware is designed to have fewer transistors and shorter critical paths, where boolean functionality deviates from an exact specification to a limited extent. Instantiating such approximate hardware has a two-fold effect on energy: fewer transistors cause less toggling activity and shorter paths allow for voltage scaling or *synthesis relaxations*, i.e., circuits can be composed of smaller transistors that require less power at the same performance. In this way, the slack in shorter paths can be exploited by the synthesis tool. The evident disadvantage is that the approximations on these circuits are fixed and hardwired. It is not possible to configure their accuracy at runtime.

Circuit-level timing speculations scale either voltage or frequency aggressively, and beyond guardbands, inducing timing errors upon activation of critical paths [8, 42, 62, 89]. The motivation behind introducing timing speculations is that most input combinations do not invoke the critical path and can be accomplished in a shorter time. In exchange, increasing the frequency improves the performance or we can reduce the voltage, resulting in a quadratic reduction in dynamic power

consumption. Thus, scaling of frequency or voltage and introducing timing errors creates a knob on accuracy. However, unless the delay constraints are exceedingly large, traditional synthesis algorithms result in circuits that contain a large number of near-critical paths. In consequence, circuits synthesized by traditional tools possess a characteristic in which under aggressive scaling either no error occurs, or a very large number of paths fail at the same time. Although this is desirable for area and power reductions to exact computation, it hinders the benefits of timing speculations and prohibits gradual configuration of the accuracy.

1.2 Accuracy-Configurable Approximations

Accuracy configurability is essential in practice for two main reasons: (i) Output quality of approximate hardware strongly depends on its inputs, and (ii) A workload may tolerate significantly different levels of approximation depending on its context and environment [128]. These reasons for accuracy configuration are further detailed in Section 2.3. Runtime methods have shown that a fixed accuracy may be too conservative and accuracy configuration is necessary to maximally exploit the opportunities of approximate computing for energy efficiency improvement [11, 64, 72, 128]. In particular, an offline profiler in [128] has shown that there is a significant variation in precision requirements between different applications and also between different phases of an application.

Considering the size and complexity of modern computing systems, automated and general methods are necessary to benefit from approximate computing at scale. Ideally, this automation should span across the abstraction layers and components to enable system-level trade-offs. The currently established, well-proven, and powerful electronic design automation (EDA) flow has a rich set of optimizations. However, it does not consider the varying accuracy requirements. Therefore, it does not yield optimal results when used for accuracy-configurable approximate computing.

1.3 Dissertation Contribution

This dissertation aims at steering computer hardware and EDA towards accuracy-configurable approximate computing goals and optimizations. The methodologies proposed in this dissertation automate and design hardware for runtime variable accuracy with exposing knobs for accuracy configurability, and optimize the hardware for accuracy-configurable use towards maximally exploiting the approximation benefits while maintaining quality targets. They utilize the existing hardware techniques, functional approximations and timing speculations, previously described in Section 1.1 They are general in terms of circuit, application, and required accuracies as they are built on commercially available logic synthesis that is used for all circuits. Additionally, the methods developed in the scope of this dissertation analyze and utilize the underlying circuit properties, such as gate-level topology, to maximally exploit approximate computing benefits. In particular, the novel contributions of this dissertation are as follows:

- First, this dissertation introduces a novel methodology to automatically synthesize circuits with enhanced timing-error resilience under aggressive frequency scaling. This methodology identifies primary outputs with a remaining timing slack margin, given any arbitrary circuit. The key idea is to optimize non-critical paths for the delay. Consequently, it reduces the probability of timing errors and thus improves the accuracy with frequency scaling. It also makes the timing errors occur more gradually compared to the conventional tools. It extends the design space to higher frequencies and finding favorable trade-offs between performance

and accuracy for both standalone and cross-layer techniques that utilize timing speculations for approximate computation.

- To optimize circuits for energy under varying accuracy use, a throughput-constrained circuit synthesis methodology is proposed. Uneven complexities of circuit topologies lead to most critical path delay dominated circuit delay, power, and area requirements, while achieving limited, best-effort improvements on low-complexity paths. The second contribution in Chapter 4 alters delays of primary outputs during logic synthesis. It relaxes the delay constraints that apply to the majority of the circuit while tightening it for a smaller part used for reduced precision operations. Consequently, the resulting circuits require less area and average energy, when used for accuracy-configurable approximate computing, while being able to maintain the initial average throughput. These circuits possess the property of dynamic frequency-precision scalability as they can operate at high precision with a slow clock and at a low precision with a fast clock.
- Chapter 5 addresses the necessity of accuracy-configurable hardware systems with the exploration of applying gating mechanisms to existing circuits together with instantiating more efficient circuits into the architecture. Jointly, the architecture and circuit layer together present a larger design space where non-trivial cross-layer decisions are necessary to find optimal solutions. In this chapter, a methodology is proposed to ensure Pareto-optimal combinations towards minimizing energy consumption under given workload and area constraints. The cross-layer design space offers dynamic accuracy configurable hardware with significantly reduced energy compared to existing gating solutions when more area can be utilized.

2 Background

Definitions

In the scope of this dissertation, the following definitions are used:

Approximate Computing: given the algorithm, the set of generic computing techniques that produce different but *acceptable* results. Approximate computing trades accuracy against a design point such as accuracy vs. energy, performance, area, or economic cost. The approximate computing techniques are generic in the sense that they can be applied to any approximation tolerant application without the application expertise, to a varying degree that is acceptable by the defined quality metric.

Quality: an application demand to satisfy an application-relevant metric. Therefore, it should be defined accordingly by an application expert. For instance, PSNR 30 dB or SSIM 0.9 for multimedia, 70 % Top-1 classification accuracy for deep neural network (DNN) applications, 10 % magnitude difference to the exact result can be all considered appropriate results for different applications.

Accuracy: correctness of the underlying computation, independent of the application. It can be defined as mean relative error distance (MRED), absolute error distance, or another similar mathematical metric.

Error: a difference between the exact and approximate output.

***Disciplined* Approximate Computing**: Clear partitioning of approximation tolerant part of the application, defining acceptable limits of error rate and magnitude.

Means of Approximation: Techniques that directly apply approximations.

Means for Utilization of Approximations: Techniques that define approximation tolerant parts of an application and map it to techniques that are means of approximation.

2.1 Research at KIT - Chair for Embedded Systems

The research outcome of this dissertation is, in part, emerged through the expertise and focus of the research projects presented in the following.

Hardware/Software Co-Design

Hardware/Software partitioning of applications is a widely used approach for optimizing system-level design metrics, such as performance, and satisfying design constraints, e.g., chip area. Chair for Embedded Systems (CES) has a history of research on this topic [35], spanning from adaptive estimation of the costs [48] to design techniques to optimize for power and energy [43] with a case study [46], and to flexible application-specific processors with runtime reconfiguration capability [12, 13].

Low Power Design

Advances in technology scaling in the nano-CMOS era have steadily increased the consumed power per chip area (i.e. power density). High power densities and thus elevated temperatures seriously jeopardize the chip's reliability. CES has focused on thermal-awareness [32, 33] and energy-minimization [78] topics and developed novel techniques to mitigate the related problems.

Network-on-Chip (NoC)

With the emergence of manycore computing systems, communication has taken a key role in defining system performance and energy-efficiency. Prior research of CES on this topic [47] had a special focus on runtime adaptations [2, 3].

Approximate Computing

In the last decade, approximate computing has emerged to improve energy-efficiency and performance by trading them against computational accuracy. Early examples from CES include an accuracy-delay configurable adder [111] and a multiplier [14].

DFG SPP 1500 Dependable Embedded Systems

Under SPP 1500 project, the research on dependable embedded systems aims at developing new methods and architectures to eliminate the effects associated when migrating to new technology nodes, such as malfunctions, performance degradation, and increase power consumption at the system level. This project is composed of five research areas: technology abstraction, dependable hardware architectures, dependable embedded systems, design methods, and operation, observation, adaptation [63].

The contributions of this dissertation particularly relate to dependable architecture and methods research areas. Approximate Computing applies to the approximation tolerant part of the applications. The degree of tolerable approximations, however, varies with multiple factors. The methods proposed in this dissertation aim at designing hardware for accuracy configurability and exposing knobs. Therefore, they allow keeping the hardware accuracy within tolerable approximations, making the computation dependable and approximate, by benefiting from approximations while maintaining a quality constraint.

DFG Transregio TR89 – Invasive Computing

"The Transregional Collaborative Research Center Invasive Computing (abbr. InvasIC), is investigating the design and resource-aware programming of future parallel computing systems. For systems with 1000 and more cores on a chip, resource-aware programming is of utmost importance to obtain high utilization as well as computational and energy efficiency numbers. InvasIC is funded by the Deutsche Forschungsgemeinschaft in its third period of four years (July 2018 - June 2022), aggregating researchers from three excellent sites in Germany (Friedrich-Alexander-Universität Erlangen-Nürnberg, Karlsruher Institut für Technologie, Technische Universität München). This scientific team includes specialists in algorithm engineering for parallel algorithm design, hardware architects for reconfigurable MPSoC development as well as language, tool and application, and operating system designers" (InvasIC, described on its website [55]). Prominent research outcome of InvasIC[45] include distributed resource management [67] and parallel operating systems [96], among many other related contributions.

2.2 Approximate Computing Across the Stack

This section briefly summarizes Approximate Computing techniques in different abstraction layers of a computer system with some prominent examples of prior work. Given a problem that can tolerate approximations in some part of its calculations, a variety of techniques are proposed to approximate in different ways and also to utilize the approximation techniques.

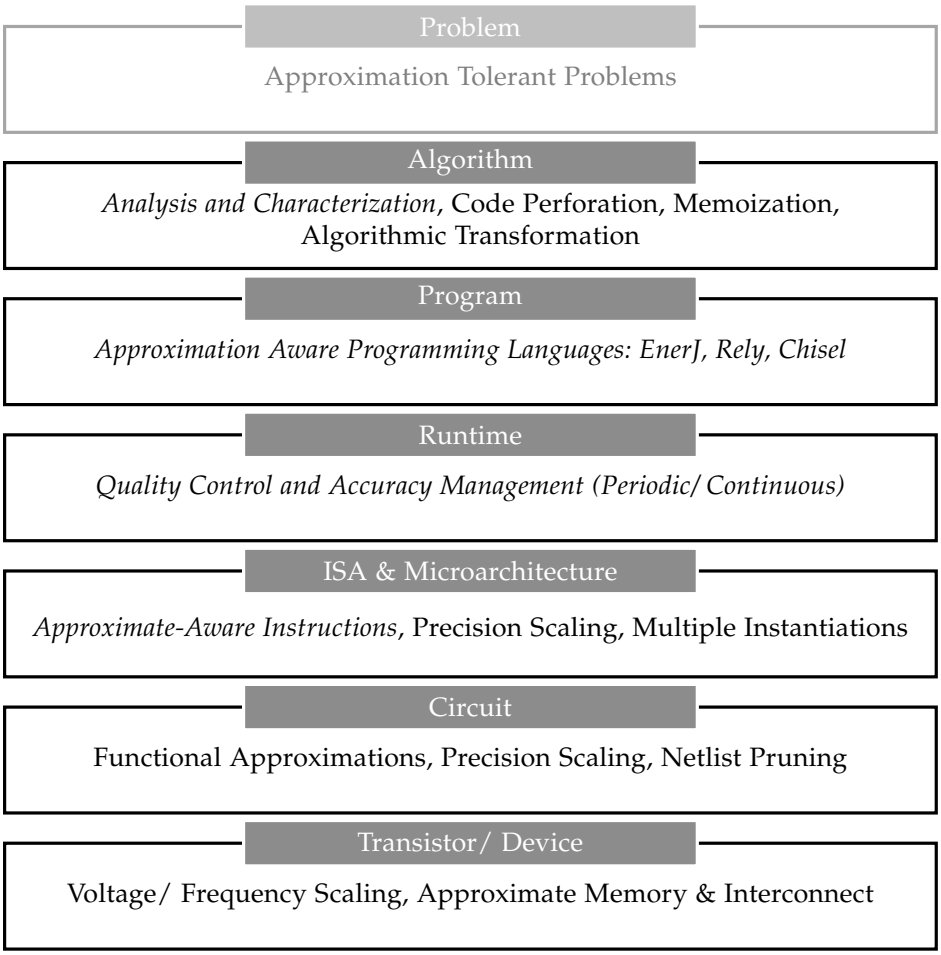


Figure 2.1: Approximate Computing across the computing stack. My classification of existing work to the abstraction layers of a computer system, as defined in [100]. Upright techniques represent different means of approximation, *Italics* represent *means for utilization* of underlying approximations.

Algorithm

A fundamental challenge is to benefit from approximations while maintaining the correct execution of the application. Addressing this challenge requires isolating the approximation tolerant part from the sensitive part. This can be done by manually marking the kernels or the code, or automatically at the algorithm level by systematically dissecting kernels, injecting errors, and testing the application. Application Resilience Characterization (ARC) Framework [26] partitions an application into resilient and sensitive parts. It uses a dynamic binary instrumentation tool (valgrind) to dominant kernels that require more than 1% of the run time. It injects unconstrained random errors to the dominant kernels and classifies them as sensitive if the application crashes, hangs, or produces an unacceptable output. Otherwise, it classifies the kernel as resilient. Hence, it automatically separates the approximation tolerant part of the program.

Approximations can also be applied directly at the algorithm level by skipping loops, data fetches, or transforming kernels. These techniques can be applied to a pseudo-code, before using a programming language. A prominent example is loop perforation [49, 88, 113]. Aiming at iteratively refining algorithms, it is shown that reducing the number of loops can offer a favorable trade-off for accuracy vs. compute time. The loops to be perforated can be found in a similar way to the ARC framework: by testing perforated code against crashes, hangs, or producing unacceptable output. This is a general technique that applies to any loop to a varying degree. A practical way to utilize this technique is to combine it with a compiler such as LLVM.

Similarly, data extrapolation by predicting its value from the previous data is proposed. In this method, data fetches on cache misses can be perforated for approximate workloads and can be predicted to mitigate the limited off-chip bandwidth bottleneck and reduce the effective memory latency [126]. This idea is shown to apply in SIMD load instructions with significant value similarity across accesses of adjacent threads.

Finally, algorithmic transformations are proposed to mimic a region of the imperative code [37]. At compile time, the code is used to train a neural network and the output binary can be separated for the CPU and the accelerator. This way, a neural network hardware accelerator can be used as a generic and flexible approximate accelerator.

Programming Language

Several programming languages are proposed to manually mark the approximation tolerant code and isolate it from the sensitive part. Two main marking strategies are variable annotation and operation annotation. EnerJ [107] extends Java with approximate data types. It enables type checking in compilers to ensure correctness where needed by defining variables as approximate or precise. By limiting the exact operations to which all of its variables are precise, and inferring approximate operations when at least one variable is approximate, it ensures exactness for the critical, approximation intolerant part of the code. Thus, it offers a mean for utilizing approximations in lower layers: The annotated approximated variables can be calculated with approximate units.

Rely [18] annotates approximate arithmetic, logic, and memory operations. It assumes the underlying hardware unreliable with a known error rate and unbound error magnitude. Given the error rate of the individual operations, it finds the error rate of the application, allowing the programmer to choose between approximate or exact operations in order to meet the quality target manually. Chisel [87] proposes automating this search for a given reliability specification.

Runtime

The output quality of an approximated application depends on its inputs. A static approximation may be too conservative, leaving benefits of further approximations untapped, or too aggressive and violate the quality target. Runtime systems with periodic and continuous quality monitoring capabilities are proposed. These systems run a periodic or a smaller sized (canary) exact computation in parallel to the approximate one and compare the results to monitor the quality. In case the quality deviates from the target, they make the necessary accuracy adjustments to maintain it. They do not offer a guarantee on the application quality but rather some control over it by responding to changes in input characteristics. Despite the monitoring overheads of parallel exact computations, previous works have reported significant energy savings directly or through computation speed-up over conservative static approximations.

Green [11] and SAGE [106] propose periodic monitoring and adjustment of general and parallel approximate computations, respectively. The monitoring period offers a fine-grained trade-off between the energy overhead and control for accuracy adjustments.

Rumba [64] and the input responsive system in [72] propose continuous monitoring, by running a smaller circuit and using smaller, canary inputs, respectively. The quality with the smaller circuit and the canary inputs are taken as representatives in setting approximation degree for the larger computation. Similarly, Topaz [1] detects unacceptable outputs of a task to re-execute in exact computation.

ISA & Microarchitecture

Several architecture-level techniques are proposed to utilize hardware approximations. Some hardware architectures are designed to enable approximations: A vector processor involving first-in-first-out buffers (FIFOs) is designed with data precision scaling capability to reduce the data bitwidth in its systolic array [25, 27]. The FIFOs propagate an adjustable number of bits, which can reduce the toggles in the datapath and thereby reduce the dynamic power consumption. The processor is designed with a proportional-integral-derivative controller (PID controller) to self-adjust the data precision. Later, this processor is extended to a custom instuction set architecture (ISA) with *approximate-aware instructions* and thus exposed accuracy knobs to higher layers [120].

Multiple instantiations of datapath units are explored for CPUs [36] and CGRAs [16]. Having such instantiations in the architecture benefit from circuit-level approximations to reduce energy consumption beyond data precision scaling, as these hardwired static-accuracy instantiations require less energy per operation.

Alternatively, the idea of having concise loads and stores maintains the datapath precision of the functional units and reduces the data precision that is loaded or stored. Consequently, it can hold more variables in the caches, increase the cache-hit rate, and reduce the computation time [56].

Finally, together with algorithmic transformation, utilizing a neural network hardware accelerator as an approximation of other kernels is proposed in [92].

Circuit

Circuit-level techniques focused on manual design [14, 133] and automation [80, 103, 112, 122, 123] of functionally approximate circuits. The techniques in this class aim to reduce logic complexity within certain quality bounds. Many fundamental units such as adders [85, 124, 132, 133] and multipliers [71, 77, 129] are manually optimized for accuracy vs. energy or performance trade-offs. Additionally, using underlying circuit properties, such as data packing by multiple simultaneous lower-precision operations using disjoints parts of the adders and multipliers is proposed [90].

Accuracy increases the design space of circuit synthesis by one dimension. This challenge can be tackled in 2 different subcategories. First, given many known manually-optimized approximate fundamental circuits, a question arises on how to use them optimally to build larger circuits such as accelerators. By propagating the error distribution of individual units such as adders, their compiler-driven design space exploration and selection is proposed to build larger circuits [20]. A similar design space exploration is integrated into high-level synthesis flows in [22, 73, 74].

Alternatively, another subcategory is automated techniques that are general, i.e., not specific to a certain circuit structure or functionality. For instance, modifying the boolean function realized by the circuit is proposed in [86, 112]. Removing gates with low toggling probability, *probabilistic pruning*, is

proposed in [80]. Synthesis of a simplified combinational circuit via setting some output bits to *don't cares* for a given quality function is proposed in [123]. This technique is used for sequential logic synthesis in [103]. Simplifying circuits by finding gates that have high correlation and removing one of them is proposed in [122]. This technique finds gates that produce the same output for a majority of the time and substitutes the output of one with another, making the other gate redundant and remove it together with the logic in its fan-in cone.

Transistor & Device

To change state between conducting and not (i.e., to toggle), the gate of the transistor requires an amount of charge to be collected as a function of transistor dimensions. This state change requires a certain time at a certain voltage, previously measured and characterized by the producers, and it is subject to process, voltage, temperature (PVT), and aging variations which are all considered [44]. Techniques at the transistor level utilize charge starvation in the sense that some set of transistors cannot operate exactly and switch their state too late. Traditionally, frequency is determined conservatively, by giving sufficient time plus a guardband to transistors to ensure sufficient charge build-up at the transistor and correctness at the circuit and higher levels. Static timing analysis tools set circuit delays for the worst case, i.e., the longest paths in the circuit (most critical path) and maximum amount of toggles. Approximate computing reduces the delay or voltage and induces timing errors to a degree tolerable by the application. The motivation behind charge starvation is that timing errors occur rarely and gradually with scaling the delay or voltage. Most paths are not critical, often only a subset of gates toggle each cycle, and PVT and aging variations are most probably not the absolute worst-case, leaving the delay and voltage values very conservative.

General techniques that apply to logic are reducing the voltage [42], increasing the frequency [8]. Fine-grained scaling of the timing violations to the criticality of sub-operations is proposed in [62]. Additionally, there are custom techniques that take advantage of the underlying technology. For instance, DRAM holds data in leaky capacitors, and it needs to be periodically refreshed [65, 81, 82]. Approximate DRAMs are proposed that reduce the refresh frequency [59, 84, 101]. Phase change memory and resistive random access memory register distinct analog values to store digital data. the margin and distance between values create a trade-off between density vs. accuracy [95]. To write, the data is checked and corrected multiple times. Also, they eventually age and wear out. These checks during the write process can be reduced at the cost of increased error possibility and the worn-out blocks can be used to store approximation tolerant data [108].

2.3 Dynamic Accuracy Reconfiguration

Dynamic accuracy reconfiguration, as interpreted in the scope of this dissertation, aims to maximally exploit the performance or energy efficiency benefits of approximate computing while meeting a given quality target at runtime. This section gives the background information for a runtime accuracy configurable system and argues for its necessity. The next part gives two motivational case study examples and discusses the background on accuracy configurability.

2.3.1 Accuracy-Configurable System

An ideal accuracy-configurable system would monitor the quality and react by adjusting the computation accuracy for power or performance benefits. Accuracy of such hardware can be adjusted by an internal control mechanism [25, 27], or it can expose knobs to higher levels, such as runtime and software and knobs can be driven through ISA extensions [120].

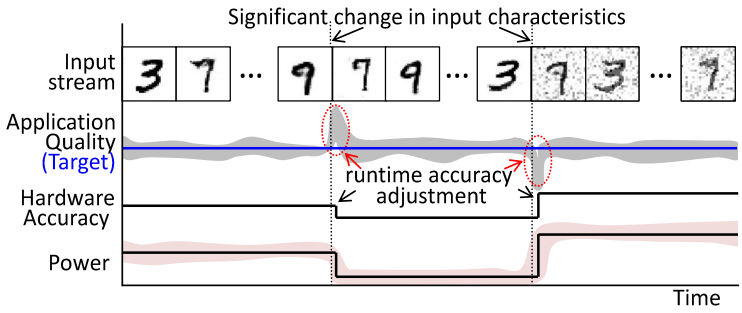


Figure 2.2: An application example targeting a quality, while input characteristics change over time.

Figure 2.2 shows a handwritten digit recognition example in which the input stream characteristics change in handwriting style and noise. A monitoring runtime management system reacts to maintain the application quality at a target. It switches the hardware to a different accuracy at a different power requirement and minimizes the energy consumption when possible. Runtime accuracy management is orthogonal to and out of the scope of this dissertations. Such runtime systems are detailed in Section 2.2. As previously defined, the quality and accuracy terms are distinguished as follows: quality is an application demand, for instance, a classification error rate of $\leq 1\%$. Accuracy is the correctness of the underlying hardware and software, with metrics such as 95% correctness in mean magnitude (5% mean relative error distance). Hence, quality changes with input characteristics and also with hardware accuracy. Accuracy can be changed by, e.g., changing the input data precision.

Apart from the input characteristics, there are several other drivers of the accuracy configuration necessity. Different applications, application parameters, quality targets all require a re-configuration of computation accuracy [106, 128]. Additionally, there can be hardware-sourced reasons, such as changes in maximum power and latency constraints.

Accuracy increases the hardware design space by one dimension. Apart from the question of which unit or accelerator to instantiate, it opens up which accuracy. These reasons make approximate hardware extremely specialized, not only in terms of the application but also the accuracy, and hence significantly limit the utilization and energy or performance benefits of fixed accuracy systems.

2.3.2 Case Study: Edge Detection in Video Frames

Image and video processing are among application domains of approximate computing. Often, it is difficult to distinguish between the raw and compressed or approximated image due to limitations in human perception. A widely accepted quality metric for images and videos is the structural similarity index measure (SSIM). It measures the perceived similarity to the original reference. Sobel Filter is a common image processing kernel and used for edge detection. It works on 8-bit grayscale images. Approximations of the Sobel Filter can be obtained by discarding the least significant bits (LSBs) of image pixels. A higher number of discarded LSBs result in higher dynamic power savings. In Figure 2.3, exact and approximate Sobel Filters are applied to the first 4000 frames of a video [1]. The exact output is taken as the reference in SSIM calculation. The Sobel Filters have the corresponding hardware accuracy given in parentheses in the legend in terms of 1-MRED (mean relative error distance).

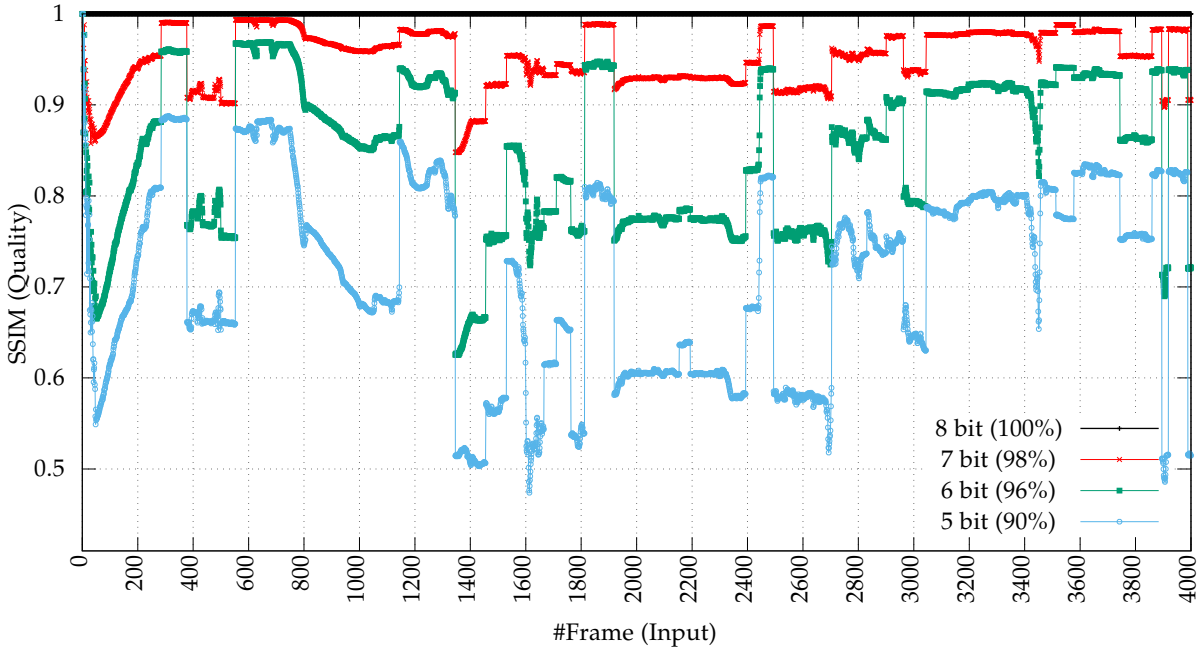


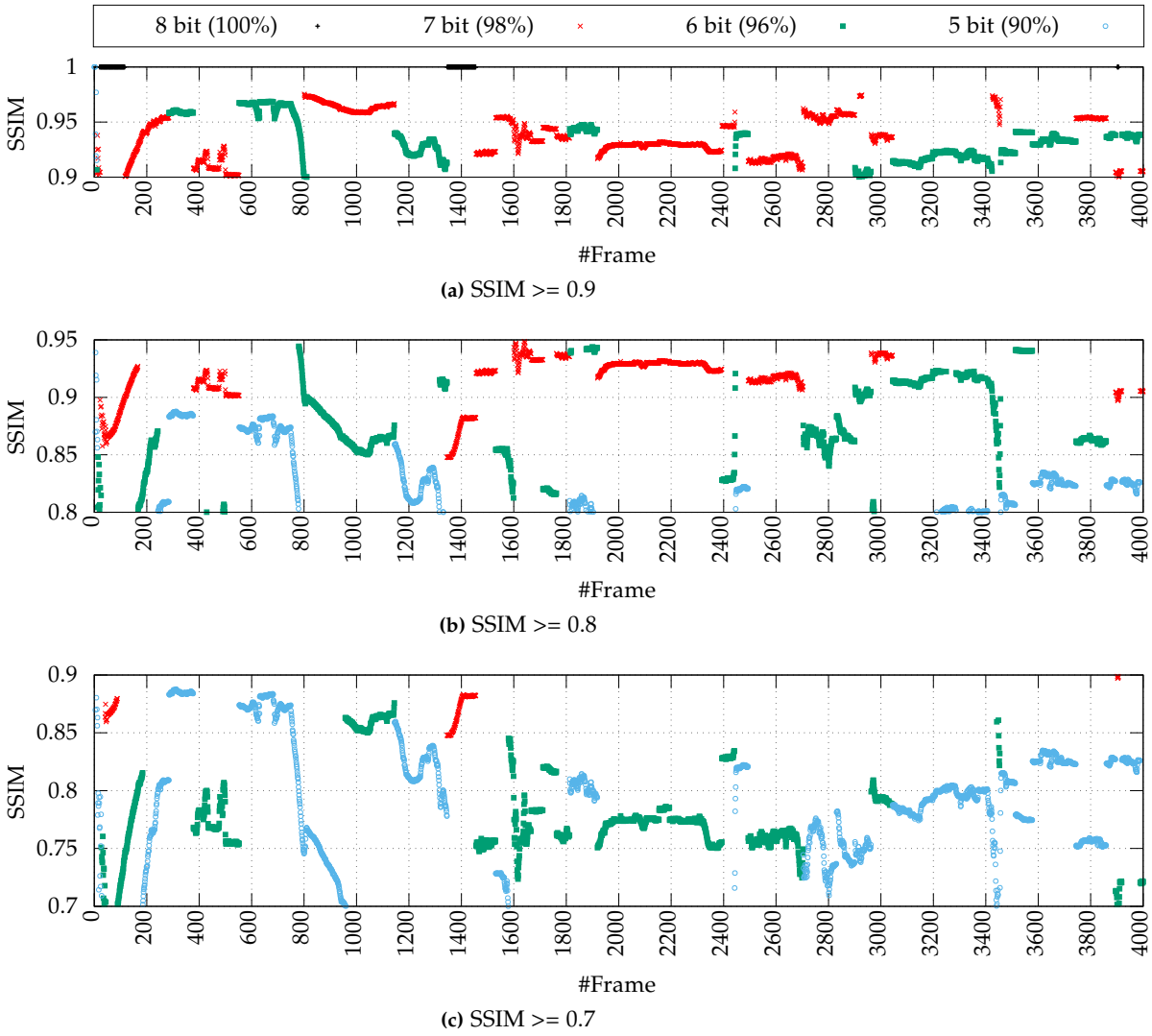
Figure 2.3: Edge detection on Big Buck Bunny [105] with exact (8 bit) and approximate Sobel Filters (7 bit, 6 bit, and 5 bit). Only the first 4000 frames are drawn for space reasons. Accuracy values are shared in the legend in parentheses in terms of '$1 - MRED$', (Mean Relative Error Distance). 8-bit filter represents the exact computation on grayscale frames. The approximate filters are taken from [5, 6].

Notice that, the input frames with the same hardware accuracy result in a large variation in output quality. In other words, to satisfy a given quality constraint such as SSIM 0.9, using any of the units with a fixed hardware accuracy either violates this quality constraint or gives only sub-optimal dynamic power savings, or both. In order to maximally exploit the approximate computing benefits while maintaining the quality constraints, different frames need to be processed with a different accuracy hardware. Thus, the computation accuracy is input dependent.

