Automated Design of Approximate Accelerators

Jorge Castro-Godínez

Automated Design of Approximate Accelerators

by
Jorge Castro-Godínez

This work has been supported financially by the Instituto Tecnológico de Costa Rica through the "Costa Rican Higher Education Improvement Project" (Proyecto Costa Rica BIRF 8794-CR), and the DAAD-STIBET-Abschlussstipendium.

TEC | Tecnológico de Costa Rica

Cover designed by the author. The base image corresponds to the *cameraman*, a standard test image in image processing (http://www.imageprocessingplace.com/root_files_V3/image_databases.htm).

Automated Design of Approximate Accelerators

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

M.Sc.

Jorge Castro-Godínez
aus San Isidro de El General,
Costa Rica

Tag der mündlichen Prüfung:	19.11.2020
Erster Gutachter:	Prof. Dr.-Ing. Jörg Henkel Karlsruher Institut für Technologie
Zweiter Gutachter:	Prof. Dr. rer. nat. Wolfgang Karl Karlsruher Institut für Technologie
Dritter Gutachter:	Prof. Dr.-Ing. Muhammad Shafique New York University Abu Dhabi

Jorge Castro-Godínez
Steinstr. 18
77815, Bühl

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.

Jorge Castro-Godínez

To Silvia, Fabiana, Ignacio, and Lucía

People think that computer science is the art of geniuses but the actual reality is the opposite, just many people doing things that build on eachother, like a wall of mini stones.

Donald Knuth

Contents

I've never been a good estimator of how long things are going to take.

Donald Knuth

Acknowledgements

First of all, I express my sincere thanks to Prof. Dr.-Ing. Jörg Henkel for letting me, an unknown guy from Costa Rica with very little research experience, to carry out my doctoral studies at the Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT). Prof. Henkel provided me the freedom to explore and propose my own ideas, and he set the bar very high to help me discover my potential as a researcher. For this, I will ever be deeply thankful.

I deeply thank Prof. Dr.-Ing. Muhammad Shafique for his constant motivation, questions, discussions, and feedback that helped me to shape my ideas and write my papers in such a way that they were accepted. I express my gratitude to Prof. Dr. rer. nat. Wolfgang Karl for accepting the invitation to be one of my reviewers.

I thank Prof. Dr.-Ing. Rüdiger Dillmann and Prof. Dr.-Ing. Gregor Snelting for accepting the invitation to be examiners for my oral defense; and Prof. Dr. Mehdi B. Tahoori and Prof. Dr.-Ing. Michael Beigl for being part of the defense as members of the doctoral committee.

I would like to thank my colleagues at CES, both present and past, for their friendship, support, and counsels; they were always really appreciated. I particularly thank my colleagues with whom I have the chance to share the office (in alphabetical order): Tanfer Alan, Dr.-Ing. Santiago Pagani, and Dr.-Ing. Farzad Samie. The interesting discussions, funny anecdotes, and experiences shared have made my time at CES more pleasant. I also thank Dr.-Ing Lars Bauer for being always reachable to answer questions and for the funny moments.

A special thank goes to all the students that worked with me during this time, in many cases coming from Costa Rica for short research stays. I am particularly indebted to my student co-authors for the wonderful work done and the friendship developed (in alphabetical order): Humberto Barrantes-García, Sven Esser, Deykel Hernández-Araya, Luis G. León-Vega, and Julián Mateus-Vargas.

The possibility of doing this doctorate was originated by the *Instituto Tecnológico de Costa Rica*, through the "Costa Rican Higher Education Improvement Project" (Proyecto Costa Rica BIRF 8794-CR). I am profoundly thankful for the given opportunity and the trust placed in me. In general, I am thankful for the public education system in Costa Rica. Education is one of the greatest social mobility vectors, which has allowed me, the son of a widow and domestic worker, to get to this point. I thank the scholarship committee for help and support with paperwork and questions. In the last year, I received the support of the DAAD-STIBET-Abschlussstipendium, for which a deeply thank the support of Oliver Kaas from the International Scholars & Welcome Office (IScO) at KIT.

My deepest gratitude goes to my family, especially to my kids, that have had an always busy dad for the last years, and to my wife, for her constant support, love, and care. She has been always there to celebrate our achievements and to help me stand during my difficult times. To my extended family, thanks for their never-ending good wishes.

Karlsruhe, 19.11.2020 Jorge Castro-Godínez

The first draft is just you telling yourself the story.

Terry Pratchett

List of Publications

The following publications provided a major contribution to this dissertation:

[Cas+20c] J. Castro-Godínez, J. Mateus-Vargas, M. Shafique, and J. Henkel. "AxHLS: Design Space Exploration and High-Level Synthesis of Approximate Accelerators using Approximate Functional Units and Analytical Models". In: *2020 IEEE/ACM 39th International Conference on Computer-Aided Design (ICCAD)*. 2020. DOI: 10.1145/3400302.3415732

[Cas+21] J. Castro-Godínez, H. Barrantes-García, M. Shafique, and J. Henkel. "AxLS: A Framework for Approximate Logic Synthesis based on Netlist Transformations". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* (2021)

[Her+20] D. Hernández-Araya, J. Castro-Godínez, M. Shafique, and J. Henkel. "AUGER: A Tool for Generating Approximate Arithmetic Circuits". In: *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*. 2020, pp. 1–4. DOI: 10.1109/LASCAS45839.2020.9069045

[CSH19] J. Castro-Godínez, M. Shafique, and J. Henkel. "ECAx: Balancing Error Correction Costs in Approximate Accelerators". In: *ACM Trans. Embed. Comput. Syst.* 18.5s (2019). DOI: 10.1145/3358179

[Cas+18] 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

The following publications provided a minor contribution to this dissertation:

[CSH20b] J. Castro-Godínez, M. Shafique, and J. Henkel. “Towards Quality-Driven Approximate Software Generation for Accurate Hardware: Work-in-Progress”. In: *2020 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*. 2020, pp. 12–14. DOI: `10.1109/CASES51649.2020.9243814`

[Cas+20b] J. Castro-Godínez, D. Hernández-Araya, M. Shafique, and J. Henkel. “Approximate Acceleration for CNN-based Applications on IoT Edge Devices”. In: *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*. 2020, pp. 1–4. DOI: `10.1109/LASCAS45839.2020.9069040`

Other co-authored publications:

[Kha+20] N. Khan, J. Castro-Godínez, S. Xue, J. Henkel, and J. Becker. “Automatic Floorplanning and Standalone Generation of Bitstream-Level IP Cores”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2020). DOI: `10.1109/TVLSI.2020.3023548`

[LCH20] L. G. León-Vega, J. Castro-Godínez, and J. Henkel. “Measuring Traffic Dynamics at the Edge”. In: *International Work Conference on Bioinspired Intelligence (IWOBI)* (2020)

[CCC14] L. Cabrera-Quirós, R. Campos-Gómez, and J. Castro-Godínez. “Critical steps in camera pose estimation: an evaluation using LTI-LIB2 library”. In: *Revista Tecnología en Marcha* (2014), pp. 60–69. DOI: `10.18845/tm.v0i0.1656`

Workshops and Summer Schools:

[Cas+20a] J. Castro-Godínez, H. Barrantes-García, M. Shafique, and J. Henkel. “AxLS: An Open-Source Framework for Netlist Transformation Approximate Logic Synthesis”. In: *3rd Workshop on Open-Source EDA Technology (WOSET), co-located with ICCAD '20*. 2020

[CSH20a] J. Castro-Godínez, M. Shafique, and J. Henkel. “Towards Designing and Implementing Approximate Accelerators”. In: *16th International Summer School on Advanced Computer Architecture and Compilation for High-performance Embedded Systems (ACACES)*. 2020

[Cas19] J. Castro-Godínez. “Approximate Software for Accurate Hardware”. In: *NiPS Summer School, OPRECOMP Summer of Code Initiative*. 2019

[CH18] J. Castro-Godínez and J. Henkel. “Error Propagation Estimation on Approximate Designs with Compiler-Driven Support”. In: *3rd. Workshop on Approximate Computing (AxC ’18), co-located with the IEEE European Test Symposium 2018*. 2018

What is not started today is never finished tomorrow.

Johann Wolfgang von Goethe

Abstract

In the last decade, the need for computing efficiency has motivated the coming forth of new devices, architectures, and design techniques. *Approximate Computing* has emerged as a modern energy-efficient design paradigm for applications that present inherent tolerance to errors. By reducing the accuracy of the results in current applications, such as image processing, computer vision, and machine learning, to an acceptable amount, savings in the circuit area, delay, and power consumption can be achieved.

With the emergence of the approximate computing paradigm, many approximate functional units have been reported in the literature, particularly approximate adders and multipliers. For a plethora of such approximate circuits, and considering their usage as building blocks for the design of approximate accelerators for error-tolerant applications, a challenge arises: selecting those approximate circuits for a given application that minimize the required resources while satisfying a defined accuracy.

This dissertation proposes automated methods for designing and implementing approximate accelerators built with approximate arithmetic circuits. To achieve it, this dissertation addresses the following challenges and provides the subsequent novel contributions:

- Many approximate adders and multipliers have been reported in the literature, either by proposing approximate designs from accurate implementations, such as the ripple-carry adder, or by generating them through Approximate Logic Synthesis (ALS) methods. A representative set of these

approximate components is required to build approximate accelerators. In that sense, this dissertation presents two approaches to generate such approximate arithmetic circuits. First, AUGER is introduced, a tool capable of generating Register-Transfer Level (RTL) descriptions for a broad set of approximate adders and multipliers for different data bit-width and accuracy configuration. A Design Space Exploration (DSE) of approximate components can be performed with AUGER to find those Pareto-optimal for a given bit-width, approximation range, and circuit metric. Then, AxLS is presented, a framework for ALS that allows the implementation of state-of-the-art methods, and the proposition of novel ones, to perform structural netlist transformations and to generate approximate arithmetic circuits from accurate ones. Moreover, both tools provide an error characterization, in the form of error distribution, and circuit characteristics (area, delay, and power) for each approximate circuit they generate. This information is essential for the scope of this dissertation.

- Despite the tolerance to errors, approximate accelerators must be designed to satisfy accuracy constraints. Hence, for the design of such accelerators using approximate arithmetic circuits, it is imperative to assess how the errors introduced by approximate circuits propagate through other computations, either accurate or inaccurate, and finally accumulate at the output. This dissertation proposes analytical models to describe the error propagation through exact and approximate calculations. With them, an automated, compiler-based methodology is proposed to estimate the error propagation on approximate accelerators designs. This methodology is integrated into a tool, CEDA, to perform fast, simulation-free accuracy estimations of approximate accelerator models described using C code.

- In the design of approximate accelerators, repetitive gate-level simulations and circuit synthesis consume a significant time to explore many, or even all, possible combinations for a given set of approximate arithmetic circuits. On the other hand, current trends for designing accelerators are based on High-Level Synthesis (HLS) tools. This dissertation presents analytical models for estimating the required computational resources when using approximate adders and multipliers in approximate accelerators' designs. Furthermore, together with the proposed analytical models for accuracy estimation, these models are integrated into a DSE methodology for error-tolerant applications, DSEwam, to identify Pareto-optimal, or near Pareto-optimal, solutions for approximate accelerators. DSEwam is integrated into an HLS tool to automatically generate RTL descriptions of approximate accelerators from C language descriptions, for a given error threshold and minimization goal.

- The use of approximate accelerators must ensure that errors generated due to approximations remain within a defined maximum value for a given accuracy metric. However, the errors produced by approximate accelerators depend on the input data, which can be different regarding the data used for the design. This dissertation presents ECAx, an automated methodology to explore and apply fine-grained, low-overhead error correction in approximate accelerators, to reduce the cost of error correction at the software level, as reported in the literature. This is performed by selectively correcting the most significant errors produced by approximate components in terms of their magnitude without losing approximations' gains. The experimental evaluation shows speedup improvements for the application in exchange for a small area and power increment in the approximate accelerator design.

Was heute nicht geschieht, ist morgen nicht getan.

Johann Wolfgang von Goethe

Zusammenfassung

In den letzten zehn Jahren hat das Bedürfnis nach Recheneffizienz die Entwicklung neuer Geräte, Architekturen und Entwurfstechniken motiviert. *Approximate Computing* hat sich als modernes, energieeffizientes Entwurfsparadigma für Anwendungen herausgestellt, die eine inhärente Fehlertoleranz aufweisen. Wenn die Genauigkeit der Ergebnisse in aktuellen Anwendungen wie Bildverarbeitung, Computer Vision und maschinellem Lernen auf ein akzeptables Maß reduziert wird, können Einsparungen im Schaltungsbereich, bei der Schaltkreisverzögerung und beim Stromverbrauch erzielt werden.

Mit dem Aufkommen dieses Approximate Computing Paradigmas wurden in der Literatur viele approximierte Funktionseinheiten angegeben, insbesondere approximierte Addierer und Multiplizierer. Für eine Vielzahl solcher approximierter Schaltkreise und unter Berücksichtigung ihrer Verwendung als Bausteine für den Entwurf von approximierten Beschleunigern für fehlertolerante Anwendungen, ergibt sich eine Herausforderung: die Auswahl dieser approximierten Schaltkreise für eine bestimmte Anwendung, die die erforderlichen Ressourcen minimieren und gleichzeitig eine definierte Genauigkeit erfüllen.

Diese Dissertation schlägt automatisierte Methoden zum Entwerfen und Implementieren von approximierten Beschleunigern vor, die aus approximierten arithmetischen Schaltungen aufgebaut sind. Um dies zu erreichen, befasst sich diese Dissertation mit folgenden Herausforderungen und liefert die nachfolgenden neuartigen Beiträge:

- In der Literatur wurden viele approximierte Addierer und Multiplizierer vorgestellt, indem entweder approximierte Entwürfe aus genauen Implementierungen wie dem Ripple-Carry-Addierer vorgeschlagen oder durch Approximate Logic Synthesis (ALS) Methoden generiert wurden. Ein repräsentativer Satz dieser approximierten Komponenten ist erforderlich, um approximierte Beschleuniger zu bauen. In diesem Sinne präsentiert diese Dissertation zwei Ansätze, um solche approximierte arithmetische Schaltungen zu erstellen. Zunächst wird AUGER vorgestellt, ein Tool, mit dem Register-Transfer Level (RTL) Beschreibungen für einen breiten Satz von approximierten Addierern und Multiplizierer für unterschiedliche Datenbitbreiten- und Genauigkeitskonfigurationen generiert werden können. Mit AUGER kann eine Design Space Exploration (DSE) von approximierten Komponenten durchgeführt werden, um diejenigen zu finden, die für eine gegebene Bitbreite, einen gegebenen Approximationsbereich und eine gegebene Schaltungsmetrik Pareto-optimal sind. Anschließend wird AxLS vorgestellt, ein Framework für ALS, das die Implementierung modernster Methoden und den Vorschlag neuartiger Methoden ermöglicht, um strukturelle Netzlistentransformationen durchzuführen und approximierte arithmetische Schaltungen aus genauen Schaltungen zu generieren. Darüber hinaus bieten beide Werkzeuge eine Fehlercharakterisierung in Form einer Fehlerverteilung und Schaltungseigenschaften (Fläche, Schaltkreisverzögerung und Leistung) für jede von ihnen erzeugte approximierte Schaltung. Diese Informationen sind für das Untersuchungsziel dieser Dissertation von wesentlicher Bedeutung.

- Trotz der Fehlertoleranz müssen approximierte Beschleuniger so ausgelegt sein, dass sie Genauigkeitsvorgaben erfüllen. Für den Entwurf solcher Beschleuniger unter Verwendung von approximierten arithmetischen Schaltungen ist es daher unerlässlich zu bewerten, wie sich die durch approximierte Schaltungen verursachten Fehler durch andere Berechnungen ausbreiten, entweder genau oder ungenau, und sich schließlich am Ausgang ansammeln. Diese Dissertation schlägt analytische Modelle vor, um die Fehlerpropagation durch genaue und approximierte Berechnungen zu beschreiben. Mit ihnen wird eine automatisierte, compilerbasierte Methodik vorgeschlagen, um die Fehlerpropagation auf approximierten Beschleunigerdesigns abzuschätzen. Diese Methode ist in ein Tool, CEDA, integriert, um schnelle, simulationsfreie Genauigkeitsschätzungen von approximierten Beschleunigermodellen durchzuführen, die unter Verwendung von C-Code beschrieben wurden.

- Beim Entwurf von approximierten Beschleunigern benötigen sich wiederholende Simulationen auf Gate-Level und die Schaltungssynthese viel Zeit, um viele oder sogar alle möglichen Kombinationen für einen gegebenen Satz von approximierten arithmetischen Schaltungen zu untersuchen. Anderer-

seits basieren aktuelle Trends beim Entwerfen von Beschleunigern auf High-Level Synthesis (HLS) Werkzeugen. In dieser Dissertation werden analytische Modelle zur Schätzung der erforderlichen Rechenressourcen vorgestellt, wenn approximierte Addierer und Multiplizierer in Konstruktionen von approximierten Beschleunigern verwendet werden. Darüber hinaus werden diese Modelle zusammen mit den vorgeschlagenen analytischen Modellen zur Genauigkeitsschätzung in eine DSE-Methodik für fehlertolerante Anwendungen, DSEwam, integriert, um Pareto-optimale oder nahezu Pareto-optimale Lösungen für approximierte Beschleuniger zu identifizieren. DSEwam ist in ein HLS-Tool integriert, um automatisch RTL-Beschreibungen von approximierten Beschleunigern aus C-Sprachbeschreibungen für eine bestimmte Fehlerschwelle und ein bestimmtes Minimierungsziel zu generieren.

- Die Verwendung von approximierten Beschleunigern muss sicherstellen, dass Fehler, die aufgrund von approximierten Berechnungen erzeugt werden, innerhalb eines definierten Maximalwerts für eine gegebene Genauigkeitsmetrik bleiben. Die Fehler, die durch approximierte Beschleuniger erzeugt werden, hängen jedoch von den Eingabedaten ab, die hinsichtlich der für das Design verwendeten Daten unterschiedlich sein können. In dieser Dissertation wird ECAx vorgestellt, eine automatisierte Methode zur Untersuchung und Anwendung feinkörniger Fehlerkorrekturen mit geringem Overhead in approximierten Beschleunigern, um die Kosten für die Fehlerkorrektur auf Softwareebene (wie es in der Literatur gemacht wird) zu senken. Dies erfolgt durch selektive Korrektur der signifikantesten Fehler (in Bezug auf ihre Größenordnung), die von approximierten Komponenten erzeugt werden, ohne die Vorteile der Approximationen zu verlieren. Die experimentelle Auswertung zeigt Beschleunigungsverbesserungen für die Anwendung im Austausch für einen leicht gestiegenen Flächen- und Leistungsverbrauch im approximierten Beschleunigerdesign.

It is better to be vaguely right than exactly wrong.

Carveth Read

1 Introduction

The discontinuation of Dennard scaling and the fading of Moore's law have motivated the coming forth of new devices, architectures, and design techniques for computing [TW17]. Nowadays, power and energy efficiency have become significant design concerns for modern computing systems. On the other hand, current applications and workloads, such as image processing, computer vision, graphics, machine learning, data mining, and financial and physical simulations, are part of a set of applications classified as Recognition, Mining, and Synthesis [Che+08], which have been reported as error-tolerant. This means that even in the presence of deliberately introduced errors, these applications produce acceptable results as a golden result does not exists, the application deals with noisy input data, or even it presents iterative refinement [Ven+15].

1.1 Approximate Computing

In recent years, *Approximate Computing* has emerged as a novel energy- and latency-efficient design paradigm relevant to applications with inherent resilience to errors [HO13; XMK16]. By accepting *good enough* results caused by imprecise calculations, for instance, in image and video processing where the human perception plays a major role, the computational *quality* (accuracy of results) is traded-off to reduce the required computational *effort* (execution time, area, power, or energy). As depicted in Figure 1.1, this tolerance to inexactness has been exploited at different abstraction layers [Sta+20]. For instance, non-critical computations are skipped at the software level, architectural changes are introduced in arithmetic blocks at the architecture level, or operation voltage is lowered at the circuit level [XMK16]. Many more

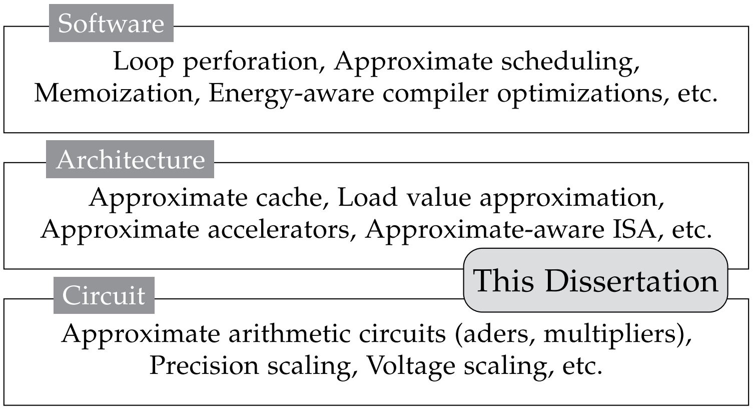

Software
Loop perforation, Approximate scheduling, Memoization, Energy-aware compiler optimizations, etc.
Architecture
Approximate cache, Load value approximation, Approximate accelerators, Approximate-aware ISA, etc.
This Dissertation
Circuit
Approximate arithmetic circuits (aders, multipliers), Precision scaling, Voltage scaling, etc.

Figure 1.1: Approximate computing at different abstraction layers. Different approximate computing techniques have been proposed and applied at different abstraction layers (based on [LLS20]).

approximate computing techniques have been reported in the literature than those in Figure 1.1. Still, these show the need to exploit this design paradigm to improve computation efficiency at all layers where computation occurs.

Recent work has proposed to exploit inherent resilience to errors in applications by using approximate accelerators [XMK16]. In general, hardware accelerators have reported significant benefits for reducing energy consumption, and they have been used to overcome the utilization wall challenge [Con+14]. In a nutshell, an accelerator is used to offload a highly-frequent and compute-intensive section of an application to dedicated hardware, while a host processor executes the rest of the application. Accelerators can be in the form of a GPU, a DSP, or a specialized FPGA design. From the approximate computing perspective, approximate accelerators exploit error resilience as frequently-executed, but error-tolerant sections of an application are performed by dedicated approximate hardware designs [Sha+16; Esm+12].

One approach proposed is to implement these accelerators as neural networks and take advantage of the approximate nature of the results produced by this computational model [Esm+12; Mor+15]. Another approach proposes the usage of approximate arithmetic circuits to replace exact calculations in hardware accelerator designs [Maz+16; Sha+16]. Nevertheless, many approximate adders [Mah+10; Gup+13; Sha+15] and multipliers [Mah+10; BMH14; HBR15; Zen+17] have been reported in the literature. For an ongoing number of such approximate arithmetic circuits, and considering their usage in building approximate designs, such as approximate accelerators, a question arises: *given a design for an error-tolerant ap-*

plication and a set of approximate components, which approximate arithmetic circuits should be used to minimize the computational effort, for instance, the required area, delay, power, or energy, while satisfying a defined accuracy? Traditional approaches required exhaustive synthesis and simulation of by-hand designed approximate accelerators, which might be infeasible due to the large design space even considering a reduced set of approximate arithmetic circuits. For instance, to satisfy accuracy constraints in these accelerators, required to guarantee *good enough* results despite the on-purpose errors, it is imperative to assess how the errors introduced by approximate circuits propagate through other exact and approximate computations, and finally accumulate at the output. This is, in particular, crucial to enable the high-level synthesis of approximate accelerators.

Bridging the gap between many approximate arithmetic circuits and the automated design and implementation of approximate accelerators is crucial to further enable cross-layer approximate computing [Sha+16].

1.2 Dissertation Contribution

This dissertation proposes the automated design of approximate accelerators using approximate arithmetic circuits. As depicted in Figure 1.1, this dissertation can be placed between the circuit and architecture layers, but from the computer-aided design perspective. The contributions of this dissertation, and their relationship, are depicted in Figure 1.2.

Approximate Arithmetic Circuits

In the literature, many approximate adders and multipliers have been reported. To automate the design of approximate accelerators, a representative set of these approximate components is required. In Chapter 3, this dissertation presents two approaches to generate such approximate arithmetic circuits. First, AUGER is introduced, a tool capable of generating Register-Transfer Level (RTL) descriptions for a broad set of approximate adders and multipliers for different data bit-width and accuracy configuration. A Design Space Exploration (DSE) of approximate components can be performed with AUGER to find those Pareto-optimal for a given bit-width, approximation range, and circuit metric. Then, AxLS is presented, a framework for Approximate Logic Synthesis that allows the implementation of state-of-the-art methods, and the proposition of novel ones, to perform structural netlist transformations and to generate approximate arithmetic circuits from accurate ones. Moreover, both tools provide an error characterization, in the form

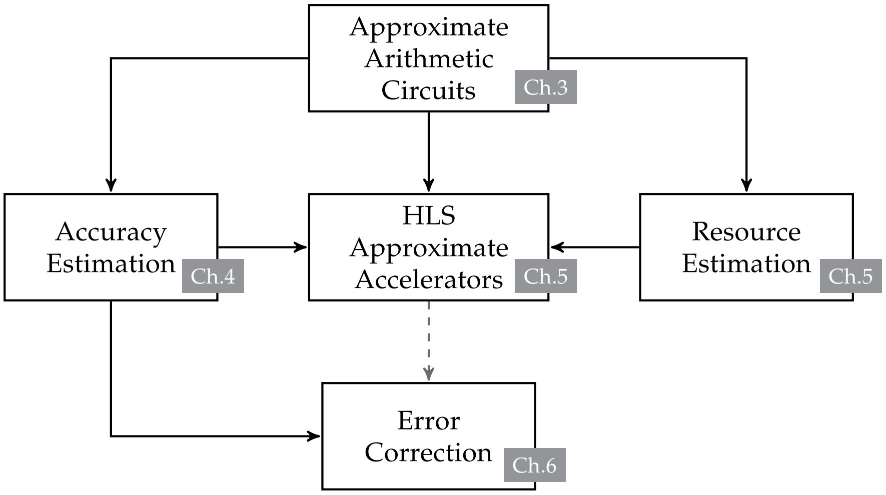


Figure 1.2: Contributions of this dissertation. The chapter number is indicated where each contribution is presented in this dissertation.

of error distribution, and circuit characteristics (area, delay, and power) for each approximate circuit they generate. This information is required to define analytical models for accuracy and resource estimation presented within this dissertation, as those models are based on the information of the approximate arithmetic circuits.

Accuracy Estimation

For the automated design of approximate accelerators built with approximate arithmetic circuits, it is essential to count with models to estimate the accuracy of approximate accelerator designs. Despite the tolerance to errors, approximate accelerators must limit the output error to satisfy accuracy constraints. In Chapter 4, this dissertation proposes analytical models to describe the error propagation through exact and approximate computations, considering approximate additions and multiplications as source of errors. These models are defined with the individual error characteristics of approximate arithmetic circuits, as provided by the contribution in Chapter 3. With these analytical models, an automated, compiler-based methodology is proposed to estimate the error propagation of approximate accelerators models described at the software level.

Resource Estimation

In the design of approximate accelerators, repetitive gate-level simulations and circuit synthesis consume a significant time to explore many, or even all, possible combinations for a given set of approximate arithmetic circuits. To speed up the exploration of many approximate accelerator designs, this dissertation presents resource estimation models in Chapter 5. As the accuracy models, these models are derived using the information of approximate circuits, adders and multipliers, provided by the contribution in Chapter 3.

High-Level Synthesis of Approximate Accelerators

With the generation of approximate arithmetic circuits and analytical models to estimate accuracy and resources in approximate accelerator designs, in Chapter 5, this dissertation presents a high-level synthesis approach to automatically generate RTL descriptions of approximate accelerators from C language descriptions. This is performed for a given error threshold and minimization goal, and using approximate adders and multipliers. To achieve it, a DSE methodology is proposed that takes advantage of the analytical models for accuracy and resource presented in this dissertation, to find Pareto-optimal, or near Pareto-optimal, solutions for approximate accelerators.

Error Correction

The use of approximate accelerators must ensure that errors generated due to approximations remain within a defined maximum value for a given accuracy metric. In Chapter 6, this dissertation presents an automated methodology to explore and apply fine-grained, low-overhead error correction in approximate accelerators. These accelerators can be automatically generated, as proposed in this dissertation or designed by hand. The main idea behind this error correction methodology is to reduce the cost of error correction at the software level, as currently reported in the literature. This is performed by selectively correcting the most significant errors produced by approximate circuits, in terms of their magnitude, without losing the gains of approximations. For this contribution, the analytical models for accuracy estimation presented in Chapter 4 are crucial to enable a fast exploration of error correction's effect in the output accuracy of approximate accelerator designs.

Man muss viel gelernt haben, um über das, was man nicht weiß, fragen zu können.

Jean-Jacques Rousseau

2 Background and Related Work

This chapter presents common principles, notions, and definitions used throughout this dissertation.

2.1 Approximate Arithmetic Circuits

As discussed in Chapter 1, with the coming forth of *Approximate Computing* (AxC) as an energy- and accuracy-aware design paradigm, many approximate arithmetic circuits have been proposed, mainly approximate adders [VBI08; Mah+10; KK12; Sha+15] and multipliers [BMH14; HBR15]. A quick search in the Scopus database[1] shows more than 1400 publications related to AxC in the last decade, and about 37% of those works correspond to approximate adders or multipliers. The main idea behind this approximate circuits is to perform the mathematical operations faster or with less area, power, or energy than the accurate circuits while introducing errors in the results.

In the literature, several methods have been proposed to generate approximate circuits from accurate descriptions, for instance, by transforming gate-level representations of the circuit [Ven+12; HMV16; Zer+16; SAP18; Cas+20a] or exploiting delay in non-critical circuit paths [AH18]. However, the dominant trend has been to propose approximate designs directly from accurate counterparts [Jia+17a].

[1] The search was performed on September 22th, 2020.

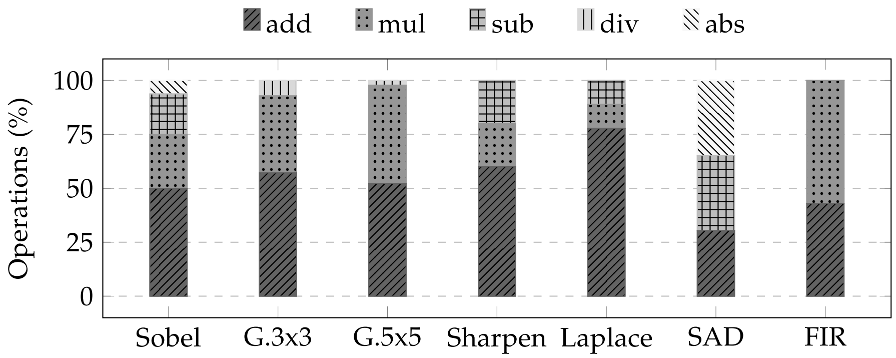


Figure 2.1: Operations profile for a set of error-tolerant applications. For most of these applications, additions and multiplications are the dominant operations.