There is no single, universally accepted quality constraint. Application developers may choose a higher or lower quality, depending on the approximation tolerance of the outputs or the next

[1] This experiment is done in collaboration with Dr.-Ing Jorge Castro-Godínez at KIT-CES.

computation stages. In Figure 2.4, three different quality constraints are considered, such that the application quality should meet or surpass the constraint. The processed frames are the same among the subfigures (a,b, and c) and also Figure 2.3. Figure 2.4 shows that according to the quality constraint, the same frames require a different accuracy.



(a) SSIM >= 0.9

(b) SSIM >= 0.8

(c) SSIM >= 0.7

Figure 2.4: Required accuracy for edge detection with different quality constraints. (a) SSIM $\geq$ 0.9 (b) SSIM $\geq$ 0.8 (c) SSIM $\geq$ 0.7

Comparing the figures, while Figure 2.4a, requiring mostly 7 and 6-bit units, Figure 2.4c requires 6 and 5-bit units for most of the frames. In other words, Figure 2.4 shows that as the quality constraint is reduced, the lower accuracy units can be used more often. Consequently, more approximation benefits can be expected.

This case study has shown the input and quality constraint dependency of the required computation accuracy. Next, the required computation accuracy for different applications, i.e., application dependency of computation accuracy is analyzed.

2.3.3 Case Study: DNN Inference

Workloads of deep neural networks (DNNs), especially inference tasks can be significantly quantized without sacrificing the application quality. Table 2.1, shows a motivational case study. It is a summary of the results of [130], showing *Top-1* and *Top-5* accuracies for several deep neural network models at varying complexity (ResNet-18, ResNet-34, etc.) with ImageNet input set.

Table 2.1: A brief comparison of state-of-the-art quantization methods on ImageNet. "FP" denotes "Full Precision"; the "W/A" values are the bitwidths of weights/activations. Summarized from Table 6 in [130].

Methods	Bitwidth (W/A)	Accuracy (%) Top-1	Top-5
ResNet-18			
FP [41]	32/32	69.6	89.2
XNOR-Net [104]	1/1	51.2	73.2
ABC-Net [79]	3/3	61.0	83.2
ABC-Net [79]	5/5	65.0	85.9
LQ-Nets [130]	1/2	62.6	84.3
LQ-Nets [130]	2/2	64.9	85.9
LQ-Nets [130]	3/3	68.2	87.9
LQ-Nets [130]	4/4	69.3	88.8
LQ-Nets [130]	32/32	70.3	89.5
ResNet-34			
FP [41]	32/32	73.3	91.3
ABC-Net [79]	3/3	66.7	87.4
ABC-Net [79]	5/5	68.4	88.2
LQ-Nets [130]	1/2	66.6	86.9
LQ-Nets [130]	2/2	69.8	89.1
LQ-Nets [130]	3/3	71.9	90.2
LQ-Nets [130]	32/32	73.8	91.4
ResNet-50			
FP [41]	32/32	76	93
ABC-Net [79]	5/5	70.1	89.7
LQ-Nets [130]	1/2	68.7	88.4
LQ-Nets [130]	2/2	71.5	90.3
LQ-Nets [130]	3/3	74.2	91.6
LQ-Nets [130]	4/4	75.1	92.4
LQ-Nets [130]	32/32	76.4	93.2
AlexNet			
FP [70]	32/32	57.1	80.2
BNN [53]	1/1	41.8	67.1
ABC-Net [79]	5/5	70.1	76.3
LQ-Nets [130]	1/2	55.7	78.7
LQ-Nets [130]	2/2	57.4	80.1
DenseNet-121			
FP [52]	32/32	75.0	92.3
DoReFa-Net [131]	2/2	67.7	88.4
LQ-Nets [130]	2/2	69.6	89.1
VGG-Variant			
FP_{-HWGQ} [17]	32/32	69.8	89.3
FP_{-HWGQ} [17]	1/2	64.1	85.6
LQ-Nets [130]	1/2	67.1	87.6
LQ-Nets [130]	2/2	68.8	88.6
LQ-Nets [130]	32/32	72.0	92.5
GoogLeNet-Variant			
FP_{-HWGQ} [17]	32/32	71.4	90.5
FP_{-HWGQ} [17]	1/2	63.0	84.9
LQ-Nets [130]	1/2	65.6	86.4
LQ-Nets [130]	2/2	68.2	88.1
LQ-Nets [130]	32/32	72.9	91.3

Quantization of DNN weights and activation values can be considered as precision scaling, a general approximate computing technique, applied to the neural network domain. Quantizing the values to a smaller bitwidth reduces the computation accuracy lower, and consequently higher approximate computing benefits can be expected. The table shows a large variation between the required bitwidths for a variety of reasons analyzed below.

- There is no single universally accepted classification accuracy (i.e., DNN application quality) target. An application developer may choose a higher or lower target accuracy with the incurring computation costs or savings.
- For a fixed accuracy target, the required precision depends on the model and the learning method:
  - In Table 2.1, a 3-bit implementation of ResNet-34 and 2-bit implementation of ResNet-50 both achieve over 70% Top-1 classification accuracy with LQ-Nets. Alternatively, the same quality target is achieved using ABC-Net with 5-bit precision. In fact, the table represents a large variation of bitwidths and resulting classification accuracy.
  - These state-of-the-art DNN models are from the last 5 years. Due to the recent and active nature of this domain, we can extrapolate that a future, improved model or training method would require a different precision.
- Apart from the DNN application designer's targets, hardware design targets such as latency and power limits can define the required precision. Microarchitectural decisions such as the size of the systolic MAC array can determine the MAC accumulator size to avoid overflow.
- DNN classification accuracy is input-dependent. Unlike these test scenarios on a known input dataset (ImageNet), in real-life we do not have control over the input. Depending on the input complexity, noise, resolution, the required precision would reduce or increase.

Thus, for the reasons listed above, a general and fixed quantization solution cannot be optimal: It either violates the accuracy target or delivers sub-optimal classification-accuracy or energy benefits. Considering the currently massive scale of effort for gradual improvement of DNNs, violating classification accuracy targets or sub-optimal trade-offs are very much undesirable.

Different DNN models can be considered as different applications, that can use the same hardware accelerator. NPUs can be considered as multi-purpose hardware accelerators. The same hardware can be used for multiple different applications and DNN models. Similarly, DNN weights can be considered as application parameters. Given a model, different learning methods set different weights. This case study has shown that apart from the input and the quality constraint, computation accuracy depends on the applications and application parameters.

In summary, the two case studies show that the required computation accuracy depends on application, application parameters, quality target, and input characteristics.

2.3.4 Accuracy-Configurable Hardware

Although approximate computing has received significant interest, the majority of the hardware research efforts explored targeting a single accuracy in manual [124, 132] and automated design [21, 73, 86, 110] of functionally approximate circuits. Runtime monitoring techniques, however, have shown that with temporal variations in input characteristics, the optimal accuracy to meet an application quality target also changes [11, 64, 72]; the single accuracy circuit delivers suboptimal benefits or violates the quality targets [128].

Several runtime methods and strategies for accuracy configurations are discussed in Section 2.2. Next is a discussion of background on hardware methods that realize and benefit from runtime accuracy configurations.

Transistor/ Device

Scaling either voltage lower or frequency higher and thus introducing timing violations is a fine-grain accuracy configuration technique. More number of transistors and paths fail with more aggressive scaling and produce an incorrect signal. The failing paths lead to an error rate depending on their activation probability. A higher error rate leads to an error magnitude depending on the significance of the failing signal. Due to variations in path delays, each path is expected to have a different minimum delay before failing. Thus, scaling voltage or frequency beyond safe limits is a fine-grain technique for introducing errors and thus reducing the accuracy. Timing speculation has received considerable interest in better than worst case design and approximate computing fields.

Graceful error under scaling has been investigated under better than worst-case design, primarily to target recovery based architectures [34]. For instance, Blueshift [38] utilizes a commercial design flow to optimize dynamically critical paths using forward body biasing. DynaTune [125] analyzes a circuit and then improves timing for dynamic critical paths assigning low-Vt cells. The proposed approach in Chapter 3 improves timing on hidden non-critical paths of circuit topologies, which removes the necessity to resort to such physical techniques and can be applied more generally. Cell sizing is proposed to redistribute the slack of frequently exercised near-critical timing paths [61]. This is a subset of standard delay optimizations that may require upsizing the fan-in cone as well to achieve speedups. It is also limited within the boundaries of design rules, such as maximum fan-out and capacitance. This technique occurs in post-layout and miss the opportunities in synthesis. Additionally, they are application dependent on selecting which paths they speed-up, which limits their circuits generality.

Several efforts have exercised timing speculations in the field of approximate computing to find trade-offs between accuracy and energy. Previous work in this class took advantage of applications with known input distribution such as low pass filters [42] and different significance of the paths on the result [62]. Cross-layer techniques [23, 27, 120] utilized timing speculations together with other methods to achieve more favorable trade-offs. Techniques from other abstraction layers such as software [18, 107] and architecture [36] have relied on timing speculations to showcase their benefits. The trade-off for all these techniques rely on error characteristics of the underlying circuit under timing speculations. Additionally, *retiming* is proposed in [102], a technique to improve timing with moving flip-flops earlier or later in between stages. A widespread delay distribution between outputs in a stage is a prerequisite for this technique which traditional synthesis optimizations eliminate. Timing speculations are often performed by voltage scaling [90, 98]. Although voltage scaling reduces the energy when designs are analyzed in isolation, it comes at often ignored system-level costs of multiple additional voltage supplies, rails, and switches.

In contrast to the better than worst-case design techniques, SlackHammer in Chapter 3 takes a logic synthesis approach for graceful errors under timing speculations. This methodology transforms the problem on a well-studied traditional synthesis problem. Unlike And-Inverter-Graph specific techniques [28, 83], it casts the problem on existing commercial synthesis tools hence it leverages a vastly enhanced scope of optimizations and remains compatible with traditional EDA flows. The above distinguishing features make SlackHammer promising in synthesis for graceful errors under aggressive scaling.

Circuit & Architecture

To utilize the temporal variations with approximations, accuracy-configurable hardware [25, 27, 51, 56, 120], and generic design methodologies are proposed [57, 66, 93]. The *de facto* method to configure accuracy is data precision scaling, i.e., not propagating the LSBs of data. In general, precision can be scaled by *gating* the signals and thus blocking their propagation in the combinational paths. Energy-aware precision scaling of floating-point data is proposed in [51]. Data packing: using non-interfering paths of a circuit for simultaneous calculation is proposed in [90]. At the system level, precision scaling is applied in the memory controller [56]. A vector co-processor with data precision reducing FIFO input buffers is proposed in [27]. It is extended with an internal PID controller [25] and later with accuracy-aware ISA extensions [120]. All of these designs apply precision scaling on data, and by doing so, they reduce the dynamic power consumption of the existing circuit. Moreover, function-specific ways are proposed for configurable approximate units [54, 60, 77, 127]. which also benefit from the reduced toggling activity. In contrast to gating existing circuits, when new circuits are synthesized with specific precisions (or accuracy in general), synthesis tools can simplify and optimize them to be more energy-efficient than the existing circuits for the given accuracy, giving up on configurability. A system with energy-efficiency of such static-accuracy approximate hardware and also the accuracy configurability is very desirable to achieve a low energy consumption and retain generality at the same time.

Generic design methodologies can offer improving performance by synthesizing partially faster circuits [8, 90, 98] or energy efficiency by means of disabling low significance logic groups in hardware [57, 66, 93]. Two gating mechanisms are utilized in [57]: (1) Masking logic groups by inserting control gates to their combinational path and (2) power gating the logic groups to partially switch off the hardware. To group the gated logic, a genetic programming search is proposed in [93]. In [66] clock gating for approximations, *clock overgating*, is proposed. By disabling the clock signal of flip-flops, power savings are achieved in their fan-out cone. Similar to existing accuracy configurable hardware designs, gating mechanisms reduce the energy via reducing the toggling activity only.

Several previous works have proposed using the instantiation of distinct circuits that can benefit from synthesis relaxations. In [15], two instantiations of adders and multipliers are used for reliability purposes, targeting circuit delay. The results are shown to even increase the energy, contrary to our primary goal. Multiple instantiations of floating-point units with different accuracies are used in [119]. Some of the exact processing elements of a CGRA are replaced with approximate ones in [16].These architectural solutions may not always be beneficial when leakage power is considered and they make a subset of the architectural proposal in Chapter 5 of this dissertation.

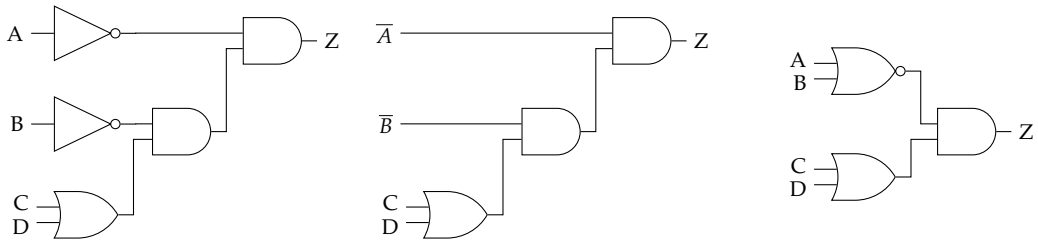
2.4 Logic Synthesis

Logic synthesis is the process of mapping the hardware described in a human-readable language (e.g., Verilog, VHDL) into a *netlist*, i.e., a description of connected cells that exist in the fabrication technology library. This process involves multiple steps for satisfying constraints, such as maximum capacitance, drive strength, etc., and also optimizations prioritizing delay and afterward area and leakage power [31]. An analogy can be made between hardware compiling and software compiling. Many optimizations, such as dead code elimination, constant propagation, common sub-expression elimination, are done during this process.

The goal of logic synthesis can be defined as an optimization problem of meeting delay constraints for every output bit while minimizing area and power. The synthesis algorithm takes delay and area constraints as inputs of a cost function and goes through iterative steps of cost minimization. Delay optimization steps pick the path with the worst timing violation (i.e. negative slack) and apply the speed-up techniques until none of the primary outputs (single output bits) violate the timing constraint. Combinational circuit synthesis is done in 4 phases: mapping, gate-level delay optimization, design rule fix, and area recovery [31]. Particularly the first 2 phases, boolean mapping and gate-level delay optimization determine the circuit delay.

Boolean Mapping

This phase maps the human-readable hardware description to a suitable circuit topology, using gates in the technology library. There can be multiple different representations of the same boolean function, corresponding to different circuit topologies. For example, a boolean function can be mapped directly to the circuit as given in Figure 2.5a. In case inverted A and B are available in the circuit, the not gates can be eliminated as in Figure 2.5b. The circuit given in Figure 2.5c is a boolean optimized topology of the same function with less delay and area.



(a) Original **(b)** Optimization 1 **(c)** Optimization 2

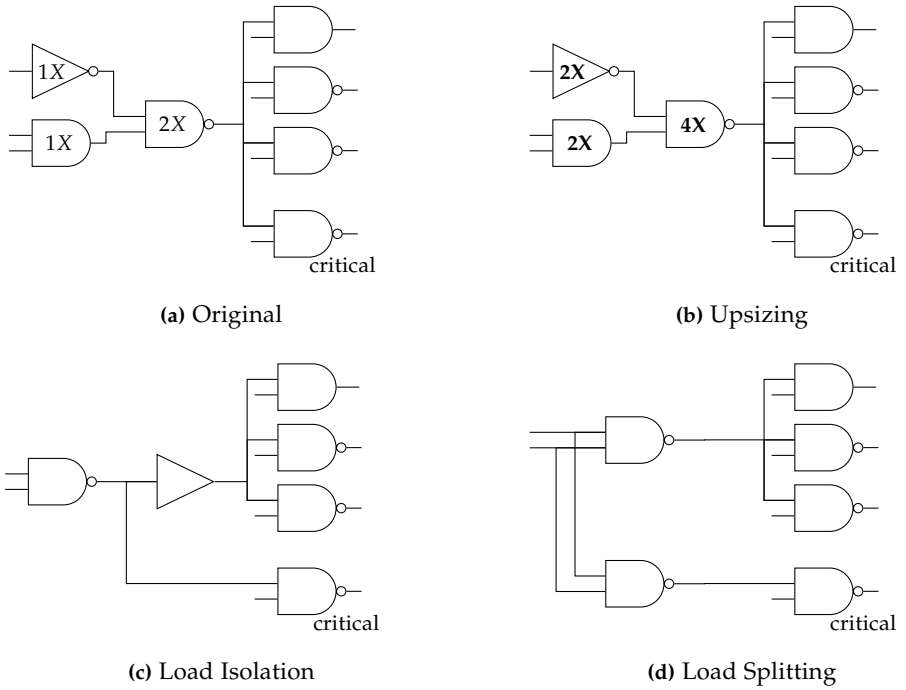
Figure 2.5: Synthesis and optimizations of the function $Z = \overline{A}(\overline{B}(C + D))$ (a) Original, (b) Common sub-expression elimination, and (c) Boolean Optimization.

For common circuits, the synthesis tool can contain a variety of topologies in its library. For instance, the tool contains multiple adder topologies, designed with different optimizations in focus [40, 69]. A common issue among them is that they suffer from long carry chains. These carry chains form a critical path and limit the maximum achievable frequency. A simple Ripple Carry Adder (RCA) has a carry chain that is in linear relation with its bit-width ($O(n)$) resulting in limited maximum frequency. Faster topologies such as Parallel Prefix Adders generally have a logarithmic relation between their critical path and bit-width at increased hardware requirements and, as a result, increased power consumption. Given a delay constraint, the synthesis tool picks the most suitable circuit topology in its circuit library and creates an initial netlist that roughly meets the constraints. The following phases do the fine-tuning.

Gate-Level Delay Optimizations

Circuit topologies chosen in the boolean mapping phase have varying delays between their paths. The paths violating the circuit delay constraint can be corrected with gate-level delay optimizations. At each iteration, the algorithm finds the slowest path, i.e., *the most critical path*, and performs delay optimizations to meet this circuit delay.

There are three different gate-level delay optimization methods: *upsizing*, *load isolation*, and *load splitting*. These methods speed up the circuit by increasing fan-out or decreasing load on critical paths.



(a) Original **(b)** Upsizing

(c) Load Isolation **(d)** Load Splitting

Figure 2.6: Gate-level delay optimizations to reduce the most critical path delay [31]

Upsizing (Fig.2.6b) increases cell size, enables higher output current and faster transition of consequent cells. However, it increases the capacitance and transition time of the upsized cell itself. To achieve speedups on a path, the entire fan-in cone needs to be considered for upsizing within a certain range of cell sizes in the technology library. Load isolation (Fig.2.6c) and load splitting (Fig.2.6d) decrease the load of a gate which drives the critical path among others, by driving the other, non-critical paths via a buffer or another equivalent gate respectively. All 3 methods increase the area by increasing cell size in upsizing or placing additional buffers in load isolation and equivalent gates in load splitting.

Tracing the gate-level delay optimizations backward, these optimizations can reduce the critical path delay further at an increasing logic, area, and power cost. In Figure 2.7, the possible cost of iterative delay improvements is shown. The synthesis tool first chooses the minimum cost, maximum delay reduction option (shown as *'1'*). In case improving the delay by load splitting on one gate is not sufficient in the next iterations, an increasing number of gates may need to become faster. This can lead to significant costs. The example shows an exponential increase in iterations 1-2-3 as a worst-case scenario.

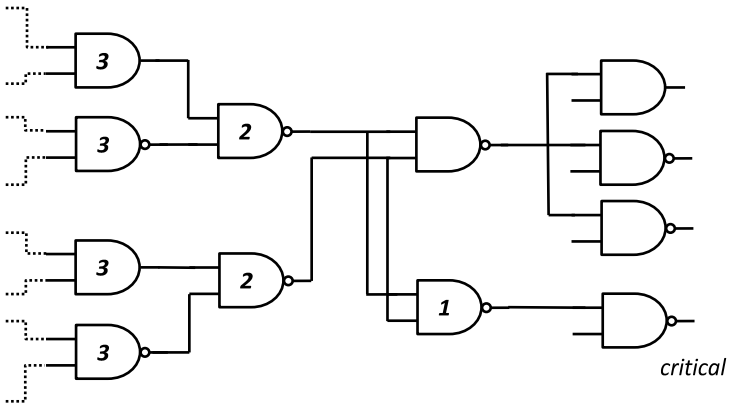


Figure 2.7: Iterative critical path improvements at increasing cost. After load splitting in iteration 1, two gates need to be faster in iteration 2 and four gates need to be faster in iteration 3.

Synthesis with Zero Delay Constraint

To maximize the delay improvement effort, a delay constraint of '0ns' can be set. The delay constraint of '0ns' cannot be realized as a circuit cannot have such delay. In this case, the synthesis tool chooses the fastest circuit topology and applies a maximum amount of gate-level delay optimizations. To obtain the fastest circuit, these optimizations extend from the output to the input of the combinational part. The tool eventually stops when it is unable to improve the worst delay any further, but still produces an output circuit, reporting a negative timing slack.

During synthesis, in every iteration, the tool picks the most critical path and searches for maximum delay improvements with minimum cost. If the optimizations are successful, the most critical path can change between the iterations. The second slowest path can become the most critical. This iterative optimization of only the most critical paths results in a high number of paths with near-identical delays.

Synthesis with Minimum Delay Constraint

Circuits can be synthesized to the minimum delay by setting a constraint that can be obtained as the negative slack in the previously explained *zero delay case*. Upon meeting the delay constraint, the synthesis tool proceeds to the next phases: design rule fix and area recovery. Briefly, design rule fix checks if the circuit complies to design constraints, such as maximum capacitance or fan-in, fan-out of the inputs and outputs. Area recovery phase can be considered as the inverse of gate-level delay optimizations. After the delay optimizations, some paths will remain to have timing slack. This phase searches for opportunities to undo gate-level optimizations wherever possible. For instance, in Figure 2.7, the region above gate '*1*' separating from the critical path can be area optimized. Thus, an ideal and successful traditional logic synthesis results in evenly distributed path delays.

It is a powerful tool, boolean mapping and gate-level delay optimizations together offer a big range for the delay, area, and power. Therefore, any circuit-level comparison should be made considering the synthesis impacts. when considering circuit-level trade-offs, synthesis impacts should be a prime concern.

2.4.1 Synthesis of Approximate Units

Accuracy as a design metric increases the design space by one dimension. One main category of approximate computing in hardware is functional approximations, where the boolean functionality of approximate units deviates from the exact to a limited extent. Functional approximations in general reduce the size of the logic and also shorten the critical path. Therefore, the energy benefits of functional approximations are two-folds: The reduced logic results in less area, leakage, and fewer transistor toggles and dynamic power. The shorter critical path allows for synthesis relaxations. The circuits can be synthesized more energy efficiently. However, there can be many different ways of functional approximations of the same unit and with the same accuracy, differing in circuit implementation, and hence in critical path length.

This subsection shows that, for the same accuracy, the critical path length is a dominating factor for circuit metrics such as delay, power, area, and leakage with a case study on lower-part approximate adders.

Lower-Part Approximate Adders [2]

As a fundamental building block for integrated circuits, adders have grabbed a lot of attention to showcase approximation methods. Particularly, lower-part approximation has become a major category. Methods in this category approximate a number of LSBs while they compute the MSBs exactly. Considering the exponential reduction of significance from MSB to LSB, a large number of bits can be approximated without a big loss of accuracy. Thus, the lower-part approximation can address the common issue of adders by reducing the length of the carry chain significantly. Here, the approximation method used on LSBs has a direct impact on computation accuracy. In other words, for the same inaccuracy tolerance, the number of approximated bits and the length of the carry chain can change significantly between the methods.

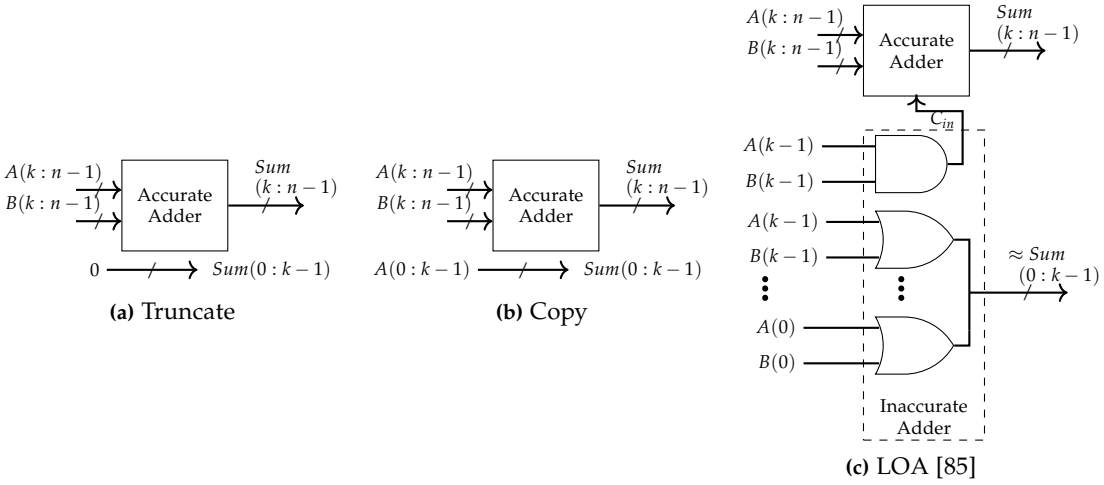


Figure 2.8: An abstracted circuit-level comparison of lower-part approximate adders (a) Truncation, (b) Copy, and (c) Lower-Part OR-Gating Adder [85].

Figure 2.8 shows three different lower-part approximation methods. Truncation simply provides a zero value instead of adding the lower part of operands. Copy method propagates the lower part of an operand to the output. Lower-Part OR-Gating Adder (LOA) [85], uses OR gates as an

[2] The content of this subsection is based on work originally published in [7].

approximate unit and an AND gate to provide carry to the accurate part. OR gating is a good approximation of bitwise addition. It provides the correct result when there is no carry and when there is carry, it fails small. Using information entropy of both operands, LOA is expected to show better accuracy than truncation and copy methods. For all three circuits shown in Figure 2.8, the critical path is in the exact adder part. The critical path of a single OR gate for the approximated unit also translates as relaxed timing constraints and can be synthesized with slower cells that characteristically have a small area and leakage current for the lower, approximate part.

Impact of Approximations on Critical Path Length

Figure 2.9 shows a comparison of lower-part approximation methods using behavioral models of each adder in MATLAB, for a varying number of approximated bits with 100.000 uniform random values as operands. In this figure, each approximated bit reduces the exact part. It shows two key insights. First, for each method, there is an accuracy vs. critical path trade-off. Second, the arrow shows a significant difference in critical path length between the methods at the same accuracy. For the same error tolerance, the truncation method can approximate 4 LSBs while LXOA [9] can approximate 7. As another example, a 16-bit approximate adder with the truncation method requires a 12-bit exact part while with LXOA requiring only 9-bit.

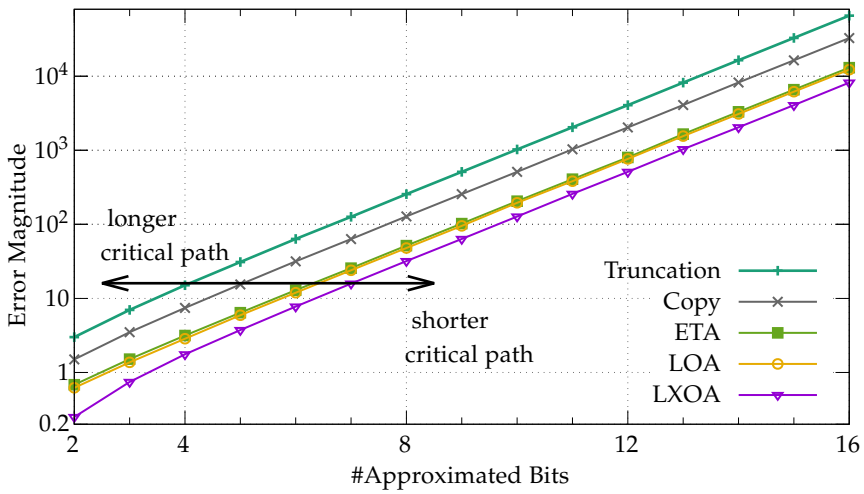
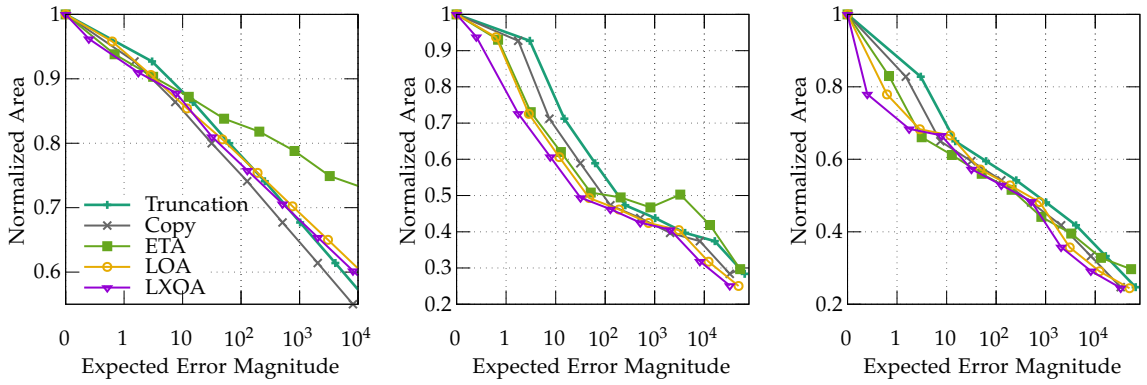


Figure 2.9: Error magnitude comparison of lower-part approximate adders, comparing the truncation, copy, and LOA [85] methods previously described in Section 2.4.1, and also ETA [132] and LXOA [9]. The arrow shows that, for the same accuracy, a significantly different number of bits can be approximated where each approximated bit reduces the carry chain, and hence the critical path in the exact part.

2.4.2 Impact of Critical Path Length on Circuit Trade-offs

As shown above, accuracy is proportional to the critical path length for all of the shared approximation methods. The critical path length directly impacts circuit trade-offs such as area and power, as discussed in detail below.



(a) CLA at 125 MHz **(b)** KSA at 2 GHz **(c)** DWA at 4.5 GHz

Figure 2.10: Area comparisons of lower-part approximate adders. (a) 32-bit Carry-lookahead Adder. (b) 32-bit Kogge-Stone Adder with radixes 4-4-2. (c) 32-bit DesignWare Adder.

Area vs. Accuracy

The area plots in Figure 2.10 start with and normalized to the same accurate adder, at 0 expected error magnitude. We synthesized the adders with lower-part approximation by increasing the approximate LSBs by 2 at each sample point ($k = 0, 2, 4, ...$).

As a corner case, Figure 2.10a shows a very low-frequency design where the critical path length does not become a design concern. A clock period is enough to synthesize the circuits with the smallest and slowest gates in the technology library. The area of the accurate part of the adder reduces linearly. LOA and LXOA methods add gates to the lower part at linear cost. The additional gate costs of the ETA are the highest. Here, methods without gate-costs offer better area vs error trade-offs with a small difference.