Other approximate arithmetic circuits have been proposed, for example, approximate dividers [HBR16]. However, this dissertation focuses on approximate adders and multipliers to built approximate accelerators, as they are the most common operations in a wide range of applications [VBI08]. For instance, consider the following error-tolerant applications: Sobel filter, 3×3 and 5×5 Gaussian filter (G.3×3 and G.5×5), Sharpen filter, Laplace filter, sum of absolute differences (SAD), and a 5-tap FIR filter. Figure 2.1 presents a profile of the required mathematical operations for these applications. As it can be noticed, in most of these applications, excluding SAD, 75% or more of the operations correspond to additions and multiplications, and, for instance, none have division as a significant operation.

As many approximate arithmetic circuits have been proposed, and are still being proposed in the community, this chapter does not cover them extensively. Although the main idea is to reduce the complexity of the arithmetic circuits, some of the key concepts behind these approximate units are here mentioned. Extensive evaluations of approximate arithmetic circuits have already been reported in the literature [JHL; Jia+16; Jia+17b; Jia+19].

Approximate Adders: In the literature, two main approaches have been proposed regarding approximate adders. The first considers the substitution of 1-bit full adder (FA) for simplified versions [Mah+10; Gup+13; Yan+13; Dut+16; AKL16], aiming to reduce the power consumption of the addition. This type is known as low-power (LP) approximate adder. For instance, Figure 2.2 depicts an 8-bit approximate adder built from a Ripple-Carry Adder (RCA). This adder, named Lower-part-OR adder (LOA), replaces the 1-bit additions of least significant bits

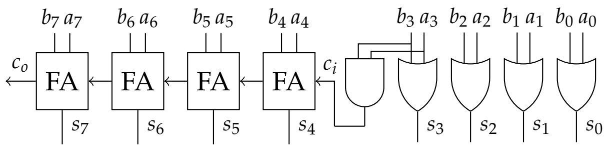


Figure 2.2: An 8-bit LOA approximate adder. This approximate adder, called Lower-part-OR adder (LOA), approximates the addition by using an OR gate to add the LSB [Mah+10].

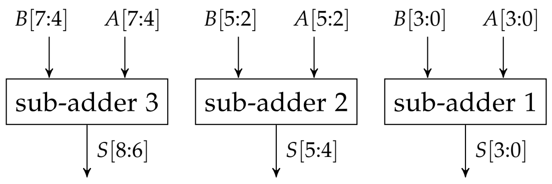


Figure 2.3: An 8-bit GeAr approximate adder. An 8-bit approximate adder, Generic Accuracy Configurable adder (GeAr), uses multiple overlapping sub-adders to perform the addition [Sha+15].

(LSB) performed by a FA with a single OR gate. By doing so, for instance, the required area is reduced as fewer logic gates are needed. Also, the circuit delay is reduced, as the carry propagation chain is cut. For this LOA example, area and power delay are reduced by about 35%, respectively, compared to an 8-bit RCA.[2]

The second type of approximate adder is known as high-performance (HP) approximate adder. For this approximate adders, the addition is computed by breaking the carry propagation chain of the exact addition and using multiple (sometimes overlapping) sub-adders to generate the addition result, aiming to reduce the latency of the computation. Figure 2.3 depicts an example of an HP approximate adder. This adder, called Generic Accuracy Configurable adder (GeAr), performs an N-bit addition using multiple sub-adders of smaller size. In this example, an 8-bit addition is done by three 4-bit adders. The most significant R-bits of the sub-adders are considered as resultant bits, and they are used in the actual result. The remaining P-bits, known as previous bits, are used to estimate the carry propagation to the upper bits. As shown in Figure 2.3, only the sub-adder 1 contributes with all its partial result to the final result, while sub-adder 2 and 3 provide 2 and 3 bits to the result. This 8-bit GeAr example reduces the delay in 40% with respect to an 8-bit RCA.[3]

[2] Considering a synthesis performed with Synopsys Design Compiler and the NanGate 15 nm technology library.
[3] Same synthesis tool and technology library as before.

Most of the reported HP approximate adders in the literature, such as ACA-I [VBI08], ACA-II [KK12], ETAII [ZGY09], and GDA [Ye+13], can be implemented as a configuration of GeAr [Sha+15]. This helps to reduce the representation of various HP approximate adders to a single proposed design.

Approximate Multipliers: For approximate multipliers, the technique applied to reduce their complexity depends on the topology of the accurate multiplier used as a base. For instance, carry-in predictions can be made to reduce latency in a Wallace Tree Multiplier [BMH14], dynamic and fast bit selection can be applied to reduce the size of the multiplier [HBR15], or rounding of operands performed to the nearest exponent of two [Zen+17]. Also, partial product perforations have been applied to accurate multipliers [Zer+16], approximate multipliers have been built from smaller approximate 2x2 multipliers [Reh+16], approximate 4-2 compressors have been proposed to design approximate Dadda multipliers [Mom+15]. Even the LOA LP approximate adder has been used to present approximate versions of array-based multipliers [Mah+10].

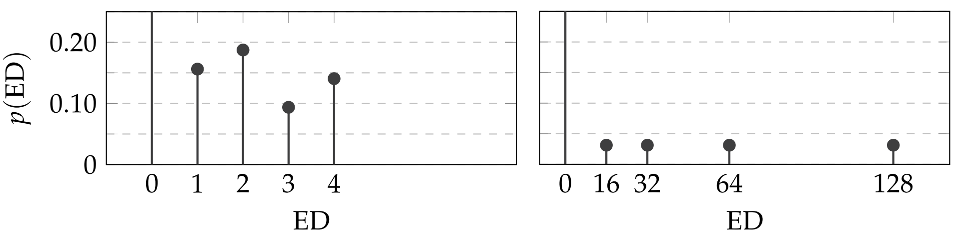
2.2 Approximation Error

As part of the design of approximate accelerators, the error generated due to approximate arithmetic circuits needs to be represented and measured. This is particularly useful, as the use of approximations in error-tolerant applications is limited to allow the application to produce results with the sufficient accuracy required.

2.2.1 Representation

In this dissertation, a Probability Mass Function (PMF) is the main form used to represent the error distribution of approximate circuits and accelerators. A PMF denotes the probability $\mathbf{P}$ of a discrete random variable X to be equal to a determined value x. This is expressed as $p_X(x) = \mathbf{P}(X = x)$. For the case of approximate circuits and accelerators, a PMF allows the representation of each Error Distance, ED, and its probability of ocurrance, p. The ED is defined as:

$$\mathrm{ED} = |O_i^{ac} - O_i^{ax}| \tag{2.1}$$



(a) LOA, ER = 0.581 **(b)** GeAr, ER = 0.125

Figure 2.4: Error distribution for two 8-bit approximate adders. Errors in LP approximate adders appear with low ED but higher frequency in comparison to HP approximate adders, where adders have a higher ED but with lower frequency. These two 8-bit approximate adders correspond to LOA [Mah+10] and GeAr [Sha+15], previously described.

which is the arithmetic distance between the result generated by an accurate (O^{ac}) and an approximate (O^{ax}) circuit (or accelerator) for identical input data (i) [LHL13].

Figure 2.4 presents two PMFs representing the error distribution of two approximate adders. As can be noticed, errors in LP approximate adders, like the LOA, appear with low ED values and a higher probability of occurrence than the errors for an HP approximate adder, like the GeAr, for which errors have a higher ED but with lower probability. Interestingly, the errors for an HP approximate adder correspond to power values of 2, or combinations of them, due to errors are produced by carry-in mispredictions, as the original adder carry chain is broken.

The error distribution of any approximate arithmetic circuit can be estimated through circuit simulations and using representative input data. However, for some of these circuits, their error distribution can be generated through analytical methods [Maz+17b; Maz+17a; Wu+19; Han+20], eliminating the need of time-consuming simulations.

For the case of approximate accelerators, this dissertation proposes a methodology to estimate their error distribution (Chapter 4). Using only the error distributions of individual approximate arithmetic circuits used in an approximate accelerator design, analytical rules are presented to model the interaction of errors generated with other accurate and approximate computations and obtain an output error distribution.

2.2.2 Metrics

An error metric measures how much the results an approximate circuit or accelerator deviates from the accurate ones. Different error metrics have been reported and used in the AxC literature [LHL13; Mra+17; Vas19], which can be derived from the error distribution represented as a PMF. While each of them presents challenges and advantages, three error metrics are widely used for the scope of this dissertation.

The first is Error Rate (ER), which denotes the total probability of error for an approximate circuit or accelerator. As depicted in Figure 2.4, the values in the x-axis, the ED, represent the error values produced by each approximate adder. The probability of having no errors, i.e., $p_X(0)$, is $1 - \text{ER}$. For both PMFs in Figure 2.4, the probability of no having errors is depicted out of the axis. Also, in Figure 2.4, for the LP approximate adder case, it can be observed that it presents a higher ER, 0.581, compared to the HP approximate adder, 0.125.

A second error metric is Worst-Case Error (WCE). WCE represents the maximum error generated by an approximate circuit or accelerator, despite its probability of occurrence, and it is defined as:

$$\text{WCE} = \max_{\forall i} |O_i^{ac} - O_i^{ax}| \tag{2.2}$$

From the PMFs in Figure 2.4, WCE = 4 and WCE = 128 for the LP and HP approximate adders, respectively. A third error metric used widely in this dissertation is Mean-Error Distance (MED). MED is the weighted average of all ED [LHL13]:

$$\text{MED} = \sum_{i=0}^{n} \text{ED}_i \cdot p(\text{ED}_i) \tag{2.3}$$

which can be calculated from the information in the PMF. For the approximate adders in Figure 2.4, the LP and HP approximate adders have 1.37 and 7.50 for this MED metric, respectively.

2.2.3 Accuracy and Quality

Accuracy and quality are two important terms that have been used indistinctly equal in AxC contributions reported in the literature. For this dissertation, the accuracy of an approximate circuit or accelerator is measured with error metrics, as the accuracy is considered not application-dependent. On the other hand, quality is an application-dependent evaluation of an approximate accelerator’s results, and for it, specific quality metrics are required. In this dissertation, the automated design of approximate accelerators is performed for given accuracy constraints.

Still, quality evaluations take place to assess the degradation of the results at the application level.

In this dissertation, three quality metrics are used. Mean Squared Error (MSE) is one of them. MSE is a way to assess the fidelity of a signal, and it is defined as [WB09]:

$$\mathrm{MSE}(\mathbf{ac}, \mathbf{ax}) = \frac{1}{N} \sum_{i=0}^{N} (O_i^{ac} - O_i^{ax})^2 \tag{2.4}$$

where **ac** and **ax** are sets containing all accurate and approximate results, respectively, for the same input data set. With MSE, a degree of similarity or error/distortion between two signals can be assessed.

In the context of image processing, the Peak-Signal-to-Noise Ratio (PSNR) is widely used, which is a converted version of the MSE. PSNR is defined as [WB09]:

$$\mathrm{PSNR} = 10 \log_{10} \frac{L^2}{\mathrm{MSE}} \tag{2.5}$$

where L is the dynamic range of the value a pixel can take, in the context of image processing. For the case of a grayscale image and using 8-bits for representation, that means $L = 255$.

Also related to image processing, in this dissertation, the Structural Similarity (SSIM) index is used as an image fidelity metric [Wan+04]. The SSIM aims to represent better the way the human visual system evaluates the quality in an image, and it is defined as:

$$\begin{aligned} \mathrm{SSIM} &= l(\mathbf{ac}, \mathbf{ax}) \cdot c(\mathbf{ac}, \mathbf{ac}) \cdot s(\mathbf{ac}, \mathbf{ax}) \\ &= \left(\frac{2\mu_{ac}\mu_{ax} + C_1}{\mu_{ac}^2 + \mu_{ax}^2 + C_1} \right) \cdot \left(\frac{2\sigma_{ac}\sigma_{ax} + C_2}{\sigma_{ac}^2 + \sigma_{ax}^2 + C_2} \right) \cdot \left(\frac{\sigma_{ac,ax} + C_3}{\sigma_{ac} + \sigma_{ax} + C_3} \right) \end{aligned} \tag{2.6}$$

where $l(\mathbf{ac}, \mathbf{ax})$ is the similarity of local patch luminances (brightness values), $c(\mathbf{ac}, \mathbf{ac})$ is the similarity of local path contrast, and $s(\mathbf{ac}, \mathbf{ax})$ is the similarity of local patch structures [WB09]. These local similarities are expressed in terms of statistics that are computed to obtain a local SSIM evaluation.

One challenge faced by AxC is that the errors produced by an approximate accelerator not only depend on the approximations applied but also in the data that is processed [Mah+16]. For instance, an approximate accelerator designed to meet a

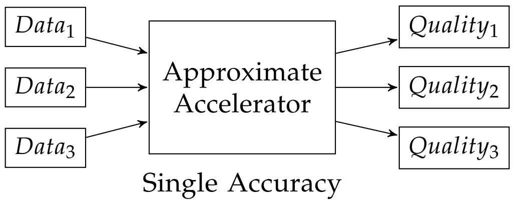


Figure 2.5: Quality as a function of input data. In approximate computing, the quality depends on the data processed and the quality metric used.

Table 2.1: Quality results for a single accuracy Sobel approximate accelerator and different data set. Even though the quality evaluated with PSNR metric is similar for these four test images, the quality estimated with SSIM produce different results.

Data set	MED	PSNR (dB)	SSIM
sails	14.23	22.92	0.86
plate	15.05	22.46	0.65
cameraman	16.92	21.53	0.61
lena	16.30	22.00	0.59

specific, single accuracy can produce results with different quality of results for different data processed, as motivated in Figure 2.5.

Consider a Sobel filter designed to meet a MED = 15 as an accuracy constraint. Four different test images are used as test data with this accelerator. As shown in Table 2.1, just for one of the test images, *sails*, the accuracy threshold is met. On the other hand, for all test images, the quality is similar when evaluated with the PSNR metric, approximately 22 dB. However, when the SSIM index is used for assessing the quality of the results, very different results are obtained, ranging from 0.86 to 0.59 for the *sails* and *lena* test images, respectively. These results confirm that accuracy and quality in AxC are two related but different aspects.

2.3 Approximate Accelerators

By definition, hardware acceleration is the use of specialized hardware to perform tasks more efficiently compared to a general processor [Con+14]. In that sense, an accelerator is used to offload a highly-frequent and compute-intensive section of an application to dedicated hardware, while the rest of the application is executed

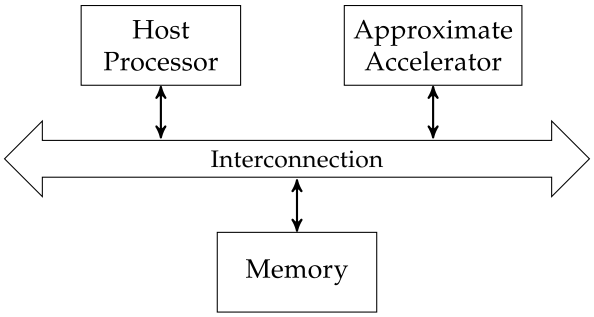


Figure 2.6: Simplified diagram of a system architecture. This diagram considers that a host processor connects to one (or more) approximate accelerator.

by a host processor, aiming to reduce the execution time of the application and increasing the throughput, and in many cases, reducing the energy consumption. Accelerators can be in the form of a GPU, a DSP, or a specialized design on an FPGA.

From the point of view of AxC, approximate accelerators exploit error resilience as frequently-executed, but error-tolerant sections of an application are executed by dedicated approximate hardware designs [Sha+16; Esm+12]. By doing so, additional benefits can be achieved, such as application speedup improvement or energy reduction. Figure 2.6 shows a simplified diagram of a system architecture depicting a host processor and an approximate accelerator interconnected through a standard on-chip interface alongside a memory system. In this sense, the approximate accelerator is designed to execute error-resilient parts of the host processor's applications. Such an approximate accelerator is built to achieve a defined accuracy during design time. Although just one approximate accelerator is shown, many could be designed and deployed in a system. It is even foreseen the development of multi-accelerator platforms, where a diverse set of accelerators with different accuracy is available to a group of processors, providing high flexibility and adaptivity and allowing applications to meet their accuracy and quality constraints while minimizing the required resources [Sha+16].

Existing work has proposed the design of approximate accelerators using neural networks [Esm+12] or approximate functional units, particularly approximate adders and multipliers [Sha+16; Maz+16]. This dissertation focuses on the automated design of approximate accelerators using approximate arithmetic circuits.

As many approximate adders have been proposed considering an RCA as a base, a skeptical observation could point to better use parallel-prefix adders that present smaller delays compared to RCA, if the goal is to reduce the delay of an accelerator

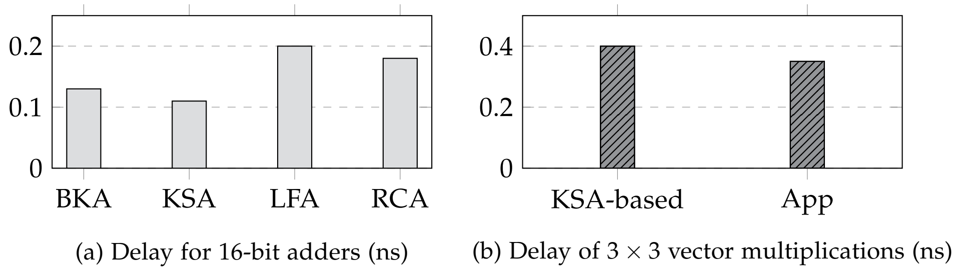


Figure 2.7: Comparison of parallel-prefix and approximate adders. Delay comparison of parallel-prefix adders and RCA-based approximate adders, individually and in approximate accelerator designs.

by using approximations. As can shown in Figure 2.7, parallel-prefix adders such as Brent-Kung Adder (BKA) and Kogge-Stone Adder (KSA) present delays 38% and 63% smaller than an RCA, respectively. Just for the case of Ladner-Fischer Adder (LFA), the RCA presents a delay smaller, about 10%.[4]

However, combinations of state-of-the-art LP and HP approximate adders can produce smaller delays than using these parallel-prefix adders when building approximate accelerators. As depicted in Figure 2.7, using LP and HP approximate adders, such as LOA and GeAr, a delay nearly 15% smaller is obtained by an approximate version (App) of a 3×3 vector multiplication compared to an accelerator built using KSA (KSA-based). On the other hand, the approximate version for this evaluation consumes about 35% less power.

[4] Evaluation performed using 10^6 uniform distributed input test vectors and Synopsys Design Compiler with the TSMC 65nm technology library.

$3 * 3 = 7$

Result of a 2x2-bit approximate multiplier to keep the result to 3 bits [KGE11].

3 Generating Approximate Arithmetic Circuits

The goal of this dissertation is the automated design of approximate accelerators, particularly using approximate arithmetic circuits as building blocks. To achieve it, first, it is necessary to have these approximate arithmetic circuits to then proceed with the design of accelerators.

This chapter presents two contributions for the generation of approximate arithmetic circuits. First, AUGER, a tool to generate and characterize approximate arithmetic circuits reported in the literature, is presented. This tool provides register-transfer level (RTL) implementations and functional models, at the software level, of approximate adders and multipliers. This is performed for different proposed designs of approximate adders and multipliers, and different approximation levels. It also provides a characterization of the circuits generated in terms of area, delay, and power, for a given technology library.

Second, AxLS, a framework for approximate logic synthesis (ALS) techniques based on netlist transformations, is presented. AxLS is a novel, open-source framework with which existing ALS methodologies can be implemented, and new techniques can be proposed. With AxLS, automated methods can be developed to automatically

The content of this chapter is based on the work originally published in [Her+20]:
D. Hernández-Araya, J. Castro-Godínez, M. Shafique, and J. Henkel: "AUGER: A Tool for Generating Approximate Arithmetic Circuits". In: *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*,
and [Cas+20a]:
J. Castro-Godínez, H. Barrantes-García, M. Shafique, and J. Henkel: "AxLS: An Open-Source Framework for Netlist Transformation Approximate Logic Synthesis", In: *3rd Workshop on Open-Source EDA Technology (WOSET)*, 2020.

generate approximate versions of accurate arithmetic circuits described at RTL. Additionally, AxLS performs a characterization of the circuit resources and error distribution of the circuits generated.

Both contributions have been made open-source, allowing their utilization by the research community and their extension to newer approximate arithmetic circuits and newer ALS techniques based on netlist transformations.

3.1 A Tool for Generating Approximate Arithmetic Circuits

As discussed in Section 2.1, an increasing number of approximate arithmetic circuits has been reported in the literature, mainly approximate adders and multipliers. Selecting those that properly fit an application, reducing resources, and achieving a specific accuracy constraint is cumbersome. Comparisons of state-of-the-art approximate adders and multipliers have been reported [JHL; Jia+16; Jia+17b], but if such approximate circuits are expected to be used in approximate designs, they must be first implemented. Additionally, the characterization of such approximate arithmetic circuits, in terms of the resources required and error generated, can speed up the design space exploration of approximate accelerators. This is because the impact in savings and accuracy degradation can be faster estimated than performing iterative circuit synthesis and simulations [Cas+18; Cas+20c], as it is presented in Chapters 4 and 5 of this dissertation.

This section presents AUGER, an Approximate Units GenERator. AUGER[1] is a tool to generate approximate adders and multipliers that have been proposed and reported in academia. AUGER generates the register-transfer level (RTL) descriptions of the circuits. It provides a software model for the arithmetic circuit generated, which can be used to assess its impact on the application accuracy or quality of results at the software level. AUGER uses industrial circuit simulation and synthesis tools to provide a characterization of the components generated in terms of area, power, delay, and error distribution. This information is useful for comparison among other similar approximate circuits, as presented in this section.

[1] AUGER is an open-source contribution and it is available at `https://git.scc.kit.edu/CES/AUGER`.

3.1.1 Description

Figure 3.1 depicts AUGER and its interaction with commercial tools to synthesize and simulate circuits to characterize each circuit generated. AUGER receives the design for one of the supported approximate arithmetic circuits to be generated (see ❶ in Figure 3.1). This implies the type of component, defined by its name, and parameters, such as bit-width and approximation level, according to the approximate circuit design. For instance, for approximate adders, this can include the number of approximate less significant bits (LSB) or bit-width of the sub-adders. Due to approximate arithmetic circuits can be built with different configurations, they can present different accuracy results and savings.

Using the design characteristics, AUGER employs a set of predefined templates (see ❷) to generate: the corresponding RTL description in Verilog HDL (.v), a software model of the component in C language (.c), a set of 10^6 random uniformly distributed input vectors (.txt), generated accordingly to the input bit-width, and a set of scripts to perform synthesis and simulations (.tcl) for the generated approximate circuit (see ❸). A simple formal verification of the design is performed using the results of the RTL simulation (see ❹) and the internal execution of the generated software model. This allows guaranteeing that the design generated corresponds to the given parameters, but also, the software model can be used to carry out accelerator design explorations using high-level models.

For the scope of AUGER, ModelSim is used for circuit simulations, both at pre-synthesis and post-synthesis level. The results of the verification are used to form an error distribution of the approximate arithmetic circuit generated, esentially for those approximate circuits that no analytical model has been proposed to obtain their error distribution [Maz+17b; Maz+17a; Wu+19]. Figure 3.2 shows two probability mass function (PMF) to represent the error distribution (see Section 2.2) for two different 8-bit approximate adders generated by AUGER. As described in Section 2.2, from this distribution, different error metrics, for instance, error rate (ER), mean error distance (MED), and worst-case error (WCE), can be obtained, which are useful to assess and compared the accuracy of the generated approximate circuit. This error distribution can be later used for error propagation estimation when using the generated approximate circuit in the design of approximate accelerators [Cas+18].

The approximate arithmetic circuit design is then synthesized for minimum delay using Synopsys Design Compiler and for a given technology library (see ❺). Area and delay characteristics, as well as preliminary power estimations, are obtained from the synthesis reports. A post-synthesis characterization is performed with

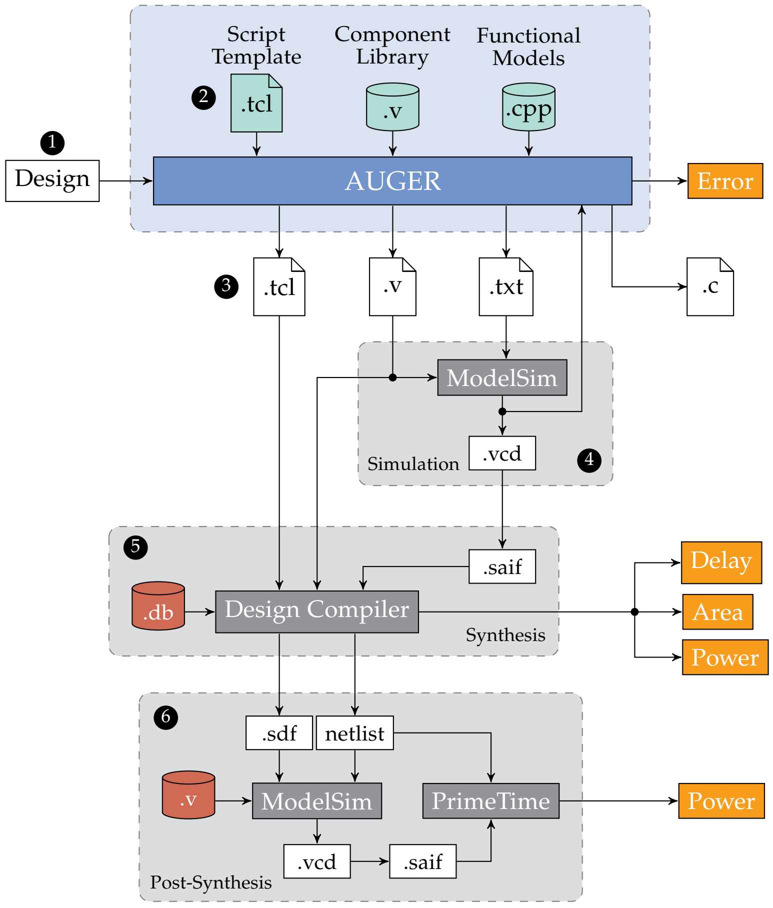
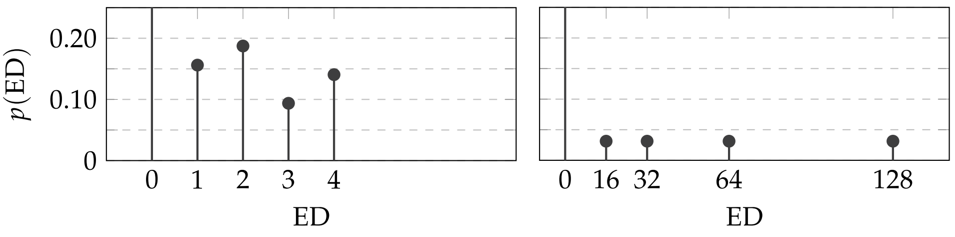


Figure 3.1: Main components in the AUGER tool. The AUGER tool generates an RTL for the bit-width and approximate configuration provided. AUGER relies on standard commercial tools for circuit simulation and synthesis to provide a characterization of the approximate arithmetic circuits.

Synopsys PrimeTime and a gate-level netlist simulation to obtain better power consumption estimations for the approximate circuit (see 6).

3.1.2 Evaluation

In this section, three scenarios of analysis for different approximate arithmetic circuits generated with AUGER are presented, particularly for approximate adders and multipliers, as they are the primary operations present in error-tolerant appli-



(a) LP, ER = 0.58 **(b)** HP, ER = 0.13

Figure 3.2: Error distribution for two 8-bit approximate adders generated with AUGER. The error characterization is obtained with AUGER, either with simulations or analytical methods.

cations, as discussed in Section 2.1. Results have been generated using the TSMC 65 nm technology library.

LP approximate adder

A first comparison scenario can be performed for low-power (LP) approximate adders. In such adders, exact 1-bit full adders are replaced for inexact counterparts, and they are used to calculate the LSB of the addition. AUGER can be used to compare new proposed adders against existing ones. For this, consider that an approximate full adder has been proposed, such that s and c_{out} bits are calculated as:

$$s = (a \oplus b) \vee c_{in} \quad \text{and} \quad c_{out} = a \wedge b$$

Seven 10-bit low-power approximate adders from the state-of-the-art, including one using the proposed approximate full adder, are compared. These adders include the Lower-part-OR Adder (LOA) [Mah+10], two variants of the Approximate Mirror Adder (AMA) [Gup+13], one from the Approximate XOR/XNOR-based Adders (AXA) [Yan+13], the Carry-Free Adder (CFA) [Dut+16], and one variant of the Inexact Adder by Cell Replacement (InXA) [AKL16]. For each adder, the 5 LSB bits have been approximated. Figure 3.3 presents normalized values, with respect to an accurate 10-bit ripple-carry adder (RCA), for the delay, area, power, and power-delay-product (PDP). The lower these normalized values are, the more savings are achieved. The error rate (ER) is here used to assess the accuracy of the approximate adder. In this case, a lower ER means a smaller probability of the adder to generate erroneous results.

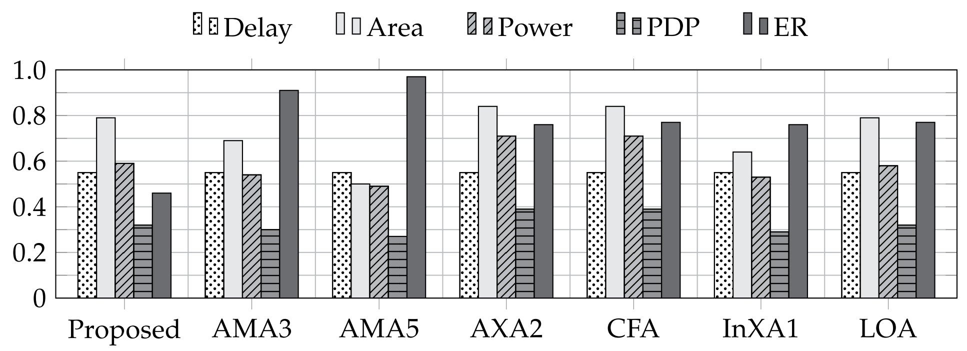


Figure 3.3: Comparison of LP approximate adders. The proposed replacement for a 1-bit full addition is compared against other reported LP approximate adders. Each one corresponds to a 10-bit approximate adder with the 5 LSBs approximated.

As can be noticed from Figure 3.3, the proposed approximate adder presents good savings in terms of PDP and, in comparison to its peers, slightly surpassed by the AMA3 and InAX1. However, the proposed approximate adder has the smallest ER compared to all approximate adders presented, being more accurate in this sense.