Critical path becomes a prime design concern at higher frequencies than the previous example. We implemented 2 high-frequency adders. Figure 2.10b shows Kogge-Stone adders [68] with radixes 4-4-2 (2 at last level) and synthesized for 2 GHz. Our implementations combine the Kogge-Stone adder for the higher part with approximation methods for the lower part. In Figure 2.10c, we use the fastest adder in Synopsys DesingWare library and optimize with *compile_ultra* option. Figures 2.10b and 2.10c show that the methods with higher accuracy relax the critical path further and lead to better area vs. error trade-offs.

Power vs. Frequency vs. Accuracy

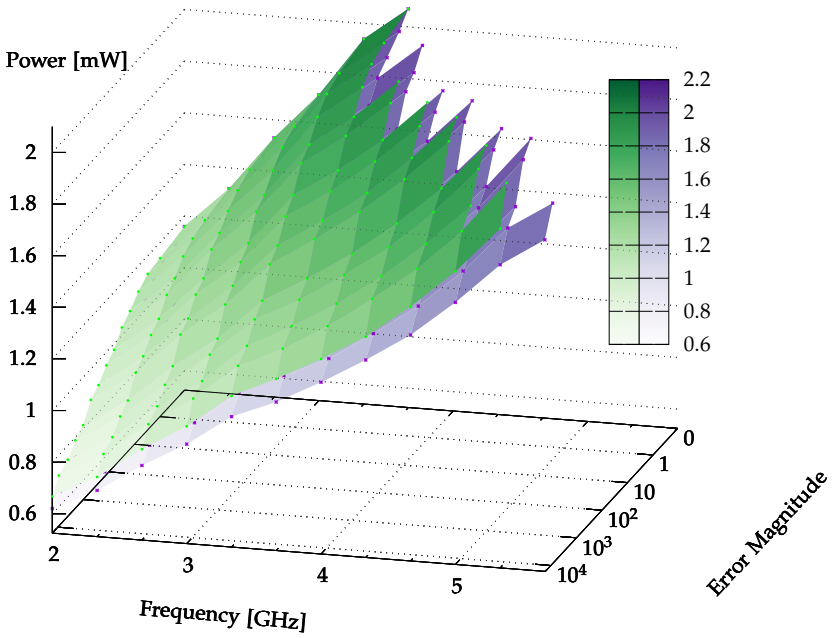


Figure 2.11: 32 bit Adder with varying lower-part approximation
Truncation in green tints, *LXOA* in purple tints

In Figure 2.11, we present power vs. frequency vs. error for the implementations of truncation and LXOA methods under varying approximation. Starting with the same, accurate DesingWare adder, we synthesized both methods with increasing approximated LSBs one by one. We ran gate-level power simulations using Synopsys PrimeTime and *.vcd* files with 333 MHz frequency increments from 2 GHz until getting timing errors. Each corner of a shaded rectangle is a value from the power simulation. Both designs synthesized to be accurate at 4.5 GHz managed to reach 5.66 GHz in our tests with 16-bit approximation, however, with an 8× difference in average error magnitude. When the other parameters are the same, LXOA achieves up to 10% power savings or ∼300 MHz higher performance. For both methods, power is inversely linear with approximation. With 16 approximated bits, at any frequency, the power consumption reduces to 52% with truncation (with relatively higher error magnitude) and to 53% with LXOA, however, at different accuracies. We can observe that the purple LXOA plane is consistently under the truncation plane. In other words, the method that results in the shortest critical path has better power, frequency, or accuracy metrics than the method with a longer critical path.

2.5 Summary

This chapter described the necessary background to convey the contributions of this dissertation in technical depth. There is a necessity of *disciplined* approximate computing to isolate approximation tolerant and sensitive parts of an application. Section 2.2 details the prior work across the stack to realize and benefit from approximate computing.

Although exactness is relaxed, to maintain the quality constraints of an application with changing input characteristics, to generalize the computing system and support multiple applications, or to support different quality needs of the same application Section 2.3 has shown that accuracy configuration is essential. Furthermore, this accuracy configuration necessity can be driven completely externally, with a change in the complexity of the inputs. Therefore, the configuration needs to be addressed at runtime. This runtime accuracy configuration necessity is highlighted by two case studies.

Accuracy increases the design space by one dimension and makes it significantly larger for hardware synthesis. As discussed in Section 2.4, critical paths are a dominating factor for circuit metrics such as delay, power, area, and leakage. Circuit-level approximate computing techniques can shorten the critical path by reducing the accuracy and, as a result, significantly improve the circuit metrics.

Outline

Following the analyses shared in this chapter, this dissertation makes the following novel contributions:

Chapter 3 introduces a novel combinational circuit synthesis methodology that aims at delay optimizing the non-critical paths so that a minimum number of paths fail when frequency is scaled beyond safe limits. Consequently, it makes the accuracy vs. frequency trade-off more gradual.

Chapter 4 introduces a novel circuit synthesis methodology that relaxes the critical paths to improve the overall circuit energy efficiency, and tightens the delay constraints on non-critical paths, making it possible to run low precision operations at a faster frequency to meet an average throughput.

Chapter 5 introduces a novel hardware synthesis methodology that instantiates multiple hardware blocks with the same functionality at different accuracies. The approximate instantiations have shorter critical paths, and hence significantly less dynamic power and area cost than the exact hardware.

3 Synthesis for Graceful Timing Violations

Motivation [1]

Traditionally, error resilience is ensured by providing sufficient guardbands on the circuit level. However, alternative approaches have emerged, to reduce the increasingly high safety margins due to process and runtime variations in advanced technology nodes [44]. For instance, the idea of recovery-based computing depends on detecting and recovering from timing errors, but which has high recovery overheads [34]. On the other hand, approximate computing relaxes the correct execution strictness for application domains that possess intrinsic error resilience. One of the main hardware approximation category is circuit-level timing speculations, where voltage or frequency is scaled aggressively, and beyond guardbands, inducing timing errors upon activation of critical paths [42, 62, 89]. Several previous efforts utilized both hardware techniques together with software in a cross-layer fashion and reported more favorable trade-offs than any singular technique [23, 27, 120].

The motivation behind introducing timing speculations is that most input combinations do not invoke the critical path and can be accomplished in a shorter time. A frequency increase or voltage reduction while introducing timing errors creates a trade-off. Given an error tolerance, energy and performance improvements depend on the error resilience of the underlying circuit. If the circuit produces rare or insignificant errors under aggressive voltage or frequency scaling, bigger energy and performance improvements are possible. However, unless the delay constraints are exceedingly large, traditional synthesis algorithms result in circuits that contain a large number of near-critical paths. In consequence, these circuits possess a characteristic in which under aggressive scaling either no error occurs, or a very large number of paths fail at the same time. Although this is desirable for area and power reductions in exact computation, it hinders the benefits of timing speculation.

Figure 3.1 showcases traditional syntheses vs. a motivational example. We synthesized a 32-bit adder and a 16-bit adder which operate completely independently but with the same clock cycle time. Characteristically adders have their critical path at MSB because of the carry chain. The figure shows that most primary outputs (PO) have an almost identical delay to MSB, despite that logic depth and complexity to produce them are different. Majority of their paths having identical delays prevent graceful errors under aggressive scaling. Previous work [61] has called this phenomenon as *wall of slack* and several efforts proposed modifications on the circuit to redistribute path slack [38, 61, 125]. Although they are promising, since they perform after synthesis, they are limited in the scope of optimizations.

[1] The content of this chapter is based on the work originally published in [8].

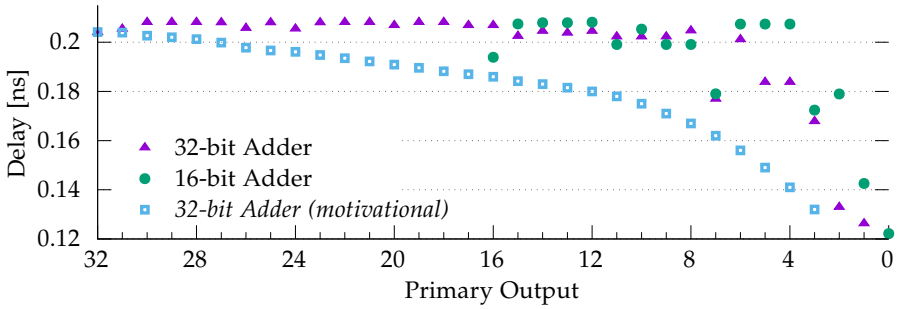


Figure 3.1: Circuits synthesized with a constraint: $t_{ckt} = 0.21ns$. Most primary outputs have almost identical delays.

In this chapter, we present a novel systematic synthesis methodology to enhance circuit-level error resilience under timing speculations. Given any circuit in RTL description, our method finds and applies tighter constraints in synthesis towards minimizing timing violation when frequency is overscaled. It works in 2 phases. First is an assessment of path delay values in isolation, i.e., without considering the rest of the paths. This phase exposes potential delay improvements in the circuit topology. Second is an iterative synthesis phase based on the the delay values in isolation. In this way, for the first time, delay optimizations in the traditional synthesis tools target non-critical paths which also violate timing under aggressive scaling. The key advantages of this work are as follows:

- The proposed methodology minimizes the number of near-critical paths, thus facilitates circuits with more favorable accuracy and performance trade-offs in the presence of timing induced errors.
- Topology based delay optimization that we introduce can be applied to any circuit, independent of the application which makes our approach general.
- Our methodology maps synthesis for timing speculations to a well-studied traditional logic synthesis problem hence it leverages the entire scope of optimizations in commercial tools and remains compatible with traditional EDA flows.

We evaluated our methodology in the synthesis of a wide range of simple and complex arithmetic circuits. The experimented circuits have shown up to 93% reduction in the number of near-critical paths, i.e., paths within 5% of the circuit delay. In consequence, error under aggressive frequency scaling is reduced up to 7x in magnitude and 15x in rate. Additionally, we observed carry chains that exist in many arithmetic circuits confine the biggest speedup margins to LSBs paths. We demonstrate utilizing these margins with precision scaling, as it is commonly used together in cross-layer techniques [23, 27, 120], an additional 27% frequency increase is possible. The area and power overheads of the circuits synthesized with our methodology are up to 14% and 12% respectively.

3.1 SlackHammer: Preliminaries and Approach

The problem statement for synthesis of circuits for timing speculations can be described as follows. The large number of near-critical paths in circuits synthesized with a traditional algorithm prevent graceful occurrence of errors under aggressive voltage or frequency scaling. Given any hardware in RTL description, SlackHammer aims at synthesizing circuits with tighter delay constraints towards

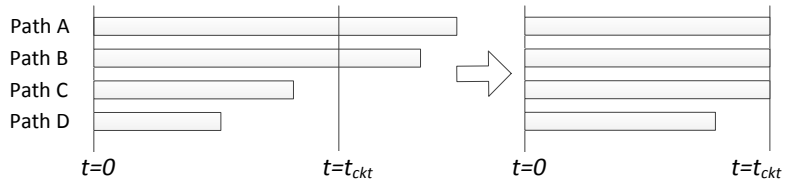


Figure 3.2: Path delay optimizations in traditional synthesis aim at meeting the single, worst case circuit delay t_{ckt}

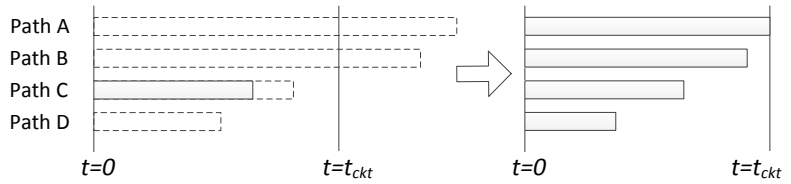


Figure 3.3: Our non-critical path delay optimizations aim at meeting individual path delays, e.g. path C, better than t_{ckt}

minimizing the path delays. This section gives an overview of a traditional synthesis algorithm, standard delay optimizations and our approach utilizing them in addressing the problem.

3.1.1 Traditional Logic Synthesis

The goal of logic synthesis is to solve an optimization problem of meeting delay constraints for each primary output while minimizing area and power. Synthesis algorithm takes delay and area constraints as inputs of a cost function and goes through iterative steps of cost minimization. Traditionally, circuits are synthesized with a circuit delay to meet a clock cycle. At each iteration, the algorithm finds the worst violator, i.e., the most critical path, and performs delay optimizations to meet this circuit delay.

In case the synthesis cannot meet the delay constraint for the most critical path, it does not spend effort on reducing the delay of other, less critical paths. We also observe that existing synthesis options such as setting a critical delay range do not optimize the less critical paths to the extend where their delays become unbalanced, despite an unbalanced depth in the circuits topology. As a result, some paths never become the worst violator and receive delay optimizations. The synthesis proceeds to a consequent phase such as area recovery: for the paths with positive slack, it searches for opportunities to reduce area via undoing delay optimizations on non-critical paths, while still meeting delay constraints.

Figure 3.2 gives an example on how the traditional optimizations result. This circuit contains 4 independent paths with varying depths and delays. It is constrained with the minimum delay (t_{ckt}) limited by the path A. Paths A and B receive delay optimizations to meet the circuit delay. Paths C and D receive area optimizations. In this example, D is a short path for which the circuit delay is exceedingly large. It is composed of the smallest available gates of the technology library and cannot be any slower to recover any further area. This example shows how the traditional synthesis algorithms result in almost identical path delays despite that example circuits topology has varying depth and delays.

As a side note, our observation is that reducing delay of a path has increasing area costs. For example, in Figure 3.2 a unit speedup on path A has a higher area cost than path B or C. This can be accounted to diminishing returns from the delay optimizations as speedups need to be applied to the fan-in cone, to an increasing number of gates. On the other hand, it reveals an opportunity to speedup the non-critical paths with lesser area overhead than paths that are already highly delay optimized.

3.1.2 Non-Critical Path Optimization

In this work, we propose applying delay optimizations on non-critical paths, as shown in Figure 3.3. Timing errors occur when a long path is invoked without sufficient time to propagate the correct values. Traditional synthesis algorithms, by maximizing the number of near-critical paths, produce circuits that have high probability of timing errors. For example in Figure 3.2, error probability of the circuit after optimizations $P_E(Ckt)$ is error probability on any of the paths A, B and C since they are synthesized with the same delay.

$$P_E(Ckt) = P_E(A \cup B \cup C) \geq P_E(A) \tag{3.1}$$

Path A being the longest, it may not be possible to optimize it any further due to technology limitations. But the remaining paths B and C could be further optimized for delay to reduce the probability of timing errors. This limited example scales up significantly as most circuits have a large number of near-critical paths with the potential to reduce their delay. For example in Figure 2.6, cells that are not marked *critical* and their entire fan-out cone is not considered for delay optimizations. We call these paths whose delay could be reduced further *hidden* non-critical paths as in traditional synthesis tools it is not possible to identify them and assess their margin for further delay reductions.

In order to reduce a circuits error probability, $P_E(Ckt)$, we need to minimize the number of *hidden* non-critical paths. The blackbox nature of commercial EDA tools makes identifying them and their improvement margin a non-trivial task.

3.1.3 Synthesis for Graceful Errors

We next describe our strategy to transform the graceful error degradation problem into a traditional logic synthesis problem. As mentioned, the synthesis tool optimizes only the worst timing violator of each iteration. Our solution is to individually constrain non-critical paths as tightly as possible, so they become the worst violator during the iterations of synthesis as many times as possible. Timing violation can be defined as the difference between the constrained value and the signal arrival time according to static timing analysis. By finding the right path delay constraints, we can transform the non-critical paths to critical.

The question that arises is how to find right delay constraints for all paths of a given circuit. The constraints must be tight to maximize the delay optimizations on the path and yet, they should not create a bottleneck which causes the synthesis to fail and discard the delay optimizations on other paths. We tackle this question in 2 steps. First we find a good estimate of the constraints by composing the boolean functions of primary outputs (PO) individually, in isolation. We detail this in Section 3.1.4. Second, we iterate over the estimates until we find a successful constrain set, taking cell sharing effects within the topology into account which we detail in Section 3.1.5.

Granularity of Primary Outputs: Our approach is suitable for the synthesis of independent logic blocks to achieve the minimum path delays in each circuit. When multiple logic are encapsulated such as a simple ALU that consists of an adder, a shifter, constraining them together on the primary outputs of the ALU with the same, tightest, yet worst case delays may lead to suboptimal trade-offs for the simpler logic. In such a case, after synthesizing the logic independently, the resulting netlists can be encapsulated. A compiler level technique can be used in their joint error analysis [20].

3.1.4 Path Analysis in Isolation

Non-critical paths receive limited delay optimizations when the circuit delay is determined by a critical path. To find out path delays in the absence of circuits critical path, we propose *path isolation*.

We isolate the circuit at the granularity of primary outputs, i.e. the single output bits of the logic, and synthesize this path group separately to find a minimum delay value for each primary output (PO). Path isolation exposes the minimum delay to implement the boolean function of a single output bit by virtue of: (i) targeting the delay optimization only on the subset of paths with the designated output and (ii) eliminating load from the other POs on shared cells which may slow down both paths. This minimum delay value roughly correlates with the logic depth but more accurate as different logic gates would have a different gate delay.

We obtain the isolated delay values this way, through synthesis, for all primary outputs, and heuristically understand the PO with biggest isolated delay value constitute the critical path. By comparison, we assess the non-critical paths delay improvement margin: The margin between a PO delay and the critical PO reveals how much delay improvement is possible on the paths leading to this non-critical PO. This way we utilize the synthesis tool for the analysis of path delays in isolation. As explained, we use path analysis in isolation for the purpose of finding good estimates of PO delays of a given circuit. The resulting isolated circuits, despite being faster, are not really feasible because of very large area and power overheads. For instance, for the 64-bit Kogge-Stone adder that we later use in our experiments, these overheads are 22.7x in area and 20.6x in power.

Most circuit topologies share a significant portion of the logic in computation of their primary outputs. In other words, intermediate values (signals) of a PO are used in computation of many other POs. Common subexpression elimination in synthesis avoids duplication of these logic when possible. Sharing the intermediate values increase the fanout of shared cells and cause them to slow down to a small degree.

3.1.5 Constraining Path Delays

To take load increases on shared cells into account, delay constraints of POs must be larger than the isolated delay values from Section 3.1.4. But exactly how much larger remains as a question. The degree of cell sharing and consequent slow down effect differ for each path. We use an agnostic approach to solve this problem. Initially we increase the isolated delay values conservatively, using a constraining function (detailed in Section 3.2). This function estimates delay values smaller than synthesizable constraints. Following that, we iteratively increase the delay constraint of worst violating PO with a small value until reaching a successful synthesis. Such iterative process enables a rigorous search for delay optimizations.

In Figure 3.4 we compare primary output delays of circuits synthesized with iteratively updated constraints vs. direct (non-iterative) constraints. Our baseline is PO delays of synthesis with circuit delay constraint t_{ckt} = 0ns. Setting relaxed delay constraints such as slightly higher than our baseline:

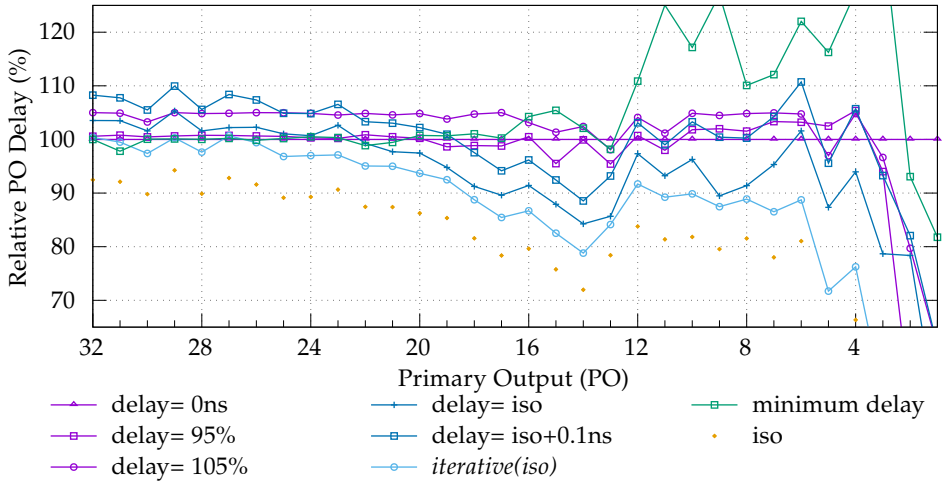


Figure 3.4: Synthesis of 16-bit multiplier circuits with different PO delay constraints, relative to the synthesis with circuit delay constraint t_{ckt} = 0ns

delay= 105% or *minimum delay* ($min(t_{ckt})$) leads to a successful synthesis where all delay constraints are met with slow POs. Setting tighter delay constraints such as slightly lower than our baseline: *delay= 95%* or the isolated delay values: *delay= iso* or relaxed isolated values: *delay= iso +0.1ns* leads to a failed synthesis; delay optimizations stop when the worst violator cannot be improved further, before meeting the constraints, leaving many POs delays sub-optimized. In contrast iteratively constraining based on isolated values: *iterative(iso)* converges to a successful synthesis by iteratively relaxing the worst violators delay values and achieving lower PO delays than comparison.

3.2 Design Methodology

This section presents a systematic methodology for synthesizing circuits using isolated path analysis and iterative constraining as previously explained.

Algorithm 1 describes the SlackHammer methodology for circuit synthesis with delay optimizations on non-critical paths. Given a hardware design described in RTL (RTL_{orig}), the algorithm finds the minimum timing constraints on primary outputs (PO_i) and produces a circuit in netlist form (Ckt_{out}) with widespread delay distribution towards minimizing timing violations under aggressive scaling. For baseline comparison, we synthesize the circuit with a standard approach, '0' delay constraint on all primary outputs (lines 3-4).

Phase 1: We obtain the minimum delay values (D_i) for all outputs with isolated path analysis. This process is explained in Algorithm 2. Using these isolated delay values we calculate our output delay constraints ($CN.O_{iso}$) for each individual PO_i with a constraining function (line 8). This function reshapes delay distribution of primary outputs. We employ a simple constraining function to minimize the path delays with parameters α and a bias β. We multiply D_i values with α with a rationale that slowing effect of sharing cells is relative to the path length in general. Some POs, especially LSBs of arithmetic units output, have high fanouts and very small delay values that do not increase enough with α and become bottlenecks in synthesis. We use a bias value β to relax their constraints.

Algorithm 1 Pseudo-code for non-critical path optimization

Input: Original circuit in RTL description: RTL_{orig}
Output: Netlist with widespread delay: Ckt_{out}

1: **Begin**
2: Read the RTL_{orig}
3: $CN.O_{orig}$ = Output delay constraints set '0' for all endpoints
4: Ckt_{orig} = synthesize (RTL_{orig}, $CN.O_{orig}$)
5: D_{orig} = Worst negative slack of Ckt_{orig}

Phase 1 – Path Analysis

6: **for** each PO_i : Primary Outputs $\in Ckt_{orig}$ **do**
7: $D_i = get\_isolated\_path\_delay(RTL_{orig}, PO_i)$
8: $CN.O_{iso}[PO_i] = \alpha * D_i + \beta$ ▷ Initialize constraints
9: **end for**

Phase 2 – Iterative Synthesis

10: Ckt_{out} = synthesize (RTL_{orig}, $CN.O_{iso}$)
11: $<WNS, PO_{WNS}>$ = Worst negative slack & PO pair
12: **while** $WNS < 0$ **do**
13: $CN.O_{iso}[PO_{WNS}]$ = min ($CN.O_{iso}[PO_{WNS}] + \delta$, D_{orig})
14: Ckt_{out} = synthesize_incremental (Ckt_{out}, $CN.O_{iso}$)
15: $<WNS, PO_{WNS}>$ = Worst negative slack & PO pair
16: **end while**
17: **return** Ckt_{out}
18: **End**

The constraining function gives us flexibility to meet the requirements of different design points. For instance, quality constraints such as output significance can be incorporated into the constraining function when the circuits are not synthesized for their highest frequency. Also reducing delay partially, on a subset of POs, may create hold violations. In case minimum delay constraints on the outputs are necessary, they should be considered in the constraining function as a lower bound.

Phase 2: As explained in Section 3.1.5, in case there are POs with negative slack we iteratively converge to final constraints and the circuit by increasing the associated POs constraint with a small value (δ) in each step until the synthesis is successful (lines 12-16). This way we obtain the tightest output delay constraints our circuit can fit in. Reducing the worst negative slack by a δ changes the most critical path of the circuit. In consequence, an increased number of paths become critical during the iterations of synthesis. They receive additional delay optimizations which in a traditional synthesis they do not. When the constraints for a successful synthesis are reached, the algorithm returns Ckt_{out} (line 17).

Algorithm 2 Pseudo-code for isolated path delays

Input: Original RTL description: RTL_{orig}, Primary Output: PO_i
Output: Isolated delay of the Endpoint, PO_i: D_i

1: **Begin**
2: Read the RTL_{orig}
3: $CN.O_i = \infty$
4: $CN.O_i[PO_i] = 0$ ▷ Set Delay Constraint for PO_i =0
5: Ckt_i = synthesize (RTL_{orig}, $CN.O_i$)
6: D_i = Worst negative slack of Ckt_i
7: **return** D_i
8: **End**

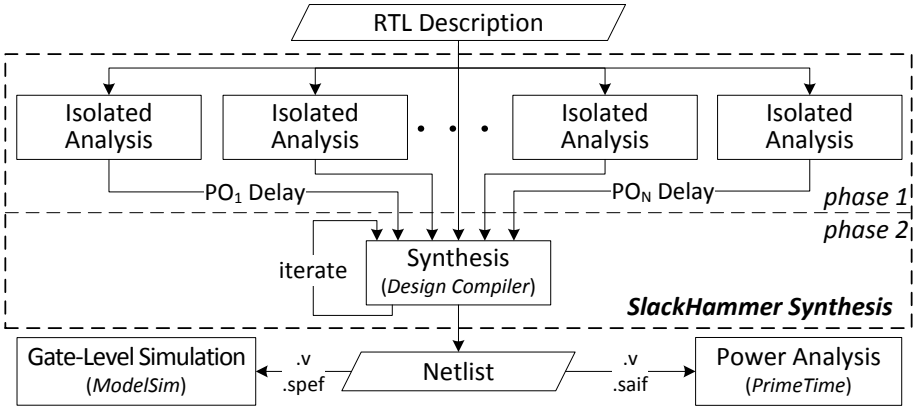


Figure 3.5: Proposed synthesis flow with 2 phases: Isolated analysis to obtain initial constraints and iterative synthesis.

Algorithm 2 presents the pseudo code we used to obtain isolated path delay of a PO as explained in Section 3.1.4. For a given RTL description (RTL_{orig}) and a primary output (PO_i) of RTL_{orig}, it finds an isolated minimum delay value. It sets PO_i with '0' delay constraint while others with an exceedingly large value (lines 3-4). This has 2 effects: (i) it limits paths that can produce a negative slack to PO_i and focuses the delay optimizations on them. (ii) it diminishes load that may occur on cells that are shared in computation of other POs.

Thus, SlackHammer effectively implements the topology based optimization approach explained in Section 3.1.3 by reformulating the graceful error degradation problem and using traditional logic synthesis operations. Figure 3.5 depicts the design flow of the proposed methodology. We utilize an off-the-shelf synthesis tool for both analysis and synthesis of a circuit. This way we inherit the tools capabilities and leverage the entire scope of optimizations in reducing the number of near-critical paths for any underlying technology library. Our methodology is limited with physical boundaries of the technology library for any given RTL description, which makes it agnostic to output quality constraints.

Heuristics on Parameter Selection

We define the Algorithm 1 parameters α and β empirically following some heuristics. The proposed methodology enables delay optimizations on the non-critical paths. Maximum delay in both SlackHammer and in the traditional synthesis is defined by the original most critical path. This path is already rigorously optimized yet remained critical in both cases. Therefore it is not feasible to achieve a faster circuit in terms of maximum delay except possible variations in the synthesis algorithm. Thus, setting the constraining function parameters α and β to match the maximum delay constraint in our methodology and the maximum delay of standard synthesis is a good practice.

To that end, we set the parameter values conservatively; initialize our delay constraint estimates slightly smaller than the maximum delay constraint of standard synthesis. Afterwards, through iterations, we let the delay constraints converge into values that result in successful synthesis (line 13 of Algorithm 1). This way, during the iterations, almost all POs become critical and examined for delay optimization possibilities.

Design Time Overhead

Proposed methodology incurs computational expenses in design time, meaning offline and one time. Therefore, we are not primarily concerned with the design time overheads. Also these expenses have a high degree of parallelism in *phase 1*: isolated path analysis for each primary output, and also in synthesis of different circuits. As a result, the design time primarily depends on *phase 2*: the number of iterations to find tightest constraints and achieve a successful synthesis. This overhead is related to number of primary outputs and appropriate selection of parameters α, β, δ in Algorithm 1.

For instance, when done sequentially, *phase 1* of a 16-bit multiplier required 33.7x in design time, relative to synthesis time of a minimum delay circuit (D_{orig}). Following the same example for *phase 2*, setting the constraining function (parameters α, β) to meet 98% of D_{orig} (max($CN.O_{iso}$) = 0.98*D_{orig}) and setting a small δ: 0.2% of D_{orig} required an additional 104.6x in design time, totaling 117.3x or 58 minutes. To reduce the design time of *phase 2*, we can reduce the distance between the constraining function and D_{orig} (e.g. from 98% to 99%) or increase δ. Doubling δ parameter reduced the time of *phase 2* to 13.3x with a small drop in quality.

Proposed algorithm iterates on primary outputs. The number of primary outputs affects the design time of *phase 2* as it changes the number of iterations. For instance, a 64-bit Kogge-Stone adder with constraining function set to meet 96% of D_{orig}, a relatively large distance, with a δ: 0.8% D_{orig} required 623x ($\sim$13.5 hours) in design time. These values are subject to variation as a consequence of heuristics in the search of synthesis.

3.3 Experimental Methodology

We evaluated the proposed methodology in Section 3.2 on a range of popular arithmetic circuits given in Table 5.1.

Table 3.1: Circuits used in experiments

Name	Function	Bitwidth	I/O
CLA32	Carry Look-ahead Adder	32	64/33
CLA 64		64	128/65
KSA32	Kogge-Stone Adder	32	64/33
KSA64		64	128/65
MUL16	Multiplier	16	32/32
MAC	Multiply and Accumulate	8	48/33
SAD	Sum of Absolute Differences	8	48/33
EU-DIST	Euclidian Distance (without square-root)	8	16/16

All syntheses are done with Synopsys Design Compiler using ultra high effort (*compile_ultra* option) and mapped to TSMC 65nm Generic-Plus library in typical corner. In our comparisons, we used '0' delay constraint (*zero_delay*). We used the worst negative slack of this synthesis as a delay constraint to synthesize with minimum delay (*min_delay*). In all experiments we set the Algorithm 1 parameter α and β conservatively, to meet a slightly lower value than the circuit delay, as previously explained in Section 3.2. For adders, MAC and SAD we set $\alpha = 1.1$. For MUL16 and EU-DIST this α value created delay constraints higher than *zero_delay*. To constraint path delays conservatively, we set $\alpha = 1.04$ for MUL16 and $\alpha = 1$ for EU-DIST. We show the delay distribution using the proposed methodology, performing non-critical path optimizations (*Proposed*), as well as the isolated delay values (*iso*) obtained in Algorithm 2. Gate-level simulations for error characteristics are done with

Mentor Graphics ModelSim. We use error rate and mean relative error as our metrics to evaluate error. Error rate is the ratio of outputs with one or more incorrect bits. Relative error, as given in Eq.3.2 is the distance between the correct and the approximate value divided by the value of correct output.

$$E_{rel} = \frac{|O_{correct} - O_{approx}|}{O_{correct}} \tag{3.2}$$

Synopsys PrimeTime is used for power estimations. We obtained the power values for both circuits using the same input stream with uniform distribution and at the same clock period where they work correctly. To compare with the state-of-the-art we have implemented cell resizing for slack redistribution from [61] in PrimeTime. We applied *OptimizePaths* procedure for all paths until no further speedup is possible, without power and error bounds and avoided *power aware post-processing* in order to maximize performance benefits.