Besides this comparison of potential new LP approximate adder designs, with AUGER, it is possible to perform exhaustive explorations to find those Pareto-optimal configurations from a set of reported approximate adders. In Figures 3.4 and 3.5, different configurations have been explored for 17 different LP approximate adders reported in the literature. The exploration has been made for 16 and 32 bit-width and considering up to half of the output bits to be approximate. This means 137 and 273 different approximate adders explored for each case. Figures 3.4 and 3.5 show a set of dominant configurations for each circuit metric and considering MED as error metric. Consider the particular case of area exploration for 16-bit LP adders in Figure 3.4. As depicted in the graph, different configurations of the AMA5 [Gup+13] LP approximate adder are Pareto optimal for moderate large MED tolerable. This also confirms what is presented in Figure 3.3, where the AMA5 configuration is the one with a lower area concerning the other LP approximate adders compared. Similar to for area, for the other metrics, the Pareto-optimal configurations can be extracted and selected, if desired, for the design of approximate accelerators.

HP approximate adders

High-performance (HP) approximate adders correspond to those in which the addition is performed by a set of sub-adders, as described in Section 2.1. In many

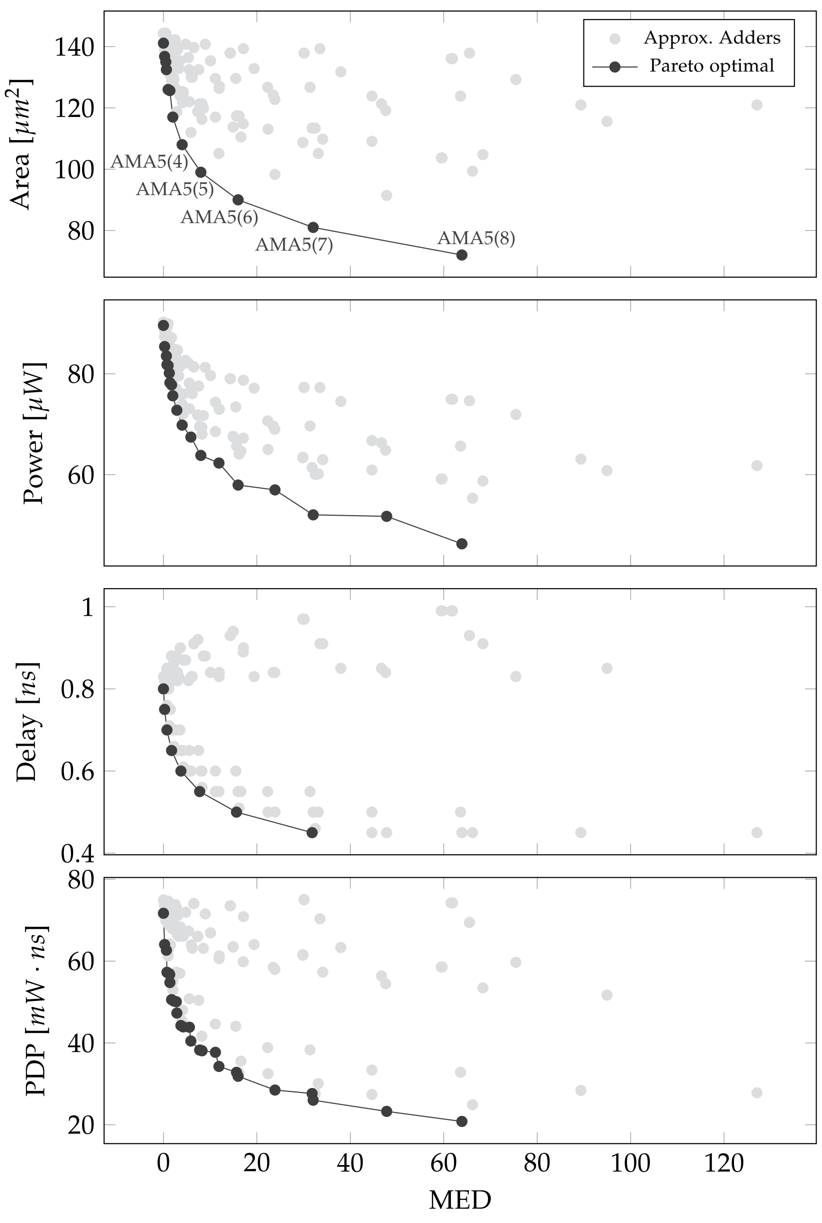


Figure 3.4: Exploration for 16-bit LP approximate adders. With characterization performed by AUGER, it is possible to find those Pareto-optimal LP approximate adders and their aproximate configuration, in this case for 16-bit adders and MED as error metric.

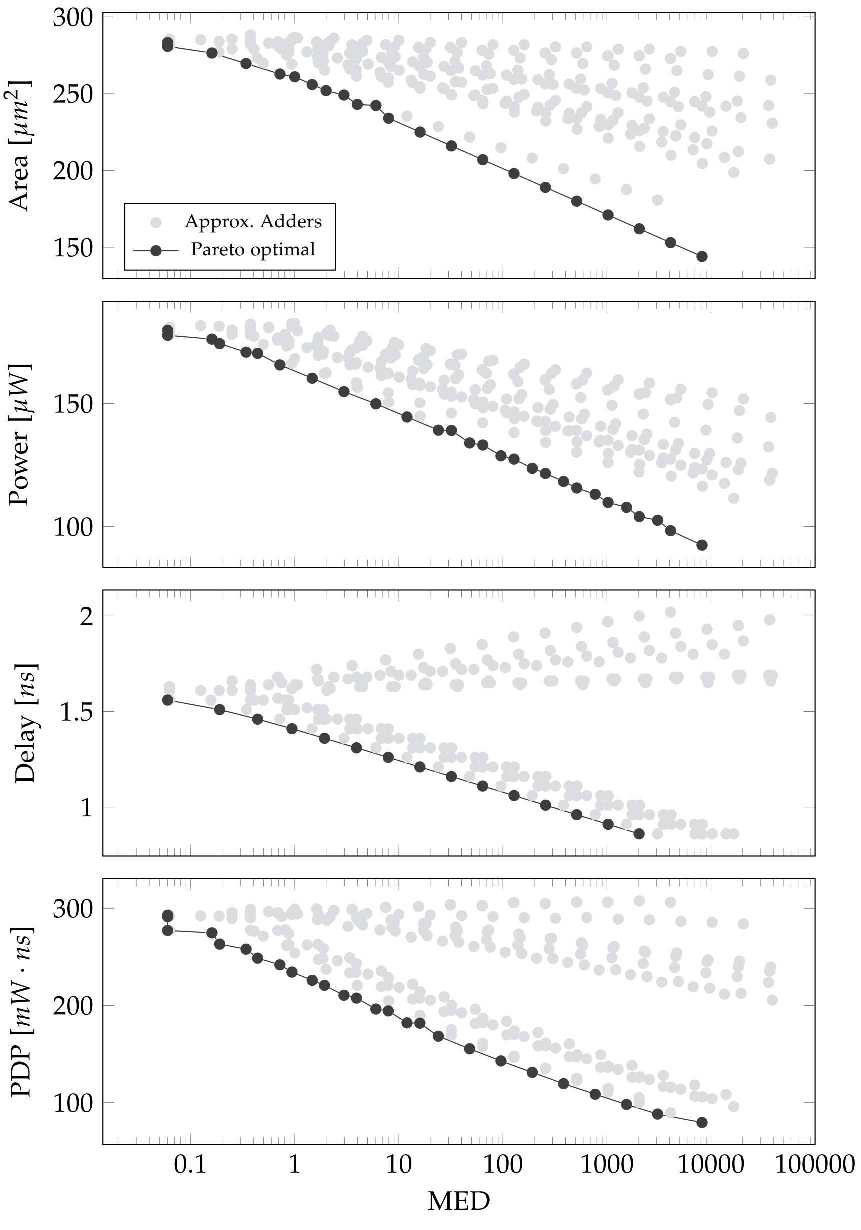


Figure 3.5: Exploration for 32-bit LP approximate adders. With characterization performed by AUGER, it is possible to find those Pareto-optimal LP approximate adders and their aproximate configuration, in this case for 32-bit adders and MED as error metric.

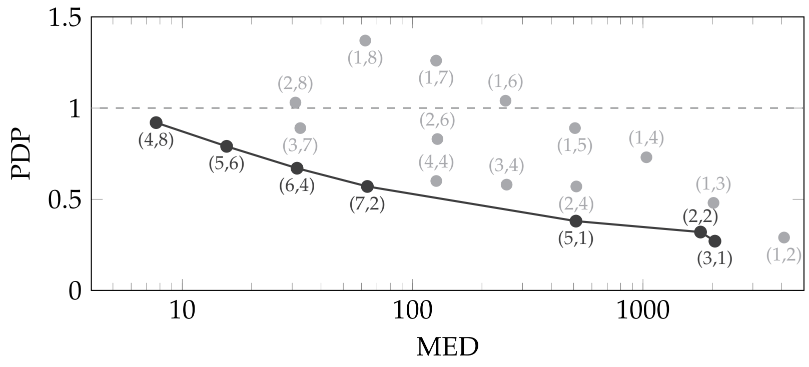


Figure 3.6: Exploration of 8-bit GeAr approximate adders. Comparison of all possible and valid 8-bit combinations for GeAr approximate adders. A set of these configurations are Pareto-optimal solutions with respect to the rest.

cases, for a given bit-width, several possible configurations of overlapping sub-adders can be generated. For instance, for a 16-bit GeAr adder [Sha+15], there are 22 different valid configurations. In fact, the GeAr adder can assume other HP approximate adders in its configuration, such as ACA II [KK12] and GDA [Ye+13]. When using GeAr, and for an N-bit addition, the most significant R-bits of the sub-adders are considered as resultant bits, and they are integrated as part of the final result, while the remaining P-bits, defined as previous bits, are used to estimate the carry propagation to upper bits.

Figure 3.6 presents a comparison in terms of PDP and MED for 20 of the valid 16-bit GeAr adder configurations. The GeAr configurations are denoted as (R,P) in Figure 3.6. As it can be noticed, and considering PDP as an energy estimation, there is a set of optimal configurations placed along the Pareto front. For instance, the (6,4) configuration present smaller PDP with respect to (2,8) and (3,7), for similar MED values. Although the delay is significantly reduced for some non-optimal designs, power consumption exceeds due to the additional hardware used in the several sub-adders required. This means that that GeAr designs consume more energy (normalized PDP $>$ 1) compared to a 16-bit RCA. From this type of comparison that can be performed with characterized approximate arithmetic circuits generated by AUGER, it is possible to select better those that minimize the required resources while meeting a defined accuracy threshold.

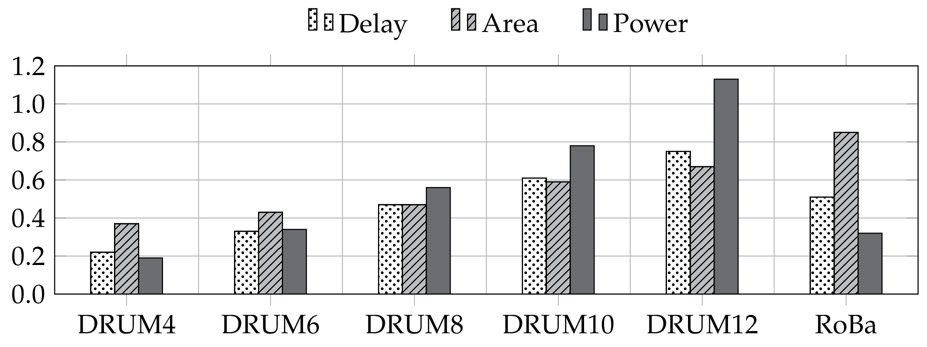


Figure 3.7: Comparison of two types of approximate multipliers. Comparison of two different 16-bit approximate multiplier approaches.

Approximate multipliers

To compare approximate multipliers generated with AUGER, consider two very distinct designs from the state-of-the-art, particularly one that presents different levels of approximations per input bit-width [HBR15], and another that presents a single configuration per input bit-width [Zen+17]. Figure 3.7 presents characterization values for some of these approximate multipliers normalized with respect to an exact array multiplier. Although the RoBa multiplier presents a single configuration for each bit-width, it is capable of obtaining good total power consumption improvement, about 70%. DRUM configurations can achieve even better power improvement, such as DRUM4, but introducing a MED about two orders of magnitude higher with respect to the RoBa.

It is also noticeable that DRUM configurations can overpass an accurate multiplier power consumption while still reducing its delay, but keeping the errors smaller than for other DRUM variants. If the delay is the design constraint, this could be the right candidate as delay improvement is obtained while not introducing significant errors.

3.2 A Framework for Approximate Logic Synthesis

Many efforts have been made to design approximate arithmetic circuits from their accurate implementations to reduce their size, delay, and power consumption [XMK16]. For instance, one effort focuses on reducing the circuit's supply voltage to lower the power required, allowing timing errors [Mia+12], and relaxing the synthesis of specific non-critical delay paths [AH18]. Other significant effort aims

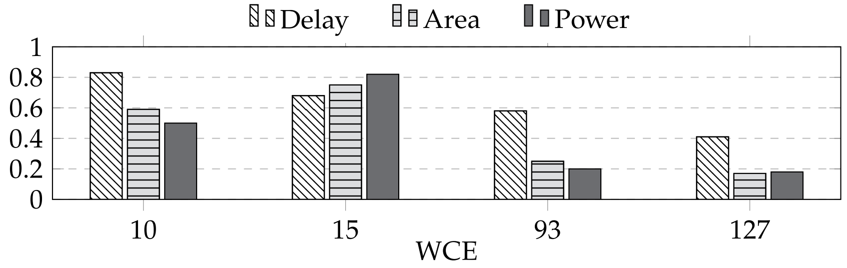


Figure 3.8: Characterization of four 8-bit approximate adders generated from an exact adder. Normalized values for delay, area, and power are depicted for different accuracy threshold. The adders were generated using the circuit carving technique [SAP18].

to perform functional approximations in which the functionality of an application, at register-transfer level (RTL) circuit or gate-level netlist, is simplified to trade-off accuracy for performance.

Regarding this second effort, different Approximate Logic Synthesis (ALS) methods have been proposed to generate approximate circuits from accurate implementations automatically. In the literature, three main approaches to exploit functional simplification have been reported [Sca+20]: structural netlist transformation [Sch+17; SAP18], Boolean rewriting [WQ16; HTR18], and approximate high-level synthesis [LJG17; Cas+20c].

For netlist transformations, various techniques have been reported, mostly based on the node pruning of a Direct Acyclic Graph (DAG) representations of the circuit netlist. For instance, nodes are pruned iteratively according to the significance they present to the final output and the toggle activity they have [Sch+17]. Another technique performs exhaustive exploration of all possible subsets of nodes that can be removed without surpassing a given error threshold [SAP18]. Figure 3.8 depicts the circuit properties for 8-bit approximate adders generated for a given worst-case error (WCE) (see Equation (2.2)) under this technique. As depicted, the goal is to reduce the resulting circuit's complexity, reflected in a reduction in its circuit metrics, while having a target accuracy for a specific error metric as a design constraint.

For the case of Boolean rewriting techniques, open-source contributions have been made [HTR18; MHR19]. However, to explore and implement current ALS techniques based on netlist transformation, develop methods for error modeling to avoid exhaustive simulations, and propose new netlist transformation approaches, an open-source framework that enables it is still missing.

This section presents AxLS[2], a novel, open-source framework for ALS techniques based on structural netlist transformations. With AxLS, existing ALS methodologies can be implemented and new ones proposed. This section describes this novel framework and provides an experimental evaluation for arithmetic circuits.

3.2.1 Description

AxLS is a framework for ALS techniques based on the concept of structural netlist transformations. Figure 3.9 depicts the main components of this framework. As shown, AxLS takes as an input the Verilog RTL description of the circuit to be approximated. A netlist is generated for a specific standard cell technology library using a synthesis tool. From the same technology library, particularly from the Verilog simulation models, a description of the basic gates is extracted into an XML file. Then, using this cell description, a representation of the netlist is generated with a custom netlist to XML function (`v2xml`). XML files are easy to retrieve, manipulate, and save; for that reason, it was chosen for the netlist's internal representation. Figure 3.10 presents a code snippet of a toy netlist representation with XML, using the NanGate 15nm technology library. Figure 3.11 shows a DAG representation for this netlist example, generated from the XML representation. Each node corresponds to a gate in the netlist, while primary inputs and outputs are indicated.

As depicted, a post-synthesis simulation can be performed to obtain gate-level switching activity values (saif file). This information can be included in the XML netlist representation as part of each node's properties, and it can be further used to guide the netlist transformation criteria [Lin+11]. Other characteristics, such as significance or weight, and fan-out can be calculated from the netlist and added to each node [Bro+15].

With an XML representation of the netlist, approximation criteria can be applied to transform the netlist. As previously mentioned, gates can be pruned one by one considering their switching activity and impact on the output error [Sch+17]. Genetic algorithms can also be used to mutate the netlist into approximate versions by interchanging gates with wire connections [MVS19], or output pruning can be applied by removing all logic gates that affect a specific circuit output bit. AxLS provides the capability to implement such types of techniques to modify and transform the netlist.

[2] AxLS is an open-source contribution and it is available at `https://github.com/ECASLab/AxLS`.

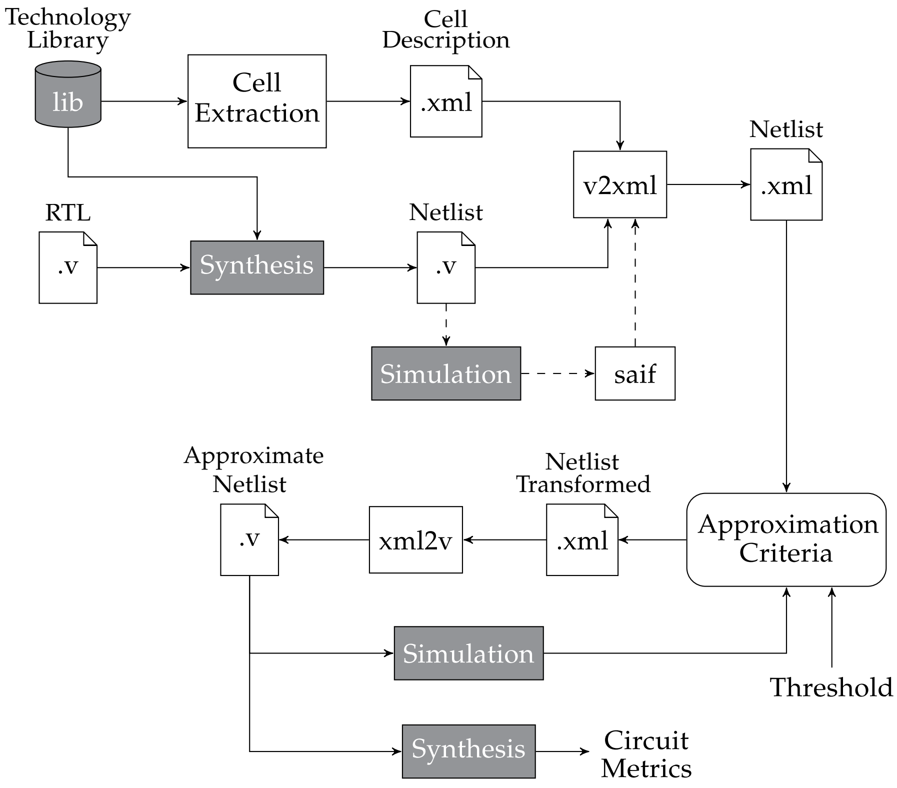


Figure 3.9: Overview of the AxLS framework. An XML representation of the synthesized netlist is created to manipulate it according to approximation criteria that can be defined within AxLS. AxLS uses external tools for synthesis and simulation of the accurate and approximate versions of the netlist.

Regardless of the netlist transformation technique followed, an accuracy threshold is required for a defined error metric. This accuracy target is used to assess if the approximations applied creates an approximate netlist that still produces acceptable results. According to the approximation criteria implemented within AxLS, a Verilog file with the approximate netlist can be generated from the transformed XML description, using a custom XML to netlist function (`xml2v`). This Verilog file generation is performed every time the accuracy of an approximate version is evaluated. The generated netlist is always based on the gates available in the cell library used for the synthesis.

Although recent techniques have been proposed to reduce the simulation dependency [Sca+19; Ech+20], by default, AxLS relies on gate-level simulations for accuracy assessment. The approximate netlist can be simulated to produce their