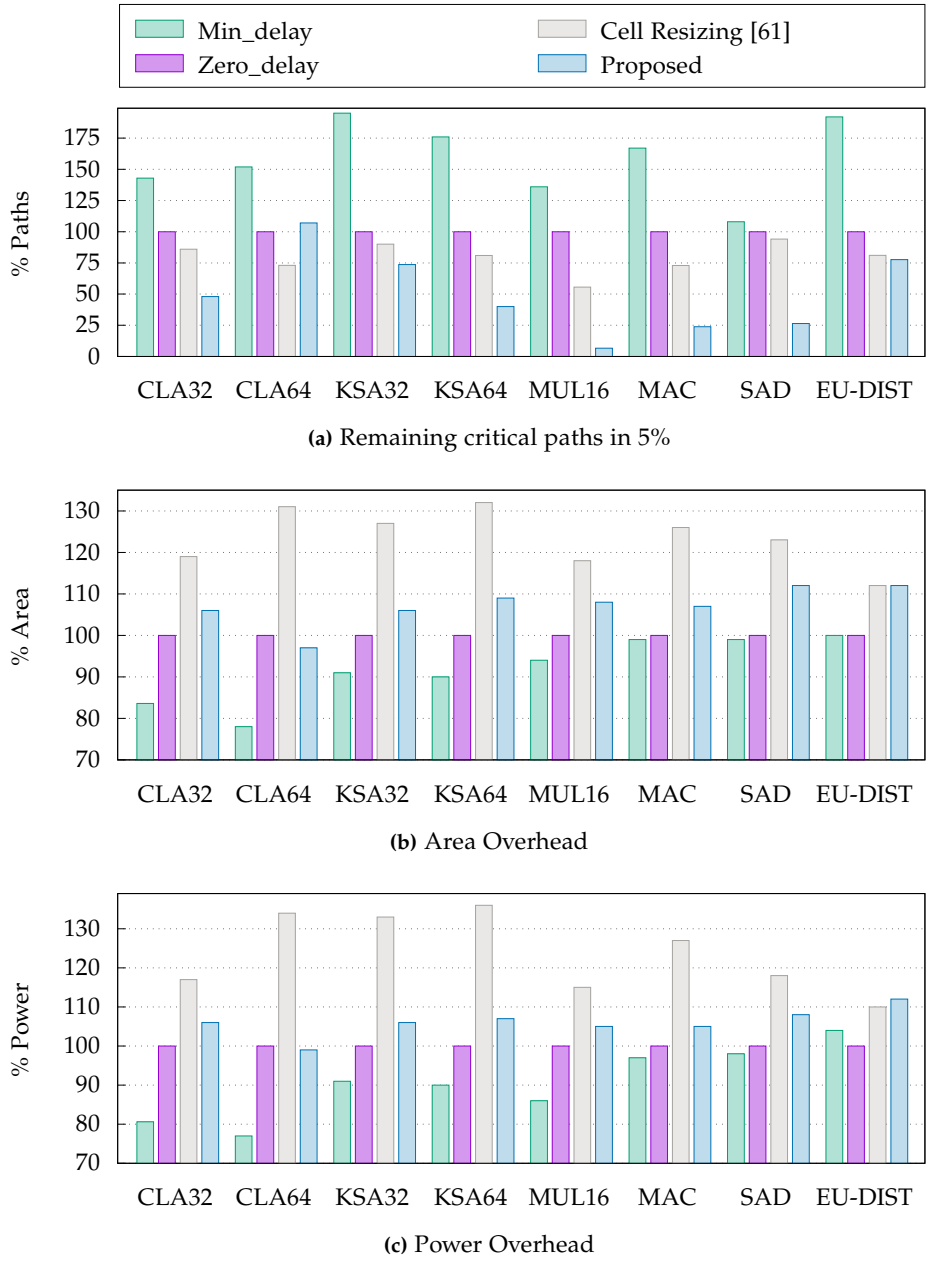
3.4 Results

We first present the results on near-critical paths and the overheads. Among the traditional methods *zero_delay* produces circuits with highest amount of path delay optimizations. Therefore we set *zero_delay* as baseline and present all values relative to it. Figure 3.6a shows the remaining near-critical paths in percentage. We use the following definition for a path: A unique set of consecutive logic from a primary input to a primary output. We counted the paths that are within 5% of the *zero_delay* circuit delay as near-critical. The number of near-critical paths are around 10^6 to 10^7 range.

Min_delay results show that area recovery phase of synthesis increases the number of near-critical paths consistently and by up to 1.9x at the circuit level. In contrast, proposed method has generally reduced the number of near-critical paths beyond *zero_delay*, and further than the cell resizing method. In the case of MUL16, the number of near-critical paths has decreased by 93%. In the case of CLA64, synthesis with our methodology using *iso* based delay constraints ($CN.O_{iso}$) could not meet the circuit delay of the traditional synthesis. This also reflected to area and power overheads as negative values. Similarly, in *DesignWare* adder our algorithm did not find a favorable optimization. We attribute this to a heavily delay balanced topology. Delay reduction on a path increases the worst case delay through incurring loads on the most critical path.

Figure 3.6b and 3.6c presents area and power overheads of the synthesis methods in comparison. Comparing *zero_delay* and *min_delay*, the area recovery phase results in generally more significant area and power improvements in parallel-prefix adders (CLA, KSA) when compared to multiplier circuits (MUL, MAC, EU-DIST). This can be attributed to more sequential topology of multipliers where LSBs are used as intermediate values in computation of MSBs and cannot be relaxed to recover area to the extent of parallel-prefix adders. The overheads of SlackHammer strongly depend on how close the delay values became to the isolated minimums (*iso*). Such reductions initially have low area cost, but this cost increases exponentially as paths require more of the logic to be sped-up. Circuit topology and selection of Algorithm 1 parameters effect these overhead values. Such overheads are an expected outcome when more paths receive delay optimizations. Therefore, they confirm that proposed methodology succeeds in optimization of more paths for delay.

Relatively small overheads with reductions in near-critical paths indicate that low cost optimizations are found during the synthesis with our methodology. This can be observed in comparisons with cell resizing method. Using the full scope of optimizations in synthesis, proposed methodology finds better area and power vs. delay trade-offs. Applying cell resizing to all paths have a considerably higher overhead than selectively applying to some paths and relaxing others as reported in [61].



(a) Remaining critical paths in 5%

(b) Area Overhead

(c) Power Overhead

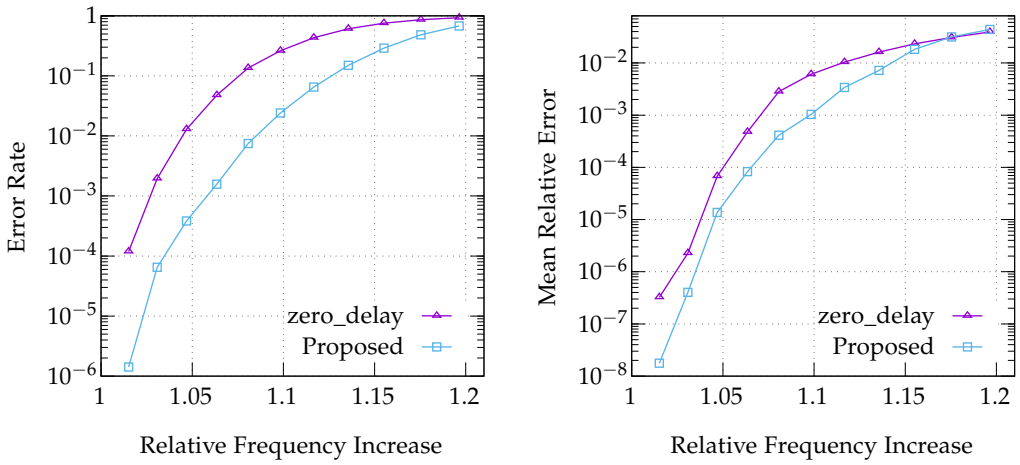
Figure 3.6: Comparison of synthesis methods normalized to synthesis with 't_{ckt} = 0ns' delay constraint, *zero_delay*. (a) Percentage of near-critical paths (b) Area overhead (c) Power overhead.

3.4.1 Accuracy-Frequency Trade-Off

Figure 3.7 demonstrates the improvement in error resiliency under timing speculations. In these comparisons we used circuits that start producing error at the very same frequency to eliminate variation effects of synthesis heuristics.

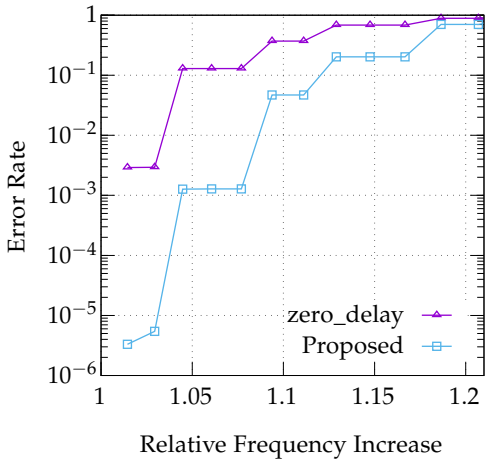
In Figure 3.7a and 3.7c, reductions in error rate reach over 2 orders of magnitude. The improvement margin in mean error in figure 3.7b and 3.7d is considerably less than the error rate margins. This is because by their topology, MSBs in arithmetic circuits require long paths. Although our methodology reduces the error rate in general, the error rate on the long paths remains the same which leads to less reduction on significant errors. The improvement in error resiliency is correlated

with the reduction in the number of near-critical paths shown in Figure 3.6a. The other factors are significance of the failing bits and activation probability of the failing paths. In some synthesis results, delay reductions were only possible on LSBs which lead to improvements in error rate but worsened the mean error. This can be attributed to an increase in significant errors and also glitches. Proposed methodology reduces many of the PO delays, it leaves less room for glitch optimizations which could be an interest for further research. Increasing frequency beyond ∼15% shows no difference between the 2 circuits. At this point the paths our methodology optimized also start to fail.

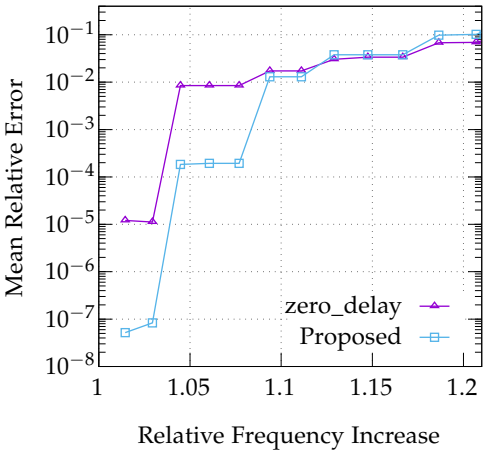


(a) MUL16 Error Rate

(b) MUL16 Error Mean

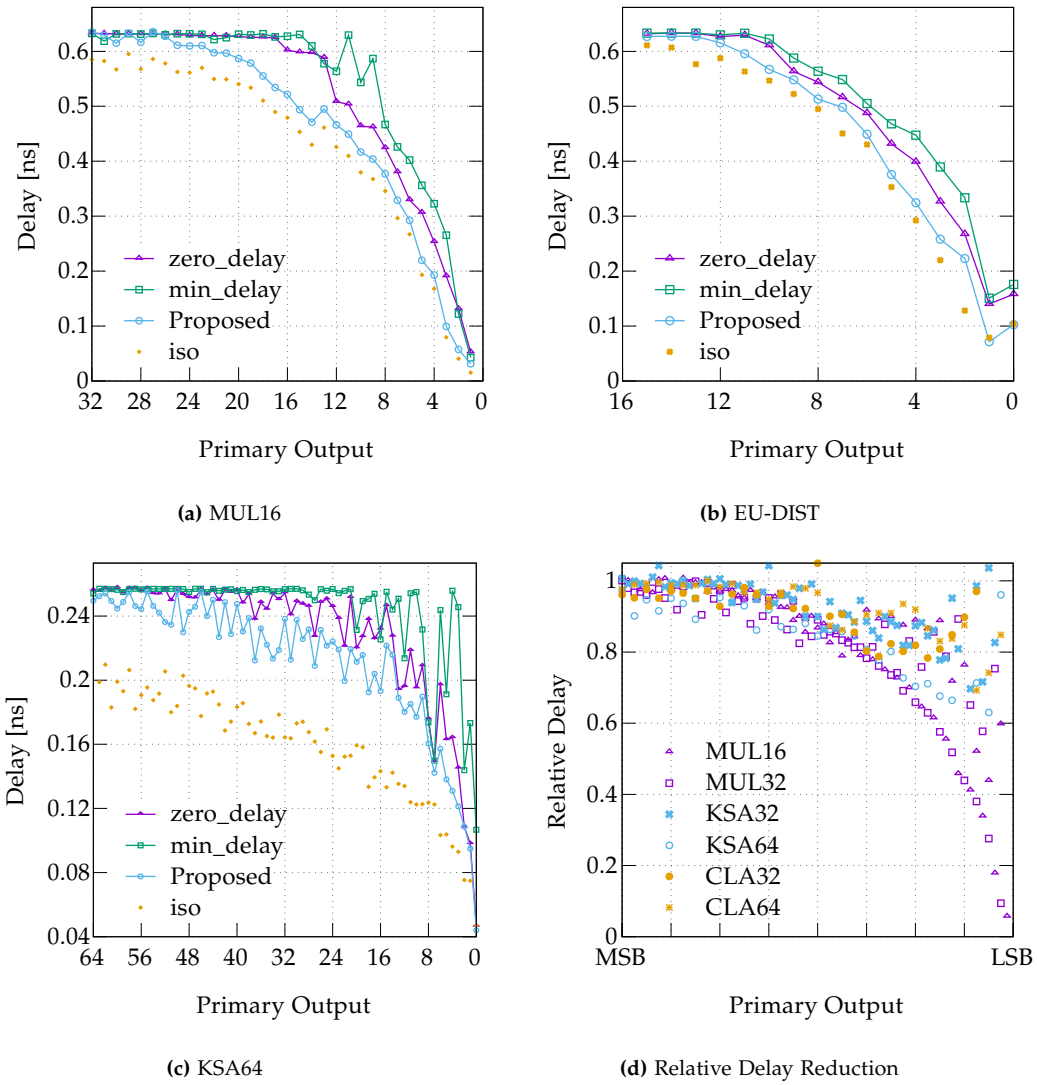


(c) KSA64 Error Rate



(d) KSA64 Error Mean

Figure 3.7: Error characterization of circuits under frequency scaling. (a) 16-bit Multiplier error rate and (b) error mean, (c) 64-bit Kogge-Stone adder error rate, and (d) error mean.



(a) MUL16 **(b)** EU-DIST

(c) KSA64 **(d)** Relative Delay Reduction

Figure 3.8: Circuit delay distributions in comparison. Proposed method reduces delays for the majority of primary outputs, especially on the LSB side.

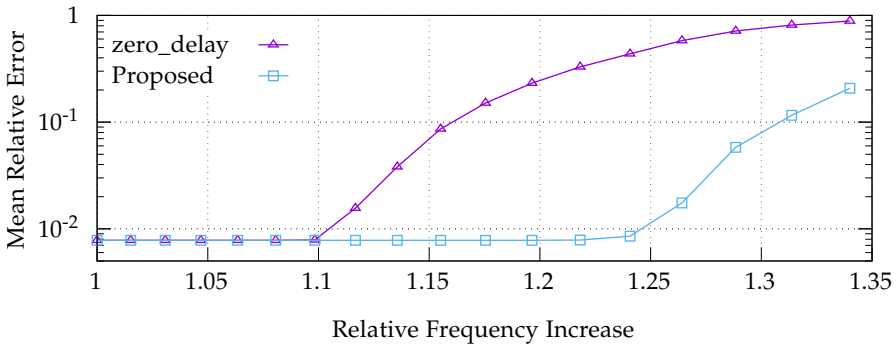
3.4.2 Delay Distribution Comparison

Figure 3.8 presents the primary output delays of *zero_delay* and *min_delay* syntheses (detailed in Section 3.3) in comparison to the proposed methodology. SlackHammer matched the traditional synthesis in worst case delays. Additionally, it has improved most of the PO delays in the experimented circuits as a result of non-critical path optimizations. Carry chains that exists in many arithmetic circuits confine the biggest improvements to LSBs. Minimum logic depth required to produce the LSBs are usually smaller than the MSBs. Our methodology results in more significant delay reductions on them. For brevity, we just explain on the basis of these 3 circuits.

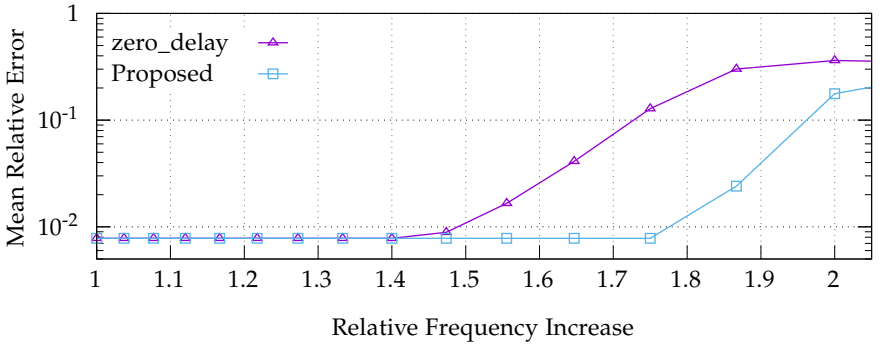
We present delay improvements on POs relative to *zero_delay* in Figure 3.8d to show the margin for further optimizations for various circuits. For most POs, delay can be reduced further by 10 to 20% with the proposed methodology in our experiments.

3.4.3 Cross-Layer Effectiveness

As described in Section 3.4.2, our approach allows reducing LSBs delays significantly. Timing speculations are commonly used together with precision scaling in a cross layer fashion [23, 27, 120]. In these examples, and when approximations are applied end-to-end in general, precision scaling utilizes LSB part of the circuits. We demonstrate its effectiveness in cross-layer fashion when used together with precision scaling in architectures such as QUORA [120]. We consider an initial mean relative error of 2^{-7} due to the quantization of inputs to 8 bit with precision scaling. Also we assume only the expected part of sum is considered as otherwise glitches cause errors in higher MSBs which results in significant errors under timing approximations. Figure 3.9a shows that our methodology enables up to 14% speedup for a 16-bit multiplier, Figure 3.9b shows up to 27% speedup for a 64-bit Kogge-Stone Adder at the same error mean. These speedup values increase further with lower precision.



(a) MUL16 with 8-bit precision scaling



(b) KSA64 with 8-bit precision scaling

Figure 3.9: Cross-layer effectiveness of SlackHammer with precision scaling.

3.5 Summary

Allowing approximations in computation via relaxing circuit level error resilience can lead to performance improvements. This chapter introduced SlackHammer, a systematic methodology to automatically synthesize circuits with enhanced timing-error resilience under aggressive frequency scaling. This methodology leverages any underlying synthesis tool and identifies primary outputs with remaining slack margin given any arbitrary circuit. The key idea behind is to optimize non-critical paths for delay to reduce the probability of timing errors. Our experiments demonstrate that SlackHammer methodology is promising in extending the design space to higher frequencies

and finding favorable trade-offs between performance and accuracy for both standalone and cross-layer techniques that utilize timing speculations.

4 Synthesis of Frequency-Precision Scalable Circuits

Motivation

This chapter applies delay optimizations of logic synthesis to minimize energy when the circuit is used for variable accuracy. The computation accuracy can be changed in hardware by dynamically scaling the quantization, i.e., switching to lower-precision operands when possible, which reduces the transistor toggles, and therefore the dynamic power consumption. At circuit level, LSB gating has been used for dynamic precision scaling. Alternatively, a growing number of applications [29, 128], architectures [120, 121], and in general end-to-end use of approximations employ dynamic precision scaling that exercises LSB paths of circuits while turning the MSBs off. The latter can further exploit smaller complexity of LSB logic vs. MSB when running in reduced precision mode: Traditional synthesis tools aim at meeting a single clock delay for all paths and leverage the imbalances in logic depths and complexity, e.g., by using slower and more energy-efficient gates, when possible [8]. However, many of the LSB paths provide intermediate signals, i.e., they are shared and hence constrained by MSB. Their slack cannot be fully leveraged for energy savings. In addition, LSB logic is small; thus, gains are very limited.

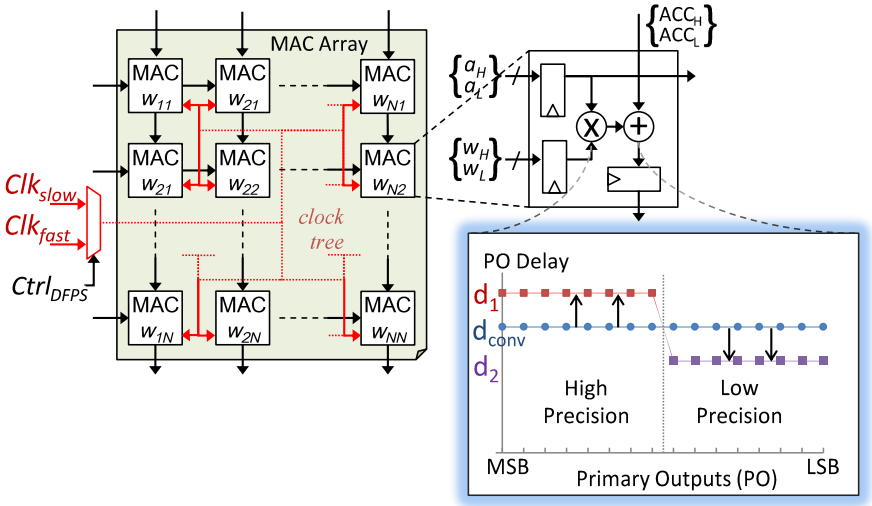


Figure 4.1: Dynamic frequency-precision scalable MAC unit in a systolic array. Unlike conventional, our circuits can utilize different clocks for high and low precision inputs to improve energy-efficiency.

This chapter proposes a novel circuit synthesis methodology that alters path delays for energy optimization under dynamic precision scaling. Given any circuit in RTL description, we explore the energy-optimum delay values for paths used by different precision levels while matching the initial average throughput. Our approach obtains net energy gains by relaxing the MSB output delay that constrains the majority of paths, at the cost of reducing LSB output delays. As such, we synthesize the whole circuit to be more energy efficient but slower, while low-precision mode can run at a faster frequency to compensate and meet average throughput. In Figure 4.1, we illustrate our goal of finding the energy optimum delays, *d1* and *d2*, for fused Multiply and Accumulate (MAC), the dominating arithmetic operation in DNNs [118]. The inputs, activation (a), weight (w), and accumulate (ACC), can be high or low precision. We formulate the search for the ideal delays as a function of the circuit topology in which the logic depth and delay of primary outputs differ and the application features, i.e., required precisions and their utilization. These features are general and allow tailoring our method for any application. Our systematic methodology automates a gradient descent-based design space exploration and logic synthesis using commercial tools. It is especially effective for high-performance implementations of imbalanced circuit topologies, in which the circuit-level delay optimizations are maximized. In summary, the work in this chapter makes the following key contributions:

- We propose a novel circuit synthesis methodology that alters the path delays for energy minimization under dynamic frequency-precision scaling. It is compatible with standard EDA flow and general in terms of the application, circuit, and target precisions.
- Circuits synthesized with our methodology have the property of dynamic frequency-precision scalability. They can run at high precision with a slower clock or low precision with a faster clock, at the same voltage, while ensuring timing-error-free operation.

In our evaluations of a wide range of variable-precision usage, we reduce the MAC dynamic energy consumption by up to 34% while also reducing the area by up to 32% and leakage by up to 43%.

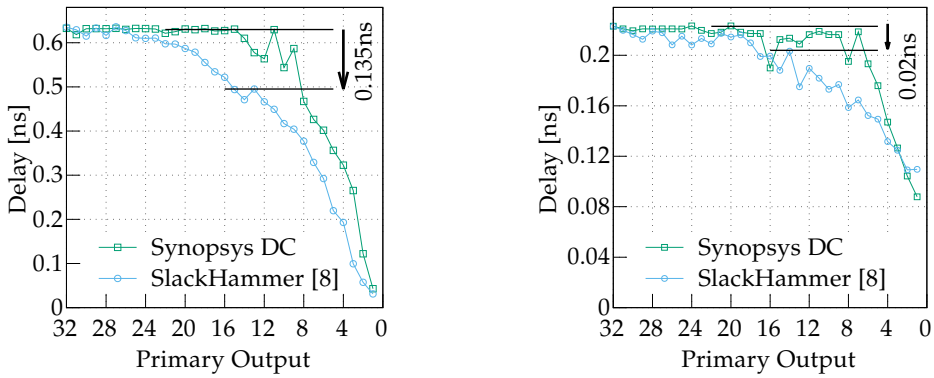
4.1 Logic Synthesis with Multiple Delay Constraints

This section describes the conventional and state-of-the-art circuit delay optimizations and our novel approach in applying them for energy optimization.

4.1.1 Delay Variations in Circuit Topology

Circuit topologies typically have an uneven depth. Conventionally, they are synthesized such that all paths are aimed at matching a given delay constraint. As detailed in Section 2.4, the synthesis tool first selects a circuit topology, and then iteratively applies gate-level delay optimizations to the current most critical path until the slack is met. Afterward, area optimizations undo gate-level optimizations on non-critical paths for area and energy savings. Hence, the conventional synthesis results an even delay distribution between the primary outputs.

Alternatively, SlackHammer, introduced in Chapter 3, finds ideal delay constraints for each primary output, which results in applying delay optimizations to a maximum number of paths [8]. In Figure 4.2, we compare the circuits generated with SlackHammer against the minimum possible delay circuits generated with a standard commercial tool, Synopsys Design Compiler. The delay reductions are limited to LSB side input to output and varying according to circuit topologies: in Figure 4.2a the delay can be reduced by 21% for a multiplier and in Figure 4.2b, it can be reduced by



(a) 16-bit DesignWare Multiplier

(b) 32-bit Carry Look-Ahead Adder

Figure 4.2: Primary output delays of (a) a multiplier and (b) an adder with fastest conventional synthesis against the state of the art [8].

10% for an adder. Adders, adder-based units (e.g., Sobel and Gaussian Filters), multipliers, and units based on adders and multipliers (e.g., MAC, Euclidian Distance, DCT, IDCT,) all inherently have a varying degree of path imbalance in their circuit topology where LSBs can be faster than MSBs [8].

4.1.2 Exploiting Delay Variations for Energy Gains

In this chapter, we exploit the delay variations in circuit topology for energy optimization. For the same logic, energy is inversely related to circuit delay: Reduction of the delay constraint leads to parallel computation of intermediate signals by a change in circuit topology or gate-level optimizations (i.e., load isolation, load splitting, gate upsizing), which increases the number or size of toggling gates and hence increases the energy. Traditional tools achieve energy gains by relaxing LSBs. As illustrated in Figure 4.1, we instead increase the delay constraint of MSBs and reduce it for LSBs while meeting the initial average throughput target.

An energy reduction that materializes with our approach can be explained in two terms: First, the MSBs typically make the majority portion of the logic and hence benefit from equal increase more than LSBs. Depending on the topology, the complexity increases beyond linear with bit-width. We can consider the Ripple-Carry adder a corner case where the logic size and depth increase linearly with bit-width. Faster parallel prefix adderss (e.g., Kogge-Stone, Brent-Kung, etc.) have a logic size and depth $O(n \log n)$ or higher, and multipliers are even quadratic. As such, when divided for the same number of bits, the LSB portion of the logic is much smaller than the MSB.

Second, LSBs have inherently shorter paths, and they are shared, thus primarily constrained by MSB delays. Consequently, relaxing LSB POs only apply to the unshared part and can gain less, if any. By contrast, relaxing MSBs also increases the delay of shared logic while still being able to meet tighter LSB output constraints. In Figure 4.3, we present an example with faster LSB output. An intermediate signal is necessary for the MSB output. MSB paths are typically slower than LSB paths as they require intermediate signals, such as carry-in in adders. The paths *(2-3)* and *(1-3)* form the critical paths. With a standard gate-level delay optimization method such as gate up-sizing, we can reduce the circuit delay by first speeding-up gate *(3)*. Then, we have to speed-up both gates *(1)* and *(2)*. Tracing it back, the energy and area cost of speed-ups shows an exponential-like increase. Conversely, relaxing the MSB delay shows reverse-exponential savings in energy and area, surpassing the costs of reducing LSB delays.

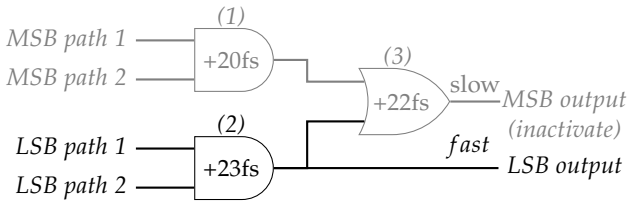


Figure 4.3: MSB output requires an intermediate signal and hence it is slower. Gate (2) is on the critical path and cannot gain from any relaxations. By contrast, relaxing the MSB constraint applies to all gates, while still being able to meet tighter LSB constraints.

In our approach, we group the primary outputs for the given quantization requirements. Within each group, we individually assign the same delay constraint. Our approach differs from the conventional synthesis where a single circuit delay is assigned to all primary outputs. Operating our circuits at different precisions results in different energy, but also different performance requiring different frequency. The energy-optimal delay values are heavily topology-dependent. Additionally, each quantization level may be used to a different degree. We search for the ideal delay values that minimize the energy while matching the average throughput in Section 4.2.

4.1.3 Dynamic Frequency-Precision Scaling System

A dynamic frequency-quantization scaling system, as we define it in the scope of this work, is able to tune the operating clock frequency upon a change in the precision of data it operates on. We pair each given quantization level with the suitable clock frequency. At system level, we impose input-aware clock frequency adjustments: With a change in input quantization, we change the clock at its source. Hence, the system can run at a high frequency when the clock domain is used for low-precision operation and vice versa. The circuits designed with our approach do not need any internal control logic. Our approach ensures timing-error-free operation with correct clock frequency and quantization pair, without changing the voltage.

At a large scale, systems that incorporate proposed circuits still need a single clock tree for the clock domain. A clock domain runs independently and communicates through an asynchronous FIFO interface. In order to benefit from frequency scaling, we make the assumption that multiple clocks are available for our use. This is a standard function in modern processors and can be done at runtime dynamically and immediately, within a cycle. We can switch between them using clock multiplexers at the source of clock tree s.t. the entire clock domain runs at the same frequency.

4.2 Design Methodology

In this section, we mathematically derive energy-optimal delay values for each precision group as a function of delay and the number of operations at this precision. We isolate the search parameter and find the energy-optimal delay values with a gradient descent algorithm.