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <node name="INV_X2" var="u19">
    <input name="A1" wire="n14" />
    <output name="ZN" wire="n16" />
  </node>
  <node name="OAI21_X1" var="u23">
    <input name="A1" wire="n14" />
    <input name="A2" wire="n20" />
    <output name="ZN" wire="n21" />
  </node>
  <node name="NAND2_X2" var="u33">
    <input name="A1" wire="n16" />
    <input name="A2" wire="n14" />
    <output name="ZN" wire="S[1]" />
  </node>
  [ ... ]
  <circuitinputs>
    <input var="in[0]"/>
    <input var="in[1]"/>
  </circuitinputs>
  <circuitoutputs>
    <output var="S[0]"/>
    <output var="S[1]"/>
  </circuitoutputs>
</root>
```

Figure 3.10: Example of XML code for netlist description. Primary inputs and outputs, gates and their connections are easily represented with XML code.

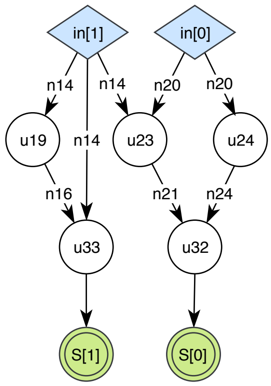


Figure 3.11: DAG representation of an XML netlist example. AxLS provides custom functionality to perform graphical representations of the netlists during the transformation process.

innacurate outputs. AxLS provides functionality to compare the approximate results against accurate ones to generate a PMF for the error representation, from which error metrics can be calculated, as described in Section 2.2. Although not depicted in Figure 3.9, the corresponding testbench is required for any simulation, which generates an output file with the approximate results. For this, representative test vectors are also required according to the input bit-width of the circuit to be approximated.

Once an approximate netlist is generated with satisfactory accuracy, according to the defined approximation criteria, the synthesis tool can be used to obtain circuit metrics such as area, delay, and power consumption.

3.2.2 Evaluation

The current state of AxLS has been implemented using Python language, and the internal representation of the netlist using XML files, as previously mentioned. As depicted in Figure 3.9, AxLS relies on external tools for synthesis and simulation. Currently AxLS uses the Yosys tool [Wol] for circuit synthesis and circuit area estimation, and Icarus Verilog [Wil] for netlist simulation. The results here presented were obtained using the NanGate 15nm technology library. For the simulations performed, 500K test vectors from a random uniform distribution were used for each input data of the circuit. Other distributions, such as normal, Weibull, or exponential, can also be integrated into AxLS.

An evaluation of AxLS for arithmetic circuits, particularly standard adders, is presented. As approximate criteria, here called as InOuts, the following netlist transformation steps have been followed:

- Considering a primary input constant, all affected nodes (gates) are explored. A node is considered constant if all its inputs are constant, and then all dependencies of such node are explored. Each of the considered nodes is pruned, one by one, and the accuracy is checked at each step. If the output error goes beyond the given threshold, the last pruning step is reversed, and other nodes are further explored.
- After exploring from the perspective of the inputs, nodes are explored considering a primary output constant. All nodes affecting such output are considered, and a one by one pruning is also performed with accuracy evaluations after every step. Similar to before, if the output error goes beyond the given threshold, the last pruning step is reversed, and other nodes affecting the output are explored.

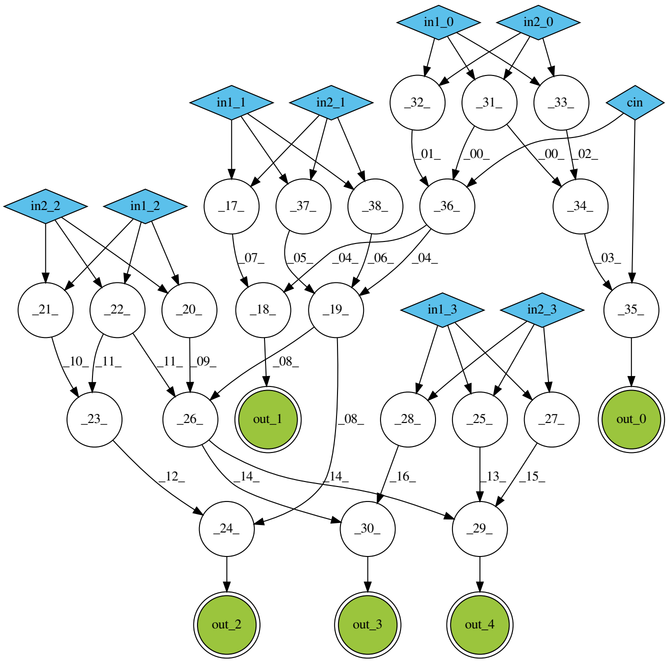


Figure 3.12: Accurate netlist. DAG representation of the accurate netlist of a 4-bit ripple-carry adder synthesized with the NanGate 15 nm technology library.

- For those nodes and outputs removed, a 0 value is assigned. For instance, nodes depending on others pruned will receive a 0 logic value as input instead of the expected result previously provided by the missing node.
- For this evaluation's scope, these steps are repeated for the half LSB of the inputs and outputs. For instance, for a 16-bit adder, this is applied first for the first LSB at the inputs and then for the 8 LSB of the output.

Consider a 4-bit RCA for an example with AxLS. Figure 3.12 depicts the netlist for the accurate implementation. The approximate netlist, shown in Figure 3.13, has been generated following the steps described and for a WCE $= 8$ as an accuracy threshold. As can be noticed from the error distribution in Figure 3.14, the higher error produced has an ED=6 with a very low probability (about 2%), which meets the accuracy constraint given. From this error distribution, internally generated by AxLS, other error metrics [MVS19] can be included and calculated in AxLS. Besides WCE, AxLS can generate approximate circuits using the MED as accuracy metric.

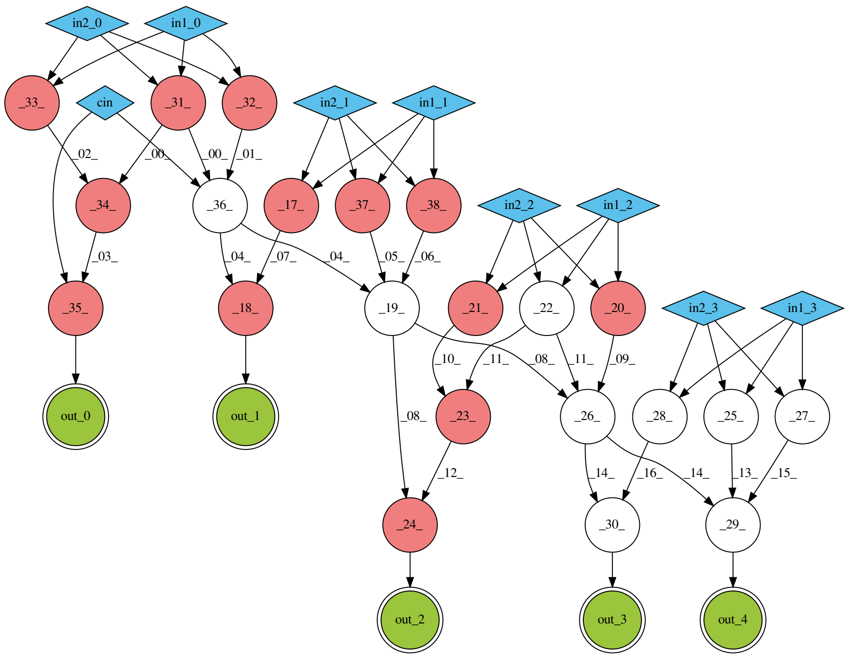


Figure 3.13: Approximate netlist for WCE = 8. DAG representation of an approximate 4-bit ripple-carry adder, generated with AxLS for a WCE = 8 constraint, from the netlist depicted in Fig. 3.12. Nodes marked (in red) are selected to be pruned.

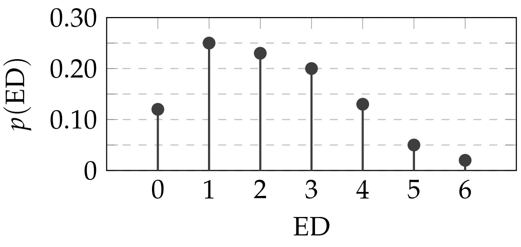


Figure 3.14: Error distribution for the approximate netlist. The higher error generated has a ED = 6, which meets the WCE constraint used to guide the netlist transformation.

Figure 3.13 shows that the resulting approximate netlist depends solely on two of the most significant bits of the inputs. Just two of the output bits are calculated, while the others (not depicted in the diagram) are defined as 0. For nodes that had a dependency on pruned gates, for instance, `_036_`, the remaining wires `_00_` and `_01_` will drive a 0 logic value, as described for these exemplary netlist

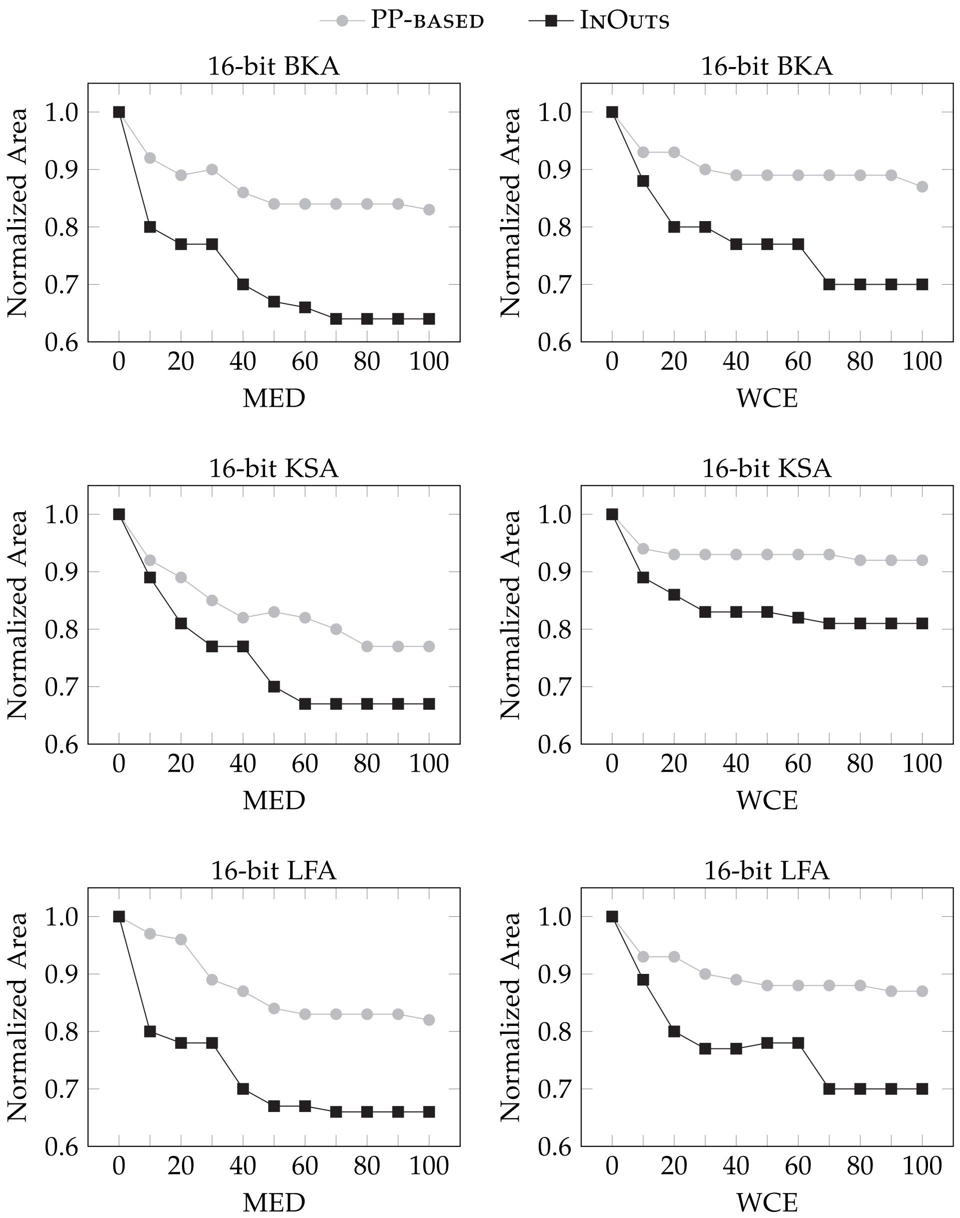


Figure 3.15: Evaluation using AxLS for three 16-bit adders. Both described approximate criteria, InOuts and PP-based, were used to generate approximate netlist for different accuracy thresholds.

transformation steps. Also, from Figure 3.13, it is possible to observe the remaining dependency on the carry-in, `cin`, for calculating the output bits `out_3` and `out_4`, as most of the gates that produce these primary outputs are preserved in order to achieve the desired accuracy.

To compare results with those generated with InOuts, an implementation based on the probabilistic pruning technique (PP-based) [Lin+11] has been implemented within AxLS. The goal of this technique is to prioritize the pruning of those nodes that change little. For instance, it begins by pruning nodes that produce 100% of the time a result equal to 1 or 0, for all the test input data, which means it is not toggling. Once a node is removed, other nodes depending on that pruned node's output receive a constant input value, which corresponds to the value the pruned node was driving most of the time. Whenever a node is pruned, the circuit's accuracy is evaluated and compared against the given accuracy threshold. Then, PP-based continues with those nodes with a higher percentage until the accuracy is reduced but satisfying the given threshold.

Figure 3.15 presents results for the evaluation for three different 16-bit parallel-prefix adders: Brent-Kung adder (BKA), Kogge-Stone adder (KSA), and Ladner-Fischer adder (LFA). Approximate netlists have been generated for different accuracy thresholds and WCE and MED error metrics, using both described approximation criteria, InOuts and PP-based. As the circuit's metrics are obtained with the Yosys tool, just the circuit area values are reported.

In general, Figure 3.15 illustrates how the circuit's area is reduced as higher errors are tolerate. With the steps in InOuts, it is possible to obtain more area savings then the results from PP-based, achieving up to 18% more area savings with the InOuts technique. However, it is interesting to notice that for some accuracy targets, the same area savings are obtained, for instance, for MED and WCE metrics from value 70. This is because no further netlist modifications can be applied without degrading the accuracy beyond the defined threshold.

3.3 Summary

In this chapter, two approaches have been presented for the generation of approximate arithmetic circuits. The first, AUGER, is a tool capable of generating RTL descriptions and characterizations of approximate adders and multipliers reported in the literature. The second, AxLS, is a framework that allows the implementation of ALS techniques based on netlist transformations to generate approximate netlist descriptions from accurate RTL implementations. The evaluation performed shows both approaches' capability to provide approximate arithmetic circuits that can be used for the automated design of approximate accelerators. As both contributions have been made open-source, this allows their usage by the scientific community and their extension to newer approximate arithmetic circuits and newer ALS techniques based on netlist transformations.

All models are wrong, but some are useful.

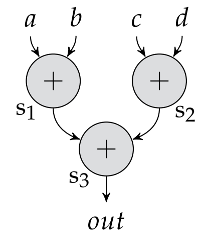
George E. P. Box

4 Modeling Error Propagation

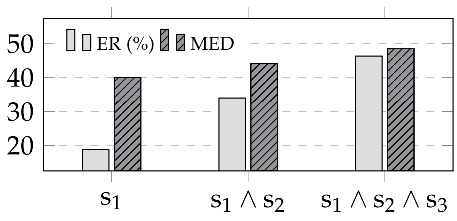
Error estimation is a cornerstone of *Approximate Computing* (AxC) [Sca+20]. This is essential for the automated design of approximate accelerators built with approximate arithmetic circuits. Despite the tolerance to errors, approximate accelerators must limit the output error to satisfy quality constraints, whether defined by the application, the developer, or even the end-user. The use of approximate arithmetic circuits in accelerators imposes the challenge of understanding how the error introduced by each approximate circuit behaves and propagates through other exact and approximate computations, and finally, how it accumulates at the output.

This chapter presents a novel compiler-driven methodology for estimating the error propagation to the output of an approximate accelerator. Using as a base a set of analytical rules to model the error propagation for individual calculations, a model is defined to estimate the propagation of error distributions represented as a probability mass function (PMF). These rules consider the propagation of errors through other approximate and accurate arithmetic computations. Since the design of accelerators is carried out in conjunction with the source code of the application, this methodology proposes to use the source code of the function to be accelerated as an input. A custom-defined pragma directive is defined to annotate the code and indicate which accurate operations are replaced by approximate ones. A modified compiler is proposed to handle these annotations and append metadata to the

The content of this chapter is based on the work originally published in [Cas+18]:
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.



(a) *Tree* of additions



(b) ER and MED for different adders replaced by approximated ones

Figure 4.1: Motivation example. In this example, up to three additions can be replaced by approximate ones. The ER and MED increase with the number of approximate adders.

intermediate representation of the code. This information is later used to statically analyze the code, using error propagation models previously determined, to obtain an error distribution of the output.

The contribution of this chapter has been integrated into a tool called CEDA.[1] This tool uses pragma-based annotated C/C++ source code as input. With these annotations, exact calculations are replaced by approximate to perform a static analysis of the error propagation and estimate the output error distribution.

4.1 Motivation

Consider three additions arranged as a *tree* of additions, as presented in Figure 4.1a. Each adder (s_1, s_2, s_3) can be replaced by an approximate adder that shows an Error Rate (ER) of 0.19 and a Mean Error Distance (MED) of 40.0. Figure 4.1b depicts the increment of ER and MED as the number of additions replaced by approximated changes. When the approximate adder replaces just s_1, the error metrics at *out* correspond to the ones produced by only one adder (ER = 0.19, MED = 40.0). In this case, the other exact additions do not have an impact on the error metrics.

When all adders are replaced, the ER and MED at *out* are 0.46 and 48.5, respectively. Estimating these values make possible the selection of adequate approximate circuits to comply with a defined error tolerance. For instance, if the additions presented have an accuracy limit of 45.0 for the MED metric, replacing s_1 and s_2 for the mentioned approximate adder can satisfy this constraint. These two

[1] CEDA is an open-source contribution and it is available at `https://git.scc.kit.edu/CES/CEDA`.

approximate additions produce a MED of 44.1, as depicted in Figure 4.1b. These error estimations are required to guide the design space exploration of approximate accelerators, and thus select the approximate circuits that meet the accuracy threshold of a design while, for instance, reducing delay or power consumption.

4.2 Error Estimation for Approximate Desings

Existing work has proposed different approaches to estimate the output error at the approximate component and accelerator level. Statistical and analytic models have been presented to characterize the error produced by approximate circuits such as adders [Maz+17b; AHS17] and multipliers [Maz+17a]. Given that this dissertation's focus is on approximate accelerators, these models [Maz+17b; AHS17; Maz+17a] can serve as an input.

For approximate accelerators, some current works employ exhaustive simulations or Monte Carlo simulations [Li+15], while others define an input data set [KCS16] or use interval and affine arithmetic [HLR11; HLR12]. Capturing error statistics through simulations produces accurate characterizations, but they are time-consuming, particularly when considering diverse configurations for an arithmetic circuit. For the other cases, the characterization precision is limited because the error metrics are sensitive to the input distribution. Other works use analytical formulations [Cha+13; Lee+16] but limiting the applicability to a small set of approximate circuits. Once the error characterization for an arithmetic circuit is achieved, the output error in a hardware block, involving exact and approximate calculations, is determined by defining error propagation rules [HLR11; HLR12; Li+15], or by performing regression analysis [Cha+13] or curve fitting [KCS16] to establish prediction equations. For the latter approaches, if the design or the circuits used changes, it is necessary to obtain new prediction equations by performing the proposed methods again.

Most of these works require detailed simulations to estimate final error propagation. This is a limitation to explore a vast design space when multiple configurations of approximate arithmetic circuits are considered, for instance, while performing high-level synthesis of approximate accelerators. This chapter proposed an analytical methodology for estimating the error propagation, not requiring simulations, and using only the error distributions of the approximate arithmetics used.

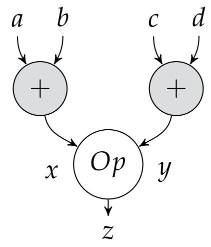


Figure 4.2: Set up for error propagation modeling. Results x and y, produced by approximate adders, are used as inputs to other accurate and approximate computations to model the error at the output z.

4.3 Models for Error Propagation

Error propagation estimation requires models representing the interaction between errors produced by approximate arithmetic circuits and other accurate and approximate computations. Through exhaustive experimentation [Tic98], and using results produced by approximate adders, a set of propagation rules or patterns can be established.

Consider the diagram in Figure 4.2. The inputs a, b, c, and d are error-free values added by two approximate adders, whose error distribution is known. The error in x and y depends on whether the addition produces an error or not, and depending on the input values and the adder configuration. Here, Op is replaced by accurate operations: addition, subtraction, multiplication, shift left, shift right, or division. Also, an approximate addition and multiplication are considered for Op.

By analyzing the results of the individual computations for every replacement of Op, a set of analytical rules has been derived to estimate the error propagation. For addition, the output error (e_z) is defined as:

$$e_z = e_x + e_y + e_+ \tag{4.1}$$

which is the addition of the input error of both addends, e_x and e_y). If an approximate addition replaces Op the output error considers the error introduced by the approximate adder itself, e_+, along with the error of the addends.

For the subtraction, the output error is the difference between the error of both operands:

$$e_z = e_x - e_y \tag{4.2}$$

As in a regular subtraction, the order of the operands affects the output error, and this can lead to errors that can be reduced and even eliminated.

The multiplication presents an especial condition, as the output error has an intrinsic dependency on the input values of the multiplication:

$$e_z = x \cdot e_y + y \cdot e_x + e_x \cdot e_y + e_\times \tag{4.3}$$

where $e_\times$ is the error introduced by the approximate multiplier. In case of an accurate multiplication, $e_\times = 0$. This means that even the error propagated through an accurate multiplication does not just depend on the input error, as for the addition, but also on the values to be multiplied.[2] However, for many error-tolerant applications, such as in a FIR filter or a neural network, the multiplication is performed between a variable, which can have an error associated, and a constant value, which reduces the expresion in Equation (4.3) to:

$$e_z = y \cdot e_x + e_\times \tag{4.4}$$

considering y as the constant value, which is known, and y as the variable with error. The previous expression even considers the general case of using an approximate multiplier.

For shift left and shift right, the error propagates through the computation experimenting an increment or decrement of its magnitude, respectively. For the case in Figure 4.2, y represents the number of bits that x needs to be shifted. Thus, the input error of x is multiplied or divided by 2 to the power of y, if a shift left or right is required, respectively.

$$e_z = e_x \cdot 2^y \tag{4.5}$$

$$e_z = e_x / 2^y \tag{4.6}$$

For the case of division by a constant value (which is common in error-tolerant applications, such as image processing kernels), the output experiences an effect

[2] The propagation rules obtained for addition and multiplication confirm previous presented equations in [Li+15].

similar to the shift right, so that it is decremented. Considering y as the constant value:

$$e_z = e_x / y \tag{4.7}$$

The models previously presented have been determined for single error values. However, they can be applied to error distributions represented by a PMF. Considering the PMFs of the errors produced by approximate adders as distributions of independent discrete and random variables, the addition of two PMFs is the convolution of their probability distributions. If $p_Z(z) = \mathbf{P}(Z = z)$ is the error distribution after an accurate addition (as for Op in Figure 4.2), and $Z = X + Y$, then,

$$p_Z(z) = \sum_y p_X(z - y) p_Y(y) \tag{4.8}$$

where $p_X(x)$ and $p_Y(y)$ are the PMFs for x and y. This formulation also applies to subtraction. However, before performing the convolution, it is required to invert the ED values for the subtrahend PMF. For the product of two PMFs, this is calculated as the joint probability of $p_X(x)$ and $p_Y(y)$, $Z = XY$, for two independent discrete random variables. Then,

$$\begin{aligned} p_Z(z) = p_{X,Y}(x,y) &= \sum_{x \in X, y \in Y} \mathbf{P}(X = x, Y = y) \\ &= \sum_{x \in X, y \in Y} \mathbf{P}(X = x)\mathbf{P}(Y = y) \end{aligned} \tag{4.9}$$

considering all pairs of values (x, y) that X and Y take. For shift left and right, the ED of the input error distribution is scaled by a factor of 2^y, as previously described. So,

$$p_Z(z) = 2^y \cdot p_X(x) \tag{4.10}$$

$$p_Z(z) = \frac{p_X(x)}{2^y} \tag{4.11}$$

represent the error propagation after shift left and shift right, respectively. In these shift operations, the ER for each ED does not change. When the multiplication is

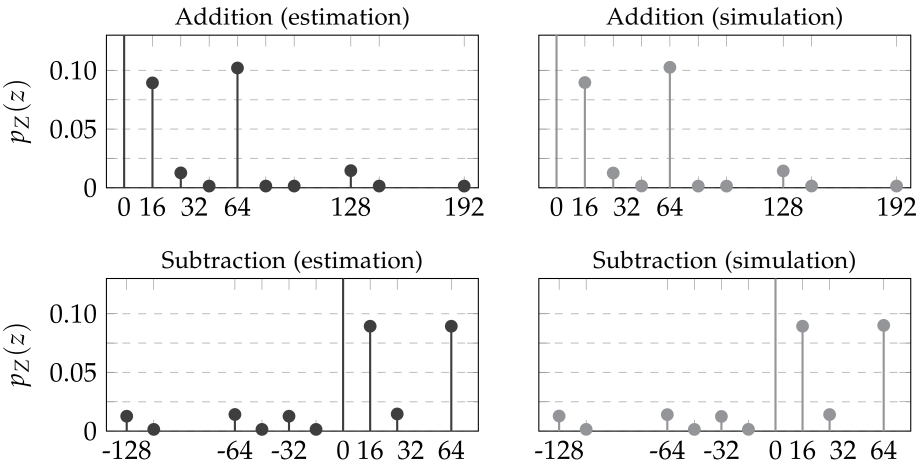


Figure 4.3: Example of error propagation for addition and subtraction. Error propagation estimation after an accurate addition and a subtraction. The error distribution of two HP approxiamte adders was used as input error distributions x and y.

between a constant value and one with errors, the effect in the error propagated is similar as for shift left:

$$p_Z(z) = y \cdot p_X(x) \tag{4.12}$$

and for the division, similar to the shift right:

$$p_Z(z) = \frac{p_X(x)}{y} \tag{4.13}$$

Figure 4.3 depicts the PMFs for the error propagated after an accurate addition and subtraction, considering that the errors distributions of x and y have been generated by HP approximate adders. In this case, the results obtained by applying the propagation models for error estimation, produce identical results as for Monte Carlo simulations.

4.4 Estimation of Error Propagation

Using the previous rules, by exploring where the errors are introduced and how they propagate, it is possible to estimate the error propagation at the output of

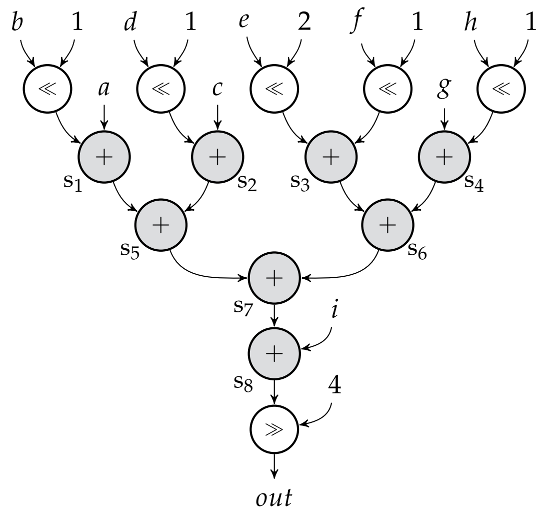


Figure 4.4: DFG representation of a 3×3 Gaussian kernel. Approximate adders potentially replace the additions in the kernel.

an approximate accelerator design. Representing the accelerator as a Data Flow Graph (DFG), an algorithm based on Depth-First Search (DFS) [SW11] is proposed to explore all the nodes in the DFG starting from the output to the inputs. All the nodes on which the output has a data dependency must be visited.

Consider the DFG for a 3×3 Gaussian kernel, as depicted in Figure 4.4. The output error distribution is calculated starting from the shift right operation at the output. From it, the error distribution of its predecessors is requested, in this case, the adder s_8. This adder, in turn, needs the error propagated by the adder s_7, and so on, until the inputs are reached. For instance, considering that just adders s_1 and s_2 are replaced by approximate adders, once those nodes are reached, the error they generate is determined according to its configuration, with analytical models for HP approximate adders, or previous characterization for the LP approximate adder. Because the inputs are considered as error-free, the error propagated after s_1 is the error generated by itself. The same occurs for s_2. Both inputs to adder s_5 now present errors, and the error propagated after s_5 is computed for the case of an accurate addition of two input error distributions, as introduced in Equation (4.8). This error distribution is propagated through s_7 and s_8 with no change because no other errors are generated in other computations, and the final error distribution propagated is only affected by the shift right at the end. This estimation process is generalized in Algorithm 1.

ALGORITHM 1: DFS-based Error Propagation Estimation.

```
Input: DFG;
Output: PMF;
computeError(node);
foreach parent $\in$ node.parent do
    if parent == instruction then
        parent.Error $\longleftarrow$ computeError(parent);
    end
    else
        parent.Error $\longleftarrow$ 0 ;
    end
end
errorPropagation(node.Error, node.parent.Error);
```

4.5 CEDA Tool

The proposed methodology has been implemented into a tool called CEDA: Compiler-driven Error Analysis for Designing Approximate Accelerators. Figure 4.5 shows its major components. An annotated C/C++ source code and an error model file are required to perform an estimation of the error propagation. The source code corresponds to the application where the introduction of errors in a function or kernel is explored. A custom directive, `#pragma approx`, indicates to a modified Clang compiler when an accurate addition or multiplication is replaced by an approximate one, as depicted in the code example of Figure 4.6. In Figure 4.6, the `gear wrp 8 1 4` of the pragma points the type of adder used for the evaluation (in this case GeAr) with its configuration (bit width `w`, resultant `r`, and propagation bits `p`; in this example 8, 1, and 4, respectively).

The modified Clang compiler handles this pragma and appends this information as metadata in the Intermediate Representation (IR) of the code, as shown in Figure 4.6. The error analysis stage (Fig. 4.5) takes the code as IR and creates a DFG representation. The metadata allows identifying which operations, represented as DFG nodes, are then approximated. The error model input corresponds to the definition of the error representation (i.e., as PMF) and the set of error propagation models previously presented. The error analysis follows the proposed methodology to estimate the error propagated to the output as a PMF. This information can then be related to specific error metrics such as MED, as described in Section 2.2.2.

The error propagation analysis is performed statically, eliminating the need for simulations and the associated computing time. However, the CEDA tool provides the possibility to obtain output error distributions through simulations. The

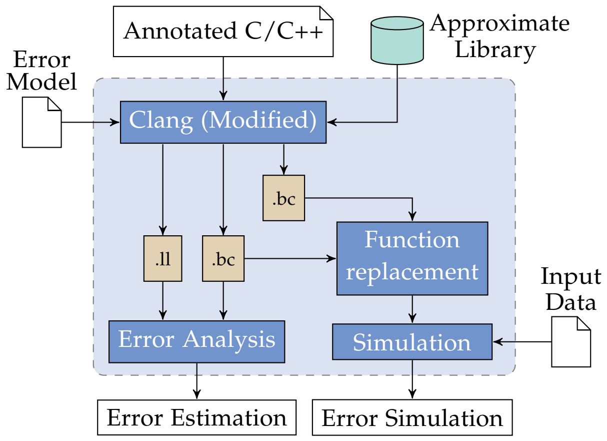


Figure 4.5: Main components of the CEDA tool. The .bc and .ll files contain the LLVM intermediate representation (IR) of the annotated source code, the error model, and the approximate library. Both error analysis and simulation are carried out using the IR.

```
void prog(int64_t* out, int64_t a, int64_t b, int64_t c) {
  #pragma approx gear wrp 8 1 4
  int64_t r = a + b;
  int64_t m = r + c;
  out[0] = r;
  out[1] = m;
}
```

```
define void @prog(i64* nocapture %out, i64 %a, i64 %b,
i64 %c) #0 {
entry:
  %add = add nsw i64 %a, %b, !approx !1
  %add1 = add nsw i64 %add, %c
  store i64 %add, i64* %out, align 8
  %arrayidx2 = getelementptr inbounds i64, i64* %out, i64 1
  store i64 %add1, i64* %arrayidx2, align 8
  ret void
}