4.2.1 Energy Optimization with a Throughput Target

The total energy E_{all} can be defined as a product of number of operations and energy per operation E_{op}:

$$E_{all} = N_{all}.E_{op}(d_{orig}, in) \tag{4.1}$$

where N_{all} is the total number of operations, d_{orig} is the original circuit delay per operation. Similarly, per function unit, throughput is the inverse of circuit delay d_{orig}. It can also be written as number of computations over the computation time:

$$Throughput_{avg} = \frac{N_{all}}{N_{all}.d_{orig}} \tag{4.2}$$

We address a problem, in which multiple different precisions are required (N_{P1}, N_{P2},... N_{Pn} or N_i as a general term).

$$\begin{aligned} N_{all} &= N_{P1} + N_{P2}... + N_{Pn} \\ &= \sum_{i=1}^{n} N_i \end{aligned} \tag{4.3}$$

To maintain the same throughput, we need to maintain the same average computation time (CT):

$$CT = N_{all}.d_{orig} = \sum_{i=1}^{n} N_i.d_i \tag{4.4}$$

The energy consumption of proposed topology-driven circuits can be written using Equation (4.1), in terms of the number of operations at a particular precision N_i and the energy consumption per operation E_i, which itself is a function of circuit delay d_i and the inputs:

$$E_{topo} = \sum_{i=1}^{n} N_i.E_i(d_i, in) \tag{4.5}$$

where E_i is ultimately determined by and obtained from the synthesis tools. We aim at minimizing the energy consumption given in Equation (4.5), while maintaining the throughput given in Equation (4.4), and formulate our goal as follows:

$$\begin{aligned} \min_{N,d,in} \quad & \sum_{i=1}^{n} N_i.E_i(d_i, in) \\ \text{s.t.} \quad & \sum_{i=1}^{n} N_i.d_i = N_{all}.d_{orig} \end{aligned} \tag{4.6}$$

4.2.2 Leverage & Distance

In this work, we limit the problem to using 2 different precisions: *low* and *high*. Starting from the original circuit delay d_{orig}, we search for ideal path delays used by each precision that minimizes Equation (4.6) while matching the initial average throughput target. We solve Equation (4.4) as follows:

$$\begin{aligned} CT &= N_{low}.d_{orig} + N_{high}.d_{orig} \\ \frac{CT}{N_{low}} &= d_{orig} + \frac{N_{low}}{N_{high}}.d_{orig} \\ &= (d_{orig} - \Delta d) + \frac{N_{high}}{N_{low}}.(d_{orig} + \frac{N_{low}}{N_{high}}.\Delta d) \end{aligned} \tag{4.7}$$

We name the term Δd as *distance*. Distance is our search parameter s.t. for the ideal Δd, the energy is minimized in Equation (4.6). Hence, the delays of the circuit at precision 1 (d_{low}) and precision 2 (d_{high}) can be written as follows:

$$d_{low} = (d_{orig} - \Delta d), \quad d_{high} = (d_{orig} + \frac{N_{low}}{N_{high}}.\Delta d) \tag{4.8}$$

Thus, we reduce our search to a single parameter, Δd. Note that d_{high} and d_{low} are changed unevenly from the original delay d_{orig}, with a multiplier N_{low}/N_{high}, i.e., the number of operations in precision 1 over precision 2. We call this term $\mathcal{L}$ for *leverage*. A high leverage means we can significantly increase the delay constraint of high precision part while only slightly reducing the delay constraint of low precision part.

From the hardware perspective, synthesis with a high leverage means a significant increase in delay margins result in more relaxed, energy-efficient circuit paths. However, it also means a high ratio of low precision to high precision operations, i.e., rare usage of highly efficient circuit paths. The opposite holds for a low leverage value, i.e., frequent but small energy savings on high precision paths. Typically, energy consumption of high precision paths dominates the low precision paths. However, it is not clear whether small frequent energy savings surpass large and rare savings. Next, we address this question with a design space exploration.

4.2.3 Design Space Exploration

Given a circuit with dual precision usage, we propose an automated search to find the distance from the original design delay that minimizes the energy consumption as previously derived in Equation (4.6).

Algorithm 4 implements a gradient descent algorithm that iteratively approaches energy-optimal delay values for low precision and high precision operations. We take any given RTL description of the circuit as input. Behavioral RTL description gives freedom in topology choice (e.g., carry look-ahead adder vs ripple carry adder). The synthesis tool automatically picks the best suiting topology from its library. This leads to a greater range of optimizations and applies to a greater range of delay values. It is also possible to give a highly structured RTL description or even a netlist to enforce a higher control in the design flow.

Our algorithm updates the gradient descent with an adaptive step size ε. We initialize the design space exploration between lines 1 to 3. In line 5, we update the delay constraints of the design according to Equation (4.8) by changing the distance asymmetrically, with a leverage $\mathcal{L}$. We set these output delay alterations relative to the original clock delay constraint, using *set_output_delay* command in Synopsys Design Compiler. This command is common in commercial synthesis tools and it is used for back-end optimization by the industry. Thus, we satisfy Equation (4.4) and maintain the throughput target. In line 6, we assign d_{low} to the LSB primary outputs that are in the low quantization group and d_{high} to the rest of the primary outputs and synthesize the circuit.

Algorithm 3 Throughput constrained synthesis for energy minimization under dual frequency & precision use

Input: Circuit RTL: RTL_{in} , delay constraint: d_{orig}, Number of operations in low and high precisions: $< N_{\text{low}}, N_{\text{high}} >$, representative input: In_{rep}
Output: Frequency-Precision Scalable Circuit: Ckt_{DFPS}

1: Ckt_{init} = synthesize(RTL_{in}, d_{orig})
2: E_{init} = get_power(Ckt_{init}, in_{rep}) $\times$ d_{orig}
3: n=0, $d_{\text{low}} = d_{\text{high}} = d_{\text{orig}}$, $\mathcal{L} = (N_{\text{low}}/N_{\text{high}})$ ▷ initialize
4: **while** $|\varepsilon| \geq Threshold$ **do**
5: n=n+1, $\text{d}_{\text{low}} = \text{d}_{\text{low}} - \varepsilon$, $\text{d}_{\text{high}} = \text{d}_{\text{high}} + (\mathcal{L} \times \varepsilon)$
6: Ckt_{DFPS_n} = synthesize (RTL_{in}, $<\text{d}_{\text{low}}, \text{d}_{\text{high}}>$)
7: E_{n} = get_power($\text{Ckt}_{\text{DFPS}_n}$, in_{low}) $\times$ d_{low} $\times$ N_{low} + get_power($\text{Ckt}_{\text{DFPS}_n}$, in_{high}) $\times$ d_{high} $\times$ N_{high}
8: **if** $\text{E}_{\text{n}} > \text{E}_{\text{n-1}} \vee \text{WNS} < 0$ **then**
9: $\varepsilon = -\varepsilon/2$
10: **end if**
11: **end while**
12: $\Delta d = d_{\text{orig}} - d_{\text{low}}$
13: **return** $Ckt_{DFPS_{n-1}}$, Δd

In line 7, we compute the total energy consumption weighted to the number of operations in this quantization level. This calculation follows the formula given in Equation (4.5): Energy per operation can be obtained by multiplying the average power and delay. We compute the energy per quantization separately, by querying the synthesis tool for power and using inputs with corresponding precision, and weigh them according to the number of operations in this precision. In line 8-10, we update the step size parameter, ε, if the synthesis cannot successfully meet the delay constraint and create a worst negative slack or if the energy increases. Unusually, if the MSB delays can be less than the LSB delays, with a reversal of ε in line 9, our algorithm also searches in the other direction. The commercial synthesis tool we use, Synopsys DC, performs as a black box to us. The results of its delay optimizations are stochastic to a degree in which fine adjustments do not necessarily improve the output. Hence, we set a threshold for minimum step size empirically and share the values we used in Section 4.3. Finally, we return the last successfully synthesized circuit in netlist form as the output and Δd for clock generation.

Thus, for a given number of high and low precision operations, Algorithm 4 does a gradient descent based search to find delay values that minimize the energy that satisfies Equation (4.6).

Table 4.1: Circuits used in experiments.

Name	P_{High} I/O	P_{Low} I/O	delay [ns]	Area [μm^2]	Architecture
MUL Multiplier	8/16	4/8	0.62	1256.8	CSA Carry Save Array
ADD Adder	24/24	12/12	0.21	810.72	CLA Carry Look-Ahead
MAC Multiply and Accumulate	8/24	4/12	0.83	2067.5	CSA + CLA

4.3 Experiments

We synthesize the circuits given in Table 4.1 using Synopsys Design Compiler and the commercially proven TSMC 65 nm generic plus technology library. All circuits are from Synopsys DesignWare library. Our method is circuit topology-based and technology library independent. We do not expect a significant change when a different, possibly smaller technology node is used. All circuits are initially synthesized for the smallest possible delay. Afterward, Algorithm 4 individually searches for delay-optimal implementations while matching the same average throughput. Figure 4.2 shows higher path delay imbalance for multipliers than adders. We empirically set the search step size ε and stop granularity threshold as 0.05 ns for the multiplier and 0.01 ns for the adder, respectively. For each search step, we synthesized a new circuit with the current distance, Δd (the difference between the original delay and new delay for the low precision).

Following the values given in [19, 58, 116, 130], we present the results on a MAC unit with 8 bit multiply and 24 bit accumulate at high precision. At low precision, it operates a 4 bit multiply and 12 bit accumulate. These quantization levels can be considered as the representatives. We are, however, not bound with these quantization levels. It is also possible to design for different precisions such as 9 bit high and 3 bit low independently for the multiplier, the adder, or for both.

Leverage is the parameter we use for generalizing our experiments. Under different use cases (applications, parameters, input sets, etc.), the usage ratio of low precision to high precision can vary. We abstract our method from the use cases by covering an extensive range of leverage. All values are normalized to the conventional circuit synthesis with a single delay target, given in Table 4.1. In our nomenclature, this represents $leverage = 0$.

4.3.1 Evaluation of DSE Iterations

We begin our evaluations by presenting the iteratively improved outputs of our design space exploration for the multiplier in Table 4.2 in a brief form. Starting from a conventional design, in which all operations have the same delay, we iteratively increase the delay distance (Δd) between high and low precision operations. Increasing the Δd forms PO delays that fit more closely to the intrinsic topology of the multiplier. Iteration 4 at $\Delta d = 0.20ns$ results in negative slack, i.e., the synthesis does not succeed. We use the circuit from the last successful synthesis, iteration 3. Table 4.2 shows a single case detailing the DSE under $leverage = 1$, the equal utilization of high and low precision operations. Next, we generalize our work with experiments on a range of leverage values.

Table 4.2: Iterative search steps of Algorithm 4 for MUL, *leverage=1*

Iteration	0	1	2	3	4
Δd [ns]	0	0.05	0.10	0.15	0.20
Dyn.Power - P_{High} [mW]	0.903	0.889	0.776	0.728	-
Dyn.Power - P_{Low} [mW]	0.704	0.594	0.501	0.488	-
Leakage Power [nW]	9.12	7.07	5.77	5.29	-
Area [μm^2]	1256.8	984.2	830.2	785.5	-

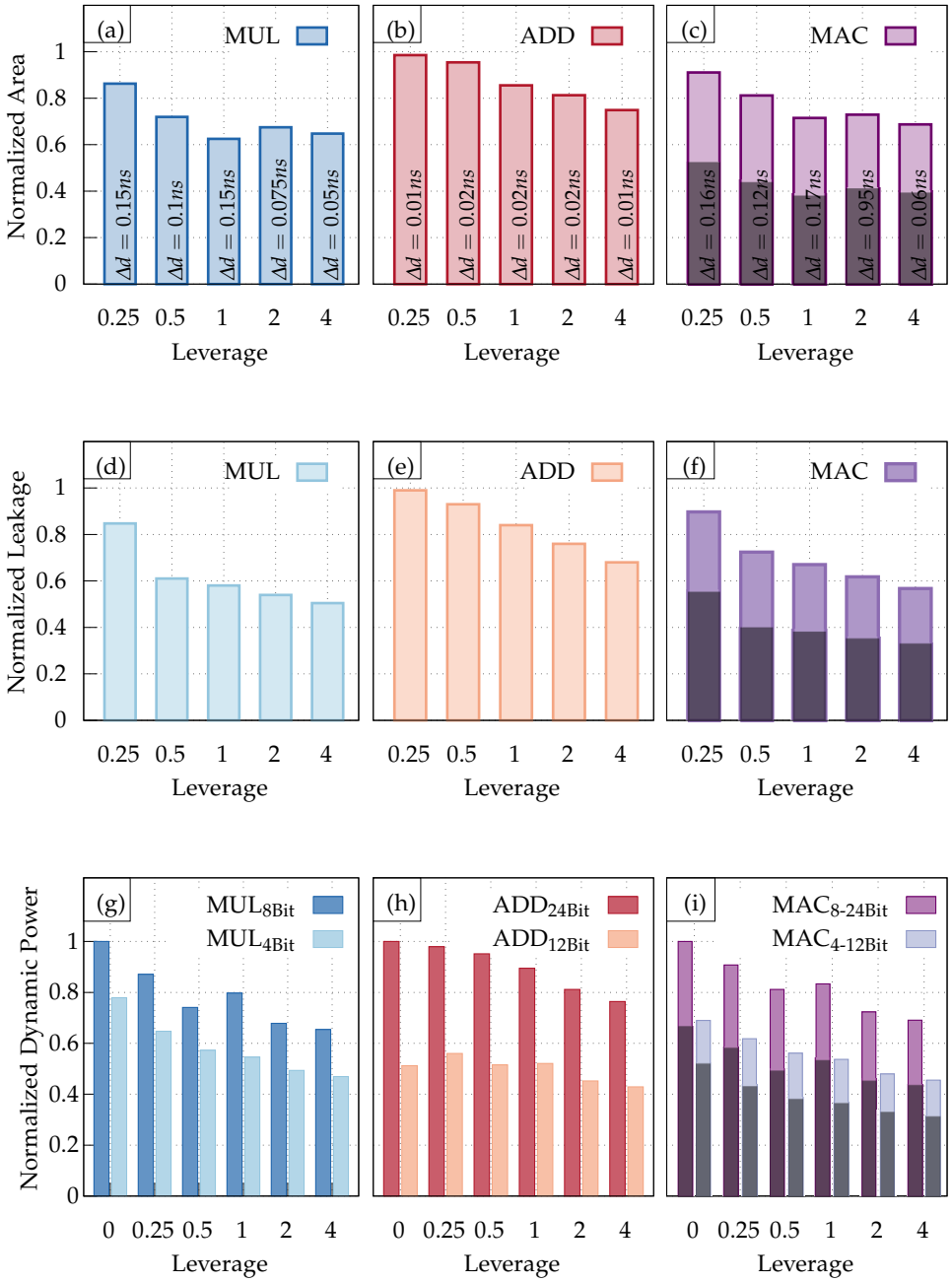


Figure 4.4: Comparisons of proposed frequency-precision scaling methodology against conventional synthesis targeting a single frequency, for circuit given in Table 4.1, in terms of (a,b,c) area, (d,e,f) leakage, and (g,h,i) dynamic power consumption.

4.3.2 Circuit Level Trade-Offs

In Figure 4.4, we share the area, leakage, and dynamic power values for the circuits used in our experiments. We obtained these results individually for each leverage value: Algorithm 4 iteratively optimized the MUL, ADD, and MAC circuits from Table 4.1 for a given leverage. We also label the final Δd values for each result. For the multiplier inputs, we used uniform random inputs. The adder is used as an accumulator in the MAC unit. An input operand of the adder comes from the multiplier. We set this adder input operand toggle frequency equal to the multiplier output

operand. Accordingly, we set the bits 16 to 23 as '0'. For the accumulation operand we used uniform random inputs. We show the dynamic power for high and low precision separately.

With two delay targets instead of one, the proposed methodology better fits the delays of activated paths to the circuit topology than the conventional synthesis. By comparing Figure 4.4a and 4.4b, we can see that the multiplier primary output delay constraints are changed more than the adder, and hence, returned higher Δd values. In other words, the multiplier has intrinsically more significant path delay imbalances in its topology. This path imbalance reflects the results as higher area, leakage, and dynamic power savings with the proposed methodology. The resulting multiplier circuit is 37.5% smaller, has 42% less leakage, and 21% less dynamic power at leverage =1.

In Figures 4.4c, 4.4f and 4.4i, we show the area, leakage, and dynamic power for the MAC unit, respectively. The darker shaded portion shows the contribution of the multiplier unit, which is on average 56% in area and 58% in leakage, and in dynamic power 63% for high precision and 69% for low precision. The remaining values are from the adder. In total, the MAC unit synthesized with the proposed method has up to 32% smaller area, 43% less leakage, and 31% and 34% less dynamic power at high and low precision, respectively, than the conventional synthesis.

Within the range of topological and gate-level delay optimizations, with higher leverage, we generally obtain smaller area, leakage. and dynamic power in Figures 4.4a to 4.4i. At high leverage, we can significantly relax the MSB delay constraints while slightly reducing the LSB delay constraints. An outlier to this situation is the adder circuit when operating at low precision mode. We observe at leverage values 0.25, 0.5, and 1, the 12-bit addition consumes more energy than the initial circuit. This means a part of the logic is aggressively optimized for the delay and became less energy-efficient. However, a larger and/or more frequently used part of the logic became more energy-efficient in return. Our algorithm aims at minimizing the total weighted energy consumption, which we examine next.

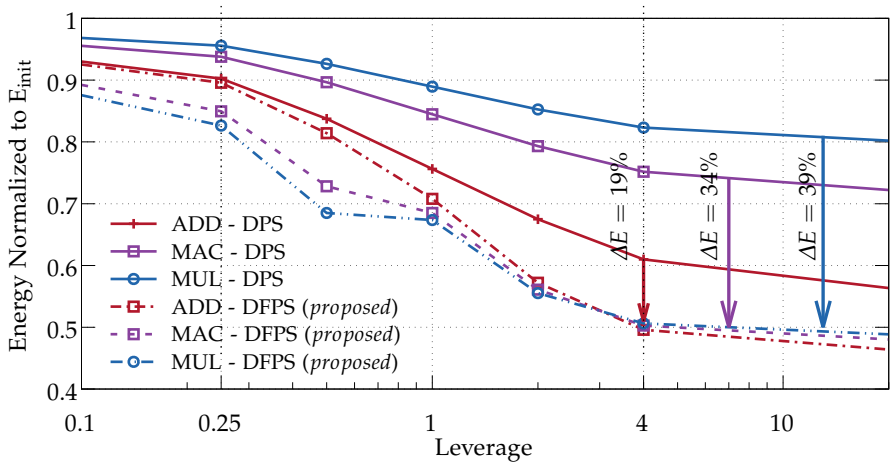


Figure 4.5: Average energy under dynamic precision scaling with conventional synthesis against dynamic frequency precision scaling with the proposed.

4.3.3 Energy vs. Leverage

In Figure 4.5, we show the average energy consumption under dynamic frequency and precision scaling with the proposed methodology (DFPS), compared to state-of-the-art implementations of dynamic precision scaling with conventional synthesis (DPS) [29, 120, 121, 128]. The energy values

are obtained from Figure 4.4, weighted to the utilization ratio of the precision mode (i.e., leverage). Lower quantization, i.e., low precision operations are more efficient. For applications with a high leverage, most operations are in low precision and the overall energy consumption is also lower in general. When compared the state of the art (DPS), proposed method (DFPS) reduces the MAC unit energy consumption by up to 34%.

4.4 Summary

This chapter has proposed automated design of throughput constrained dynamic frequency-precision scalable circuits. As the experiments of Chapter 3 has revealed, many circuits have uneven complexity in their topologies. Especially LSBs of arithmetic circuits have significantly less complexity than their MSBs and they can be synthesized faster. Utilizing this underlying circuit property, the method proposed in this chapter applies the delay optimizations in synthesis described in Section 2.4 for energy minimization under configurable accuracy use. It finds the energy-optimal delay values for two different precisions while matching the average throughput. It is a promising design method for applications that have dynamic changes in precision requirements, such as DNNs.

5 Architecture and Circuit Co-Synthesis

Motivation [1]

Majority of the current approximate hardware designs and design methodologies target a single and static accuracy, as previously detailed in Section 2.2. Existing accuracy-configurable hardware proposals [25, 27, 51, 56, 120] and design methodologies [57, 66, 93] primarily utilize circuit-layer *gating* mechanisms: They disable a configurable portion of the paths by not propagating data or inserting control circuitry into the exact hardware with a small area overhead. Notwithstanding their potency, such approaches only benefit from reduced toggling activity. Because they do not structurally simplify the circuit, e.g. shorten the critical path, they cannot exploit the full extent of power savings that static approximate hardware can achieve with synthesis relaxations.

A potential solution can be instantiating multiple static accuracy circuits in the architecture and switching between them despite their area and leakage power costs [16, 119]. In Figure 5.1, we compare gating an exact hardware, similar to existing work [57, 66, 93], against static accuracy approximate hardware by means of precision scaling their inputs, i.e., discarding a number of LSBs for Sobel Filter and Euclidian Distance computation hardware blocks. Additional dynamic power savings reach up to 46% when leakage is neglected. Notably, the additional area cost of instantiating a circuit is reduced significantly as the accuracy is reduced. For instance in Figure 5.1b at 90% accuracy, the circuit requires only $0.19\times$ the area of the exact circuit. This example demonstrates that instantiating and connecting additional approximate circuits can be a lower-power and higher-area overhead alternative to gating. However, instantiating an additional circuit incurs leakage cost even when this circuit is not used. The dynamic power benefits of an instantiated logic is proportional to its utilization whereas its leakage power and area costs are fixed. At fine granularity, having many instantiations would reduce the utilization per circuit. Hence, instantiating may not always result in net energy benefits.

In this chapter, we propose a novel, cross-layer approach for the synthesis of runtime accuracy-energy configurable hardware. Our approach combines circuit-level gating mechanisms and instantiating multiple approximate circuits in the architecture, to exploit both toggling activity and also synthesis relaxations. It enables fine-grain energy vs. area trade-offs in a design space that is a superset of two distinct approaches. Finding energy optimal solutions in this joint design space is a non-trivial function of required accuracies, their utilization in the workload, power savings that can be achieved at required accuracies, and leakage in the used technology. In [6], we introduced the basic hybrid

[1] The content of this chapter is based on the work originally published in [4–6].

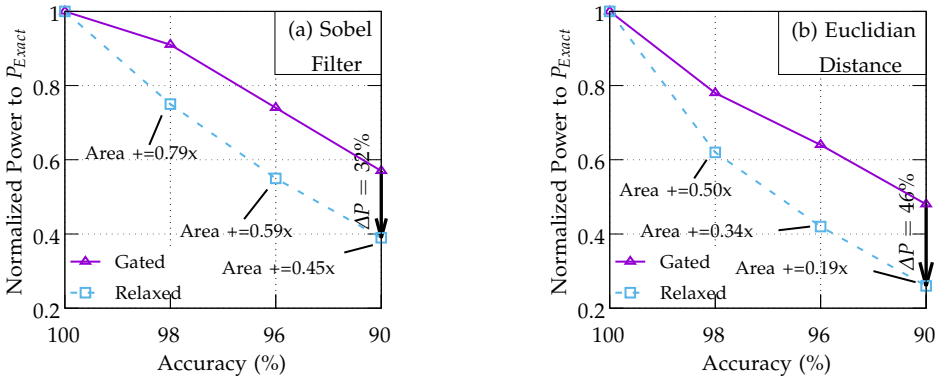


Figure 5.1: Power vs. accuracy to compare gating an exact circuit against instantiating relaxed, approximate circuits. Area costs of instantiated circuits are noted. All values are relative to the exact version of the corresponding circuit. ΔP is dynamic power savings of instantiating approach over gating.

design approach combining gating and instantiation applied in a manual fashion. This chapter extends our previous work into a systematic and automated cross-layer methodology that explores the design space efficiently yet exhaustively and finds the minimum energy requirement solution given an RTL block, a workload with specified accuracies and a maximum area constraint. Additionally, we significantly expand our evaluations to showcase generality, impact of the integration overheads, accuracy reconfiguration costs, input dependency of energy savings, and a detailed analysis of the leakage impact with 2 different technology libraries. All combined, our work makes the following key contributions:

- We propose a novel and cross-layer runtime accuracy-configurable hardware design approach. Our approach is general, supports several accuracies, and utilizes both circuit and architectural techniques in significantly reducing dynamic power consumption.
- Our work demonstrates the existence of a larger design space of accuracy-configurable hardware, with non-obvious trade-offs linked to the workload, hardware architecture, and technology.
- We present a systematic methodology to ensure selecting Pareto-optimal solutions. Our methodology creates a design-time knob on energy vs. area while matching given dynamic accuracy requirements.
- Our experiments show significant energy savings can be achieved despite the increase in area and leakage energy. We present these results in a technology-independent manner.

In our evaluations, we examine a range of circuits under different workloads and accuracy requirements. Our experiments show at 2× area cost and the same performance, up to 60% energy reduction compared to an exact hardware block and up to 32% energy reduction compared to state-of-the-art accuracy-configurable gated hardware while matching the accuracy.

5.1 Background

Figure 5.2 illustrates the background of a runtime accuracy-configurable system and highlights the position of our work. At design time (Figure 5.2a), we aim at synthesizing an energy optimum hardware that consists of multiple instantiations of approximate circuits and a gating mechanism. Here, optimality depends on the accuracy specifications that can be profiled for the given applications

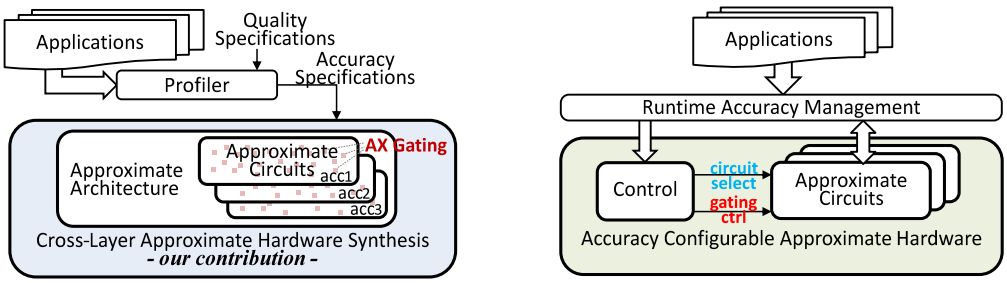


Figure 5.2: Background of a runtime accuracy-configurable system at design and run time. (a) Synthesis of approximate hardware with multiple circuit instantiations and a gating mechanism. (b) System-level abstraction of accuracy management, where the hardware can work in tandem with a runtime system.

and the quality specifications. In Figure 5.2c, hardware synthesized by our methodology offers accuracy controls to work in tandem with an external runtime system for accuracy management.

5.2 Accuracy-Configurable Hardware Architecture

In this section, we first detail *gating* mechanisms and present *instantiating* approximate circuits, which is a distinct, architecture-layer design approach for accuracy configurable hardware synthesis. Next, we introduce a novel hybrid, *cross-layer* design approach that jointly considers gating and instantiating and enables a design space with fine-grain energy vs. area trade-offs. Finally, we explain the hardware execution of cycle-by-cycle accuracy configuration, facilitating dynamic runtime adjustments.

5.2.1 Gating Groups of Paths in Circuit

We utilize the gating mechanisms discussed in Section 2.3.4 as low area cost accuracy configuration methods [57, 66, 93]. When the clock gating, power gating, and masking compared; masking by inserting control gates into the combinatorial paths increases path delays of the circuit. Thus, either the circuit delay increases or circuit area and power increase to match the same delay with more aggressive synthesis optimizations. These effects are counter-intuitive to our energy optimization design goal. Power and area overheads are reported as up to 7.6% and 8.7% [57]. Power gating mechanism, used in [57, 93] require many cycles, prohibiting cycle-by-cycle dynamic adjustments. In comparison, clock gating for approximations can eliminate such delay overheads and allow dynamic adjustments. Clock gating is a mean for disabling configurable partitions of a circuit. In [66], *significance constrained overgating* strategy, together with clock gating candidates algorithm result in configurable degrees of precision scaling on the input registers.

While our design approach will allow for any gating mechanism to be employed, we use clock gating as a baseline in our comparisons to represent the gating approach in the remainder of this chapter.

5.2.2 Instantiating Approximate Circuits with Different Accuracies

We employ adding and connecting multiple instantiations of a circuit for additional energy savings at area expense. As shown in Figure 5.1, the approximate instantiations have static accuracies and they exhibit lower power at the same delay as the exact instantiation. To design approximate instantiations

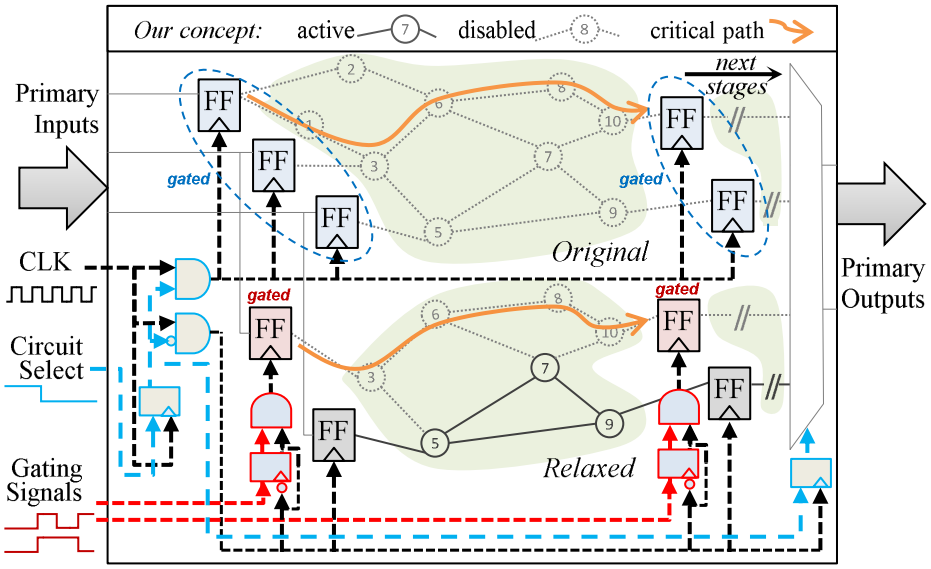


Figure 5.3: Cross-layer accuracy configurable hardware architecture. The lower positioned circuit is instantiated and gated.

of a circuit, we inherit the rich set of existing static approximation, i.e., single accuracy hardware design methods. By connecting separate instantiations of a circuit with different accuracies, the hardware is able to offer dynamic accuracy configuration to fulfill the varying requirements of the application at runtime.

Energy savings through functional simplification of the hardware architecture originate from having fewer gates and shorter paths compared to the exact circuit. When the exact and the approximate circuits are synthesized for the same clock delay, the shorter paths allow *synthesis relaxations*: Boolean remapping and undoing gate-level delay optimizations [31]. Boolean remapping converts a large number of parallel gates into a less number of serial gates while maintaining the boolean function (e.g., parallel-prefix adder to ripple carry adder). Undoing gate-level delay optimizations such as inserting buffers (load isolation) and splitting the driving gates on the critical paths (load splitting) reduce the number of gates in the design. Gate resizing re-composes the circuit with smaller transistors that require less energy. Hence, synthesis relaxations generate inherently more energy-efficient circuits.

Our approach is agnostic to the design method of the approximate instantiations. Existing work includes tools that parameterize precision scaling [73, 86], other automated functional simplifications [21, 110], and selecting manually implemented approximate circuits from a library [16]. In this chapter, we apply precision scaling to the primary inputs of the RTL and let the synthesis tool propagate this approximation using hardware compiler optimizations such as constant propagation and eliminating unused gates.