!1 = metadata !{metadata !"gear", i32 8, i32 1, i32 4}
```

Figure 4.6: Example annotated code and LLVM IR. In this example code, the custom pragma annotation (above) can be noticed and its corresponding IR (below). Also, notice the metadata added regarding the approximation at the bottom of the IR.

metadata in the IR is used to determine which additions to replace by a function invocation and to provide the parameters required. This function mimics the execution of an approximate adder. For this, a library of approximate adders and multiplier is required as an input to this tool, as shown in Figure 4.5, which has functional models that can produce the results of the approximate adders. Once this replacement is performed, the IR codes for accurate and approximate versions of the application are executed using the LLVM-JIT (just-in-time) [LA04] compiler and a given input data set. The error statistics are determined by comparing the results. Results obtained through simulations provide the possibility to compare the accuracy of the results provided by the proposed methodology and validate the propagation models.

4.6 Evaluation

Results obtained with the proposed methodology are compared against user-defined input data sets and Monte Carlo simulations. Results for estimations and simulations are generated with CEDA. Figure 4.7 presents the estimation and simulation results for the aforementioned GAUSSIAN kernel. In this example, the adders s_4 and s_7 have been replaced by HP approximate adders. The estimation performed shows almost identical error propagation on 10^6 input combinations generated by a Monte Carlo simulation. The inputs generated used by the simulation correspond to potential pixel values of an image. There is a difference in the probability of the $ED = 2$, which is also reflected in the 0.02 difference in the ER, because the values in the simulation do not cause the approximate adders to produce the errors with the exact same probability as considered by the analytic error modelling [Maz+17b], which is used by this methodology. These differences are also noticeable when using a smaller input data set, as a test image for this example, due to the distribution of the inputs provided by each image. Two test images, *peppers* and *cameraman*, were used as inputs. For the case of *peppers*, the results show the same ED values but with lower value in their probability. For the *cameraman*, the ER is basically equal to the simulation results, but it is appreciable the difference of the probability for $ED = 8$ and $ED = 32$.

The required time for the estimation does not overpass some tens of milliseconds, compared to the seconds required for the simulation, which is particularly important if this analysis is required for a diverse set of configurations of approximate adders. The results using a smaller data set can reduce the required simulation time, but they produce inaccurate results depending on the values provided by the image(s) used.

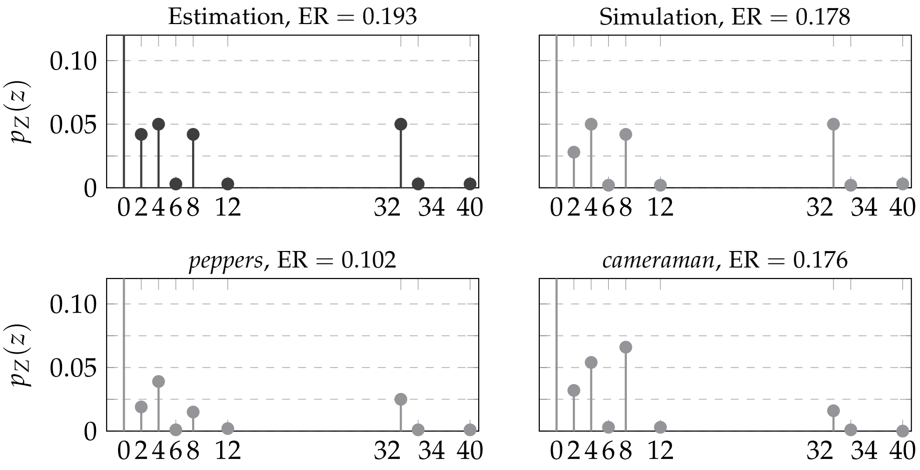


Figure 4.7: Evaluation for a 3×3 Gaussian filter. Error propagation estimation and simulation for a 3×3 Gaussian kernel. Two adders have been replaced by HP approximate adders. Results for a defined data set, such as *peppers* and *cameraman* test image are presented.

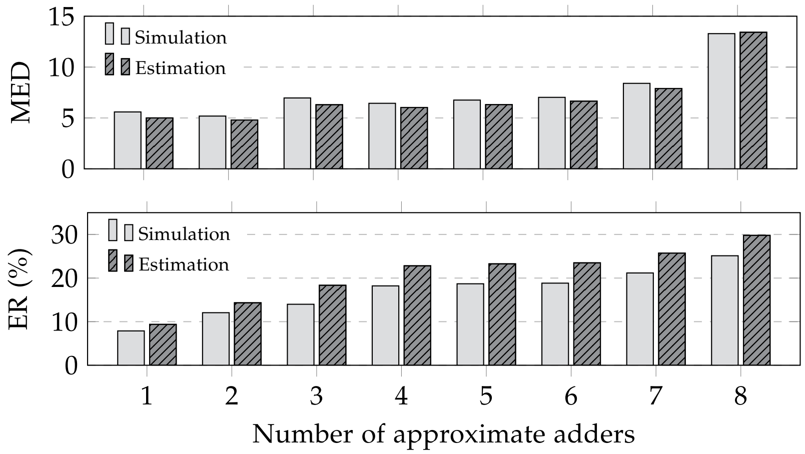


Figure 4.8: MED and ER evaluation regarding number of approximate operations. MED and ER for different error estimations and simulations of a 3×3 Gaussian kernel. The number of approximate adders varies from 1 to 8.

The CEDA tool is able to accurately estimate for different number of arithmetic circuits replaced. For example, Figure 4.8 shows the MED and ER for a Gaussian kernel, considering that the adders replaced grows from 1 (just one adder) to 8 (all additions). The MED is calculated from the resulting propagated PMF. The

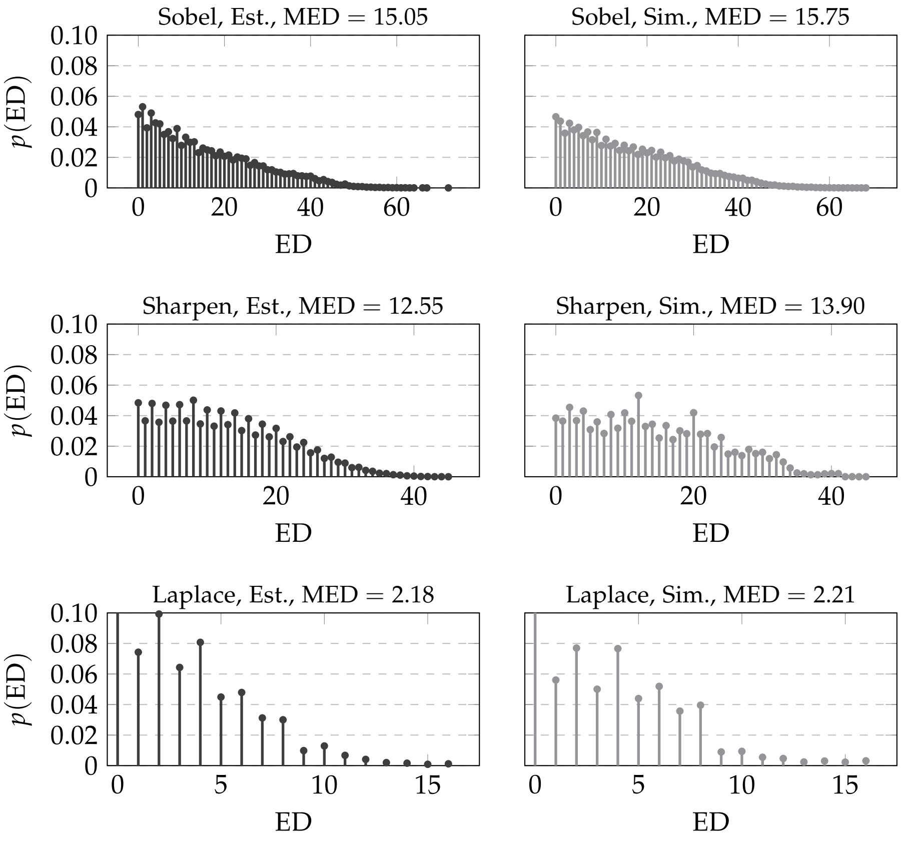


Figure 4.9: Evaluation for approximate accelerators designs. Error propagation estimation and simulation for a 3×3 Gaussian kernel. Two adders have been replaced by HP approximate adders. Results for a defined data set, such as *peppers* and *cameraman* test image are presented.

estimations shows slight differences regarding simulation, because this simulation is not exhaustive, and therefore does not consider all input combinations.

Further results are presented in Figure 4.9. Here, the output error distribution dor different approximate accelerator designs have been estimated. Results for simulations of the accelerator are also presented. As shown, the MED metric calculated from the PMFs is very close between estimation and simulation, even though they present differences.

4.7 Summary

In this chapter, a compiler-based methodology to estimate the error propagation on approximate accelerators designs have been presented. This methodology uses analytical rules proposed to model the error propagation through accurate and approximate arithmetic computations. These models are used in a DFS-based algorithm to explore a DFG representation of approximate designs and estimate the output error distribution. The proposed methodology has been integrated into CEDA, a tool released as an open-source contribution. It performs the error estimation from annotated C/C++ code, facilitating the exploration of different approximate accelerator designs without requiring time-consuming simulations. The results presented in the evaluation for various image processing kernels have shown very accurate estimations with respect to image-based input data sets and Monte Carlo simulations.

Each problem that I solved became a rule, which served afterwards to solve other problems.

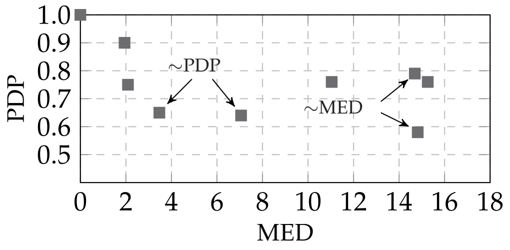
René Descartes

5 Designing Approximate Accelerators

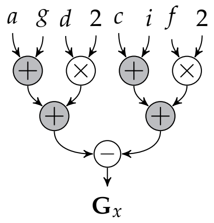
With the rise of approximate computing as a design paradigm, many approximate functional units have been proposed, particularly approximate adders and multipliers. These circuits reduce the accuracy of their results within a tolerable margin to scale down the required computational effort and energy requirements. However, for an increasing number of such approximate circuits reported in the literature, the challenge is to select those that minimize the required resources for a specific approximate accelerator description while satisfying a defined accuracy constraint.

In this chapter, a novel automated framework for High-Level Synthesis (HLS) of approximate accelerators using a given library of approximate arithmetic circuits is presented. As repetitive circuit synthesis and gate-level simulations require a significant amount of time, to enable this framework, a set of analytical models for estimating the necessary computational resources when using approximate adders and multipliers in approximate designs is introduced. These models complement those presented in Chapter 4 for error propagation estimation in approximate accelerators. A Design Space Exploration (DSE) methodology for error-tolerant applications, in which analytical models are used to estimate resources needed, and the accuracy of approximate accelerators is described. This DSE methodology

The content of this chapter is based on the work originally published in [Cas+20c]:
J. Castro-Godínez, J. Mateus-Vargas, M. Shafique, and J. Henkel: "AxHLS: Design Space Exploration and High-Level Synthesis of Approximate Accelerators using Approximate Functional Units and Analytical Models". In: *2020 IEEE/ACM 39th International Conference on Computer-Aided Design (ICCAD)*, 2020.



(a) Solutions for a small design-space exploration.



(b) DFG for $\mathbf{G}_x$.

Figure 5.1: Motivational example. Solutions for a small design-space exploration (5.1a) of an approximate $\mathbf{G}_x$ kernel of the Sobel filter (5.1b). Using approximate additions, the required energy can be reduced while increasing the errors.

allows finding Pareto-optimal solutions for approximate accelerator designs, minimizing the required resources while meeting accuracy constraints. Furthermore, this DSE methodology ins integrated into an HLS tool to automatically generate accelerators from C language descriptions.

5.1 Motivation

Consider the 3×3 $\mathbf{G_x}$ kernel of the Sobel filter, depicted as a Data Flow Graph (DFG) in Figure 5.1b. The input variables in Figure 5.1b represent pixel values from a 3×3 section of an image. Consider that the four required additions are replaced with approximate ones. For a small set of approximate adders, it is possible to obtain different solutions that set a trade-off between the required computational effort and the accuracy of results, as depicted in Figure 5.1a. In this example, the computational effort required by an approximate design is represented by the energy consumed. For this, the Power-Delay-Product (PDP) is used, a figure of merit used for energy efficiency. In Figure 5.1a, PDP values are normalized with respect to an accurate design. The accuracy of results is represented by the Mean Error Distance (MED) (Section 2.2.2).

As shown in Figure 5.1a, by allowing a degradation in the accuracy (higher MED), it is possible to reduce the required effort (smaller PDP). However, some combinations of approximate adders might reduce the required energy similarly (~PDP) while presenting different accuracy. Alternatively, some approximate designs might have similar accuracy (~MED) but reducing the energy differently. As indicated in Figure 5.1a, two solutions have a PDP≈0.65 for different MED, 3.47, and 7.06, while the other two have a MED≈15 but with different PDP. Repetitive gate-level simulations and circuit synthesis consume a significant time to explore many, or even all, possible combinations for a set of approximate adders, as in this example.[1]

5.2 Generation of Approximate Accelerators

Models to estimate the consumed resources (either area, delay, power, or energy) and the error introduced due to approximations can speed up selecting those approximate functional units that will contribute to obtain Pareto-optimal solutions. This is particularly essential to ease the design of approximate accelerators using behavioral descriptions and HLS tools.

Previous work has proposed models to estimate how the errors generated by approximate functional units accumulate and propagate to the outputs in approximate designs [Li+15; Cas+18; Sen+19]. Due to the deterministic nature of the errors produced by reported approximate arithmetic circuits, as considered in this work, their error distribution can be pre-characterized according to their bit-width and configuration. These error distributions can then be used in the DSE of approximate designs. However, analytical models to estimate the impact of approximate functional units in the required area, delay, power, and energy of approximate accelerator designs are also required. Existing work has relied solely on the evaluation of approximate designs through circuit simulation and synthesis [Nep+14; AMP18].

As previously mentioned, faster explorations of approximate accelerator designs can be performed through analytical models to estimate accuracy and required resources. For the first, state-of-the-art contributions have proposed methods to estimate the error propagation using error distributions of the approximate functional units used [Li+15; Lee+16], mainly using a Probability Mass Function (PMF) to depict such error distributions [Cas+18; Sen+19]. This dissertation presents in Chapter 4 a methodology for performing error estimations for approximate accelerators, which is used in this chapter.

The PMFs and other error characteristics of approximate arithmetic circuits can be achieved using methods reported in the literature [Maz+17b; AHS17; Maz+17a; Wu+19] or through exhaustive simulations. Other work has proposed using artificial neural networks to estimate the accuracy degradation in approximate designs [Zer+19]. However, this requires a significant data set and time to train the models.

[1] For instance, 200 seconds are required to characterize 10 different approximate designs for this $\mathbf{G_x}$ kernel example.

On the other hand, models for estimating the savings due to approximations have not been significantly exploited [LJG17]. In most cases, circuit synthesis is performed to assess the impact of approximations [Nep+14; Zer+16; BIM16; AMP18; LN19]. This chapter proposes analytical models to estimate circuit metrics when using approximate adders and multipliers in approximate accelerator designs.

Besides contributions that propose approximate designs from accurate RTL descriptions [Nep+14; Zer+16; AMP18], work has been reported regarding the generation of approximate designs using HLS tools. The main idea behind standard HLS is to automate the generation of digital hardware from a described behavior by interpreting its algorithmic description. With the notion of approximate computing added, the goal is to generate such digital hardware, but inaccurate, while meeting a defined accuracy constraint.

Two main contributions have proposed additional stages to an existing HLS tool, LegUp [Can+13], to generate approximate designs from high-level descriptions [LJG17; LN19]. Other contributions have proposed approximate aware scheduling, allocation, and binding, but outside an HLS tool [Li+15], and the use of a software library to explore arbitrary precision in the computations before using a set of predefined reduced-accuracy components with Vivado HLS [BIM16]. In [LJG17], precision optimization is implemented as bit rounding; even operations are eliminated. Additionally, voltage scaling is applied to take advantage of latency reductions, but making the final design complex due to the different voltages required. In [LN19], variable-to-constant substitutions are employed to prune operations, reduce overall delay, and hence meet real-time constraints.

5.3 Models for Resource Estimation

As previously discussed, to accelerate the DSE of approximate accelerators designed with approximate arithmetic circuits, it is required to have analytical models to estimate the impact of such approximate components in the circuit metrics. In this section, AxME, Approximation-based circuit Metrics Estimation, is presented. With AxME, a set of analytical models enables the possibility to evaluate the impact on area, power, and delay when using approximate adders and multipliers in approximate designs. The estimations provided by AxME rely solely on individual characteristics of the components conforming to the design. Thus, instead of simulating and synthesizing each approximate variant during the DSE to determine these circuit characteristics, it can be estimated from the area, dynamic and static power, and delay values of the design's components.

Table 5.1: Sample of components in the approximate library. These correspond to some of the 8-bit adders used for the formulation of AxME.

Adder	Area $[\mu m^2]$	S. Pow. $[\mu W]$	D. Pow. $[\mu W]$	Delay $[ns]$
RCA8b	11.01	0.50	20.91	71.99
LOA8b2	9.24	0.42	18.18	55.30
LOA8b4	7.08	0.33	13.23	38.61
GeAr8b22	12.39	0.56	26.38	43.12
GeAr8b13	17.30	0.77	29.87	43.12
GeAr8b24	13.47	0.61	27.92	59.81

A set of approximate arithmetic units was characterized to propose AxME. This characterization was performed by the contribution described in Chapter 3. Table 5.1 shows the characterization for some of the 8-bit adders used for the formulation of AxME. The two most representative types of approximate adders, low-power (LP) and high-performance (HP), were considered, particularly the widely adopted open-source LOA [Mah+10] and GeAr [Sha+15] approximate adders, respectively. For approximate multiplication, the Broken-Array Multiplier (BAM) [Mah+10] was used. These components are considered for different bit-width and approximate configurations. Ripple-Carry Adder (RCA) and Array Multiplier (AM) were used as accurate counterparts.

For the analysis leading to AxME, a simple test application, here called *tree*, was used. The DFG of the *tree* is depicted in Figure 6.6. The O_i, $i \in \{1, \ldots, 7\}$, accurate operations in the *tree* were replaced for approximate ones in certain positions. Then, the corresponding circuit was synthesized to obtain the resulting power, area, and delay, and to observe and categorize the effect of approximations. This characterization was performed in two parts: first, the *tree* was studied using adders only, and then, the *tree* was characterized by including both adders and multipliers. Based on exhaustive experimentation [Tic98], observations, and analysis performed on this *tree* application, a generic analytical model applicable to other applications is proposed. For generalization, the three levels where operations are found are defined as input, output, and middle. The middle level is composed of all P components located between the input and output levels in a given application, as indicated in Figure 6.6. Therefore, for an approximate design composed by N functional units, where O_i are the operations of an application, as depicted in the DFG (Figure 6.6), the following models are defined:

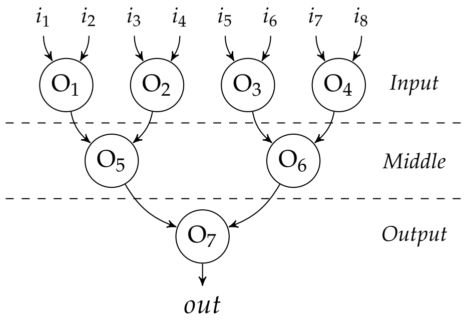


Figure 5.2: DFG of the *tree* test application. This *tree* test application was used in experiments performed for the AxME formulation.

Area: The total cell area, A_T, is approximately equal to the sum of individual cell area of each operation, A_{O_i}, composing the design:

$$A_T \approx \sum_{i}^{N} A_{O_i} \tag{5.1}$$

Power Similar to the total area, the total static power, SP_T is approximately given by the sum of the individual static power, SP_{O_i}, of each component:

$$SP_T \approx \sum_{i}^{N} SP_{O_i} \tag{5.2}$$

For the case of the total dynamic power, DP_T, a normalized estimation of the sum of individual components dynamic power values, DP_i, with respect to the dynamic power of the accurate design, DP_{Acc}, was used.

$$DP_T \approx \frac{\sum_i^N DP_i}{DP_{Acc}} \tag{5.3}$$

Delay: The delay estimation is given by the O_i operations located in the critical path of the circuit, with P operations between input and output levels:

$$D_T \approx \frac{\alpha_{O_{In}}}{\beta_{O_{In}}} * D_{O_{In}} + \sum_{i}^{P} \frac{\alpha_{O_i}}{\beta_{O_i}} * D_{O_i} + \frac{\alpha_{O_{Out}}}{\beta_{O_{Out}}} * D_{O_{Out}} \tag{5.4}$$

where α_{O_i} and β_{O_i} are factors associated with each approximate functional unit and accurate counterpart, and they are obtained as part of the characterization of such components. The α factor is defined by the position of the component in the critical path, according to the levels previously defined. β defines the contribution of the individual delay of the component to the total delay. With this delay model, it can be noticed that the effect of an approximate arithmetic circuits in the total delay depends on if it is located in the critical path of the circuit, and if so, on its relative position in the critical path.

The synthesis used for deriving AxME models, both at individual components and application level, preserves the design hierarchy and uses a medium map effort. Under other synthesis constraints, such as flattening the design or applying higher map effort, AxME models might not apply as here presented. However, these configurations can be employed once the AxME models have been used to evaluate approximate designs to obtain further optimizations.

5.3.1 Evaluation

To assess the efficiency of AxME, these models are evaluated with approximate designs for error-tolerant applications. Figure 5.3 depicts estimations obtained using AxME, alongside with values obtained through circuit synthesis and simulations, for a set of different approximate designs of a 3×3 Gaussian and a 3×3 Sobel kernels. As it can be noticed, the area estimation produced by AxME reflects significantly well the values obtained with the synthesis of the designs. Similar occurs for static power estimations. It is important to notice, that for Sobel, there is an offset between the synthesis and AxME estimation for the accurate design; however, the estimation follows the curve shape and shows a high fidelity, meaning that a design that minimizes over one metric using the proposed models can be minimizing it when considering actual synthesis. For the case of dynamic power, the synthesis values also have been normalized against the corresponding dynamic power for the accurate design implementation. In this case, AxME estimations go along with the synthesis values. AxME delay estimation is the one that differs more regarding synthesis numbers; however, AxME captures the tendency of synthesis values, as previously described.

Some approximate variants in Figure 5.3 require more area and power. This is due to the utilization of HP approximate adders, which by design consume more resources. However, for some of these variants, it can be observed that they present a smaller delay. This goes alongside with an observation noted during the experimentation leading to define AxME delay model. By replacing just accurate additions at the input or middle level with LP approximate adders, no noticeable change in the

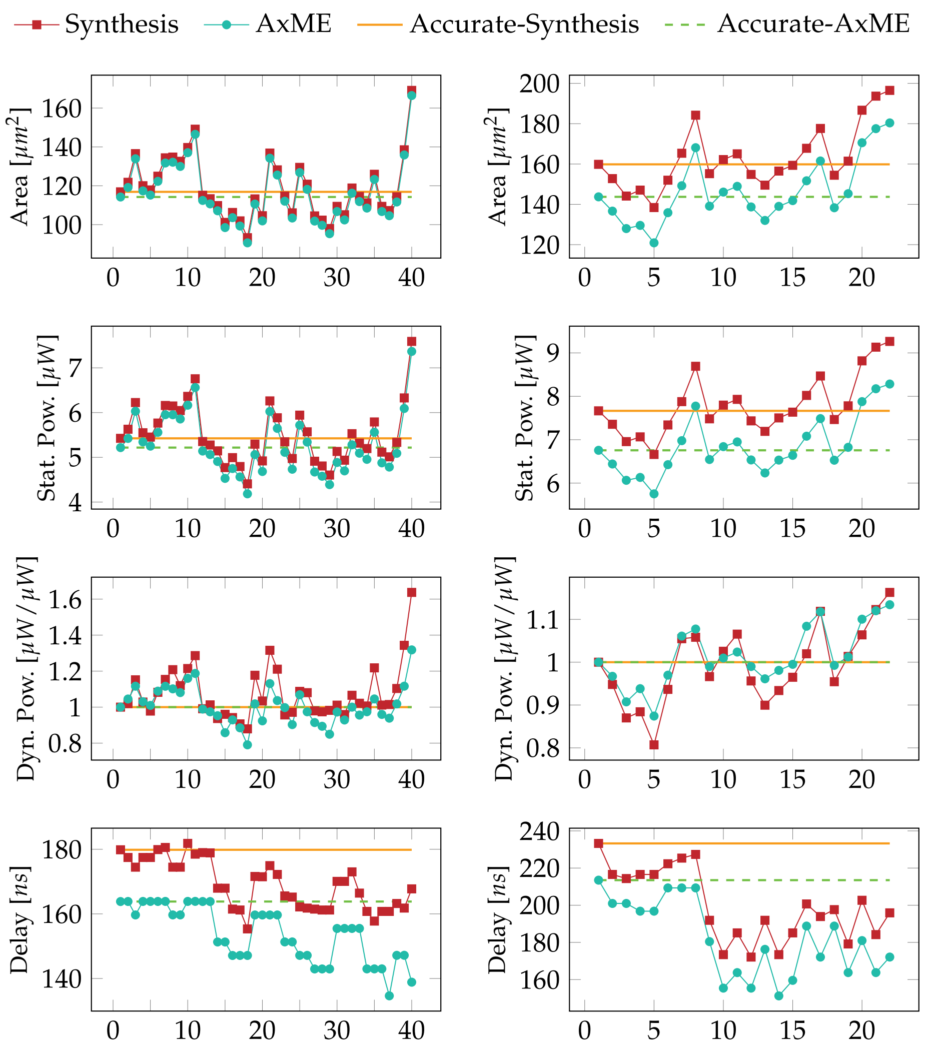


Figure 5.3: Evaluation of analytical models for resource estimation. Evaluation of the AxME analytical models for different approximate accelerator designs of Gaussian and Sobel kernels.

total delay was observed. A reduction was noticed when replacing those at the input level with LPs. On the other hand, HP approximate adders contribute to reducing the total delay when used to replaced accurate additions at the output level. In general, replacing additions, when available in the application design,

Table 5.2: Accuracy evaluation of AxME models. Evaluation was performed using the MSE metric.

Application	Area	Stat. Power	Dyn. Power	Delay
LAPLACE	0.00	0.00	0.01	8.51
SOBEL	0.43	0.02	0.01	22.12
GAUSSIAN	0.00	0.00	0.01	12.04
VECMULT	3.64	0.01	0.11	9.40

at the input with LP approximate adders and the output with HP approximate adders contribute more to reducing the delay.

The accuracy of the models in AxME was evaluated by calculating the Mean Square Error (MSE) [WB09] for a set of applications and a diverse set of 50 approximate variants per application. Table 5.2 summarizes the results obtained. As reported, for the area, static and dynamic power, AxME estimations are close to the actual number reported through circuit synthesis and simulation. However, as previously discussed, more significant differences are observed for the delay calculation, which is noticeable from Figure 5.3. This is, partly, due to the delay model does not consider the delay of operations such as division, absolute value, or shift, that might appear in the critical path of an application, as it happens in GAUSSIAN and SOBEL. This consideration keeps the model general and applicable to potentially any application, using only information about approximate adders and multipliers, and their accurate counterparts.

5.4 Design-Space Exploration with Analytical Models

With analytical models to estimate the impact in resources as approximate adders and multipliers are used in approximate accelerator designs, and considering analytical models for the accuracy estimation from Chapter 4, in this section, DSEwam is proposed, a Design-Space Exploration methodology with analytical models. DSEwam aims to find Pareto-optimal, or near Pareto-optimal, solutions for approximate accelerator designs. For a given description of an error-tolerant application and an error threshold, these solutions maximize the savings achieved due to approximations (i.e., minimizing circuit requirements) while not producing errors above the given threshold. Figure 5.4 presents the main components in DSEwam. The DSE is performed over a DFG representation of the data-path of

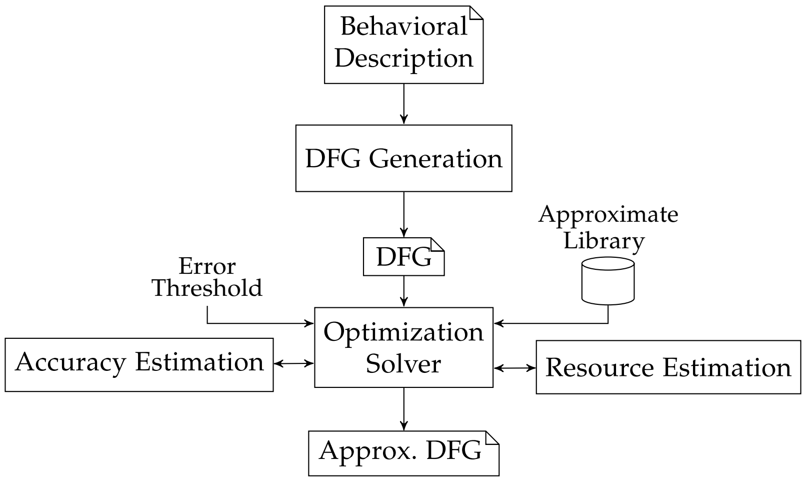


Figure 5.4: Proposed DSEwam methodology. From a behavioral description of an error-tolerant application, a DSE is performed using analytical models.

a hardware accelerator. A high-level description of an error-tolerant application is needed, from which a DFG representation is then generated. As depicted in Figure 5.4, an optimization solver is required to explore the design space built using a library of approximate functional units. This library contains the error and circuit characteristics for a set of components previously selected.

The optimization solver assesses the solutions by evaluating their accuracy and required resources. For these evaluations, the approximate functional units' parameters are used, alongside the characteristics of their accurate counterparts. For the accuracy evaluation, an error threshold is required. Accuracy and quality metrics depend on the application itself [AKK15]; thus, the error threshold is defined as a maximum tolerable value for a given accuracy metric, as reported in the literature [LHL13]. The error models of the accuracy estimation generate the output error of a potential solution, which makes it possible to evaluate if such error is below the given threshold. For the metrics estimation, models are required to estimate the resources needed by the solution. Then, for each potential solution, the optimization solver can have an estimation regarding how much is the optimization target being minimized, either area, delay, power, or energy. Once a Pareto-optimal solution is found, according to the solver parameters, a resulting DFG is provided in which the approximate operations are indicated.

```
void sobelGx(int* out, int a, int c, int d, int f, int g, int i) {
  int x1 = a + g;
  int x2 = d * 2;
  int x3 = x1 + x2;
  int x4 = c + i;
  int x5 = f * 2;
  int x6 = x4 + x5;
  int gx = x3 - x6;
  *out = gx;
}
```

```
define void @SobelGx(i32* %out, i32 %a, i32 %c, i32 %d, i32 %f,
i32 %g, i32 %i) #0 {
entry:
  %add = add nsw i32 %a, %g
  %mul = shl nsw i32 %d, 1
  %add1 = add nsw i32 %add, %mul
  %add2 = add nsw i32 %c, %i
  %mul3 = shl nsw i32 %f, 1
  %add4 = add nsw i32 %add2, %mul3
  %sub = sub nsw i32 %add1, %add4
  store i32 %sub, i32* %out, align 4
  ret void
}
```

Figure 5.5: Model for the G_x kernel. From the behavioral description of the kernel in C code (above), a IR representation is produced (below). Then, a DFG representation of the accelerator design is generated.

5.4.1 Evaluation

In the implementation of DSEwam, the behavioral description of an accurate circuit is provided as a C code function. For the DFG generation, Clang 3.5 is used to generate an optimized intermediate representation (IR), but keeping the structure of the input code, as can be seen in Figure 5.5. The IR is then traversed to create a DFG, as depicted in Figure 5.6. For resource estimation, the AxME models are used, while for the accuracy estimation, the analytical models of Chapter 4 are used. In this implementation, MED is used as accuracy metric, which can be calculated from the resulting PMF generated by the error estimation. For this evaluation, the approximate library was composed of approximate adders and multipliers previously characterized, as well as accurate circuits, as described in Section 5.3.

An algorithm based on Tabu Search (TS), as presented in Algorithm 2, is used as the optimization solver. However, other algorithms, for instance, based on genetic

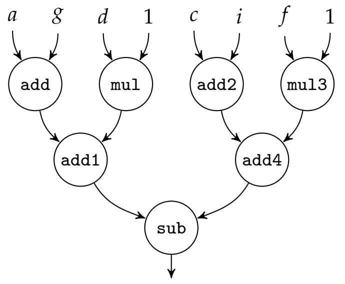


Figure 5.6: Generation of DFG from a C code description The code corresponds to a sorfware representation of the $\mathbf{G_x}$ kernel previously presented.

algorithms, can also be used as solvers, as the proposed framework is modular and provides interfaces to integrate other optimization algorithms. In the TS-based algorithm implementation, the starting solution considers all operations to be accurate functional units. An optimization step performed by the TS-based algorithm is defined as moving from one functional unit to another. Each iteration goes as follows: as a starting point, it is checked if existing taboos have expired. If this is the case, they are removed from the list. It is then iterated over all approximable operation, and the possible steps are tested for each one. The aspiration criterion for TS is used. This consideration defines that an operation is terminated after a good enough step has been found, and a defined number of further steps has been considered. To test an optimization step, it is first checked if the step is technically possible and adheres to the taboos. Then, the operation is replaced by an approximate one. At this point, error and fitness are calculated. The fitness function is defined as:

$$\begin{aligned} fitness = & w_{Delay} * (1 - Delay) + w_{DynPower} * (1 - DynPower) \\ & + w_{StatPower} * (1 - StatPower) + w_{Area} * (1 - Area) \end{aligned}$$

where weights are assigned according to the optimization goal. For the case of energy optimization, the PDP is minimized assigning values to the power and delay weights. If the solution evaluated is better than the previous solutions in one iteration, it is overwritten. If the best solution reachable in the iteration has been found, the list of taboos is then updated with the new taboos. A Pareto-optimal solution is found after the termination conditions for the algorithm are reached.

In this section, a Laplace and Sobel filter are evaluated with DSEwam, for which area and energy (PDP) are minimized, respectively. Figure 5.7 shows the design

ALGORITHM 2: TS-based algorithm used in DSEwam.

```
Input: $TabuList_{size}$;
Output: $S_{best}$;
$S_0 \longleftarrow$ PopulateStartSolution();
approximations $\longleftarrow$ GetApproximationsList();
approximableInstructions $\longleftarrow$
  GetApproximableInstructions();
$S_{best} \longleftarrow S_0$; $S_{current} \longleftarrow S_0$;
sinceLastBest $\longleftarrow 0$; iteration $\longleftarrow 0$;
while sinceLastBest $<$ MAXAFTERBEST $AND$ iteration $<$
  MAXITERATION do
    visitedInstructs $\longleftarrow 0$;
    foreach $instr \in$ approximableInstructions do
        visitedInstructs $++$;
        visitedApproxs $\longleftarrow 0$;
        foreach $approx \in$ approximations do
            visitedApproxs $++$;
            if UpdateSolutionCheckTabu($S_{new}$, $instr$, $approx$)
              then
                if HasBetterFitness($S_{new}$, $S_{current}$) then
                    continue;
                end
                else
                    delete $S_{new}$;
                end
                if visitedApproxs == MAXVISIT then
                    break;
                end
            end
        end
        if visitedInstructs == MAXVISIT $AND$ visitedInstructs $>$
          MINVISIT then
            break;
        end
    end
    UpdateTabus();
    $S_{current} \longleftarrow S_{new}$;
    if HasBetterFitness($S_{current}$, $S_{best}$) then
        $S_{best} \longleftarrow S_{current}$;
        sinceLastBest $\longleftarrow 0$;
    end
    sinceLastBest $++$;
    iteration $++$;
end
```

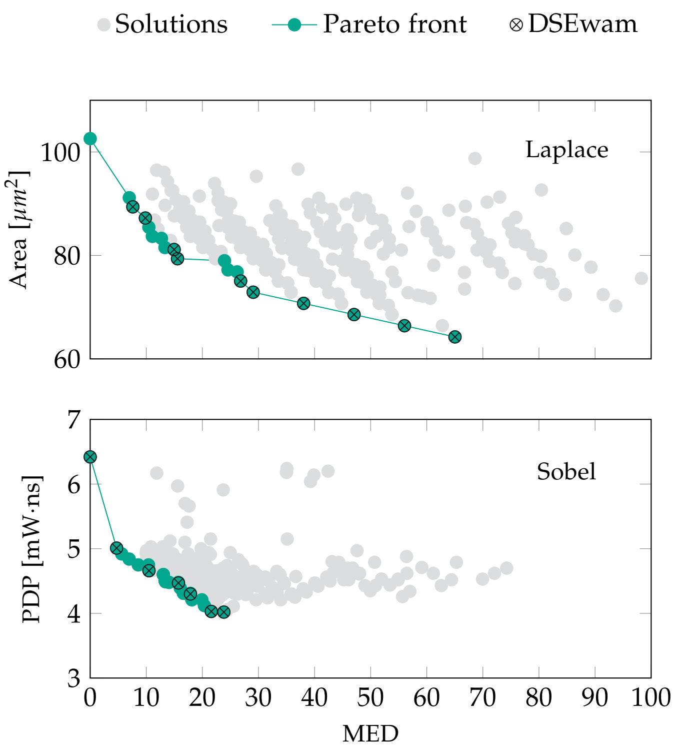


Figure 5.7: Design space for Laplace and Sobel applications. DSEwam finds Pareto-optimal solutions for a given set of approximate units.

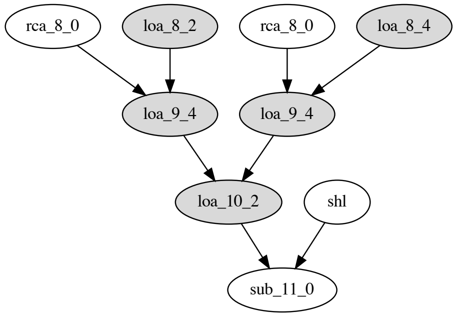
space for both applications considering an approximate library composed of about 100 components: approximate adders [Mah+10; Sha+15] and multipliers [Mah+10], and accurate counterparts. The bit-width of the inputs is provided to the DSEwam, so the exploration considers the proper bit-width for each operation and picks the approximate version accordingly. As it can be noticed, DSEwam is capable of finding the Pareto-optimal solutions for different MED constraints. Consider the case of the exploration for the Laplace filter. For instance, defining the error threshold to MED = 10 and MED = 40, DSEwam finds solutions that minimize area and keep the error within the given threshold, with resulting MED = 9.84 and MED = 38.05, respectively. The corresponding approximate DFGs for these two cases are depicted in Figure 5.8. As it can be noticed, for this Laplace designs, different approximate additions are used for each case (nodes in gray), and their name and configuration are given.

Solutions proposed by DSEwam were further evaluated. For example, the resulting DFG for the Laplace filter and MED = 40, as presented in Figure 5.8b, was

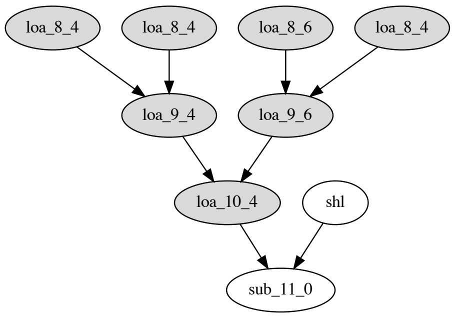
Table 5.3: Evaluation of solutions provided by DSEwam. For this case, the solution provided as Pareto-optimal in fact was dominant with respect to other solutions closed.

Solution	MED	AxME [μm^2]	Synth. [μm^2]
Accurate	0.00	102.58	101.15
Pareto-optimal	38.05	**70.73**	**69.30**
Variant 1	35.82	72.89	71.47
Variant 2	36.69	77.12	75.69
Variant 3	39.07	78.59	77.17
Variant 4	40.31	75.05	73.63

implemented. However, also other solutions close to the Pareto frontier and the Pareto-optimal solution provided by DSEwam were implemented. Table 5.3 reports the information regarding the MED value and the actual area estimated by AxME and obtained through synthesis. Characterization of the approximate library was performed with a NanGate 15nm technology library and using Synopsys Design Compiler and PrimeTime. Once implemented by hand, the approximate designs were evaluated with the same technology library and tools. As can be noticed, the Pareto-optimal solution provided by DSEwam minimizes the required area in comparison with other near solutions while keeping a MED below 40. This, in turn, allows validating the proposed AxME models. Similar evaluations were performed for different applications, error constraints, and optimization goals. OThe results reported that about 90% of the cases, the solutions provided by DSEwam were Pareto-optimal, and the remaining 10% of the cases were near Pareto-optimal.



(a) MED = 10



(b) MED = 40

Figure 5.8: Resulting DFGs for different accuracy targets. In the DFG is indicated the corresponding accurate or approximate arithmetic circuit, with its correspondin configuration.

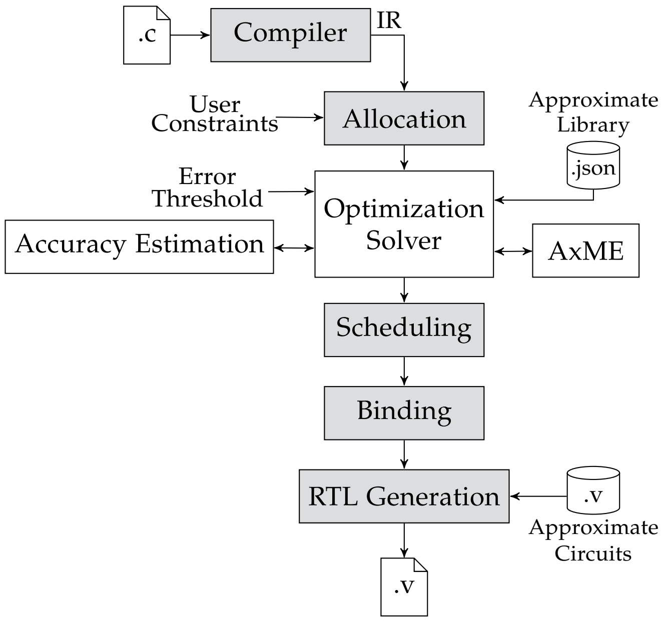


Figure 5.9: Overview of the AxHLS approach. Original steps in LegUp were adapted to include the proposed DSEwam and handle information about approximations.

5.5 Approximate High-Level Synthesis

This dissertation proposes AxHLS, an approximate HLS approach to automatically generate approximate accelerator designs. For this, approximate arithmetic circuits are used, and the academic version of the LegUp tool [Can+13] is extended to include DSEwam, the proposed design-space exploration with analytical models. Figure 5.9 depicts the main stages in LegUp, including the DSEwam methodology. As can be seen, LegUp takes as input a C program or a specific function according to the user constraints to generate a hardware description.

The initial steps of the LegUp tool are preserved. This corresponds to the compilation of the C input source code and the corresponding allocation of resources of the operations. However, to enable the AxHLS approach, it was necessary to adapt the scheduling, binding, and RTL generation stages to support and handle additional information regarding approximations selected by DSEwam as part of the proposed solutions. During the allocation, the required hardware to implement the circuit is determined. We consider that for each operation in the LLVM IR, a functional unit is assigned. By doing so, resource sharing is avoided, which increases the complexity of the error analysis, and the need for additional multiplexers.

At LLVM IR, each function to be approximated is expanded with two types of metadata. Moreover, the instructions receive information about the time step of the scheduling in which they are executed, performed during the scheduling stage. On the other hand, they receive a unique ID. This is required to transfer information regarding which operation has been replaced for an approximate one. Once the allocation has been performed, the DSE is triggered. LegUp waits until DSEwam provides a solution for the available approximate functional units and the given error threshold to continue with the following stages. The LLVM module, which contains the metadata of the approximation generated by DSEwam, is then read. Based on the ID metadata created, the approximation metadata is transferred to the corresponding instructions. The scheduling stage was adjusted to recognize this metadata, as well as the binding and the RTL generation stage, so the required approximate operations can be used and connected.

During the RTL generation, LegUp iterates over the instructions of the function to be synthesized and creates the required functional units. In the case of an approximate operation, the existing Verilog modules of the approximate arithmetic circuits must be instantiated. This is depicted in Figure 5.9 as the Approximate Circuits, a collection of the Verilog implementations for the approximate components used in the DSE, and corresponds to those in the Approximate Library. Internally, the function in charge of creating the functional units checks whether the instruction under consideration should be approximated. If so, the name of the instance is first created, and, based on the name, the path of the Verilog implementation of the approximation used is added to the list of Verilog files to be imported. Then the necessary wiring for the inputs and outputs is generated.

5.5.1 Evaluation

The proposed AxHLS is evaluated with error-tolerant applications used in image processing, Sobel and Sharpen filter. Approximate accelerators for four different error thresholds for each application were generated. The MED error metric was used as the accuracy metric and generated approximate accelerators aim to reduce the required energy. Table 5.4 summarizes the results obtained, reporting energy reduction with the PDP metric. The AxHLS implementation, include the DSEwam methodology in the LegUp HLS tool (version 4.0). The resulting accelerators were evaluated using Synopsys Design Compiler and PrimeTime with a NanGate 15nm technology library.

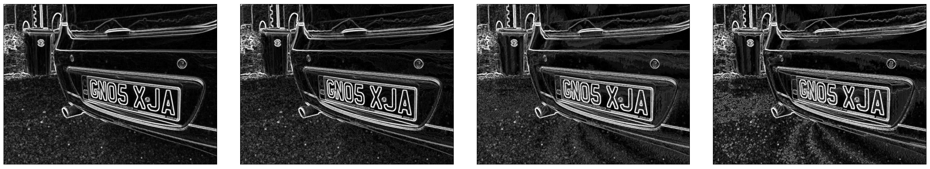
As reported in Table 5.4, with a threshold of MED $= 5$ for both applications, the generated approximate accelerators reduce their required energy by nearly 30%. As expected, a higher allowed error (a higher MED) causes more savings, achieving

Table 5.4: Results for approximate accelerators generated. Approximate accelerators for Sobel and Sharpen filters generated with the AxHLS approach.

MED	PDP [μW·ns]	PDP (Norm.)	PSNR [dB]	SSIM
		Sobel		
0	6420.57	1	∞	1
5	4461.26	0.69	29.86	0.84
11	3985.95	0.62	22.44	0.65
16	3912.11	0.61	19.52	0.58
23	3679.43	0.57	18.50	0.54
		Sharpen		
0	13202.22	1	∞	1
5	9212.01	0.70	36.00	0.98
15	8526.78	0.65	24.50	0.91
20	8120.10	0.62	21.61	0.84
30	7471.01	0.57	17.99	0.76

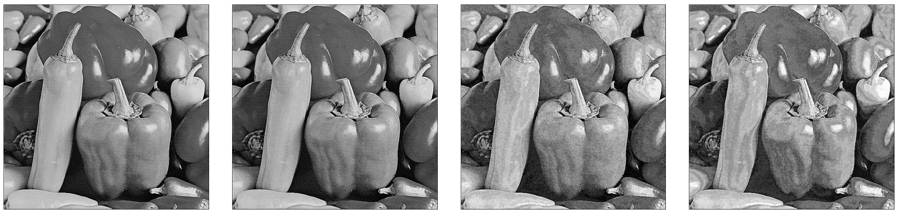
a 43% energy reduction for MED = 23 and MED = 30 for Sobel and Sharpen, respectively. However, to have a better perspective of the effect of these accuracy thresholds in the quality of the results, the approximate accelerators generated were evaluated with testing images. Figures 5.10 and 5.11 present the resulting images of applying the Sobel and Sharpen accelerators to the *plate* and *peppers* test images. With values for PSNR and SSIM metrics, commonly used to assess the quality in images, the results for these quality metrics are also reported in Table 5.4. Even when the acceptable quality depends on the application and the processed data, it is noticeable that higher MED thresholds introduced visual errors. This occurs for the accelerators generated for a MED equal to 11 and 16 for Sobel filter, and as depicted in Figure 5.10c and 5.10d; and for MED equal to 20 for Sharpen filter, and depicted in Figure 5.11d.

Although applications evaluated in reported work differ from those here presented [LJG17], the AxHLS approach achieves more savings than precision scaling. In general, when applying precision scaling, the work in [LJG17] reports about 10% energy reduction with PSNR values around 25dB. As reported in Table 5.4, for PSNR of 29.86dB and 36.00dB, the AxHLS approach can generate an approximate design with an energy reduction of 30%. In [LJG17], precision scaling is combined with voltage scaling to achieve similar energy savings, but the quality of results is reduced to around 20dB. These different voltages for different precisions in components make the final implementation more complex, while in the proposed approach, the approximate design synthesis considers a single voltage source. On



(a) Exact **(b)** MED = 5 **(c)** MED = 11 **(d)** MED = 16

Figure 5.10: Quality evaluation for Sobel approximate accelerators. Resulting images for applying approximate Sobel filter designs to the *plate* test image.



(a) Exact **(b)** MED = 5 **(c)** MED = 15 **(d)** MED = 20

Figure 5.11: Quality evaluation for Sharpen approximate accelerators. Resulting images for applying approximate Sharpen filter designs to the *peppers* test image.

the other hand, due to the generality that proposes the DSEwam methodology, approximate adders and multipliers with reduced precision can also be included as part of the approximate library.

5.6 Summary

This chapter presented a set of analytical models, AxME, to estimate the circuit metrics when using approximate arithmetic circuits, particularly approximate adders and multiplier in approximate designs. Despite not being entirely accurate, AxME allows the design space exploration of approximate accelerator designs and obtaining Pareto-optimal, or near Pareto-optimal, solutions, as demonstrated with the evaluation of the proposed DSEwam methodology. The DSEwam methodology has been released as an open-source contribution, so the analytical models, optimization solver, and approximate library can be used and further modified and improved.[2]. Also, this methodology was integrated into an HLS tool to generate approximate accelerators from behavioral level C code descriptions of error-tolerant

[2] DSEwam is an open-source contribution and it is available at `https://git.scc.kit.edu/CES/DSEwam`

applications. The results for the approximate accelerator generated show that even with a small accuracy degradation on the accuracy (MED = 5), about 30% of energy reduction can be achieved.

I know there's a proverb which that says 'To err is human,' but a human error is nothing to what a computer can do if it tries.

Agatha Christie

6 Balancing Error Correction

The use of approximate accelerators must ensure defined accuracy levels. Hence, the errors generated due to approximations must remain within a defined bound, required for the application to produce *good-enough* results. With this consideration, and as presented in Chapter 5, approximate accelerators are designed to meet a given accuracy for an error metric. However, as described in Section 2.2.3, the errors produced by an approximate accelerator depend on the input data [Mah+16]. This data may differ from the one used during the design, making the accelerator to have unacceptable errors at run time. On the other hand, the defined tolerable error threshold can be dynamically changed. In both cases, those undesired errors need to be corrected to achieve the desired accuracy.

State-of-the-art approaches address this issue by re-computing, at the software level, those accelerator invocations that produce unacceptable errors. For this, lightweight, pre-trained predictors are used to estimate if an accelerator invocation will produce an error beyond the defined accuracy threshold for a given set of input data. Nevertheless, software re-computations reduce approximate computing benefits, especially when input data variations are high at run time.

This chapter presents ECAx, a methodology to explore and apply fine-grained, selective error correction in approximate accelerators to balance the costs associ-

The content of this chapter is based on the work originally published in [CSH19]:
J. Castro-Godínez, M. Shafique, and J. Henkel: "ECAx: Balancing Error Correction Costs in Approximate Accelerators". In: *ACM Transactions on Embedded Computing Systems (TECS)*, 18, 5s, Article 48 (October 2019), ESWEEK Special Issue; and presented during the *2019 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '19)*.

ated with accuracy control. This methodology reduces the required coarse-grained correction, exact re-computations performed by the host processor, to meet a defined accuracy constraint, finding a balance between software- and hardware-level error correction, while not significantly reducing the benefits obtained due to the approximations, such as the speedup achieved by an approximate accelerator with respect to an accurate one. This methodology is particularly useful for approximate accelerators built with approximate functional units, the type of approximate accelerators targeted in this dissertation, achieving up to 20% performance gain compared to reported approaches in the literature.

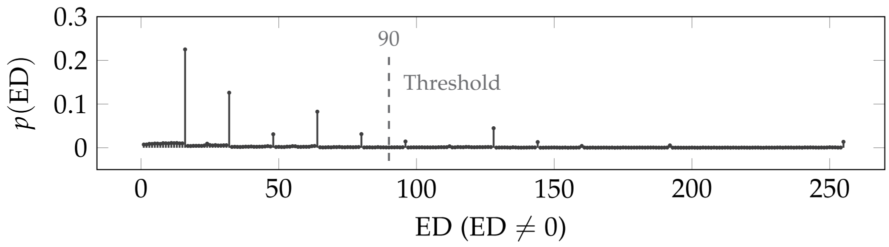
6.1 Motivation

The following two examples motivate the need for a robust and general solution to the challenge of correcting errors outside a defined limit when using approximate accelerators built with approximate adders and multipliers.

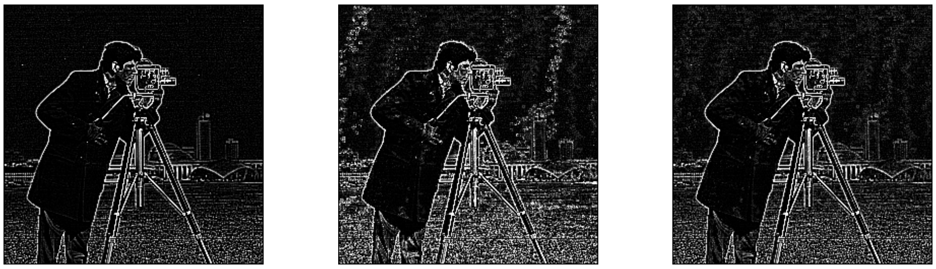
Performing re-computations. Consider a Laplace filter for image processing as an approximate accelerator. Figure 6.1a depicts its error distribution. By applying this filter to a test image, such as *cameraman*, and considering no error correction implemented (no EC), the resulting image has a PSNR of 15.4 dB (see Figure 6.1c), in comparison to the image produced by the accurate version of the accelerator (see Figure 6.1b). Although high ED values in the error distribution have a very low probability (see Figure 6.1a), they cause noticeable noise (see Figure 6.1c in comparison with Figure 6.1b).

An arbitrary error threshold is defined to $\mathrm{ED} = 90$ as an accuracy constraint for this approximate Laplace accelerator. To keep the errors at the output within this threshold, it is required to detect all errors with $\mathrm{ED} \geq 90$ once they appear, and correct them by exactly re-computing those accelerator invocations. This correction is performed using the host processor (see Section 2.3), as proposed in previous work [Khu+15; Mah+16; Wan+16]. By doing so, the quality of this test image is improved to 22.3 dB (with EC, see Figure 6.1d). However, this implies an overhead that limits the benefits of approximations. For this example, a $2.1\times$ speedup is achieved due to approximations, but this changes to $1.7\times$ when this error correction is applied.

Correcting errors at the component level. As described in Section 2.1, the errors produced by LP and HP approximate adders are very different in magnitude and



(a) Error distribution for an approximate Laplace filter.

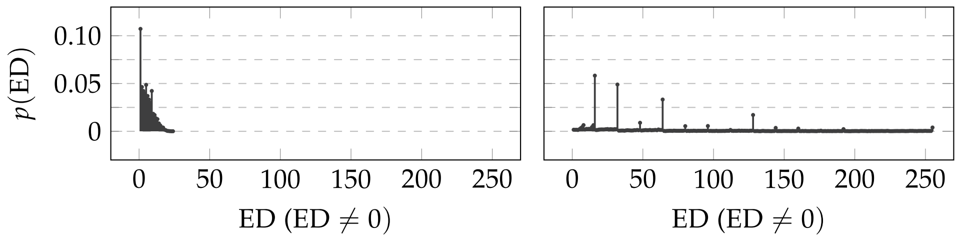


(b) Accurate **(c)** No EC (15.4 dB) **(d)** With EC (22.3 dB)

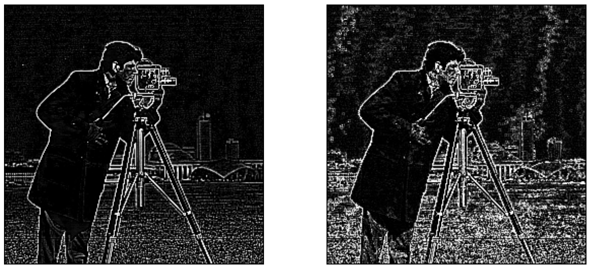
Figure 6.1: Re-computations for accuracy improvement. This motivational example shows the effect of error correction (EC) applied to an approximate Laplace filter as a threshold of $ED = 90$ is set. For the *cameraman* image, there is a quality improvement from 15.4 dB to 22.3 dB in the PSNR metric for such accuracy threshold.

frequency. Therefore, they have a different effect on the output accuracy and quality of results. Consider that all additions in the previous Laplace filter are replaced with LP approximate adders, in one case, and with HP approximate adders, in another. As shown in Figure 6.2a and 6.2b, using HP approximate adders results in a smaller overall ER for the accelerator compared with LP approximate adders (0.14 vs. 0.52). However, the presence of high ED values significantly degrades the accuracy and quality of the results, despite their low probability. For instance, for the *cameraman* test image, the filter built with LP approximate adders produces an image with a PSNR = 34.08 dB (see Figure 6.2c), while the one made with HP approximate adders has a PSNR = 13.37 dB (see Figure 6.2d). From this, it can be inferred that correcting the errors with a high ED value produced by HP approximate adders in an approximate accelerator design, even with a low ER, can have a more significant impact on the quality than correcting those caused by LP approximate adders. This goes in-line with the notion of *fail small, fail rare* in approximate computing [Chi+13].

However, correcting those errors represents a compromise. Figure 6.3a shows the ED values for a particular 8-bit HP approximate adder, where each ED $\neq$ 0



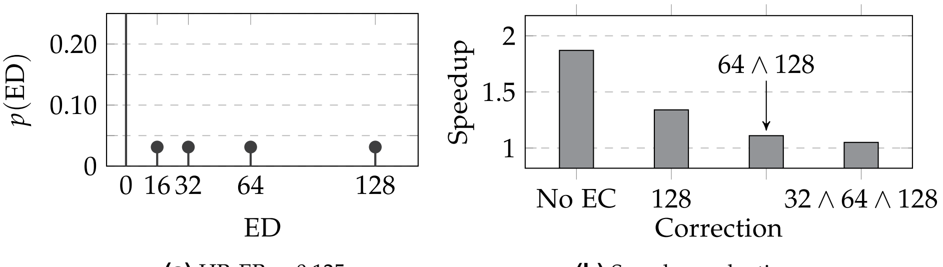
(a) LP, ER = 0.52 **(b)** HP, ER = 0.14



(c) LP, PSNR = 34.0 dB **(d)** HP, PSNR = 13.7 dB

Figure 6.2: Errors produced by LP and HP approximate adders in a Laplace filter. The type of approximate adders used in an approximate accelerator, in this case, a Laplace filter, directly affects accuracy and quality. In this case, despite a lower ER, errors from HP approximate adders have a more significant impact on the resulting image quality.

has a probability of occurrence below 5%. Without any error correction, this HP approximate adder is 1.87× faster than his accurate counterpart based on RCA, as depicted in Figure 6.3b. Consider the error detection and correction mechanism proposed for HP approximate adders [Maz+16]. By detecting and correcting the most significant error in this HP approximate adder, ED = 128, the speedup is



(a) HP, ER = 0.125 **(b)** Speedup reduction

Figure 6.3: Effect of error correction in speedup reduction of an approximate adder. For this 8-bit HP approximate adder, the delay savings achieved by approximations are reduced as errors are corrected.

reduced to 1.34×, as the critical path of the adder is increased to allow the detection and correction of this error. When ED = 64 is also corrected, the speedup goes down to 1.11×. As can be noticed from Figure 6.3, to preserve the benefits achieved due to approximations in an HP approximate adder, as in this case, the speedup, *all errors could not be detected and corrected*.

6.2 Accuracy Control

As depicted in Section 6.1, there are two main approaches to control the accuracy of results when using approximate accelerators. A coarse-grained approach corrects unacceptable errors at the accelerator level using software re-computations. A fine-grained approach corrects errors produced by the approximate arithmetic circuits used to build the approximate accelerator.

6.2.1 Correction at the Accelerator Level

At the accelerator level, two main techniques have been reported for monitoring and controlling the accuracy of results of at run time. The first technique proposes to periodically measure the error of an accelerator by comparing its output against an exact software-based computation performed by the host processor. In case the error is above a defined threshold, a process of re-calibration and adjustment is performed to improve the quality in subsequent accelerator invocations [Sam+13; BC10].

A second technique proposes to implement light-weight, pre-trained error predictors to forecast if the invocation of an approximate accelerator would produce an unacceptable error for a particular input data set [Khu+15; Mah+16; Wan+16]. Figure 6.4 depicts the general outline of this technique. If the predictor estimates an error above a defined threshold, a recovery signal indicates the host processor to recover the last invocation by re-computing it accurately at the software level.

The benefits due to approximations decrease every time the host processor re-computes an accelerator invocation to satisfy an accuracy constraint. For instance, taking into account this technique for the motivation example previously described, and considering an accuracy threshold of ED = 90, 18.22% of the accelerator invocations would need to be re-computed to meet this constraint, which reduces the gains in execution time, as described, and energy consumption improvement.

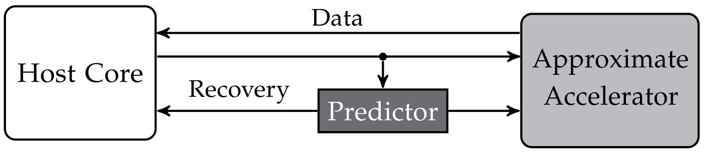


Figure 6.4: Outline of the prediction-based accuracy control. The predictor estimates errors bigger than acceptable and triggers the recovery of the accelerator invocation with an accurate re-computation at the software level (diagram based on [Mah+16]).

6.2.2 Correction at the Approximate Circuit Level

As discussed in Section 2.1, many approximate circuits, such as adders and multipliers, have been reported in the literature. For the case of many HP approximate adders' designs, techniques have been proposed to detect and correct their errors. In most cases, the error detection is based on checking the carry propagation (*cp*) and the carry output (*co*) of the sub-adders used to perform the addition, as depicted in Figure 6.5. For the error correction, the techniques depend on the approximate circuit design. For instance, for GeAr [Sha+15] and ACA-II [KK12] approximate adders, extra clock cycles are required to accomplish the correction; in the first by changing the lower part of the inputs, while in the latter by using an incrementor. For GeAr, a simple, yet effective method to correct the errors has been proposed [Maz+16]. The method proposes to toggle the output bit's values to compensate for the error, starting from the sub-adder where the carry propagation is missed. This is achieved using additional inverters and multiplexers into the approximate adder design, in the Error Correction block, as shown in Figure 6.5. This correction increases the adder's critical delay path when an error is corrected but avoids extra clock cycles.

The error correction technique described in [Maz+16] can be applied to other HP approximate adders, such as ACA-I [VBI08], ETAII [ZGY09], and GDA [Ye+13], as these HP approximate adders can be implemented as configurations of GeAr [Sha+15]. As described in Section 2.1, and depicted in Figure 6.5, the GeAr approximate adder performs an N-bit addition using multiple sub-adders of smaller size. For the case of Figure 6.5, 3 4-bit adders. The most significant R-bits of the sub-adders are considered as resultant bits and are used in the result. The remaining P-bits, called previous bits, are used to estimate the carry propagation to the upper bits. In this example, both R and P are 2.

Error detection and correction have been proposed for LP approximate adders [Dut+16]. However, as discussed in Section 6.1, the errors produced by LP approximate adders do not significantly degrade the accuracy of results, and the resulting quality, as the ones produced by HP approximate adders (see Figure 6.2).

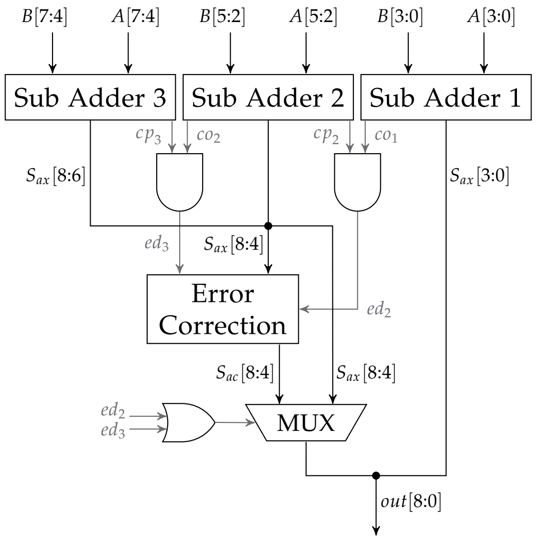


Figure 6.5: Error detection and correction outline for HP approximate adders. The detection is performed by monitoring carry propagation mispredictions. The correction is achieved by flipping the output bits from the lowest misprediction point (based on [Maz+16]).

Other reported work had proposed a self-healing approach, where approximate modules are placed together in an accelerator data path aiming to cancel their errors [MHS18; Gil+18]. These extra modules produce errors with an inverse magnitude that helps to compensate for the errors generated and to reduce their propagation to the output. Also, multispeculative adders to correct potential errors at the output have been proposed, particularly when functional units are shared in a high-level synthesis approach [Del+13]. The methodology presented in this chapter differs from these approaches, as the goal is to add the smallest overhead possible to the accelerator to improve its performance while correcting unacceptable errors, balancing the correction performed at the software and hardware level. It also considers approximate designs built using existing approximate functional units from the state-of-the-art, which is the primary motivation of this dissertation.

6.3 Early Error Correction

This section explores the effect of error correction at approximate adder level on the required re-computations and the effect on the overall accelerator delay. For this purpose, consider a set of additions arranged as depicted in Figure 6.6,

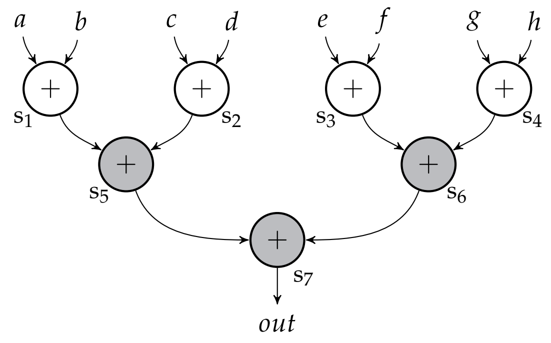
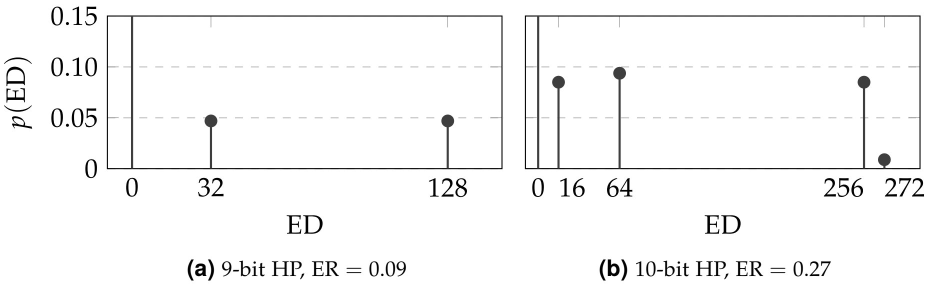


Figure 6.6: Early error correction test implementation. This tree of adders is used to explore error correction's effect at the adder level on accelerator's delay and re-computations needed. Adders are replaced with LP approximate adders at the input while the rest are substituted by HP approximate adders (shaded).



(a) 9-bit HP, ER = 0.09 **(b)** 10-bit HP, ER = 0.27

Figure 6.7: Error distributions for a 9-bit and a 10-bit HP approximate adders. Most significant errors of these approximate adders are corrected in this early error correction exploration example.

which is a typical structure found, for instance, in image processing kernels [XS17]. Consider that all adders have been replaced by approximate ones. An 8-bit LP approximate adder replaces s_1, s_2, s_3, and s_4. The other adders are replaced by HP approximate adder (shaded in the figure): s_5 and s_6 by a 9-bit, and s_7 by a 10-bit HP approximate adder. The bit-length is selected to optimize the design, in this case, knowing that all inputs ($\{a, b, \ldots, h\}$) are 8-bit long. The error distribution for these HP approximate adders is presented in Figure 6.7, while the resulting error distribution for this tree of adders is shown in Figure 6.8a. As described in Chapter 5, a combination of LP and HP approximate adders as presented for this tree of adders example provides a smaller delay than just using HP approximate adders to replace all accurate ones.

The delay of any approximate accelerator is compared to an exact implementation built with RCAa throughout this chapter. Most of the approximate adders proposed in the literature have been derived from the RCA, and their characteristics, regarding the accuracy and required resources, are compared with respect to this exact adder [Maz+16].

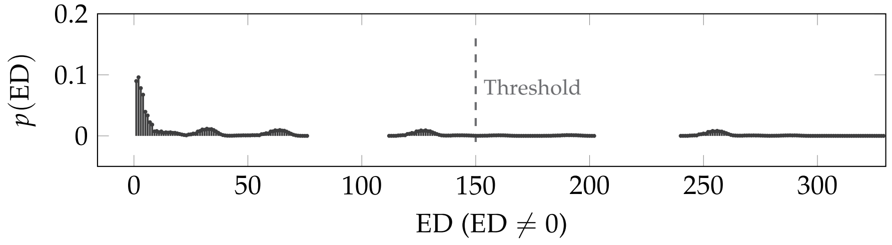
For the case of this tree of adders (Figure 6.6), an arbitrary error threshold is set to 150, which means that all accelerator outputs (at *out*) with an associated ED $\geq$ 150 would need to be re-computed. In Figure 6.9, the results for three error correction scenarios are presented. When no error correction is performed (No EC), and considering the defined accuracy threshold, 12.15% of all accelerator invocations would need to be re-computed. This means that the host processor would need to re-compute about 12 out of each 100 executions performed by the accelerator to correct those errors with ED $\geq$ 150. In comparison with the exact implementation, this approximate version presents a speedup of 1.29$\times$. Speedup S is calculated as [HP12]:

$$S = \frac{L_{acc}}{L_{app}} \tag{6.1}$$

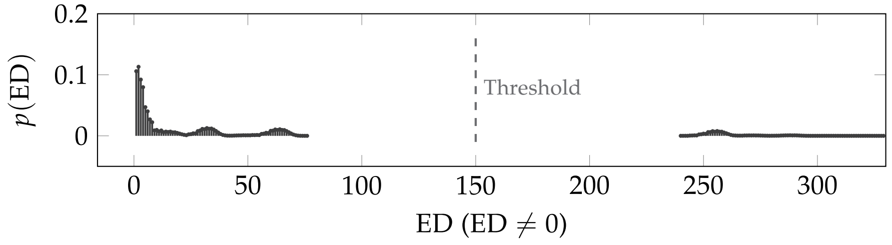
where L_{acc} and L_{app} correspond to the delay estimation for accurate and approximate accelerators, respectively.

A first error correction scenario, 9*b*, corresponds to the correction of the most significant error produced by the two 9-bit HP approximate adders, which is $ED = 128$ (see Figure 6.7a). As shown in Figure 6.9, the re-computations required are reduced, in this case by about 3%, from 12.15% to 9.34%, of the total accelerator invocations. Simultaneously, the delay of the accelerator increases, and it is then 1.23$\times$ faster than the accurate version. A second error correction scenario, 10*b*, considers the correction of the most significant error generated by the 10-bit HP approximate adder, which is ED = 256. Even though ED = 272 is the biggest ED value for this adder, as depicted in Figure 6.7b, it is generated when ED = 256 and ED = 16 are simultaneously produced; so correcting ED = 256 eliminates ED = 272. In this case, the required re-computations are reduced by 10%, from 12.15% to 2.10%, but the accelerator is 1.15$\times$ faster regarding the accurate one. Figure 6.8b and 6.8c show the remaining ED values in the error distribution once the correction has been considered, which in both cases is less than originally (Figure 6.8a), as expected.

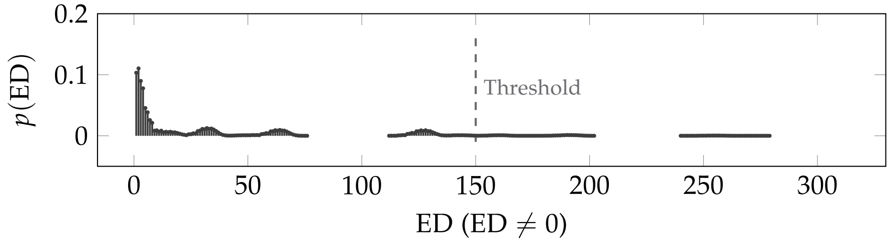
A third error correction scenario considers applying the previous two error corrections for the 9-bit and 10-bit HP approximate adders concurrently. As depicted in Figure 6.9, no re-computation is required to satisfy the defined error threshold



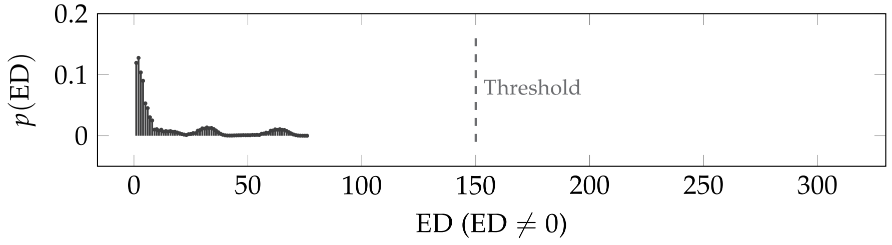
(a) No EC



(b) Correction scenario 9*b*

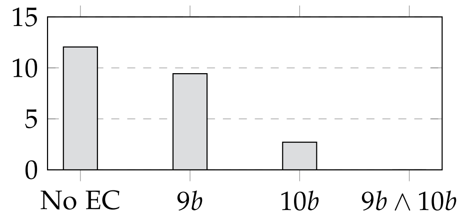


(c) Correction scenario 10*b*

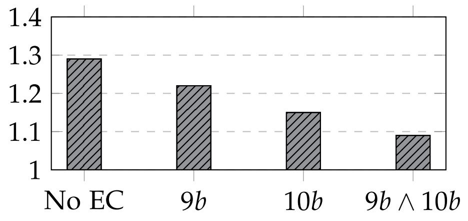


(d) Correction scenarios $9b \wedge 10b$

Figure 6.8: Error distributions for different error correction scenarios. Each correction scenario has a different impact on the remaining errors generated above the defined accuracy threshold and on the accelerator's delay.



(a) Re-computations (%)



(b) Accelerator Speedup

Figure 6.9: Reduction of re-computations and accelerator speedup due to selective error correction. Each correction scenario has a different impact on reducing re-computations needed and on the accelerator speedup when the errors are corrected at the approximate adder level.

by applying this correction scenario. However, the delay increases more, and the accelerator is then only 1.09× faster regarding the accurate version. Furthermore, when inspecting the error distribution for this correction scenario (see Figure 6.8d), the maximum ED is 76.

As noticed, different correction scenarios provide different trade-offs regarding the reduction of re-computations required and the resulting delay of the accelerator, and hence a change in the speedup achieved. *It is necessary to count on a method that allows the exploration of an approximate accelerator design to determine which error correction scenarios, in this case at approximate adder level, would produce the most significant benefit without losing the gains of approximations*. For this, two critical remarks need to be considered:

Correct more significant errors first: From the previous case, it can be observed that *by correcting the higher ED from the HP approximate adders, it is possible to achieve a higher reduction of the required re-computations*. Correcting ED = 256 from the 10-bit HP approximate adder not only reduces that specific error at the adder level (and also ED = 272, in this case). Correcting such error also reduces many more errors generated by combining this high error with others generated and propagated by different approximate functional units, such as the LP approximate adders. This correction reduces them to appear at the approximate accelerator's output, as depicted in Figure 6.9, with the reduction of the required re-computations.

Correct all adders at the same level: *If HP approximate adders at the same level are equal, correcting the highest ED for all of them reduces more required re-computations while experiencing the same delay overhead*. Consider the HP approximate adders at s_5 and s_6 for the tree of adders in Figure 6.6, which are the same and are located at the same level, which means, at the same operations distance from the output.

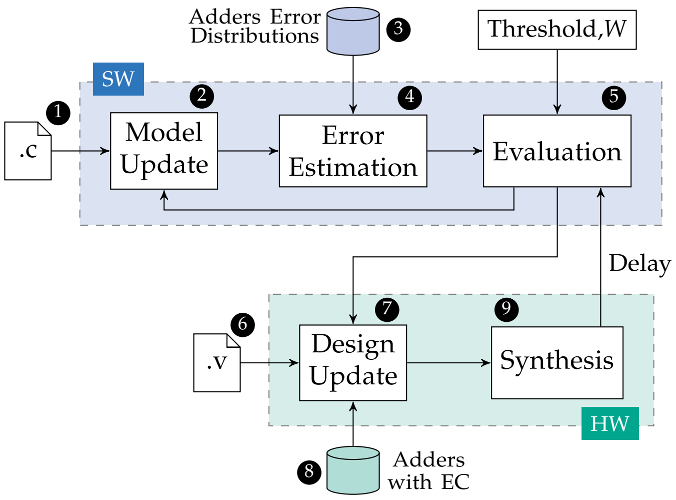


Figure 6.10: Main components in ECAx methodology. A software model of the accelerator is used to explore the error correction at the adder level. The hardware implementation is modified accordingly to determine the effect of the error correction on the delay.

Suppose that just the most significant error in s_5 is corrected. In that case, the worst-case delay of the accelerator will be the same if the correction is performed in both HP approximate adders because the critical path will be provided by the HP approximate adder where the correction occurs. As previously presented, if $ED = 128$ is corrected in both s_5 and s_6, the required re-computations decrease in 2.71%. If the correction takes place in just one of them, the re-computation reduction achieves just a 1.6%.

6.4 ECAx Methodology

This section presents ECAx, a methodology for selective Error Correction in Approximate accelerators, aiming to reduce the required exact re-computations while meeting a defined accuracy of results. This methodology balances the costs associated with error correction performed at the software and hardware level so that savings obtained by approximations are not significantly reduced while accuracy constraints are satisfied. This goal is achieved as the most significant errors in an approximate accelerator are corrected; this is, those with an ED higher than a defined tolerable threshold and produced by HP approximate adders. Figure 6.10 presents a diagram with the main components of this methodology.