A challenge in instantiation is how to integrate the instances into the final design. Integration overheads can easily overweigh the benefits. If the accuracy-configurable hardware block is a component in a larger final design that includes a interconnection network such as a crossbar or a shared bus system, new instantiations can be connected as additional system components [16]. Larger hardware blocks such as systolic arrays [25, 27, 58, 120] share and reduce the integration overheads as they are connected through a single shared system interface. In case of multiplexer-based systems [10], this will require extending slave-to-master multiplexers for each instantiation. Such an approach also allows instantiating multiple functional units within the execution stage in a processor [119]. Alternatively, instantiations can be integrated as separate accelerators with distinct

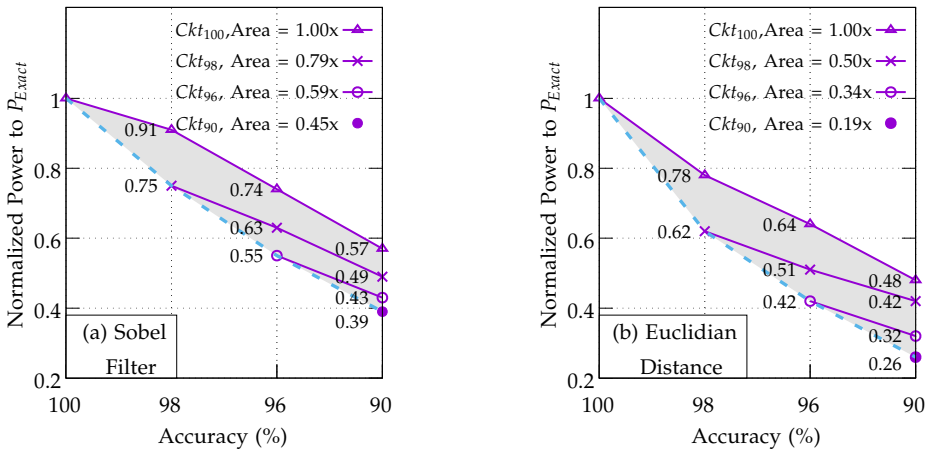


Figure 5.4: Power vs. accuracy design space of the cross-layer approach. Power values are labeled. Area costs of instantiated circuits are given in the legend. All values are relative to the exact version of the corresponding circuit.

memory-mapped IO at no multiplexer cost [94]. As a guideline, in case neither memory-mapped IO nor extending a multiplexer is possible, adding a new multiplexer to the same hardware stage should be avoided for delay and power overhead reasons, as shown by previous work [15, 57].

In this work, the instantiated circuits are at the level of at least one complete sequential stage. We call this granularity a *hardware block*. Potentially, instantiating at the combinational, sub-block level can exploit logic sharing opportunities for area savings across instantiated blocks. We utilize the final synthesis tool to find hardware common sub-expressions between instantiated combinational blocks for sharing logic and thus exploit opportunities for sharing logic in an automated manner. To connect new instantiations, we consider two architectural integration decisions: memory-mapped IO at no multiplexer cost [94] and extending slave-to-master multiplexers in shared bus systems [10].

5.2.3 Cross-Layer Design Approach

While gating brings some energy benefits at low area overhead, instantiating can significantly improve the energy benefits with a higher area overhead. Our proposed *cross-layer* approach combines the two: It instantiates distinct approximated blocks at coarse accuracy levels and selectively gates them. Consequently, it enables fine-grained intermediate accuracies and additional energy savings on their computation, without the area and leakage cost of instantiating each intermediate accuracy circuit. We take an existing RTL design (*hardware block*) as input and apply instantiating at the complete block level only. Afterwards, we apply a chosen gating method which can potentially gate instantiations internally, according to its search algorithm. Figure 5.3 illustrates the proposed cross-layer approach. Here, an instance of the original design (top) is combined with an approximate instantiation of the original circuit (bottom). The instantiation has a shorter critical path and fewer gates. It maximizes the power savings of computation at a particular accuracy. The instantiation is also clock gated to enable further power savings for a further range of accuracies.

In Figure 5.4, we extend our motivational example from Figure 5.1 with the cross-layer design approach. Starting from exact versions of the Sobel filter and Euclidian distance circuits, we synthesize approximate circuits for each accuracy. Afterward, we gate each synthesized circuit for lower accuracies. The leftmost value of each line represents power values achieved with instantiations. The lines towards the right represent power values achieved with gating each instantiated circuit. Note that the design space of prior, gating-only approaches is limited to the Ckt_{100} line. By contrast,

instantiations are limited to the left-most points on each curve (blue dashed line). The combined design space of our proposed cross-layer approach is shown by the shaded area. These examples also indicate a key insight to minimize dynamic power: We observe that for a single accuracy level, instantiating a circuit produces the most power-efficient solution, followed by gating the closest higher accuracy circuit available. In Section 5.3, we use this insight to reduce the search space.

Given an area budget, we can instantiate a set of circuits to address varying accuracy requirements. The energy-optimal selection of such a circuit set is not a trivial task. It necessitates answering the following questions: (1) how many circuits to instantiate and at which accuracies (2) which instantiated circuits to gate and (3) how to associate different accuracy requirements of workloads with the hardware in an optimum manner. The energy optimal solution is a function of hardware and workload. Within an area budget, the additional area and associated leakage cost vs. dynamic power savings of instantiations over gating should be considered. From the leakage perspective, the impact of power savings through instantiation is thereby weighted by the utilization of the corresponding accuracy level in a given workload.

A multi-layer search, i.e., independent decisions in the architecture and the circuit would lead to first instantiating the highest utilized circuit (to make it most efficient) and then gating for other accuracies. For example, following Figure 5.4, if a low accuracy is utilized 70% of the time, the best strategy seems to be instantiating an efficient circuit for this accuracy. However, if gating a slightly higher accuracy instantiation results in a very close energy consumption, yet considerably increases utilization of this efficient circuit, let's say 90% of the time, instantiating a higher accuracy circuit and gating it 70% of the time can become the ideal solution. The inter-dependency between the layers, i.e., the impact of gating on instantiations influence the optimal decision. Such decisions can only be made with a cross-layer search in the joint design space. Note that the number of solutions increases quadratically with the number of accuracies as we can see in Figure 5.4. We need to consider their combination to address all required accuracies, which is a power set with the complexity $O(2^{n^2})$. Therefore, an automated and systematic exploration methodology is necessary. We address this challenge in Section 5.3.

5.2.4 Runtime Accuracy Management

Our design approach supports dynamic accuracy configuration at cycle-by-cycle granularity. With this, the runtime accuracy changes can be set by a simple control unit. For each accuracy, there exists a single, static choice of circuit and gating configuration, determined at design time. I.e., no dynamic decisions are taken on circuit selection. Accuracy changes are orchestrated by an independent runtime system, such as [11, 72, 128], one cycle ahead of the operation. The frequency of accuracy changes is determined by the runtime system as a function of dynamic changes in the input stream or environment. The design of such a runtime system is out of the scope of this dissertation. Once an accuracy change is requested, the control unit sets *circuit select* and gating signals using a look-up table that holds the configuration for the selected accuracy. It propagates requested accuracies to the next stages in multi-stage hardware.

5.3 Exploration Methodology

In this section, we introduce our methodology to systematically and efficiently explore the joint design space. Here, energy optimality is a non-trivial function of required accuracies, their utilization in the workload, power savings that can be achieved at required accuracies, and leakage in the used

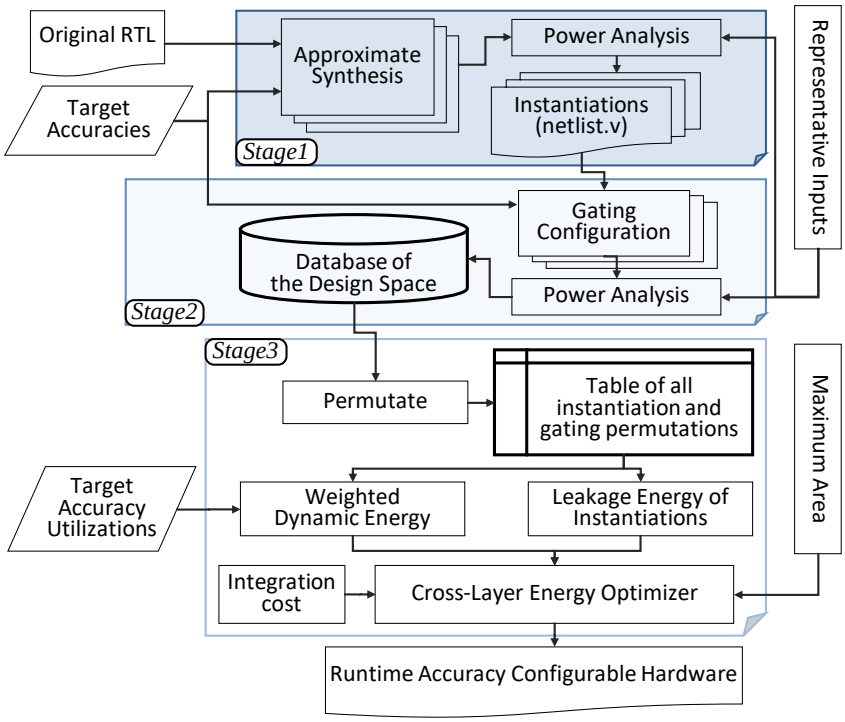


Figure 5.5: Cross-layer synthesis flow.

technology. Finding the highest energy saving combination that fits into an area constraint maps to the well studied Knapsack Problem, and it is known to be NP-complete.

In Figure 5.5, we present our cross-layer synthesis flow as a framework. We employ a divide and conquer alike algorithm, in which we search for the minimum energy solutions in 3 distinct hierarchical stages. We break down the cross-layer energy optimization problem to energy optimal instantiation and energy optimal gating problems. Both are further divided per accuracy. In stage 3, we combine the prior solutions to answer the original problem. Combinations of prior solutions, permutated over target accuracies gives us a complete table where each entry is a unique 3-tuple (target accuracy, instantiation, gating). Considering dynamic energy consumption weighted to circuit utilization, together with leakage and integration costs, we search for the lowest energy solution fitting into a given area constraint, by solving a $0/1$ Knapsack problem with dynamic weight dependencies among elements. Hence, we decouple the cross-layer search and explore the joint meta design space on top of the separate design spaces of synthesis and gating. Our approach thereby enables: (1) use of existing, effective and well-studied tools for approximate synthesis and gating without modification, (2) modularity in choosing among approximate synthesis methods and gating mechanisms (such as clock gating, masking, or power gating). (3) significant design time savings by running synthesis and gating only one time to create a database, not recursively. Our search remains complete following the assumption that the ideal instantiations will also lead to ideal gated instantiations. This assumption held true in the extent of our experiments. Typically Knapsack Problem is solved using a form of greedy search (gradient descent/ascent, simulated annealing, etc.). With our flow, we can run an exhaustive search in a feasible time to find optimum cross-layer solutions. Next, we detail our methodology using pseudo-code and analyze its computational complexity.

Algorithm 4 describes a 3-stage exploration process, namely *approximate synthesis*, *gating*, and *optimizer*. The inputs to the algorithm are an RTL description, the maximum area constraint (A_{max}) per hardware block, representative inputs (In_{rep}), and the list of required circuit accuracies with

Algorithm 4 Accuracy configurable hardware synthesis

Input: Exact RTL block, area constraint: A_{max}, representative input: In_{rep}, accuracy and utilization list: $< ACC_{\text{list}}, U_{\text{list}} >$
Output: Accuracy configurable circuit: $Ckt_{dynamic}$

Phase 1 – Approximate Synthesis

1: **for** each $ACC_i \in ACC_{list}$ **do**
2: Ckt_acc_i = Synthesize_AX (RTL, ACC_i)
3: $<P_i, P_{leak\_i}>$ = get_power(Ckt_acc_i, In_{rep})
4: $Ckt_{all} = Ckt_{all} \cup Ckt\_acc_i$
5: **end for**

Phase 2 – Gating

6: **for** each $Ckt\_acc_i \in Ckt_{all}$ **do**
7: **for** each $ACC_j < ACC_i$, in ACC_{list} **do**
8: $Ckt\_acc_i\_g_j$ = apply_gating (Ckt_acc_i, ACC_j)
9: $P_{i,j}$ = get_power($Ckt\_acc_i\_g_j$, In_{rep})
10: **end for**
11: **end for**

Phase 3 – Cross-Layer Energy Optimizer

12: $CktSet_{all} = \wp^{Ckt_{all}} \setminus \emptyset$
13: $CktSet_{candidates} = \{CktSet \in CktSet_{\text{all}} | (get\_area(CktSet) \leq A_{\max}) \wedge (Ckt\_acc_{max} \subseteq CktSet)\}$
14: **for** each $CktSet_k \in CktSet_{candidates}$ **do**
15: $CktSet_k$ = Synthesize($CktSet_k$)
16: **for** each ACC_i in ACC_{list} **do**
17: **if** $Ckt\_acc_i \in CktSet_k$ **then**
18: associate($CktSet_k.Ckt\_acc_i$, ACC_i)
19: $P_{CktSet_k} = P_{CktSet_k} + (P_{MUX}+P_i)*U(ACC_i) + P_{leak\_i} + P_{leak\_MUX}$
20: **else**
21: associate($CktSet_k.Ckt\_acc_{i+1}\_g_i$, ACC_i)
22: $P_{CktSet_k} = P_{CktSet_k} + P_{i+1,j} * U(ACC_i)$
23: **end if**
24: **end for**
25: **end for**
26: $CktSet_{minpower}$ = argmin($CktSet_{candidates}$, P_{CktSet})
27: $Ckt_{dynamic}$ = wrap($CktSet_{minpower}$)
28: **return** $Ckt_{dynamic}$

their utilization ($ACC_{\text{list}}, U_{\text{list}}$) in ascending order. The latter input pair can be obtained for a given workload using a profiler, e.g. from [26, 128].

Stage 1 synthesizes circuits at the same delay for all accuracies in ACC_{list}. Netlists can be generated by an automated tool that parameterizes precision scaling [73, 86] or functional simplification [21, 110], or synthesizing manually implemented approximate circuits from a library; our flow is agnostic to the synthesis method. We consider the synthesized approximate circuits optimal in the sense that they are the minimum power consuming circuits possible for that accuracy. We characterize the power of each circuit (Ckt_acc_i) using the given representative input in line 3. In line 4 we create a set of all synthesized circuits, Ckt_{all}.

Stage 2 applies a chosen gating method to each instantiation generated in stage 1 (Ckt_{all}), and for each target accuracy. In line 8, the *apply_gating* function returns the minimum power gated circuit that meets or exceeds the accuracy constraint, ACC_j. Note that gating can only reduce the accuracy of a netlist. So instantiated circuits are gated only to lower accuracies in the ACC_{list}. Possible gating strategies are given in [57] for masking and power gating, and in [66] for clock gating, where our flow is again agnostic to the chosen method. The output of this second stage is a database containing dynamic power, area values (as given in Figure 5.4), and also leakage power for all instantiation and gating combinations.

Stage 3 forms and evaluates circuit combinations, *CktSet*, to find the minimum power solution for a given area constraint. This decision depends on the utilization of each accuracy (U_i), the power consumption of each circuit (P_i), and also the integration cost (P_{MUX}). We show this relation in Equation (5.1) for a system that utilizes n different accuracies.

$$\begin{aligned} P_{CktSet_k} &= \sum_{i=1}^{n} P_i U_i + P_{Leakage} + P_{\text{MUX}} \\ \text{where :} \quad U_{total} &= \sum_{i=1}^{n} U_i, \;\; 0 \leq U_{total} \leq 1 \end{aligned} \tag{5.1}$$

U_{total} is the portion of cycles in which at least one circuit instantiation is active, i.e., the hardware is not idle. P_{CktSet_k} denotes the total power consumption of a circuit set out of many.

The optimizer first generates all dynamic-accuracy solutions ($CktSet_{all}$) from the combinations of all prior, static-accuracy solutions (Ckt_{all}). This is the power set of Ckt_{all}, except the empty set. In line 13, we reduce the possible dynamic-accuracy solutions $CktSet_{all}$ to $CktSet_{candidates}$. These are the solutions that fit into a given area constraint and also include the highest accuracy circuit (Ckt_acc_{max}). If a *CktSet* does not include Ckt_acc_{max}, we invalidate it because a part of the workload cannot be computed at a required accuracy. In line 15, we instantiate each *CktSet* and synthesize it with the "-incremental" switch in Synopsys DC to find and exploit the opportunities for sharing logic, i.e., hardware common subexpressions, between the circuits of each set.

The second part of stage 3 is to evaluate each candidate *CktSet*. For each circuit set, we associate the accuracy requirements of the workload with circuits. When *CktSet* contains a circuit with matching accuracy ACC_i, we associate them in line 18. The *associate* function generates directives for the control system, binding configurations to accuracies. Our insight from Section 5.2.3 has shown that when gated, the closest accuracy requires the lowest power. In case *CktSet* does not contain a matching accuracy circuit, we get the gating of the next higher accuracy circuit available in the set ($Ckt\_acc_{i+1}\_g_i$) in line 21-22, and use the configuration previously computed in line 8. This gives us a permutation of all dynamic-accuracy solutions (*CktSet*) over the target accuracies (ACC_{list}) as a complete table. With the workload associations in line 18 and 22, we know the utilization of each circuit (U_i) within each dynamic-accuracy solution (*CktSet*). We can fill our table with the power values. In lines 19 and 22, we accumulate the dynamic power consumption of each circuit weighted proportionally to its utilization. The total power consumption includes dynamic, leakage, and also integration costs. Our analysis on system integration has shown that multiplexer area, dynamic, and leakage power increase linearly with size. To consider the integration impacts, we synthesize *m-to-1* multiplexers where we changed m from 1 to 8, and used average increment in determining A_{MUX}, P_{MUX}, and P_{leak_MUX} at average primary output toggle rate. Finally, from all evaluated circuit sets, we pick the one with lowest power consumption, $CktSet_{minpower}$ in line 26. Since all circuits were synthesized to the same delay, the $CktSet_{minpower}$ is also the $CktSet_{minenergy}$. We instantiate all the circuits in $CktSet_{minpower}$ and connect them as explained in Section 5.2.2 to generate a dynamic accuracy configurable circuit ($Ckt_{dynamic}$) in line 27.

Thus, our proposed methodology systematically explores the design space of dynamic accuracy configurable hardware. It finds circuit sets, that require minimum energy, and match the required accuracies by their construction, in the joint design space of gating and instantiating approaches. With an adjustable area constraint, A_{max}, it creates a design-time knob on energy vs. area. It automates the generation of $Ckt_{dynamic}$, i.e. the energy-optimized dynamic accuracy configurable circuit.

The computations in Stage 1 and 2 are necessary only one time and can be parallelized. Afterward, our optimizer works on provided power and area values. Thus, it is completely decoupled from the previous stages. To ensure Pareto-optimal solutions, we run an exhaustive search with a complexity $\mathcal{O}(n2^n)$, where n is the number of elements in ACC_{list}. If we break it down, in line 14, the for loop operates on $CktSet_{candidates}$, which is a subset of the power set term $\wp^{Ckt_{all}}$ has the complexity of $\mathcal{O}(2^n)$. In line 16, another for loop operates on ACC_{list}, with complexity $\mathcal{O}(n)$. The insight we gained in Section 5.2.3 allowed us to reduce the gating candidates in line 21 to one. As a result, this reduced the complexity by one order. Overall, an exhaustive search is possible within seconds for the practical range of n, where our carefully structured and computation efficient algorithm is capable of exploring a design space significantly beyond the previous work.

Table 5.1: Circuits used in our experiments.

Name	Function	Bitwidth	I/O	$t_{ckt}[ns]$	Area $[\mu m^2]$	Input
FIR	4-Tap FIR Filter	8	8/16	0.30	1177	random uniform
Neuron	8 input ReLu Neuron	8	64/8	0.91	5865	random uniform
Sobel	3x3 Sobel Filter	8	64/8	0.60	1337	cameraman
Gaussian	3x3 Gaussian Filter	8	72/8	0.50	764	cameraman
Euclidian	Euclidian Distance (without square-root)	8	32/16	0.77	1380	random uniform

5.4 Experiments and Results

We evaluate the effectiveness of the proposed cross-layer methodology on a variety of circuits as listed in Table 5.1. For the synthesis, we use Synopsys Design Compiler with the ultra high effort option using an industrially proven technology library, TSMC 65nm generic plus (typical case, 1.0 Volt, 25°C). We synthesized a circuit for each given accuracy by precision scaling the primary inputs and letting the synthesis tool propagate optimizations to later stages with compiler optimizations (dead code/gate elimination, constant propagation, etc.). All instantiations of circuits are synthesized to match the same delay, i.e.,110% of the minimum delay of the corresponding exact circuit ($t_{ckt} = delay_{min} \times 110\%$). Therefore all circuits compared in our experiments have the same performance. The extra 10% delay budget was applied to give some headroom for optimizations; It is not a limitation. We instantiated each circuit directly in the HDL testbench. For gating, we reduced the number of primary inputs supplied to the netlists according to the algorithm given in the state of the art [66]. For the area and leakage cost of gating, we assumed a conservative 3% penalty per added accuracy, which is in line with [57]. We calculated energy by multiplying total power and time. To characterize the dynamic power consumption of our circuits, we generated toggling activity files (*.vcd* file) from gate-level simulations with ModelSim and provided them to Synopsys Primetime in *.saif* format. Similarly, we included the leakage power values that Primetime reported. We ignore the clock tree power of our post-synthesis netlists. In practice, there could be additional minor clock tree savings, up to 2.5% according to our experiments. The unit of length is μm in our library and we derive the area in μm^2. For a gate count comparisons, the smallest NOT and NAND gates are 1.08 and 1.44 μm^2, respectively. As inputs, in Sobel and Gaussian filter experiments we used a 512x512 pixel cameraman picture. These filters are used exclusively in image

processing. For other circuits, we used random inputs with uniform distribution. Note that the 3x3 Sobel kernel inputs 8 pixels (64 bits), excluding the center pixel. The FIR filter module inputs an 8-bit value and internally propagates it through its stages.

Workload Profiling:

The required circuit accuracies and their utilizations are application and input dependent. To abstract their effect on our methodology, we used 4 different accuracies and 3 different utilization distributions as shown in Table 5.2. In our experiments on the Sobel filter, 98%, 96% and 90% accuracies corresponded to PSNR 45, 38 and 31dB, respectively.

Table 5.2: Utilization distributions (U_i) of 4 different circuit accuracies for 3 synthetic workloads.

Utilization Distributions	Accuracy			
Workload	100%	98%	96%	90%
W_ex - *mostly exact*	0.5	0.2	0.2	0.1
W_eq - *even distribution*	0.25	0.25	0.25	0.25
W_ax - *mostly approximate*	0.1	0.15	0.05	0.7

Accuracy:

We considered circuit accuracies in terms of $1 - MRED$ (Mean Relative Error Distance), as shown in Equation (5.2), where O_{approx_n} is the nth approximate and O_{exact_n} is the nth exact output value.

$$Accuracy = 1 - \frac{1}{N}\sum_{n=1}^{N}\frac{|(O_{approx\_n} - O_{exact\_n})|}{O_{exact\_n}} \tag{5.2}$$

As we use the mean value in Equation (5.2), our error metric involves both the rate and magnitude.

Optimizer Runtime:

We profiled the runtime of our Cross-Layer Energy Optimizer (*Algorithm 1 - Stage 3*) with MATLAB R2016a using integer variables on an Intel i5 6600 with 16GB RAM. In Table 5.3, we show that an exhaustive search is feasible as it is in the order of seconds for the practical range of the number of required accuracies n (3-6 in [27, 56, 57, 66, 90, 93, 97, 98, 119, 120, 128]). In fact, our computation-efficiently structured algorithm is capable of exploring a design space for double the n, that is quadratically larger than the previous work.

Table 5.3: Optimizer run time

n	$\leq$9	10	11	12
[s]	0.01	0.14	1.42	21.3

5.4.1 Design Space Exploration

We begin our evaluations by presenting the design space of a Sobel filter obtained using our proposed methodology in Figure 5.6 under 3 different workloads from Table 5.2. Each solution on the design space corresponds to a particular combination of instantiated circuit, gating and workload accuracy association. We show the Pareto front obtained by our proposed methodology with a dashed line. The Pareto solutions we refer to in the text are labeled as follows: *G1-G3* are gating-only solutions, *Hy1-Hy5* are hybrid solutions of gating and instantiations in the cross-layer

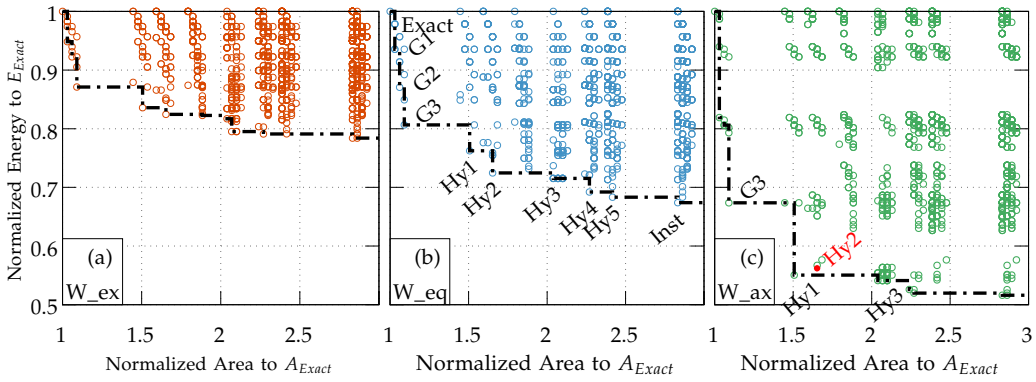


Figure 5.6: Design space of an accuracy configurable Sobel filter under 3 different utilizations given in Table 5.2. Pareto-front is obtained using the proposed methodology. Pareto solutions in Figure 5.6b are labeled to be used in Figures 5.7 and 5.12.

design space, and *Inst* is the instantiation-only solution. When examined at 2× maximum area constraint, the Pareto-optimum solution in Figure 5.6b is labeled *Hy2*. At the same area cost, under *W_ax* workload in Figure 5.6c, *Hy1* is the optimum solution, which dominates *Hy2*. *Thus, Pareto-optimal solutions are workload dependent.*

The dynamic power impact of instantiating an approximate circuit is proportional to the utilization of its accuracy. Our methodology first instantiates the highest power impact solution. At the excess area, only low impact circuits remain. As an example, under the mostly approximate workload in Figure 5.6c, 90% is the dominating accuracy. With solution *Hy1*: [$Ckt\_acc_{100}g_{98}g_{96}$, Ckt_acc_{90}] (i.e., instantiating exact and 90% accuracy circuits and gating the exact for 98% and 96%) we already achieve significant energy savings. Additionally instantiating a 96% accuracy circuit (Ckt_acc_{96}) only reduces the energy consumption by a mere 2.6% at $0.6\times A_{\text{Exact}}$ extra area cost. Similarly, under *W_eq*, where the accuracy utilization is even, an extra 0.65× area at first reduces energy by 28% w.r.t. *Exact* (10% w.r.t. *G3*), and at the end, only 1.3%, w.r.t. *Hy5*. *Thus, energy savings diminish at excess area.*

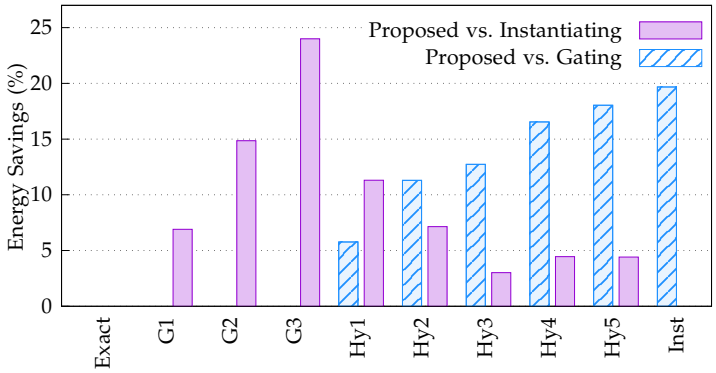


Figure 5.7: Pareto front comparison of cross-layer solutions against gating and instantiating solutions from Figure 5.6b.

5.4.2 Comparison of Pareto-Optimal Solutions

In Figure 5.7, we show the energy savings of cross-layer solutions over gating and instantiating alone for the Sobel Filter with *W_eq* workload. We obtain the gating Pareto front by providing only the exact circuit as the output of Stage 1 in Algorithm 1. Similarly, to obtain the Pareto front of the instantiating approach, we skip Stage 2 in Algorithm 1 and associate the missing accuracies with a higher accuracy circuit. The figure highlights that, as we increase the area budget, the Pareto solutions of *cross-layer* and *instantiating* approaches offer energy reduction over state-of-the-art gating approaches. *The joint design space offers solutions that are superior or equal to the two individual approaches.*

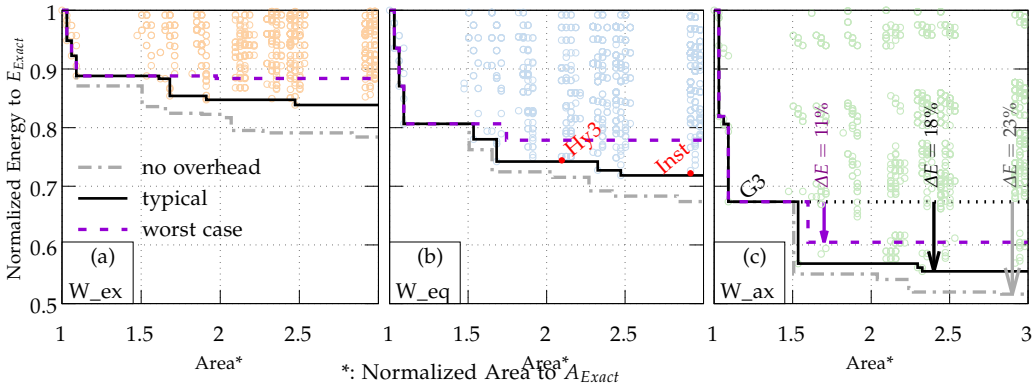


Figure 5.8: Design space of an accuracy configurable Sobel Filter with *typical* integration costs (control+MUX+clk) under 3 different utilizations given in Table 5.2. Pareto-fronts are obtained using the proposed methodology with varying integration costs.

5.4.3 Analysis of Integration and Control Overhead

The integration overheads, as we have previously discussed in Section 5.2.2, are dependent on the connection to an enclaving hardware system. The experiments in Figures 5.6 and 5.7 represent energy savings when the system integration costs can be averted, such as in adding instantiations with memory-mapped IO [94]. For a shared bus architecture, we need to extend the slave-to-master multiplexers to connect new instantiations [10].

We included an m-to-1 multiplexer in our design space exploration in the experiments shown in Figure 5.8. The system-level integration overheads manifest themselves as additional energy and area as a function of m, i.e., the number of instantiations. In Figure 5.8, we show their impact on the Pareto-front for 3 scenarios: The worst-case line represents using the fastest multiplexer generated by the synthesis tool with the delay constraint set to 0 ns to maximize synthesis effort. The typical line represents a multiplexer with an excess delay budget. In our experiments, the multiplexer delay has been a fraction of the circuit delay ($< 41\%$). Hence, if necessary, multiplexers could always comfortably fit into a pipeline stage that is delay constrained by the experimented circuit. Finally, we included the 'no overhead' line from the previous experiments, as shown in Figure 5.6. Values are normalized to the area and the energy of the Sobel Filter. Depending on the scenario, Figure 5.8c shows 11% to 23% energy savings over *G3*, the most energy-efficient solution of the state-of-the-art gating method.

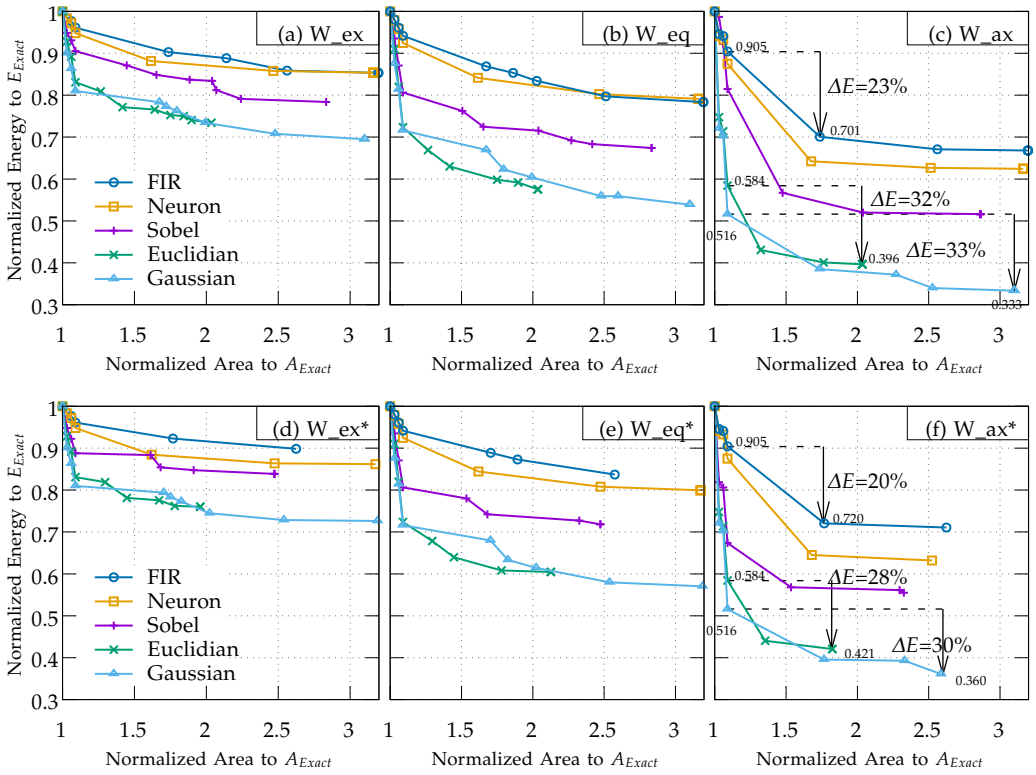


Figure 5.9: Pareto curves for a range of circuits under workloads with accuracy utilizations given in Table 5.2. The experiments denoted with a "*" in (d,e,f) include system integration overheads. Each point represents a particular dynamic accuracy configurable circuit, $Ckt_{dynamic}$, which is energy optimized for the given area and the workload.

Note that in the typical case, the solution of instantiating all circuits (*inst*) is no longer on the Pareto front for any workload. We also observe that *Hy3* is dominated in Figure 5.8b. These are the first (lowest area) solutions that use 4 and 3 instantiations and their energy reduction do not compensate for the increased integration overheads. This finding supports the previous argument of *energy savings diminish at the excess area* for a different reason: increasing overheads.

We have also explored placing multiplexers inside the pipeline stages of the hardware block, similar to the previous work [15, 57]. As such, we can enable sharing some stages and configuring accuracy in others. This setup has not given energy savings more than gating as it significantly reduces the delay margin for both the instantiations and the multiplexer and thus more aggressive delay optimizations that increase energy.

5.4.4 Area vs. Energy Trade-offs

In Figure 5.9, we generalize our exploration by repeating it for a range of datapath circuits. We share the energy-area Pareto front obtained under 3 different workloads from Table 5.2, with (d,e,f) and without integration overheads (a,b,c). For each circuit, after the exact, the first 3 data points represent the gating solutions and for Figure 5.9a,b,c the last data point represents the solution at which all circuits are instantiated. Figure 5.9 shows that *our cross-layer synthesis methodology is general and it applies to a wide range of circuits.*

We can observe that Pareto points change in number and x-axis position under different workloads. For instance, the Gaussian filter has 10 Pareto solutions under the workloads *W_ex* and *W_eq* whereas

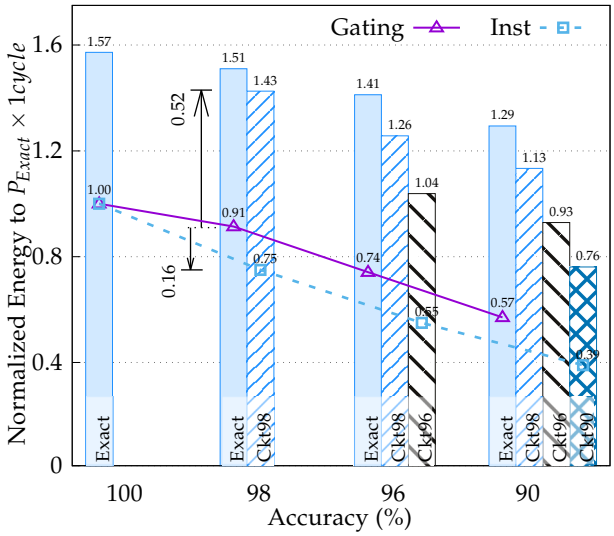


Figure 5.10: Energy required for accuracy reconfiguration of the Sobel Filter: Each bar represents one-time energy to activate a circuit for the desired accuracy. The purple line represents clock gating the exact circuit for the desired accuracy. The blue line is the runtime energy requirement of instantiations at their designed accuracy, without switching. Normalized to the average of $P_{\text{exact}} \times 1cycle$, for Cameraman picture as input.

8 under *W_ax*. This validates the workload dependency of Pareto solutions. Similarly, a maximum area constraint of $2 \times A_{Exact}$ results in an average 18%, 28%, and 48% energy savings under W_ex, W_eq and W_ax workloads, respectively. Afterward, an additional area budget of A_{Exact} only reduces energy by 4%, 5%, and 1%. These values support our previous finding on diminishing energy savings at excess area. In other terms, instantiating all accuracies solution (*inst*) has only a small, incremental energy saving over cross-layer solutions at relatively large area cost.

Previously, in Figure 5.1, we shared power savings with *instantiating* compared to an exact Euclidian distance calculator as up to 78%. Compared to gating an exact circuit, we reported additional power savings of up to 46% with instantiation at matching accuracy. These values set the maximum possible savings: under a workload utilizing constantly 90% accuracy ($U_{90} = 1$). Between our workloads, *W_ax* shows higher energy savings than *W_eq* and *W_ex*, as it utilizes the energy-efficient, approximate instantiations more. At 2x area cost, the power reduction under the mostly exact workload in Figure 5.9a is up to 26% whereas the reduction under the mostly approximate workload reaches 60% in Figure 5.9c. When compared to the state-of-the-art gating approach and at matching accuracy, in Figure 5.9c, *we achieve up to 32% additional energy savings.*

In the experiments shown in Figure 5.9(d,e,f), we considered the system-level integration overheads by including an *m-to-1* multiplexer in our design space exploration methodology, generalizing our previous overhead analysis in Section 5.4.3. We observe that some Pareto points such as solutions that instantiate all (*inst*) vanish; effectively, for the remaining Pareto-solutions, the energy overhead of integration is less than 4%. When Figure 5.9(d,e,f) are examined, apart from the initial low-cost gating solutions, *only the cross-layer solutions remain on the Pareto-front.*

5.4.5 Energy Cost of Runtime Accuracy Reconfiguration

Runtime systems decide and dictate an accuracy reconfiguration in relatively large periods [11, 128], and independently of our hardware synthesis methodology. Architecturally, as shown in Figure 5.3, an accuracy change is conducted by setting a small number of registers ($\sim log(n)$, for n accuracies).

Although this register energy cost is negligible, activating a different instantiation can result in a context switch cost.

In the real-world, due to temporal and spatial correlations in input data, we can expect that not all input bits change and lead to toggles and energy consumption in every cycle. In other words, bit-level input correlation is a factor in energy consumption. For instance, in a picture, neighboring pixels are likely to have a similar value, as do successive frames in a video stream. Running a kernel on the equal value pixels of a graphic would not lead to any toggling activity. Small changes in the input are likely to toggle only LSB paths which are inherently short and more energy-efficient [8]. However, activating a different circuit instantiation by switching accuracy or changing the input data context leads to a high amount of toggle in the hardware. Upon a context switch, the correlation in inputs is lost and the energy consumption is expected to be higher. The impact can be generalized as energy consumption upon context change.

Each time we switch to a different instantiation, we can expect the current input values to not be correlated to what the inputs of the instantiation were the last time it was used: A context switch occurs to instantiations upon activation. To evaluate this energy impact of input data correlation, we have tested the Sobel Filter with uniform random input values that have no correlation from one cycle to the next, which emulates a setup in which accuracy re-configuration occurs in every cycle. In Figure 5.10, we compare the average power consumption per cycle with accuracy configuration (bars) against the runtime cost of using the same circuit with normal inputs benefiting from correlation (lines). In other words, bars represent the energy consumption of switching to and activating a specific instantiations when changing the hardware accuracy, while the lines represent the energy consumption at a fixed accuracy, where purple is a gated exact circuit and blue is instantiations without gating. The figure can be read as follows: Normalized to an energy unit of $P_{Exact} \times 1cycle$, 1.43x energy is required to switch to the 98% accuracy instantiation. In comparison, if there are no instantiations and only the exact circuit, the circuit could be gated to 98% at no context switch cost, resulting in a fixed energy cost of $0.91x$ as shown with the purple line. Hence, by switching to a different accuracy circuit, we incur an energy overhead. However, instantiated circuits are more energy-efficient than the gated ones at the same accuracy. Considering a net switching cost of $(1.43 - 0.91) = 0.52$x and following savings of $(0.91 - 0.75) = 0.16$x, our approach can amortize an accuracy reconfiguration within 4 cycles.

This analysis is particularly useful as it abstracts the hardware from the workload and hence retains generality. We see the impact of switching between instantiations on energy, independent of the workload characteristics such as the switch frequency and previous state (as in a finite state machine).

5.4.6 Input Dependency of Cross-Layer Design Space

The dynamic power consumption depends on the characteristics of the input stream. Figure 5.11a is the cameraman picture used in our experiments. We use it is a representative of real-world data with spatial correlations (similarly valued neighbor pixels). The image processing kernels we used, Sobel and Gaussian filters, work on neighboring pixels and shift over the image pixels like a sliding window for each output. The required computation is strongly affected by spatial correlations. In Figure 5.11b, we analyze the bitwise similarity between neighboring pixels of the cameraman picture using MATLAB. The figure can be read as follows: the MSB of the pixels have the same value for 95% of the neighbors; only 5% of the neighbor pixels have a different MSBs. The LSBs of the neighbor pixels, however, have the same value for only 54%. *Using real-world data, the kernel inputs have relatively high-frequency changes between LSBs compared to their MSBs.* In comparison, uniform

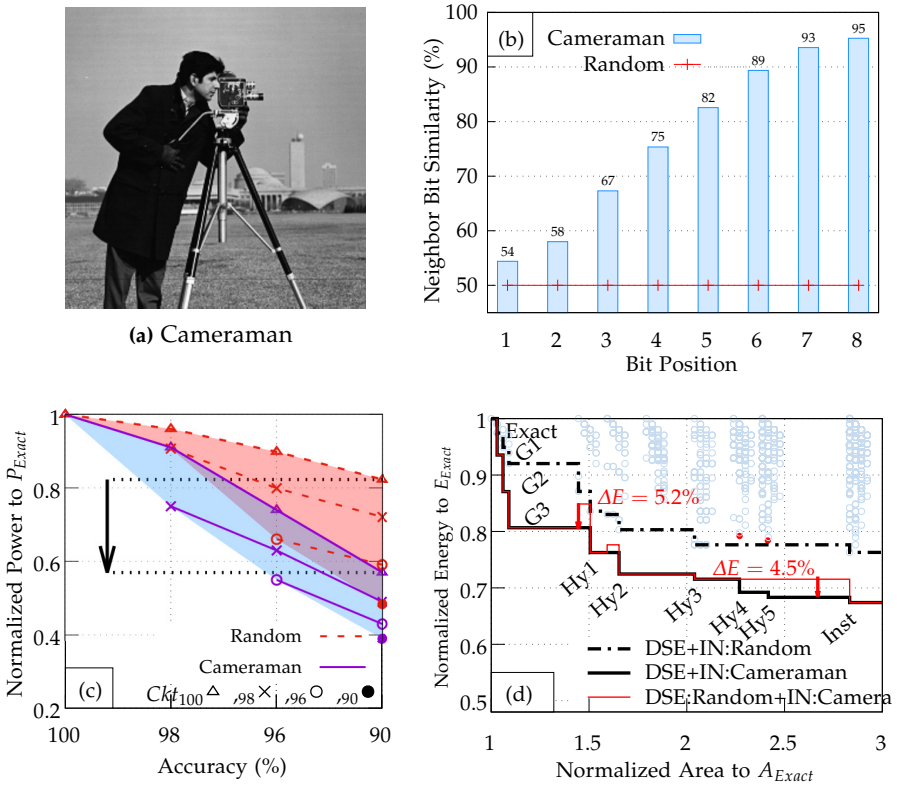
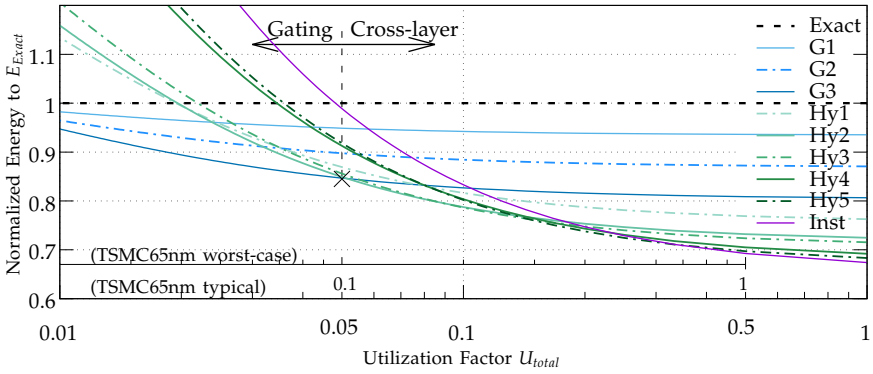


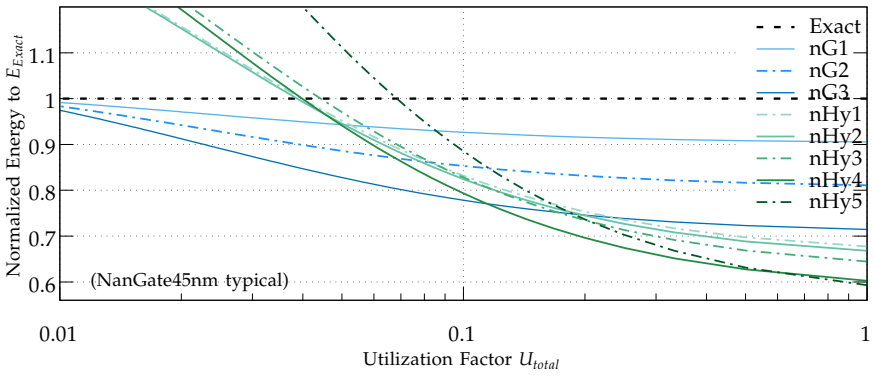
Figure 5.11: Input dependency of cross-layer design space. (a) Cameraman picture used in our experiments. (b) Bitwise similarity between neighbor pixel values of the Cameraman picture vs. uniform random numbers. (c) Power consumption of a Sobel Filter with random inputs compared to the cameraman picture as input. Lines with different markers represent the dynamic power consumption of instantiations when gated, as in Figure 5.4. (d) Design Space of accuracy configurable Sobel filter under W_eq workload from Table 5.2, with random inputs, highlighting the mismatch in solutions and its energy impact.

random input has the same, 50% toggle rate for each bit position. As we lower accuracy through precision scaling, we omit the high frequency LSB toggles, which leads to bigger normalized savings in real-world data (up to 44%, in Figure 5.11c) than random. Since both experiments are on the same circuit, the area and leakage power remain the same. To the scope of our experiments, *precision scaling leads to higher normalized energy savings with inputs from real-world data than random inputs with uniform distribution.*

We run our cross-layer synthesis methodology using both random and representative input values for comparison by changing the inputs in line 3 and 9 of *Algorithm 1*. In Figure 5.11d, we present the design space exploration of the accuracy configurable Sobel filter with random input and the Pareto-fronts for random and representative inputs when evaluated running the same inputs as used during search. Comparing the Pareto-fronts, we notice a difference in energy consumption. The red line in Figure 5.11d shows the solutions for random input evaluated by running on the cameraman input. An in-depth analysis shows that most of the solutions on both Pareto-fronts are the same configuration. However, some Pareto-solutions are input dependent. Using an unrepresentative input during design space exploration can lead to suboptimal solutions, which cost up to 5.2% energy. By contrast, performing exploration using representative inputs reveals additional Pareto-solutions (*Hy4, Hy5*) for further energy savings of up to 4.5% in this example.



(a) TSMC 65nm typical and worst-case corners



(b) NanGate 45nm typical corner

Figure 5.12: Leakage impact on accuracy configurable Sobel filters under varying utilization factor U_{total} with different technology libraries. Each line is a Pareto solution under the workload *W_eq*.

5.4.7 Leakage Energy Analysis and Technology Independence

Instantiating additional circuits increases area and hence the leakage of the hardware. The contribution of leakage energy over total energy is a function of the technology and hardware utilization, U_{total}: the portion of cycles, in which at least one circuit is active. In Figure 5.12, we present the total energy consumption of dynamic accuracy configurable Sobel filters under the workload *W_eq* for different technologies and varying utilization factors, achieved by introducing idle cycles. The y-axis shows the total energy (including the leakage), normalized to the total energy of the exact circuit. Note that the x-axes (utilization factor) are on a logarithmic scale. The graph can be read as follows: at a utilization factor of 1, there are no idle cycles. At 0.1 the hardware is used for 1 cycle out of 10 on average, i.e., 9 idle cycles. Here, the energy contribution of the dynamic part is also reduced to 0.1, while the leakage remains constant.

In Figure 5.12a, dynamic accuracy configurable circuits that we use are from the Pareto front of Figure 5.6b. We plot their energy under two different utilization scales: The main (bottom) x-axis represents our analysis with the TSMC65nm library at the typical corner. The second x-axis represents results for the worst-case corner instead. The TSMC65nm library characterized for worst-case conditions (0.9V, 125C) results in an increase in leakage to dynamic power ratio by 2x. This is equivalent to doubling the number of idle cycles, i.e., scaling utilization by a factor of 0.5 as shown in the figure. The highest energy savings are obtained with instantiation (inst) and cross-layer solutions (*Hy1-Hy5*) at high hardware utilization, such as 1 to 0.1 for the x-axis representing the typical corner. As the effect of leakage power becomes more prominent, the energy of *Hy1-Hy5*,

and inst increase above gating-only solutions *G1-G3* between 0.1 and 0.05 (marked in Figure 5.12a). Until the range of 0.05 to 0.0125, the *Hy1-Hy5, and inst* remain superior to the exact circuit.

We applied the Sobel Filter to a camera system application with a 512×512 pixel video feed at 30 frames per second, running at a clock frequency of 100MHz. In this specification, the utilization factor is 0.08. Even though this can be considered as a low utilization to justify the accelerator, the best solution is *Hy2* (with typical x-axis). When we increase the resolution and frame rate to 1280×720 pixels and 60fps, the utilization factor becomes 0.54. At this value, leakage power has a much lower impact on total energy. The instantiation solution (*inst*) offers 0.4%, 26%, and 37% more energy savings than solutions *Hy5*, *G3* and *exact*, respectively.

The data can be read in a technology-independent manner, for different leakage impact on total energy, by scaling the utilization factor axis accordingly, as we show in Figure 5.12a. Furthermore, In Figure 5.12b, we redo the experiments (synthesis, gating, design space exploration, and leakage analysis) with a NanGate 45nm library. Note that the Pareto-points in Figure 5.12a and Figure 5.12b are not the same set. For instance, the NanGate *inst* solution is dominated by *nHy5* and not drawn in Figure 5.12b. There are differences in synthesis and gating results that change the design space and Pareto-points. The NanGate experiments validate our previous leakage analysis using a completely independent technology library. With these experiments, we can generalize that *for a significant range of utilization factor, the cross-layer approach produces superior solutions.*

5.5 Summary

This chapter addressed the necessity of accuracy-configurable hardware systems with the exploration of applying gating mechanisms to existing circuits together with instantiating more efficient circuits into the architecture. Jointly, they present a larger design space where non-trivial cross-layer decisions are necessary to find optimal solutions. We proposed a systematic methodology to ensure Pareto-optimal combinations towards minimizing energy consumption under given workload and area constraints. Our work has demonstrated that dynamic accuracy configurable hardware with significantly (up to 33%) reduced energy compared to existing gating solutions can be synthesized when more circuit area can be utilized.

6 Conclusion

Approximation-resilient applications paved up avenues for new methods to improve energy efficiency and performance. Over the last decade, a significant amount of work is proposed across the computing stack to best take advantage of the approximations. One key aspect is that required computation accuracy changes with input, applications, and application quality targets. The current conventional design automation flow is optimized for exact computation and provides sub-optimal benefits when used for accuracy-configurable computations. This dissertation proposed methods and implementations to design accuracy-configurable hardware, exposing the accuracy knobs, and also to optimize the hardware for accuracy-configurable use.

In Chapter 3, this dissertation introduced a systematic methodology to automatically synthesize circuits with enhanced timing-error resilience when frequency is scaled beyond safe limits. The key idea behind is to optimize non-critical paths for the delay. Given any arbitrary circuit, this methodology identifies primary outputs with a remaining slack margin and assigns them the minimum delay constraint for successful synthesis. Consequently, it reduces the probability of timing errors and thus improves the accuracy. It also makes the timing errors occur more gradually with frequency scaling. As demonstrated by experiments, this methodology extends the design space to higher frequencies and finding favorable trade-offs between performance and accuracy for both standalone and cross-layer techniques that utilize timing speculations for approximate computation.

The experiments in Chapter 3 have exposed that many circuits have an uneven complexity in their topologies. This property leads to most critical path delay dominated circuit delay, power, and area requirements while achieving limited, best-effort improvements on low-complexity paths. Chapter 4 introduced the second contribution of this dissertation, a throughput-constrained, energy-optimized synthesis method. It relaxes the delay constraints that apply to the majority of the circuit while tightening it for a smaller part used for reduced precision operations. Consequently, the resulting circuits require less area and average energy, when used for accuracy-configurable approximate computing, while being able to maintain the initial average throughput. These circuits possess the property of dynamic frequency-precision scalability as they can operate at high precision with a slow clock and at a low precision with a fast clock.

Chapter 5, addressed the necessity of accuracy-configurable hardware systems with the exploration of circuit-level gating mechanisms together with instantiating more efficient circuits into the architecture. Jointly, they present a larger design space where non-trivial cross-layer decisions are necessary to find optimal solutions. In this chapter, a systematic methodology is proposed to ensure Pareto-optimal combinations towards minimizing energy consumption under given workload and

area constraints. This work has demonstrated that the cross-layer design space offers dynamic accuracy-configurable hardware with significantly reduced energy compared to existing gating solutions when more area can be utilized.

The design methodologies proposed in this dissertation utilize underlying circuit properties into account in an automated and general way. As they are built on the logic synthesis that is used for all logic, they inherent generality and rich set of optimizations. These methods design runtime variable accuracy hardware with exposing knobs for accuracy configurability and optimize the hardware for runtime accuracy-configurable use towards maximally exploiting the approximation benefits while maintaining quality targets.

Future Work

Runtime accuracy configuration necessitates monitoring changes in inputs or quality targets and reacting to them. This comes in some form of overhead. Therefore, it is important to make these runtime methods lightweight. A potential solution can be hardware support for runtime methods to minimize their overhead and maximize approximation benefits

A cross-layer approach includes the design space of single-layer approaches and merges them. Thus, it creates a larger design space where the solutions are better than or equal to any incorporated single layer of the computing stack, for any design metric. This dissertation has proposed a joint circuit and architecture design space exploration to reduce energy consumption. Integrating more layers and making design decision in the larger joint design space has the potential to provide further improvements.

Bibliography

[1] Sara Achour and Martin C Rinard. "Approximate computation with outlier detection in topaz". In: *Acm Sigplan Notices* 50.10 (2015), pp. 711–730.

[2] Mohammad Abdullah Al Faruque, Thomas Ebi, and Jorg Henkel. "Configurable links for runtime adaptive on-chip communication". In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE. 2009, pp. 256–261. DOI: 10.1109/DATE.2009.5090667.

[3] Mohammad Abdullah Al Faruque, Thomas Ebi, and Jorg Henkel. "Run-time adaptive on-chip communication scheme". In: *2007 IEEE/ACM International Conference on Computer-Aided Design*. IEEE. 2007, pp. 26–31. DOI: 10.1109/ICCAD.2007.4397239.

[4] Tanfer Alan, Jorge Castro-Godinez, and Jörg Henkel. "Multiple Approximate Instances in Neural Processing Units for Energy-Efficient Circuit Synthesis (WiP)". In: *International Conference on Compilers, Architectures and Synthesis For Embedded Systems (CASES)*, (2021). DOI: 10.1145/3451939.3477594.

[5] Tanfer Alan, Andreas Gerstlauer, and Jörg Henkel. "Cross-Layer Approximate Hardware Synthesis for Runtime Configurable Accuracy". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2021). DOI: 10.1109/TVLSI.2021.3068312.

[6] Tanfer Alan, Andreas Gerstlauer, and Jörg Henkel. "Runtime Accuracy-Configurable Approximate Hardware Synthesis Using Logic Gating and Relaxation". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020, pp. 1578–1581. DOI: 10.23919/DATE48585.2020.9116272.

[7] Tanfer Alan and Jörg Henkel. "Probability-Driven Evaluation of Lower-Part Approximation Adders". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* (2021). DOI: 10.1109/TCSII.2021.3093984.

[8] Tanfer Alan and Jörg Henkel. "SlackHammer: Logic Synthesis for Graceful Errors Under Frequency Scaling". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2802–2811. DOI: 10.1109/TCAD.2018.2858364.

[9] Pietro Albicocco, Gian Carlo Cardarilli, Alberto Nannarelli, Massimo Petricca, and Matteo Re. "Imprecise arithmetic for low power image processing". In: *Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar Conference on*. IEEE. 2012, pp. 983–987. DOI: 10.1109/ACSSC.2012.6489164.

[10] *ARM AMBA 5 AHB Protocol Specification*. Accessed: 2020.05.17. URL: https://static.docs.arm.com/ihi0033/bb/IHI0033B_B_amba_5_ahb_protocol_spec.pdf.

[11] Woongki Baek and Trishul Chilimbi. "Green: A system for supporting energy-conscious programming using principled approximation". In: *Conference on Programming Language Design and Implementation*. ACM. 2010, pp. 198–209.

[12] Lars Bauer, Muhammad Shafique, and Jorg Henkel. "Run-time instruction set selection in a transmutable embedded processor". In: *2008 45th ACM/IEEE Design Automation Conference*. IEEE. 2008, pp. 56–61. DOI: 10.1145/1391469.1391486.

[13] Lars Bauer, Muhammad Shafique, Simon Kramer, and Jörg Henkel. "RISPP: Rotating instruction set processing platform". In: *Proceedings of the 44th annual Design Automation Conference*. 2007, pp. 791–796. DOI: 10.1145/1278480.1278678.

[14] Kartikeya Bhardwaj, Pravin S Mane, and Jörg Henkel. "Power-and area-efficient approximate wallace tree multiplier for error-resilient systems". In: *Fifteenth International Symposium on Quality Electronic Design*. IEEE. 2014, pp. 263–269. DOI: https://doi.org/10.1109/ISQED.2014.6783335.

[15] Behzad Boroujerdian, Hussam Amrouch, Jörg Henkel, and Andreas Gerstlauer. "Trading off temperature guardbands via adaptive approximations". In: *International Conference on Computer Design (ICCD)*. IEEE. 2018, pp. 202–209. DOI: 10.1109/ICCD.2018.00039.

[16] Marcelo Brandalero, Antonio Carlos S Beck, Luigi Carro, and Muhammad Shafique. "Approximate on-the-fly coarse-grained reconfigurable acceleration for general-purpose applications". In: *Design Automation Conference (DAC)*. IEEE. 2018, pp. 1–6. DOI: 10.1109/DAC.2018.8465930.

[17] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. "Deep learning with low precision by half-wave gaussian quantization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5918–5926. DOI: 10.1109/CVPR.2017.574.

[18] Michael Carbin, Sasa Misailovic, and Martin C Rinard. "Verifying quantitative reliability for programs that execute on unreliable hardware". In: *ACM SIGPLAN Notices*. Vol. 48. 10. ACM. 2013, pp. 33–52. DOI: 10.1145/2509136.2509546.

[19] Stephen Cass. "Taking AI to the edge: Google's TPU now comes in a maker-friendly package". In: *IEEE Spectrum* 56.5 (2019), pp. 16–17. DOI: 10.1109/MSPEC.2019.8701189.

[20] J. Castro-Godínez, S. Esser, M. Shafique, S. Pagani, and J. Henkel. "Compiler-driven error analysis for designing approximate accelerators". In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018, pp. 1027–1032. DOI: 10.23919/DATE.2018.8342163.

[21] Jorge Castro-Godınez, Sven Esser, Muhammad Shafique, Santiago Pagani, and Jörg Henkel. "Compiler-driven error analysis for designing approximate accelerators". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 1027–1032. DOI: 10.23919/DATE.2018.8342163.

[22] Jorge Castro-Godınez, Julián Mateus-Vargas, Muhammad Shafique, and Jörg Henkel. "AxHLS: design space exploration and high-level synthesis of approximate accelerators using approximate functional units and analytical models". In: *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. 2020, pp. 1–9. DOI: 10.1145/3400302.3415732.

[23] Srimat T Chakradhar and Anand Raghunathan. "Best-effort computing: re-thinking parallel software and hardware". In: *Design Automation Conference (DAC)*. ACM/IEEE. 2010, pp. 865–870. DOI: 10.1145/1837274.1837492.

[24] Yen-Kuang Chen, Jatin Chhugani, Pradeep Dubey, Christopher J Hughes, Daehyun Kim, Sanjeev Kumar, Victor W Lee, Anthony D Nguyen, and Mikhail Smelyanskiy. "Convergence of recognition, mining, and synthesis workloads and its implications". In: *Proceedings of the IEEE* 96.5 (2008), pp. 790–807. DOI: 10.1109/JPROC.2008.917729.

[25] Vinay Chippa, Anand Raghunathan, Kaushik Roy, and Srimat Chakradhar. "Dynamic effort scaling: Managing the quality-efficiency tradeoff". In: *Design Automation Conference (DAC)*. IEEE. 2011, pp. 603–608. DOI: 10.1145/2024724.2024863.