```
void tree(int* out, int a, int b, int c, int d, int e, int f,
int g, int h) {
  #pragma approx loa wr 8 4
  int s1 = a + b;
  #pragma approx loa wr 8 4
  int s2 = c + d;
  #pragma approx loa wr 8 4
  int s3 = e + f;
  #pragma approx loa wr 8 4
  int s4 = g + h;
  #pragma approx gear wrp 9 2 3
  int s5 = s1 + s2;
  #pragma approx gear wrp 9 2 3
  int s6 = s3 + s4;
  #pragma approx gear wrp 10 2 2
  int s7 = s5 + s6;
  out = s7;
}
```

Figure 6.11: Example of an annotated software model. This code corresponds to the annotated software model for the tree of adders example, used in Section 6.3.

To explore the errors to be corrected at the adder level and determine the effect at the accelerator level, a software model, *swM*, of the approximate accelerator is required, and its hardware implementation at RTL, *hwD*. ECAx methodology comprises two stages: the first at the hardware level and the second at the software level. Each of these stages has a specific purpose. With the hardware one, estimations of the impact of the selective error correction in the delay are obtained. With the software one, the effect of error correction in the output error distribution is explored, as well as the number of re-computations saved is analyzed. With this, the error correction scenario is evaluated to determine if a proper balance of correction cost per software and hardware is achieved. The software model is annotated to indicate which exact additions are replaced for approximate ones, and for which LP or HP approximate adder (see ❶ in Figure 6.10). Figure 6.11 shows an example of the annotated code for the previous tree of adders example, presented in Section 6.3. The error distribution of the approximate accelerator is estimated using the methodology presented in Chapter 4 ❹. This estimation requires error models for the approximate adders, including models for the HP approximate adders, when errors are corrected ❸. The delay of the accelerator is characterized using its hardware implementation ❻ and a circuit synthesis tool ❾. A baseline error distribution and delay estimation are required, with no correction applied, to compare the cases further when corrections are performed.

The ECAx methodology proceeds as depicted in Algorithm 3. Each HP approximate adder in *swM* is replaced by a model, HP*, of the same HP approximate adder

ALGORITHM 3: Steps in ECAx methodology