[26] Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. "Analysis and characterization of inherent application resilience for approximate computing". In: *Design Automation Conference (DAC)*. 2013, pp. 1–9.

[27] Vinay Kumar Chippa, Debabrata Mohapatra, Kaushik Roy, Srimat T Chakradhar, and Anand Raghunathan. "Scalable effort hardware design". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.9 (2014), pp. 2004–2016. DOI: 10.1145/1837274.1837411.

[28] Jason Cong and Kirill Minkovich. "Logic synthesis for better than worst-case designs". In: *VLSI Design, Automation and Test, 2009. VLSI-DAT'09. International Symposium on*. IEEE. 2009, pp. 166–169. DOI: 10.1109/VDAT.2009.5158121.

[29] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "Training deep neural networks with low precision multiplications". In: *arXiv preprint arXiv:1412.7024* (2014).

[30] Robert H Dennard, Fritz H Gaensslen, Hwa-Nien Yu, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. "Design of ion-implanted MOSFET's with very small physical dimensions". In: *IEEE Journal of Solid-State Circuits* 9.5 (1974), pp. 256–268. DOI: 10.1109/JSSC.1974.1050511.

[31] *Design Compiler® User Guide*. Synopsys. www.synopsys.com, 2010.

[32] Thomas Ebi, Mohammad Abdullah Al Faruque, and Jörg Henkel. "Tape: Thermal-aware agent-based power econom multi/many-core architectures". In: *2009 IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers*. IEEE. 2009, pp. 302–309. DOI: 10.1145/1687399.1687457.

[33] Thomas Ebi, David Kramer, Wolfgang Karl, and Jörg Henkel. "Economic learning for thermal-aware power budgeting in many-core architectures". In: *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. 2011, pp. 189–196. DOI: 10.1145/2039370.2039401.

[34] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, et al. "Razor: A low-power pipeline based on circuit-level timing speculation". In: *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*. IEEE. 2003, pp. 7–18. DOI: 10.1109/MICRO.2003.1253179.

[35] Rolf Ernst, Jörg Henkel, and Thomas Benner. "Hardware-software cosynthesis for microcontrollers". In: *IEEE Design & Test of computers* 10.4 (1993), pp. 64–75. DOI: 10.1109/54.245964.

[36] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. "Architecture support for disciplined approximate programming". In: *ACM SIGPLAN Notices*. Vol. 47. 4. ACM. 2012, pp. 301–312. DOI: 10.1145/2150976.2151008.

[37] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. "Neural acceleration for general-purpose approximate programs". In: *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE. 2012, pp. 449–460. DOI: 10.1109/MICRO.2012.48.

[38] Brian Greskamp, Lu Wan, Ulya R Karpuzcu, Jeffrey J Cook, Josep Torrellas, Deming Chen, and Craig Zilles. "Blueshift: Designing processors for timing speculation from the ground up." In: *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*. IEEE. 2009, pp. 213–224. DOI: 10.1109/HPCA.2009.4798256.

[39] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. "Deep Learning with Limited Numerical Precision". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1737–1746. URL: http://proceedings.mlr.press/v37/gupta15.html.

[40] T. Han and D. A. Carlson. "Fast area-efficient VLSI adders". In: *Computer Arithmetic (ARITH), 1987 IEEE 8th Symposium on*. May 1987, pp. 49–56. DOI: 10.1109/ARITH.1987.6158699.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[42] Rajamohana Hegde and Naresh R Shanbhag. "Energy-efficient signal processing via algorithmic noise-tolerance". In: *Proceedings of the 1999 international symposium on Low power electronics and design*. ACM. 1999, pp. 30–35. DOI: 10.1145/313817.313834.

[43] Jörg Henkel. "A low power hardware/software partitioning approach for core-based embedded systems". In: *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*. IEEE. 1999, pp. 122–127. DOI: 10.1109/DAC.1999.781296.

[44] Jörg Henkel, Thomas Ebi, Hussam Amrouch, and Heba Khdr. "Thermal management for dependable on-chip systems". In: *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*. IEEE. 2013, pp. 113–118. DOI: 10.1109/ASPDAC.2013.6509582.

[45] Jörg Henkel, Andreas Herkersdorf, Lars Bauer, Thomas Wild, Michael Hübner, Ravi Kumar Pujari, Artjom Grudnitsky, Jan Heisswolf, Aurang Zaib, Benjamin Vogel, et al. "Invasive manycore architectures". In: *17th Asia and South Pacific Design Automation Conference*. IEEE. 2012, pp. 193–200. DOI: 10.1109/ASPDAC.2012.6164944.

[46] Jörg Henkel and Yanbing Li. "Energy-conscious HW/SW-partitioning of embedded systems: A Case Study on an MPEG-2 Encoder". In: *Proceedings of the Sixth International Workshop on Hardware/Software Codesign.(CODES/CASHE'98)*. IEEE. 1998, pp. 23–27. DOI: 10.1109/HSC.1998.666233.

[47] Jörg Henkel, Wayne Wolf, and Srimat Chakradhar. "On-chip networks: A scalable, communication-centric embedded system design paradigm". In: *17th International Conference on VLSI Design. Proceedings.* IEEE. 2004, pp. 845–851. DOI: 10.1109/ICVD.2004.1261037.

[48] Dirk Herrmann, Jörg Henkel, and Rolf Ernst. "An approach to the adaptation of estimated cost parameters in the COSYMA system". In: *Third International Workshop on Hardware/Software Codesign*. IEEE. 1994, pp. 100–107. DOI: 10.1109/HSC.1994.336718.

[49] Henry Hoffmann, Sasa Misailovic, Stelios Sidiroglou, Anant Agarwal, and Martin Rinard. "Using code perforation to improve performance, reduce energy consumption, and respond to failures". In: (2009).

[50] Mark Horowitz. "1.1 computing's energy problem (and what we can do about it)". In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 2014. DOI: 10.1109/ISSCC.2014.6757323.

[51] Chih-Chieh Hsiao, Slo-Li Chu, and Chen-Yu Chen. "Energy-aware hybrid precision selection framework for mobile GPUs". In: *Elsevier Computers & Graphics* 37.5 (2013), pp. 431–444. DOI: 10.1016/j.cag.2013.03.003.

[52] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. "Densely connected convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4700–4708. DOI: 10.1109/CVPR.2017.243.

[53] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Binarized Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf.

[54] Mohsen Imani, Daniel Peroni, and Tajana Rosing. "CFPU: Configurable floating point multiplier for energy-efficient computing". In: *Design Automation Conference (DAC)*. IEEE. 2017, pp. 1–6. DOI: 10.1145/3061639.3062210.

[55] *Invasive Computing (InvasIC)*. Online. (Accessed: 2021-05-19). URL: http://invasic.informatik.uni-erlangen.de/en/index.php.

[56] Animesh Jain, Parker Hill, Shih-Chieh Lin, Muneeb Khan, Md E Haque, Michael A Laurenzano, Scott Mahlke, Lingjia Tang, and Jason Mars. "Concise loads and stores: The case for an asymmetric compute-memory architecture for approximation". In: *International Symposium on Microarchitecture (MICRO)*. IEEE/ACM. 2016, pp. 1–13. DOI: 10.5555/3195638.3195688.

[57] Shubham Jain, Swagath Venkataramani, and Anand Raghunathan. "Approximation through logic isolation for the design of quality configurable circuits". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2016, pp. 612–617. DOI: 10.5555/2971808.2971951.

[58] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. "In-datacenter performance analysis of a tensor processing unit". In: *International Symposium on Computer Architecture (ISCA)*. 2017, pp. 1–12. DOI: 10.1145/3079856.3080246.

[59] Matthias Jung, Deepak M Mathew, Christian Weis, and Norbert Wehn. "Invited-Approximate computing with partially unreliable dynamic random access memory-approximate DRAM". In: *Proceedings of the 53rd Annual Design Automation Conference*. ACM. 2016, p. 100. DOI: 10.1145/2897937.2905002.

[60] Andrew B Kahng and Seokhyeong Kang. "Accuracy-configurable adder for approximate arithmetic designs". In: *Design Automation Conference (DAC)*. ACM. 2012, pp. 820–825. DOI: 10.1145/2228360.2228509.

[61] Andrew B Kahng, Seokhyeong Kang, Rakesh Kumar, and John Sartori. "Slack redistribution for graceful degradation under voltage overscaling". In: *Proc. of the 2010 Asia and South Pacific Design Automation Conference*. 2010, pp. 825–831. DOI: 10.1109/ASPDAC.2010.5419690.

[62] Georgios Karakonstantis, Debabrata Mohapatra, and Kaushik Roy. "System level DSP synthesis using voltage overscaling, unequal error protection & adaptive quality tuning". In: *IEEE Workshop on Signal Processing Systems, 2009. SiPS 2009.* IEEE, pp. 133–138. DOI: 10.1109/SIPS.2009.5336238.

[63] Heba Khdr. "Resource Management for Multicores to Optimize Performance under Temperature and Aging Constraints". PhD thesis. Karlsruher Institut für Technologie (KIT), 2019. 155 pp. DOI: 10.5445/IR/1000095963.

[64] Daya S Khudia, Babak Zamirai, Mehrzad Samadi, and Scott Mahlke. "Rumba: An online quality management system for approximate computing". In: *International Symposium on Computer Architecture (ISCA)*. 2015, pp. 554–566. DOI: 10.1145/2749469.2750371.

[65] Kinam Kim and Jooyoung Lee. "A new investigation of data retention time in truly nanoscaled DRAMs". In: *IEEE Electron Device Letters* 30.8 (2009), pp. 846–848. DOI: 10.1109/LED.2009.2023248.

[66] Younghoon Kim, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. "Designing approximate circuits using clock overgating". In: *Design Automation Conference (DAC)*. ACM/IEEE. 2016, pp. 1–6. DOI: 10.1145/2897937.2898005.

[67] Sebastian Kobbe, Lars Bauer, Daniel Lohmann, Wolfgang Schröder-Preikschat, and Jörg Henkel. "DistRM: Distributed resource management for on-chip many-core systems". In: *2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE. 2011, pp. 119–128. DOI: 10.1145/2039370.2039392.

[68] Peter M Kogge and Harold S Stone. "A parallel algorithm for the efficient solution of a general class of recurrence equations". In: *IEEE Transactions on Computers* 100.8 (1973), pp. 786–793.

[69] Israel Koren. *Computer Arithmetic Algorithms*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993. ISBN: 0-13-151952-2.

[70] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90. DOI: 10.1145/3065386.

[71] Parag Kulkarni, Puneet Gupta, and Milos Ercegovac. "Trading accuracy for power with an underdesigned multiplier architecture". In: *2011 24th Internatioal Conference on VLSI Design*. IEEE. 2011, pp. 346–351. DOI: 10.1109/VLSID.2011.51.

[72] Michael A Laurenzano, Parker Hill, Mehrzad Samadi, Scott Mahlke, Jason Mars, and Lingjia Tang. "Input responsiveness: using canary inputs to dynamically steer approximation". In: ACM, 2016, pp. 161–176. DOI: 10.1145/2908080.2908087.

[73] Seogoo Lee and Andreas Gerstlauer. "Fine grain precision scaling for datapath approximations in digital signal processing systems". In: *International Conference on Very Large Scale Integration-System on a Chip*. IFIP/IEEE. 2013, pp. 266–271. DOI: 10.1109/VLSI-SoC.2013.6673287.

[74] Seogoo Lee, Dongwook Lee, Kyungtae Han, Emily Shriver, Lizy K John, and Andreas Gerstlauer. "Statistical quality modeling of approximate hardware." In: *ISQED*. 2016, pp. 163–168.

[75] Teahyung Lee, Myung Hwangbo, Tanfer Alan, Omesh Tickoo, and Ravishankar Iyer. "Low-complexity hog for efficient video saliency". In: *International Conference on Image Processing (ICIP)*. IEEE. 2015, pp. 3749–3752. DOI: 10.1109/ICIP.2015.7351505.

[76] Teahyung Lee, Myung Hwangbo, Tanfer Alan, Omesh Tickoo, and Ravishankar Iyer. "Method and system of low-complexity histogram of gradients generation for image processing". U.S. pat. US9760794B2. 2017.

[77] Vasileios Leon, Georgios Zervakis, Sotirios Xydis, Dimitrios Soudris, and Kiamal Pekmestzi. "Walking through the energy-error Pareto frontier of approximate multipliers". In: *IEEE Micro* 38.4 (2018), pp. 40–49. DOI: 10.1109/MM.2018.043191124.

[78] Yanbing Li and Jörg Henkel. "A framework for estimation and minimizing energy dissipation of embedded HW/SW systems". In: *Proceedings of the 35th annual Design Automation Conference*. 1998, pp. 188–193.

[79] Xiaofan Lin, Cong Zhao, and Wei Pan. "Towards accurate binary convolutional neural network". In: *Advances in Neural Information Processing Systems*. 2017, pp. 345–353. DOI: 10.5555/3294771.3294804.

[80] Avinash Lingamneni, Christian Enz, Jean-Luc Nagel, Krishna Palem, and Christian Piguet. "Energy parsimonious circuit design through probabilistic pruning". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE. 2011, pp. 1–6. DOI: 10.1109/DATE.2011.5763130.

[81] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. "RAIDR: Retention-aware intelligent DRAM refresh". In: *ACM SIGARCH Computer Architecture News*. Vol. 40. 3. IEEE Computer Society. 2012, pp. 1–12. DOI: 10.1109/ISCA.2012.6237001.

[82] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G Zorn. "Flikker: saving DRAM refresh-power through critical data partitioning". In: *ACM SIGPLAN Notices* 47.4 (2012), pp. 213–224. DOI: 10.1145/1950365.1950391.

[83] Yuxi Liu, Rong Ye, Feng Yuan, Rakesh Kumar, and Qiang Xu. "On logic synthesis for timing speculation". In: *Proceedings of the International Conference on Computer-Aided Design*. ACM. 2012, pp. 591–596. DOI: 10.1145/2429384.2429512.

[84] Jan Lucas, Mauricio Alvarez-Mesa, Michael Andersch, and Ben Juurlink. "Sparkk: Quality-scalable approximate storage in DRAM". In: *The Memory Forum*. 2014, pp. 1–9.

[85] Hamid Reza Mahdiani, Ali Ahmadi, Sied Mehdi Fakhraie, and Caro Lucas. "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 57.4 (2010), pp. 850–862. DOI: 10.1109/TCSI.2009.2027626.

[86] Jin Miao, Andreas Gerstlauer, and Michael Orshansky. "Approximate logic synthesis under general error magnitude and frequency constraints". In: *International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2013, pp. 779–786. DOI: 10.1109/ICCAD.2013.6691202.

[87] Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, and Martin C Rinard. "Chisel: Reliability-and accuracy-aware optimization of approximate computational kernels". In: *ACM SIGPLAN Notices*. Vol. 49. 10. ACM. 2014, pp. 309–328. DOI: 10.1145/2660193.2660231.

[88] Sasa Misailovic, Stelios Sidiroglou, Henry Hoffmann, and Martin Rinard. "Quality of service profiling". In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. 2010, pp. 25–34. DOI: 10.1145/1806799.1806808.

[89] Debabrata Mohapatra, Vinay K Chippa, Anand Raghunathan, and Kaushik Roy. "Design of voltage-scalable meta-functions for approximate computing". In: *DATE*. IEEE. 2011, pp. 1–6. DOI: 10.1109/DATE.2011.5763154.

[90] Bert Moons and Marian Verhelst. "Dvas: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing". In: *International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE. 2015, pp. 237–242. DOI: 10.1109/ISLPED.2015.7273520.

[91] Gordon E Moore. "Cramming more components onto integrated circuits". In: *Electronics* 38.8 (Apr. 1965), p. 114. DOI: 10.1109/N-SSC.2006.4785860.

[92] Thierry Moreau, Mark Wyse, Jacob Nelson, Adrian Sampson, Hadi Esmaeilzadeh, Luis Ceze, and Mark Oskin. "SNNAP: Approximate computing on programmable socs via neural acceleration". In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2015, pp. 603–614. DOI: 10.1109/HPCA.2015.7056066.

[93] Vojtech Mrazek, Zdenek Vasicek, and Lukas Sekanina. "Design of Quality-Configurable Approximate Multipliers Suitable for Dynamic Environment". In: *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE. 2018, pp. 264–271. DOI: 10.1109/AHS.2018.8541479.

[94] *MSP430 Hardware Multiplier*. (Accessed: 2020.05.17). URL: http://www.ti.com/sc/docs/products/micro/msp430/userguid/ag_06.pdf.

[95] Jacob Nelson, Adrian Sampson, and Luis Ceze. *Dense approximate storage in phase-change memory*. ASPLOS Ideas & Perspectives. 2011.

[96] Benjamin Oechslein, Jens Schedel, Jürgen Kleinöder, Lars Bauer, Jörg Henkel, Daniel Lohmann, and Wolfgang Schröder-Preikschat. "OctoPOS: A parallel operating system for invasive computing". In: *Proceedings of the International Workshop on Systems for Future Multi-Core Architectures (SFMA). EuroSys*. Citeseer. 2011, pp. 9–14.

[97] Daniele Jahier Pagliari, Enrico Macii, and Massimo Poncino. "Automated Synthesis of Energy-Efficient Reconfigurable-Precision Circuits". In: *IEEE Access* 7 (2019), pp. 172030–172044. DOI: 10.1109/ACCESS.2019.2956679.

[98] Daniele Jahier Pagliari and Massimo Poncino. "Application-driven synthesis of energy-efficient reconfigurable-precision operators". In: *International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2018, pp. 1–5. DOI: 10.1109/ISCAS.2018.8351232.

[99] Guilherme Paim, Leandro Mateus Giacomini Rocha, Hussam Amrouch, Eduardo Antônio César da Costa, Sergio Bampi, and Jörg Henkel. "A cross-layer gate-level-to-application co-simulation for design space exploration of approximate circuits in HEVC video encoders". In: *IEEE Transactions on Circuits and Systems for Video Technology* 30.10 (2019), pp. 3814–3828. DOI: 10.1109/TCSVT.2019.2945763.

[100] Yale N. Patt and Sanjay J. Patel. *Introduction to Computing Systems: From Bits amp; Gates to C amp; Beyond*. 2nd ed. USA: McGraw-Hill, Inc., 2003. ISBN: 0072467509.

[101] Arnab Raha, Hrishikesh Jayakumar, Soubhagya Sutar, and Vijay Raghunathan. "Quality-aware data allocation in approximate DRAM". In: *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. IEEE Press. 2015, pp. 89–98. DOI: 10.1109/CASES.2015.7324549.

[102] Shankar Ganesh Ramasubramanian, Swagath Venkataramani, Adithya Parandhaman, and Anand Raghunathan. "Relax-and-retime: A methodology for energy-efficient recovery based design". In: *Proceedings of the 50th Annual Design Automation Conference*. ACM. 2013, p. 111. DOI: 10.1145/2463209.2488871.

[103] Ashish Ranjan, Arnab Raha, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. "Aslan: Synthesis of approximate sequential circuits". In: *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association. 2014, p. 364. DOI: 10.7873/DATE.2014.377.

[104] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European conference on computer vision*. Springer. 2016, pp. 525–542. DOI: 10.1145/3429945.

[105] Ton Roosendaal. "Big Buck Bunny". In: *International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH ASIA 2008, Singapore, December 10-13, 2008, Computer Animation Festival*. Ed. by Jinny H. J. Choo. ACM, 2008, p. 62. DOI: 10.1145/1504271.1504321. URL: https://doi.org/10.1145/1504271.1504321.

[106] Mehrzad Samadi, Janghaeng Lee, D Anoushe Jamshidi, Amir Hormati, and Scott Mahlke. "Sage: Self-tuning approximation for graphics engines". In: *MICRO'46 , IEEE/ACM*. 2013. DOI: 10.1145/2540708.2540711.

[107] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. "EnerJ: Approximate data types for safe and general low-power computation". In: *ACM SIGPLAN Notices*. Vol. 46. 6. 2011, pp. 164–174. DOI: 10.1145/1993498.1993518.

[108] Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. "Approximate storage in solid-state memories". In: *ACM Transactions on Computer Systems (TOCS)* 32.3 (2014), pp. 1–23. DOI: 10.1145/2540708.2540712.

[109] Syed Shakib Sarwar, Gopalakrishnan Srinivasan, Bing Han, Parami Wijesinghe, Akhilesh Jaiswal, Priyadarshini Panda, Anand Raghunathan, and Kaushik Roy. "Energy efficient neural computing: A study of cross-layer approximations". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8.4 (2018), pp. 796–809. DOI: 10.1109/JETCAS.2018.2835809.

[110] Ilaria Scarabottolo, Giovanni Ansaloni, and Laura Pozzi. "Circuit carving: A methodology for the design of approximate hardware". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 545–550. DOI: 10.23919/DATE.2018.8342067.

[111] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jörg Henkel. "A low latency generic accuracy configurable adder". In: *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2015, pp. 1–6. DOI: https://doi.org/10.1145/2744769.2744778.

[112] Doochul Shin and Sandeep K Gupta. "Approximate logic synthesis for error tolerant applications". In: *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association. 2010, pp. 957–960. DOI: 10.1109/DATE.2010.5456913.

[113] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. "Managing performance vs. accuracy trade-offs with loop perforation". In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 2011, pp. 124–134. DOI: 10.1145/2025113.2025133.

[114] Leonardo B Soares, Morgana MA da Rosa, Cláudio M Diniz, Eduardo AC da Costa, and Sergio Bampi. "Exploring power-performance-quality tradeoff of approximate adders for energy efficient sobel filtering". In: *2018 IEEE 9th Latin American Symposium on Circuits & Systems (LASCAS)*. IEEE. 2018, pp. 1–4. DOI: 10.1109/LASCAS.2018.8399938.

[115] Leonardo Bandeira Soares, Morgana Macedo Azevedo da Rosa, Cláudio Machado Diniz, Eduardo Antonio César da Costa, and Sergio Bampi. "Design methodology to explore hybrid approximate adders for energy-efficient image and video processing accelerators". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 66.6 (2019), pp. 2137–2150. DOI: 10.1109/TCSI.2019.2892588.

[116] Jinook Song, Yunkyo Cho, Jun-Seok Park, Jun-Woo Jang, Sehwan Lee, Joon-Ho Song, Jae-Gon Lee, and Inyup Kang. "7.1 An 11.5 TOPS/W 1024-MAC butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile SoC". In: *International Solid-State Circuits Conference-(ISSCC)*. IEEE. 2019, pp. 130–132. DOI: 10.1109/ISSCC.2019.8662476.

[117] Phillip Stanley-Marbell and Martin Rinard. "Error-Efficient Computing Systems". In: *Foundations and Trends® in Electronic Design Automation* 11.4 (2017), pp. 362–461. ISSN: 1551-3939. DOI: 10.1561/1000000049. URL: http://dx.doi.org/10.1561/1000000049.

[118] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. "Efficient processing of deep neural networks: A tutorial and survey". In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329. DOI: 10.1109/JPROC.2017.2761740.

[119] Giuseppe Tagliavini, Stefan Mach, Davide Rossi, Andrea Marongiu, and Luca Benini. "A transprecision floating-point platform for ultra-low power computing". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 1051–1056. DOI: 10.23919/DATE.2018.8342167.

[120] Swagath Venkataramani, Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. "Quality programmable vector processors for approximate computing". In: *International Symposium on Microarchitecture (MICRO)*. IEEE/ACM. 2013, pp. 1–12. DOI: 10.1145/2540708.2540710.

[121] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. "AxNN: energy-efficient neuromorphic systems using approximate computing". In: *2014 IEEE/ACM International Symposium on Low Power Electronics and Design*. DOI: 10.1145/2627369.2627613.

[122] Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits". In: *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium. 2013, pp. 1367–1372.

[123] Swagath Venkataramani, Amit Sabne, Vivek Kozhikkottu, Kaushik Roy, and Anand Raghunathan. "SALSA: systematic logic synthesis of approximate circuits". In: *DAC*. ACM. 2012, pp. 796–801. DOI: 10.1145/2228360.2228504.

[124] Ajay K Verma, Philip Brisk, and Paolo Ienne. "Variable latency speculative addition: A new paradigm for arithmetic circuit design". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2008, pp. 1250–1255. DOI: 10.1109/DATE.2008.4484850.

[125] Lu Wan and Deming Chen. "DynaTune: circuit-level optimization for timing speculation considering dynamic path behavior". In: *Proceedings of the 2009 International Conference on Computer-Aided Design*. ACM. 2009, pp. 172–179. DOI: 10.1145/1687399.1687430.

[126] Amir Yazdanbakhsh, Gennady Pekhimenko, Bradley Thwaites, Hadi Esmaeilzadeh, Onur Mutlu, and Todd C Mowry. "RFVP: Rollback-free value prediction with safe-to-approximate loads". In: *ACM Transactions on Architecture and Code Optimization (TACO)* 12.4 (2016), pp. 1–26. DOI: 10.1145/2836168.

[127] Rong Ye, Ting Wang, Feng Yuan, Rakesh Kumar, and Qiang Xu. "On reconfiguration-oriented approximate adder design and its application". In: *International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2013, pp. 48–54. DOI: 10.1109/ICCAD.2013.6691096.

[128] Serif Yesil, Ismail Akturk, and Ulya R Karpuzcu. "Toward Dynamic Precision Scaling". In: *IEEE Micro* 38.4 (2018), pp. 30–39. DOI: 10.1109/MM.2018.043191123.

[129] Georgios Zervakis, Kostas Tsoumanis, Sotirios Xydis, Dimitrios Soudris, and Kiamal Pekmestzi. "Design-efficient approximate multiplication circuits through partial product perforation". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.10 (2016), pp. 3105–3117. DOI: 10.1109/TVLSI.2016.2535398.

[130] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. "Lq-nets: Learned quantization for highly accurate and compact deep neural networks". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 365–382.

[131] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients". In: *arXiv preprint arXiv:1606.06160* (2016).

[132] Ning Zhu, Wang Ling Goh, and Kiat Seng Yeo. "An enhanced low-power high-speed adder for error-tolerant application". In: *ISIC*. IEEE. 2009, pp. 323–327.

[133] Ning Zhu, Wang Ling Goh, Weija Zhang, Kiat Seng Yeo, and Zhi Hui Kong. "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 18.8 (2010), pp. 1225–1229. DOI: 10.1109/SOCDC.2010.5682905.

List of Publications

[4] Tanfer Alan, Jorge Castro-Godinez, and Jörg Henkel. "Multiple Approximate Instances in Neural Processing Units for Energy-Efficient Circuit Synthesis (WiP)". In: *International Conference on Compilers, Architectures and Synthesis For Embedded Systems (CASES)*, (2021). DOI: 10.1145/3451939.3477594.

[5] Tanfer Alan, Andreas Gerstlauer, and Jörg Henkel. "Cross-Layer Approximate Hardware Synthesis for Runtime Configurable Accuracy". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2021). DOI: 10.1109/TVLSI.2021.3068312.

[6] Tanfer Alan, Andreas Gerstlauer, and Jörg Henkel. "Runtime Accuracy-Configurable Approximate Hardware Synthesis Using Logic Gating and Relaxation". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020, pp. 1578–1581. DOI: 10.23919/DATE48585.2020.9116272.

[7] Tanfer Alan and Jörg Henkel. "Probability-Driven Evaluation of Lower-Part Approximation Adders". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* (2021). DOI: 10.1109/TCSII.2021.3093984.

[8] Tanfer Alan and Jörg Henkel. "SlackHammer: Logic Synthesis for Graceful Errors Under Frequency Scaling". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2802–2811. DOI: 10.1109/TCAD.2018.2858364.

[75] Teahyung Lee, Myung Hwangbo, Tanfer Alan, Omesh Tickoo, and Ravishankar Iyer. "Low-complexity hog for efficient video saliency". In: *International Conference on Image Processing (ICIP)*. IEEE. 2015, pp. 3749–3752. DOI: 10.1109/ICIP.2015.7351505.

[76] Teahyung Lee, Myung Hwangbo, Tanfer Alan, Omesh Tickoo, and Ravishankar Iyer. "Method and system of low-complexity histogram of gradients generation for image processing". U.S. pat. US9760794B2. 2017.

List of Figures

2.1 Approximate Computing across the computing stack. 8
2.2 Motivational example: changing accuracy requirement for a fixed application quality target 12
2.3 Edge detection on Big Buck Bunny with exact and approximate Sobel Filters. 13
2.4 Required accuracy for edge detection for different quality targets 14
2.5 Synthesis Optimizations 19
2.6 Gate-level delay optimizations to reduce the most critical path delay [31] 20
2.7 Iterative improvements of the critical path delay. 21
2.8 Comparison of lower-part approximate adder circuits 22
2.9 Error magnitude comparison of lower-part approximate adders. 23
2.10 Area comparisons of lower-part approximate adders. 24
2.11 Power vs. Frequency vs. Accuracy trade-offs of a 32-bit Adder 25

3.1 Logic synthesis for graceful errors (motivational) 28
3.2 Path delays in traditional logic synthesis 29
3.3 Path delays with non-critical path optimizations 29
3.4 Search of minimum feasible delay constraints for primary outputs 32
3.5 SlackHammer synthesis flow 34
3.6 Comparison of synthesis methods in (a) number of near-critical paths (b) Area, and (c) Power. 37
3.7 Error characterization of circuits under frequency scaling. 38
3.8 Circuit delay distributions in comparison. 39
3.9 Non-critical path delay optimizations for cross-layer use with precision scaling . . . 40

4.1 Motivational: dynamic frequency-precision scalable MAC unit in a systolic array . . 43
4.2 Primary output delay comparison of traditional synthesis vs. SlackHammer [8] . . . 45
4.3 Abstracted and circuit-level comparison of high-complexity MSB paths vs. low-complexity LSB paths. 46
4.4 Comparisons of proposed frequency-precision scaling methodology against conventional synthesis targeting a single frequency, in circuit metrics: area, leakage, and dynamic power. 51
4.5 Energy comparison of circuits under dynamic precision scaling. 52

5.1 Power vs. accuracy comparisons of gating an exact circuit against instantiating relaxed, approximate circuits . . . 56
5.2 Background of a runtime accuracy-configurable system at design and run time . . . 57
5.3 Cross-layer accuracy configurable hardware architecture . . . 58
5.4 Power vs. accuracy design space of the cross-layer approach . . . 59
5.5 Cross-layer synthesis flow. . . . 61
5.6 Design space of an accuracy configurable Sobel filter under 3 different utilizations given in Table 5.2 . . . 67
5.7 Pareto front comparison of cross-layer solutions against gating and instantiating solutions from Figure 5.6b. . . . 67
5.8 Design space of an accuracy configurable Sobel Filter with *typical* integration costs (control+MUX+clk) under 3 different utilizations given in Table 5.2 . . . 68
5.9 Pareto curves for a range of circuits under workloads with accuracy utilizations given in Table 5.2. . . . 69
5.10 Energy required for accuracy reconfiguration of the Sobel Filter . . . 70
5.11 Input dependency of cross-layer design space . . . 72
5.12 Leakage impact on accuracy configurable Sobel filters under varying utilization factor with different technology libraries. . . . 73

List of Tables

2.1 Quantization for different neural network learning methods 15

3.1 Circuits used in experiments of Chapter 3 35

4.1 Circuits used in experiments of Chapter 4 49
4.2 Iterative search steps of Algorithm 4. 50

5.1 Circuits used in experiments of Chapter 5 64
5.2 Accuracy utilization distributions of experimented synthetic workloads 66
5.3 Cross-layer energy optimizer run time 66