```
Input: $swM$, $hwD$, $th$, $W$, $L_{exc}$;
Output: $hwD^*$;
1  recomp[] $\longleftarrow$ 0; speedup[] $\longleftarrow$ 0;
2  dist $\longleftarrow$ 0; $i \longleftarrow 0$;
3  foreach $HP \in swM$ do
4      ModelUpdate($swM$, $HP^*$);
5      dist $\longleftarrow$ ComputeErrorDistribution($swM$);
6      recomp[$i$] $\longleftarrow$ SavedRecomp(dist, $th$);
7      DesignUpdate($hwD$, $HP^*$);
8      $L_{app} \longleftarrow$ Synthesis($hwD$);
9      speedup[$i$] $\longleftarrow L_{exc}/L_{app}$;
10     $i \longleftarrow i+1$;
11 end
12 KP($i$, $W$);
13 Function KP($i$, $W$):
14     if speedup[$i$] $> W$ then
15         res $\longleftarrow$ KP($i-1$, $W$);
16     end
17     else
18         s1 $\longleftarrow$ KP($i-1$, $W$);
19         s2 $\longleftarrow$ recomp[$i$] + KP($i-1$, $W-$ speedup[$i$]);
20         res $\longleftarrow$ max($s1$,$s2$);
21     end
22 return
```

with a correction for the most significant ED. If two HP approximate adders present the same maximum ED, the methodology selects the level where more HP approximate adders are present. At the software level, a data flow graph is generated from *swM*, which allows finding which adders are at the same level ❹. This update of the model comprises replacing the annotation (pragma) for one that indicates a HP approximate adder with error correction considered ❷. Then, a new error distribution is computed ❹. With this new error distribution and considering a given error threshold *th*, the maximum tolerable ED, the number of saved re-computations can be calculated and stored ❺. The notion of "saved re-computations" refers to the re-computations that will not be performed by the host processor, as the error correction at HP approximate adder level is reducing errors with ED $\geq th$. Similarly, the design of the accelerator, *hwD*, is updated with the corresponding implementation of the HP approximate adder with error correction ❼, and a new delay value, L_{app}, is estimated using a synthesis tool ❾. With this value, the speedup is calculated, following (6.1), and stored.

It is necessary to notice that the design of the approximate accelerator, *hwD*, is considered as given, including its approximations through approximate functional units. This methodology performs an exploration of the selective error correction on top of the existing design, without changing the exact and approximate functional units defined. ECAx explores the cost balance achieved as per software error correction is replaced by selective and fine-grained per hardware correction.

In most cases, not all errors can be corrected for a given threshold, as this might significantly harm the speedup achieved by the approximate accelerator. Following a 0-1 Knapsack approach, the evaluation phase ❺ considers potential combinations of individual correction scenarios to find the least reduction on the accelerator speedup as re-computations are reduced. For this 1-0 Knapsack formulation, the objective is to maximize:

$$\sum_{i=0}^{n} v_i x_i$$

subject to:

$$\sum_{i=0}^{n} w_i x_i \leq W$$

where $x_i \in 0, 1$, v_i corresponds to the accelerator re-computations saved due to an i-th error correction scenario at HP approximate adder level, and w_i corresponds to the speedup reduction of the approximate accelerator due to such correction (recomp[] and speedup[] in Algorithm 3). The maximum speed reduction for the approximate accelerator, W, is defined as a proportion to its original speedup. For instance, $W = 50\%$ means that no more than this percentage of reduction in the speedup is desired. The result provided by Algorithm 3 allows to modify the hardware design and to generate a final implementation, *hwD**, with the error correction included.

In any case, the goal is to find the selective error correction that most reduces the required re-computations while producing less impact on the delay possible, and hence, balance the error correction costs. Each step in the exploration is characterized in terms of error and delay. It then provides a complete scenario of the possible design-correction solutions and their implications for the accelerator's delay. In an extreme case, all most significant errors will be corrected for all HP approximate adders if the delay degradation is within the W parameter provided. Moreover, this methodology also considers the impact of the delay degradation in the overall speedup an application can achieve due to the approximate accelerator's use. In any case, the maximum application speedup possible is reached while the accuracy is contained within a defined error threshold.

6.5 Evaluation

The ECAx methodology has been implemented in a fully automated manner, so the error correction exploration is performed automatically, as described in Section 6.4. The accelerator design and its software model need to be provided. Additionally, the corresponding models for the error distribution analysis and the HP approximate adders with selective error detection and correction implemented are also required. An error threshold for which a balance in the cost of software and hardware correction is desired and the maximum desired reduction in the approximate accelerator's speedup is also input information required by ECAx.

In the scope of this chapter, this methodology was tested with six error-tolerant applications: Sobel, Gaussian, Finite Impulse Response (FIR), and Laplace filters, Sum of Absolute Differences (SAD), and Discrete Cosine Transform (DCT), and different approximate accelerator designs for these applications were evaluated, using different configurations of open-source approximate adders, LOA [Mah+10] as LP approximate adder, and GeAr [Sha+15] as HP approximate adder. Synopsys Design Compiler was used to obtain delay estimations of the approximate accelerators using the TSMC 65 nm technology library.

The proposed methodology does not have limitations in using other technology libraries, as the selective error correction is performed at the RTL level. Also, although the applications tested in the scope of this chapter present, in general, a similar data processing structure (a chain or tree of additions), other applications where approximate adders are used can take advantage of the proposed methodology. Other approximate arithmetic functional units with selective and fine-grained error detection and correction can be integrated to this methodology, as long as the error correction does not degrade the savings achieved due to the approximations.

Figure 6.12 presents an exploration of different correction scenarios for approximate accelerator designs of each of the applications mentioned above. As it can be noticed, as the percentage of required re-computations is reduced, as an effect of the selective error correction, the speedup of the accelerator decreases. *The re-computations percentage corresponds to the percentage of the total accelerator invocations.* For these cases, W's value for the Knapsack formulation was defined up to 100% of the approximate accelerator's original speedup, allowing an exhaustive evaluation to be carried out.

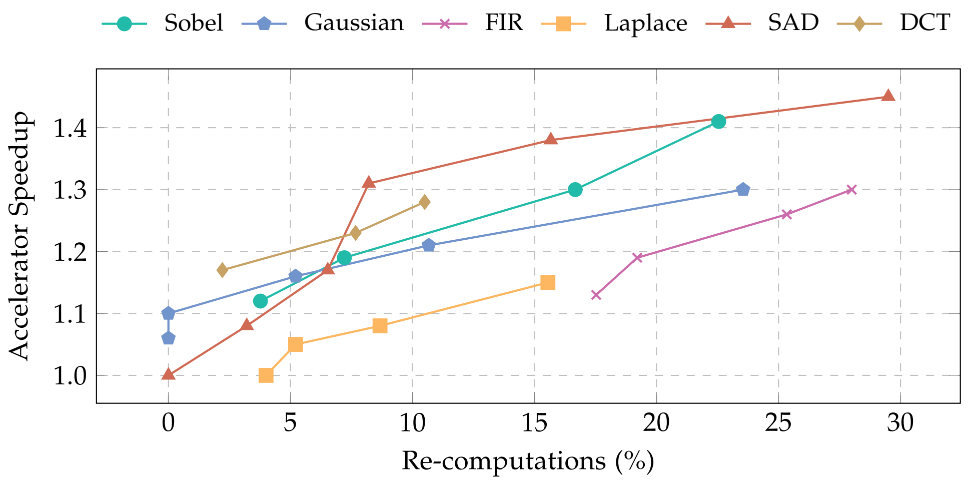


Figure 6.12: Error correction and its effect on re-computations and speedup for error-tolerant applications. As the number of required re-computations is reduced, due to selective error correction at the adder level, the accelerator's speedup decreases.

Consider the particular case of the SAD accelerator. If a smaller value of W is defined, for instance, 15%, which means that up to 15% lost in the speedup is desired due to the correction, 21% of the required computations would be reduced with a change of the accelerator speedup from 1.45× to 1.31×.

From the information in Figure 6.12, two other cases are interesting to notice. For the Laplace filter, the speedup could be reduced to 1.0×, which means that no benefit in the delay reduction due to approximations is achieved. For that case, even 4% of re-computations are still needed to satisfy the accuracy constraint. By limiting the maximum speedup degradation allowed, the ECAx methodology can reduce the re-computations required while containing the accelerator speedup's degradation. Another case to point out is the Gaussian filter, which presents two potential correction scenarios that can be used to achieve 0% re-computations needed. However, each of them presents a different accelerator speedup. One is 1.10×, while the other is 1.06×. In any case, the one with a higher speedup is desired.

The approximate accelerators with selective error correction at HP approximate adder level, alongside an error predictor for the accelerator, were integrated into a Tiger MIPS processor (as available in the LegUp tool [Can+13]), in an architecture similar to the one presented in Figure 6.4. As an error predictor, a linear model-based predictor with a table-based predictor, as proposed in [Wan+16], were implemented. For each application, the predictor was trained for a given error

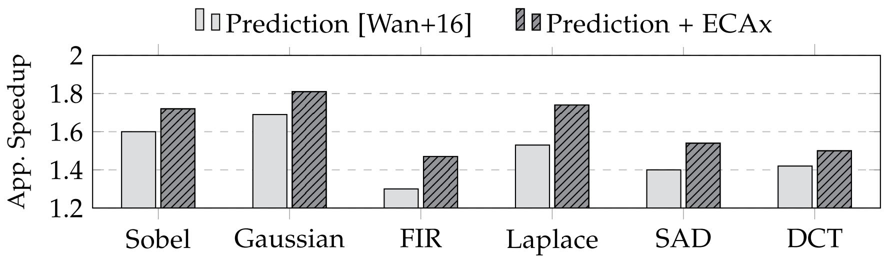


Figure 6.13: Application speedup improvement. An improvement in the speedup is obtained due to selective error correction at the approximate adder level alongside a prediction-based approach.

threshold using random input vectors from a uniform distribution, similar to the accelerator's input.

Modelsim simulations were used to estimate the required clock cycles for each implementation. With these values, a comparison of the applications' overall speedup improvement due to the reduction of re-computations can be performed. Figure 6.13 summarizes the best results obtained for designs explored corresponding to the applications tested. As depicted in Figure 6.13, in general, it is possible to obtain a speedup improvement when adding selective error correction to the HP approximate adders to state-of-the-art prediction-based methods for accuracy control. For a Laplace filter design, an improvement of 21% was achieved. For that particular case, applying error correction to the most significant ED of two HP approximate adders at the same level, and for an error threshold of ED $\geq$ 60, the required re-computations were reduced to less than 2%.

As already discussed, when applying the selective error correction, as proposed by the ECAx methodology, this correction impacts the approximate accelerator's delay. Although the detection is always active and the correction occurs on-demand, just when an error is detected, it is considered that both detection and correction take place for estimating the delay of the resulting approximate accelerator. Thus, no delay variance is considered in determining the application speedup improvement.

Figure 6.14 shows the area and power overhead required for each of the solutions presented in Figure 6.13. As can be noticed, up to 5% area overhead is required for applying selective error correction at the approximate adder level compared to the original area of the approximate accelerator. Regarding the total power overhead, it was found between 3% and 7%. The increment in the area produces an increment in static power consumption, but also, as these sections perform the error detection and correction, they consume dynamic power.

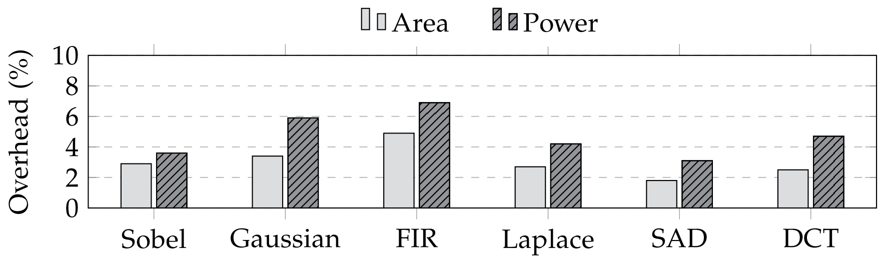


Figure 6.14: Area and power overhead. For the solutions presented in Fig. 6.13, up to 5% area overhead is required for applying selective error correction at the approximate adder level. Power overhead ranges between 3% and 7%.

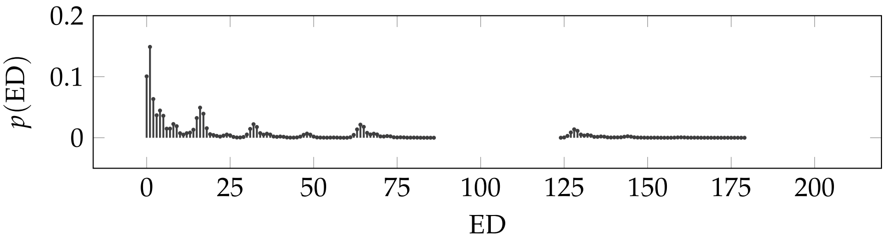


Figure 6.15: Error distribution for a Gaussian filter example. For this example, errors appear with higher ER for ED values lower than 80. However, those with high ED value do trigger re-computations even for a high accuracy degradation allowed.

A more detailed case for a Gaussian filter as an approximate accelerator is presented here. Particularly, it is evaluated the effect of selective error correction for one specific accelerator design while the defined error threshold changes. Figure 6.15 depicts the error distribution for this particular design. As shown, the errors are more present in a range between ED = 1 and ED = 80, with some errors between ED = 125 and ED = 175 but with a low ER. *According to a defined error threshold, even errors with a low probability of occurrence will trigger re-computations.*

Figure 6.16 shows how the required re-computations change as the defined error threshold is modified. As expected, a higher percentage of the accelerator invocations are required to be re-computed as the tolerable error is small. For instance, for a threshold of ED = 20, about 30% of the accelerator invocations require correction. As the error threshold is increased, this is a higher ED tolerable, there is a reduction in the re-computations needed. It is interesting to notice that for an error threshold between ED = 80 and ED = 120, the required re-computations stay constant, as there is no increment in the number of errors for those values, as shown in Figure 6.15.

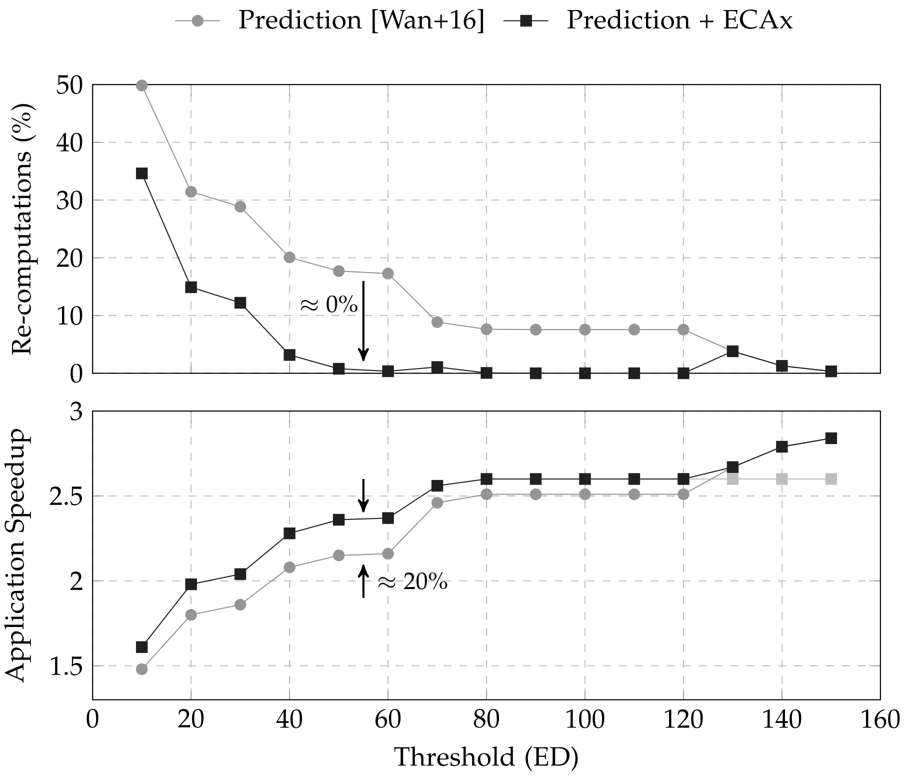
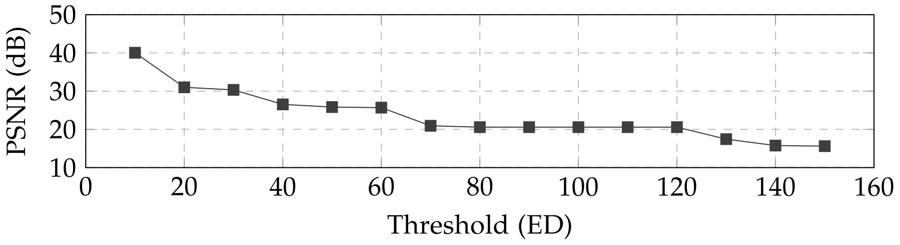


Figure 6.16: Effect of selective error correction for a Gaussian filter. As the error threshold changes, a reduction in the required re-computations is experienced, even eliminating the need of re-computations.

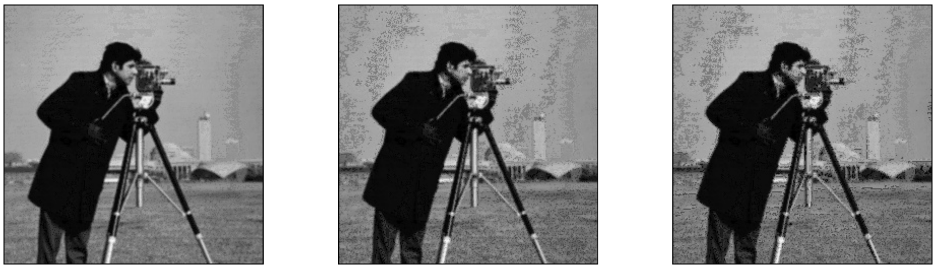
The required re-computations are reduced by applying this proposed methodology with a predictor from the state-of-the-art to this Gaussian example. As shown in Figure 6.16, for error thresholds between ED = 50 and ED = 120, the required re-computations are reduced to basically 0%, compared to just using a prediction-based approach, which it varies from 7% up to 17%. For each error threshold, a retraining of the predictor was considered.

As the error threshold changes, so do the application's speedup, as depicted in Figure 6.16. For most of the cases, by applying the proposed methodology, an improvement in the speedup of the application is achieved. For the range between ED = 90 and ED = 120, an increment of about 10% is accomplished, while for the rage between ED = 40 and ED = 60, up to 21% speedup improvement is obtained.

Two particular remarks for this Gaussian filter approximate accelerator are worthy of highlighting. First, as this methodology was applied for an error threshold of



(a) Quality evaluated as PSNR metric for different error threshold.

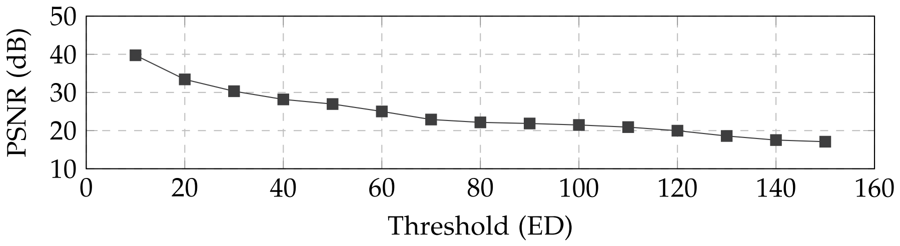


(b) ED = 20 (31.00 dB) **(c)** ED = 50 (25.83 dB) **(d)** ED = 80 (20.58 dB)

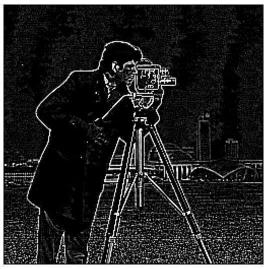
Figure 6.17: Resulting quality for a Gaussian filter example. Quality improves as selective error correction, applied with the proposed methodology, is performed for different error thresholds.

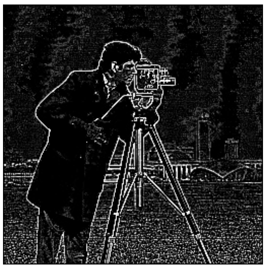
$ED \geq 130$, no re-computations were saved. As ECAx explores these cases, it is possible to achieve 0% of re-computations needed; however, this would reduce the application's speedup as the accelerator's delay would be unnecessarily increased for such a small of re-computations needed. As indicated in Figure 6.16 (faded line), if error correction at the adder level is applied for such an error threshold, it would potentially decrease the speedup up to 20%. The proposed methodology can estimate such an impact, and it does not apply any selective error correction to preserve the application speedup as high as possible.

Second, for ED $\leq$ 60 as an error threshold, additional error correction at HP approximate adder level was required. Between ED = 70 and ED = 120, ECAx found that applying one error correction to the most significant ED for a particular HP approximate adder, it was possible to reduce the required re-computations to nearly 0% and improve the application's speedup. However, for an error threshold between ED = 10 and ED = 60, the exploration performed found that an additional correction was required for another HP approximate adder to reduce the re-computations. This improved the application's speedup, and even though by



(a) Quality evaluated as PSNR metric for different error threshold.


(b) ED = 50 (27.22 dB)


(c) ED = 100 (22.30 dB)

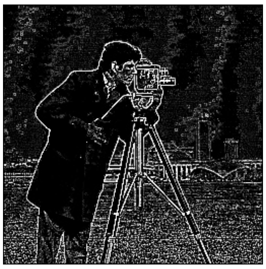

(d) ED = 150 (18.01 dB)

Figure 6.18: Resulting quality for a Laplace filter example. As for the Gaussian filter example, there is quality improvement for a Laplace filter as selective error correction is applied for different error thresholds.

increasing the selective error correction, the delay of the accelerator was decreased, the overall speedup of the application was raised.

Also, it is important to notice that for this particular design, an error threshold of ED $=$ 10 implies a significant reduction in the application speedup due to the high number of re-computations required. Although our approach improves the speedup, a redesign of the accelerator would benefit more of the approximations by limiting the required accelerator re-computations from the design. This methodology does not consider modifying a given approximate accelerator design but could potentially be extended to include it.

Figure 6.17a summarizes the effect of the ECAx methodology in the quality of the images processed by this approximate Gaussian filter for different error thresholds. As expected, the quality of the image improves as the error threshold is reduced. For this approximate accelerator, and considering the *cameraman* test image, significant quality improvement is observed for error thresholds of 40, 50, and 60, from PSNR = 15.41 when no correction is applied to PSNR = 25 dB. This mainly occurs when the required re-computations are near to 0%, and a maximum application speedup is achieved, nearly 20%, compared to an only prediction-based approach (see

Figure 6.16). Figures 6.17b, 6.17c, and 6.17d depict different output quality for the *cameraman* test image at different error thresholds explored with the presented methodology. In Figure 6.18, quality results for the Laplace filter case, described in Section 6.1, are presented. As can be observed, by an error threshold around 50, the noise in the image is not noticeable.

6.6 Summary

This chapter has presented ECAx, a methodology to selectively explore and apply error correction in approximate accelerators. ECAx has demonstrated, for the first time, the benefits of selective, fine-grained, and low-overhead error correction when using HP approximate adders in approximate accelerator designs. The methodology presented proposes that *by using selective error correction at the approximate functional unit level, it is possible to reduce the required exact re-computations performed by the host processor to keep the approximate accelerator accuracy at the output to a defined error threshold*. This implies to find a balance between the cost of correcting errors by software and hardware: reducing the required correction by software while increasing the correction by hardware should not compromise the benefits obtained due to approximations. Reducing these re-computations makes it possible to improve the performance, as correction per software requires more time. The evaluation performed shows that applying ECAx alongside a state-of-the-art error predictor for approximate accelerators improves up to 20% the application speedup compared to just using a predictor-based approach for accuracy control proposed in previous work. Moreover, this methodology has shown its potential to provide good trade-offs to improve the application speedup as the error threshold changes for a specific approximate design.

Education is not something you can finish.

Isaac Asimov

7 Conclusion

This chapter concludes this dissertation with a summary of the work presented, and it provides promising future work directions.

7.1 Dissertation Summary

With the coming forth of *Approximate Computing* as an energy-efficient design paradigm, many contributions have been made at the circuit level, particularly proposing several approximate designs for adders and multipliers. The diversity of these components impose a challenge when using them to design approximate accelerators to speed up the execution of error-tolerant applications. From this scenario, this dissertation has addressed the challenge of automated design of approximate accelerators built with approximate arithmetic circuits. To achieve it, this dissertation has presented different methodologies to enhance the design space exploration and the generation of register-transfer level implementations of approximate accelerators.

First, this dissertation introduced two frameworks to generate approximate arithmetic circuits (Chapter 3). One focuses on the generation of approximate adders and multipliers from designs reported in the literature, for different bitwidth and approximate configuration. The other focuses on netlist transformation techniques for approximate logic synthesis. In both cases, the characterization performed for the approximate circuits generated, regarding the error and circuit metrics (area, delay, and power), is fundamental for the analytical models for accuracy and resource estimations proposed within this work.

Then, a compiler-driven methodology for accuracy estimation was introduced (Chapter 4). A set of analytical models was defined to enable fast error propagation estimations for approximate designs, reducing the dependency of exhaustive simulations, and allowing to assess the accuracy degradation when using approximate arithmetic circuits in accelerator designs. These models consider only the error distribution of each approximate component used in the design to perform the propagation estimations. The compiler-based methodology introduced allows quick estimations by using software-level models of the accelerators and replacing the accurate additions and multiplications by approximate ones through custom pragma directives.

Later, a complete framework for the generation of approximate accelerators from high-level descriptions was presented (Chapter 5). Analytical models for resource estimations were introduced to complement those for accuracy estimations. These additional models estimate the required area, delay, power, and energy of approximate accelerators' designs without performing circuit synthesis. With these models, a novel design space exploration methodology was proposed to find Pareto-optimal solutions for a given error-tolerant application and set of approximate arithmetic circuits. This exploration methodology was integrated into a high-level synthesis tool to generate accelerators from C code implementations of error-tolerant applications automatically.

Finally, a method for balance the cost associated with error correction for approximate accelerators was presented (Chapter 6). Reported accuracy control mechanisms employ re-computations, performed by the host processor, to correct those results produced by an approximate accelerator with an error above a defined accuracy threshold. This undermines the performance improvements achieved due to the approximations used. This dissertation presented a methodology for fine-grained, selective error correction at the component level, applied to high-performance approximate adders. Although this error correction reduces the accelerator delay and slightly increases the area and power consumption, the overall balance improves the speedup of error-tolerant applications as the number of required re-computations is significantly reduced.

Most of the contributions presented in this dissertation have been made open-source, allowing their use by the scientific community and further expansion and improvement.

7.2 Future Work

In the following, this dissertation shares three critical ideas for further research directions motivated, in part, by the contributions of this work.

Bridging the gap between accuracy and quality. As presented in Section 2.2.3, there is a mismatch between accuracy and quality in approximate computing, starting from how these terms are understood. Even what really quality is must be reconsidered for approximate computing [SBC15]. While most of the contributions reported in the literature focus on designing accelerators, for instance, for a given accuracy threshold, the quality of results is a by-product issue. In this sense, it is necessary to develop top-down methodologies to bridge quality requirements into specific accuracy constraints for a given application and input data set.

From the accuracy perspective, current works have proposed top-down approaches. For instance, the accuracy degradation is distributed or translated to a component-level accuracy [VMS19; ASP20]. This improves the selection, or generation, of approximate components for a specific application, that at the end, will sum up to satisfy the final required accuracy constraint. On the other hand, the on-demand utilization of different instances of approximate implementations with fixed accuracy has been proposed [AGH20]. So, for a particular application, and according to a workload and accuracy requirement, one approximate instance is selected to meet the expected accuracy, while the other instances are switch off to save power.

However, to bring it the design of approximate accelerators to an applications' quality-driven level, proving statistical guarantees for quality at the output, it is required to close the bridge between current accuracy-driven design methodologies for approximate accelerators and application quality constraints.

Cross-layer accuracy and quality estimations. Since half a decade ago, the notion of cross-layer approximate computing has been hanging around in the scientific community [Sha+16; Pan+16]. Even though many efforts have been made to count on disciplined approximate computing techniques at individual abstraction layers of the computing stack, there are still many challenges to effectively have cross-layer design methodologies.

One essential aspect towards true cross-layer approximate computing is the ability to perform precise accuracy and quality estimations across the layers. This means to be able to quantify the effect of approximations taking placed at different layers, and their final accumulative effect in the application's accuracy and quality. For instance, it is required to understand how approximations performed at the circuit level, by approximate additions and multiplications in an approximate accelerator, have an impact on the applications' accuracy, when simultaneously approximations are applied at software level [CSH20b] and in combination with approximations applied at memory level [San+15]. Although approximations have been applied at different abstraction layers trying to exploit optimizations visible at each level [XS17], this has been done to design approximate circuit implementations with no cross-layer error estimation considered.

Quality-driven design for end-to-end error-tolerant applications. One particular area where little has been contributed is the development of methodologies to design end-to-end approximate implementations of error-tolerant applications. Most of the existing work has focused on individual sub-parts of a computing system, such as the memory or the computing subsystem; this last one, for instance, using approximate accelerators. However, more has to be investigated in this direction to fully take advantage of approximate computing as an energy-efficient design paradigm.

Currently, three contributions have shown promising results for end-to-end approximate implementations. In [RR18], a smart camera system example is presented where approximations are applied to sensing, memory, processing, and communication subsystem, exploiting approximations at each stage synergistically. More recently, in a similar direction, the case for approximate inference at the edge has been depicted [GRR20]. In [Has+18], the example of an iris-scanning system is presented. In this case, a combination of approximate techniques at the software and hardware level is applied to the focus assessment, iris segmentation, normalization, and encoding stages to significantly reduce the required runtime and energy while meeting the accuracy requirement of the iris encoding as defined by industry standards.

Bibliography

[AKK15] I. Akturk, K. Khatamifard, and U. R. Karpuzcu. "On Quantification of Accuracy Loss in Approximate Computing". In: 2015 (cit. on p. 60).

[AGH20] T. Alan, A. Gerstlauer, and J. Henkel. "Runtime Accuracy-Configurable Approximate Hardware Synthesis Using Logic Gating and Relaxation". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020, pp. 1578–1581. DOI: `10.23919/DATE48585.2020.9116272` (cit. on p. 97).

[AH18] T. Alan and J. 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` (cit. on pp. 7, 26).

[AKL16] H. A. F. Almurib, T. N. Kumar, and F. Lombardi. "Inexact Designs for Approximate Low Power Addition by Cell Replacement". In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2016, pp. 660–665. DOI: `10.3850/9783981537079_0042` (cit. on pp. 8, 21).

[ASP20] G. Ansaloni, I. Scarabottolo, and L. Pozzi. "Judiciously Spreading Approximation Among Arithmetic Components with Top-Down Inexact Hardware Design". In: *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Ed. by F. Rincón, J. Barba, H. K. H. So, P. Diniz, and J. Caba. Cham: Springer International Publishing, 2020, pp. 14–29. DOI: `10.1007/978-3-030-44534-8_2` (cit. on p. 97).

[AMP18] M. Awais, H. G. Mohammadi, and M. Platzner. “An MCTS-based Framework for Synthesis of Approximate Circuits”. In: *International Conference on Very Large Scale Integration (VLSI-SoC)*. 2018, pp. 219–224. DOI: 10.1109/VLSI-SoC.2018.8645026 (cit. on pp. 53, 54).

[AHS17] M. K. Ayub, O. Hasan, and M. Shafique. “Statistical Error Analysis for Low Power Approximate Adders”. In: *54th Design Automation Conference (DAC)*. 2017, pp. 1–6. DOI: 10.1145/3061639.3062319 (cit. on pp. 39, 53).

[BC10] W. Baek and T. M. Chilimbi. “Green: A Framework for Supporting Energy-conscious Programming Using Controlled Approximation”. In: *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI ’10. Toronto, Ontario, Canada: ACM, 2010, pp. 198–209. DOI: 10.1145/1806596.1806620 (cit. on p. 75).

[BIM16] M. Barbareschi, F. Iannucci, and A. Mazzeo. “An Extendible Design Exploration Tool for Supporting Approximate Computing Techniques”. In: *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*. 2016, pp. 1–6. DOI: 10.1109/DTIS.2016.7483888 (cit. on p. 54).

[BMH14] K. Bhardwaj, P. S. Mane, and J. Henkel. “Power- and Area-Efficient Approximate Wallace Tree Multiplier for Error-Resilient Systems”. In: *15th International Symposium on Quality Electronic Design (ISQED)*. 2014, pp. 263–269. DOI: 10.1109/ISQED.2014.6783335 (cit. on pp. 2, 7, 10).

[Bro+15] J. Broc, P.-E. Gaillardon, L. Amarú, J. J. Murillo, K. Palem, and G. De Micheli. “A Fast Pruning Technique for Low-Power Inexact Circuit Design”. In: *2015 IEEE 6th Latin American Symposium on Circuits & Systems (LASCAS)*. 2015, pp. 1–4. DOI: 10.1109/LASCAS.2015.7250448 (cit. on p. 28).

[CCC14] L. Cabrera-Quirós, R. Campos-Gómez, and J. Castro-Godínez. “Critical steps in camera pose estimation: an evaluation using LTI-LIB2 library”. In: *Revista Tecnología en Marcha* (2014), pp. 60–69. DOI: 10.18845/tm.v0i0.1656 (cit. on p. xx).

[Can+13] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson. “LegUp: An Open-source High-level Synthesis Tool for FPGA-based Processor/Accelerator Systems”. In: *ACM Trans. Embed. Comput. Syst.* 13.2 (2013), 24:1–24:27. DOI: 10.1145/2514740 (cit. on pp. 54, 66, 87).

[Cas19] J. Castro-Godínez. “Approximate Software for Accurate Hardware”. In: *NiPS Summer School, OPRECOMP Summer of Code Initiative*. 2019 (cit. on p. xxi).

[Cas+20a] J. Castro-Godínez, H. Barrantes-García, M. Shafique, and J. Henkel. “AxLS: An Open-Source Framework for Netlist Transformation Approximate Logic Synthesis”. In: *3rd Workshop on Open-Source EDA Technology (WOSET), co-located with ICCAD '20*. 2020 (cit. on pp. xx, 7, 17).

[Cas+21] J. Castro-Godínez, H. Barrantes-García, M. Shafique, and J. Henkel. “AxLS: A Framework for Approximate Logic Synthesis based on Netlist Transformations”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* (2021) (cit. on p. xix).

[Cas+18] 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` (cit. on pp. xix, 18, 19, 37, 53).

[CH18] J. Castro-Godínez and J. Henkel. “Error Propagation Estimation on Approximate Designs with Compiler-Driven Support”. In: *3rd. Workshop on Approximate Computing (AxC '18), co-located with the IEEE European Test Symposium 2018*. 2018 (cit. on p. xxi).

[Cas+20b] J. Castro-Godínez, D. Hernández-Araya, M. Shafique, and J. Henkel. “Approximate Acceleration for CNN-based Applications on IoT Edge Devices”. In: *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*. 2020, pp. 1–4. DOI: `10.1109/LASCAS45839.2020.9069040` (cit. on p. xx).

[Cas+20c] J. Castro-Godínez, J. Mateus-Vargas, M. Shafique, and J. Henkel. “AxHLS: Design Space Exploration and High-Level Synthesis of Approximate Accelerators using Approximate Functional Units and Analytical Models”. In: *2020 IEEE/ACM 39th International Conference on Computer-Aided Design (ICCAD)*. 2020. DOI: `10.1145/3400302.3415732` (cit. on pp. xix, 18, 27, 51).

[CSH19] J. Castro-Godínez, M. Shafique, and J. Henkel. “ECAx: Balancing Error Correction Costs in Approximate Accelerators”. In: *ACM Trans. Embed. Comput. Syst.* 18.5s (2019). DOI: `10.1145/3358179` (cit. on pp. xix, 71).

[CSH20a] J. Castro-Godínez, M. Shafique, and J. Henkel. “Towards Designing and Implementing Approximate Accelerators”. In: *16th International Summer School on Advanced Computer Architecture and Compilation for High-performance Embedded Systems (ACACES)*. 2020 (cit. on p. xx).

[CSH20b] J. Castro-Godínez, M. Shafique, and J. Henkel. "Towards Quality-Driven Approximate Software Generation for Accurate Hardware: Work-in-Progress". In: *2020 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*. 2020, pp. 12–14. DOI: `10.1109/CASES51649.2020.9243814` (cit. on pp. xx, 98).

[Cha+13] W.-T. J. Chan, A. B. Kahng, S. Kang, R. Kumar, and J. Sartori. "Statistical Analysis and Modeling for Error Composition in Approximate Computation Circuits". In: *the 31st International Conference on Computer Design (ICCD)*. 2013 (cit. on p. 39).

[Che+08] Y.-K. Chen, J. Chhugani, P. Dubey, C. J. Hughes, D. Kim, S. Kumar, V. W. Lee, A. D. Nhuyen, and M. 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` (cit. on p. 1).

[Chi+13] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. "Analysis and characterization of inherent application resilience for approximate computing". In: *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2013, pp. 1–9. DOI: `10.1145/2463209.2488873` (cit. on p. 73).

[Con+14] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, K. Gururaj, and G. Reinman. "Accelerator-rich architectures: Opportunities and progresses". In: *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2014, pp. 1–6. DOI: `10.1145/2593069.2596667` (cit. on pp. 2, 14).

[Del+13] A. A. Del Barrio, R. Hermida, S. O. Memik, J. M. Mendias, and M. C. Molina. "Multispeculative additive trees in High-Level Synthesis". In: *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2013, pp. 188–193. DOI: `10.7873/DATE.2013.052` (cit. on p. 77).

[Dut+16] S. Dutt, H. Patel, S. Nandi, and G. Trivedi. "Exploring Approximate Computing for Yield Improvement via Re-design of Adders for Error-Resilient Applications". In: *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*. 2016, pp. 134–139. DOI: `10.1109/VLSID.2016.101` (cit. on pp. 8, 21, 76).

[Ech+20] J. Echavarria, S. Wildermann, O. Keszöcze, and J. Teich. "Probabilistic Error Propagation through Approximated Boolean Networks". In: *57th Design Automation Conference (DAC)*. 2020 (cit. on p. 29).

[Esm+12] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. "Neural Acceleration for General-Purpose Approximate Programs". In: *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 2012, pp. 449–460. DOI: 10.1109/MICRO.2012.48 (cit. on pp. 2, 15).

[GRR20] S. K. Ghosh, A. Raha, and V. Raghunathan. "Approximate Inference Systems (AxIS): End-to-End Approximations for Energy-Efficient Inference at the Edge". In: *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. ISLPED '20. Boston, Massachusetts, 2020, pp. 7–12. DOI: 10.1145/3370748.3406575 (cit. on p. 98).

[Gil+18] G. A. Gillani, M. A. Hanif, M. Krone, S. H. Gerez, M. Shafique, and A. B. J. Kokkeler. "SquASH: Approximate Square-Accumulate With Self-Healing". In: *IEEE Access* 6 (2018), pp. 49112–49128. DOI: 10.1109/ACCESS.2018.2868036 (cit. on p. 77).

[Gup+13] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. "Low-Power Digital Signal Processing Using Approximate Adders". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32.1 (2013), pp. 124–137. DOI: 10.1109/TCAD.2012.2217962 (cit. on pp. 2, 8, 21, 22).

[HO13] J. Han and M. Orshansky. "Approximate Computing: An Emerging Paradigm for Energy-Efficient Design". In: *18th IEEE European Test Symposium (ETS)*. 2013, pp. 1–6. DOI: 10.1109/ETS.2013.6569370 (cit. on p. 1).

[Han+20] M. A. Hanif, R. Hafiz, O. Hasan, and M. Shafique. "PEMACx: A Probabilistic Error Analysis Methodology for Adders with Cascaded Approximate Units". In: *57th Design Automation Conference (DAC)*. 2020 (cit. on p. 11).

[HBR15] S. Hashemi, R. I. Bahar, and S. Reda. "DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications". In: *2015 International Conference on Computer-Aided Design (ICCAD)*. 2015, pp. 418–425. DOI: 10.1109/ICCAD.2015.7372600 (cit. on pp. 2, 7, 10, 26).

[HBR16] S. Hashemi, R. I. Bahar, and S. Reda. "A Low-Power Dynamic Divider for Approximate Applications". In: *53nd Design Automation Conference (DAC)*. 2016, pp. 1–6. DOI: 10.1145/2897937.2897965 (cit. on p. 8).

[Has+18] S. Hashemi, H. Tann, F. Buttafuoco, and S. Reda. "Approximate Computing for Biometric Security Systems: A Case Study on Iris Scanning". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2018, pp. 319–324 (cit. on p. 98).

[HTR18] S. Hashemi, H. Tann, and S. Reda. "BLASYS: Approximate Logic Synthesis Using Boolean Matrix Factorization". In: *55th Annual Design Automation Conference (DAC)*. 2018, 55:1–55:6. DOI: 10.1145/3195970.3196001 (cit. on p. 27).

[HP12] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. 5th ed. Morgan Kaufmann, 2012 (cit. on p. 79).

[Her+20] D. Hernández-Araya, J. Castro-Godínez, M. Shafique, and J. Henkel. "AUGER: A Tool for Generating Approximate Arithmetic Circuits". In: *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*. 2020, pp. 1–4. DOI: 10.1109/LASCAS45839.2020.9069045 (cit. on pp. xix, 17).

[HMV16] R. Hrbacek, V. Mrazek, and Z. Vasicek. "Automatic Design of Approximate Circuits by Means of Multi-Objective Evolutionary Algorithms". In: *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*. 2016, pp. 1–6. DOI: 10.1109/DTIS.2016.7483885 (cit. on p. 7).

[HLR11] J. Huang, J. Lach, and G. Robins. "Analytic Error Modeling for Imprecise Arithmetic Circuits". In: *Silicon Errors in Logic - System Effects*. 2011 (cit. on p. 39).

[HLR12] J. Huang, J. Lach, and G. Robins. "A Methodology for Energy-quality Tradeoff Using Imprecise Hardware". In: *the 49th Design Automation Conference (DAC)*. 2012 (cit. on p. 39).

[JHL] H. Jiang, J. Han, and F. Lombardi. "A Comparative Review and Evaluation of Approximate Adders". In: *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*. GLSVLSI '15. Pittsburgh, Pennsylvania, USA, pp. 343–348. DOI: 10.1145/2742060.2743760 (cit. on pp. 8, 18).

[Jia+17a] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han. "A Review, Classification, and Comparative Evaluation of Approximate Arithmetic Circuits". In: *J. Emerg. Technol. Comput. Syst.* 13.4 (2017), 60:1–60:34. DOI: 10.1145/3094124 (cit. on p. 7).

[Jia+17b] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han. "A Review, Classification, and Comparative Evaluation of Approximate Arithmetic Circuits". In: *J. Emerg. Technol. Comput. Syst.* 13.4 (2017). DOI: 10.1145/3094124 (cit. on pp. 8, 18).

[Jia+16] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han. "A Comparative Evaluation of Approximate Multipliers". In: *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. 2016, pp. 191–196. DOI: 10.1145/2950067.2950068 (cit. on pp. 8, 18).

[Jia+19] H. Jiang, L. Liu, F. Lombardi, and J. Han. "Approximate Arithmetic Circuits: Design and Evaluation". In: *Approximate Circuits: Methodologies and CAD*. Ed. by S. Reda and M. Shafique. Cham: Springer International Publishing, 2019, pp. 67–98. DOI: 10.1007/978-3-319-99322-5_4 (cit. on p. 8).

[KK12] A. B. Kahng and S. Kang. "Accuracy-Configurable Adder for Approximate Arithmetic Designs". In: *Proceedings of the 49th Annual Design Automation Conference (DAC)*. 2012, pp. 820–825. DOI: 10.1145/2228360.2228509 (cit. on pp. 7, 10, 25, 76).

[KCS16] A. Kapare, H. Cherupalli, and J. Sartori. "Automated Error Prediction for Approximate Sequential Circuits". In: *the 35th Intl. Conference on Computer-Aided Design (ICCAD)*. 2016 (cit. on p. 39).

[Kha+20] N. Khan, J. Castro-Godínez, S. Xue, J. Henkel, and J. Becker. "Automatic Floorplanning and Standalone Generation of Bitstream-Level IP Cores". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2020). DOI: 10.1109/TVLSI.2020.3023548 (cit. on p. xx).

[Khu+15] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke. "Rumba: An online quality management system for approximate computing". In: *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 2015, pp. 554–566. DOI: 10.1145/2749469.2750371 (cit. on pp. 72, 75).

[KGE11] P. Kulkarni, P. Gupta, and M. Ercegovac. "Trading Accuracy for Power with an Underdesigned Multiplier Architecture". In: *2011 24th International Conference on VLSI Design*. 2011, pp. 346–351. DOI: 10.1109/VLSID.2011.51 (cit. on p. 17).

[LA04] C. Lattner and V. Adve. "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation". In: *CGO*. 2004 (cit. on p. 47).

[LJG17] S. Lee, L. K. John, and A. Gerstlauer. "High-Level Synthesis of Approximate Hardware under Joint Precision and Voltage Scaling". In: *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2017, pp. 187–192. DOI: 10.23919/DATE.2017.7926980 (cit. on pp. 27, 54, 68).

[Lee+16] S. Lee, D. Lee, K. Han, E. Shriver, E. John, and A. Gerstlauer. "Statistical Quality Modeling of Approximate Hardware". In: *17th International Symposium on Quality Electronic Design (ISQED)*. 2016, pp. 163–168. DOI: 10.1109/ISQED.2016.7479194 (cit. on pp. 39, 53).

[LN19] M. T. Leipnitz and G. L. Nazar. “High-Level Synthesis of Approximate Designs under Real-Time Constraints”. In: *ACM Trans. Embed. Comput. Syst.* 18.5s (2019). DOI: `10.1145/3358182` (cit. on p. 54).

[LCH20] L. G. León-Vega, J. Castro-Godínez, and J. Henkel. “Measuring Traffic Dynamics at the Edge”. In: *International Work Conference on Bioinspired Intelligence (IWOBI)* (2020) (cit. on p. xx).

[Li+15] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu. “Joint Precision Optimization and High Level Synthesis for Approximate Computing”. In: *52nd Design Automation Conference (DAC)*. 2015, pp. 1–6. DOI: `10.1145/2744769.2744863` (cit. on pp. 39, 41, 53, 54).

[LHL13] J. Liang, J. Han, and F. Lombardi. “New Metrics for the Reliability of Approximate and Probabilistic Adders”. In: *IEEE Transactions on Computers* 62.9 (2013), pp. 1760–1771. DOI: `10.1109/TC.2012.146` (cit. on pp. 11, 12, 60).

[Lin+11] A. Lingamneni, C. Enz, J.-L. Nagel, K. Palem, and C. Piguet. “Energy Parsimonious Circuit Design through Probabilistic Pruning”. In: *2011 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2011, pp. 1–6. DOI: `10.1109/DATE.2011.5763130` (cit. on pp. 28, 35).

[LLS20] W. Liu, F. Lombardi, and M. Shulte. “A Retrospective and Prospective View of Approximate Computing [Point of View]”. In: *Proceedings of the IEEE* 108.3 (2020), pp. 394–399. DOI: `10.1109/JPROC.2020.2975695` (cit. on p. 2).

[MHR19] J. Ma, S. Hashemi, and S. Reda. “Approximate Logic Synthesis Using BLASYS”. In: *Workshop on Open-Source EDA Technology (WOSET)*. 5. 2019 (cit. on p. 27).

[Mah+16] D. Mahajan, A. Yazdanbakhsh, J. Park, B. Thwaites, and H. Esmaeilzadeh. “Towards Statistical Guarantees in Controlling Quality Tradeoffs for Approximate Acceleration”. In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 2016, pp. 66–77. DOI: `10.1109/ISCA.2016.16` (cit. on pp. 13, 71, 72, 75, 76).

[Mah+10] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. 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` (cit. on pp. 2, 7–11, 21, 55, 64, 86).

[Maz+17a] S. Mazahir, O. Hasan, R. Hafiz, and M. Shafique. “Probabilistic Error Analysis of Approximate Recursive Multipliers”. In: *IEEE Transactions on Computers* 66.11 (2017), pp. 1982–1990. DOI: `10.1109/TC.2017.2709542` (cit. on pp. 11, 19, 39, 53).

[Maz+16] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel. "An Area-Efficient Consolidated Configurable Error Correction for Approximate Hardware Accelerators". In: *53nd Design Automation Conference (DAC)*. 2016, pp. 1–6. DOI: 10.1145/2897937.2897981 (cit. on pp. 2, 15, 74, 76, 77, 79).

[Maz+17b] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel. "Probabilistic Error Modeling for Approximate Adders". In: *IEEE Transactions on Computers* 66.3 (2017), pp. 515–530. DOI: 10.1109/TC.2016.2605382 (cit. on pp. 11, 19, 39, 47, 53).

[MHS18] S. Mazahir, O. Hasan, and M. Shafique. "Adaptive Approximate Computing in Arithmetic Datapaths". In: *IEEE Design Test* 35.4 (2018), pp. 65–74. DOI: 10.1109/MDAT.2017.2772874 (cit. on p. 77).

[Mia+12] J. Miao, K. He, A. Gerstlauer, and M. Orshansky. "Modeling and Synthesis of Quality-Energy Optimal Approximate Adders". In: *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2012, pp. 728–735. DOI: 10.1145/2429384.2429542 (cit. on p. 26).

[Mom+15] A. Momeni, J. Han, P. Montuschi, and F. Lombardi. "Design and Analysis of Approximate Compressors for Multiplication". In: *IEEE Transactions on Computers* 64.4 (2015), pp. 984–994. DOI: 10.1109/TC.2014.2308214 (cit. on p. 10).

[Mor+15] T. Moreau, M. Wyse, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, and M. Oskin. "SNNAP: Approximate computing on programmable SoCs via neural acceleration". In: *the 21st International Symposium on High Performance Computer Architecture (HPCA)*. 2015 (cit. on p. 2).

[Mra+17] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina. "EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2017, pp. 258–261. DOI: 10.23919/DATE.2017.7926993 (cit. on p. 12).

[MVS19] Z. Mrazek, Z. Vasicek, and L. Sekanina. "EvoApproxLib: Extended Library of Approximate Arithmetic Circuits". In: *Workshop on Open-Source EDA Technology (WOSET)*. 10. 2019 (cit. on pp. 28, 32).

[Nep+14] K. Nepal, Y. Li, R. I. Bahar, and S. Reda. "ABACUS: A Technique for Automated Behavioral Synthesis of Approximate Computing Circuits". In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2014, pp. 1–6. DOI: 10.7873/DATE.2014.374 (cit. on pp. 53, 54).

[Pan+16] P. Panda, A. Sengupta, S. S. Sarwar, G. Srinivasan, S. Venkataramani, A. Raghunathan, and K. Roy. "Invited - Cross-Layer Approximations for Neuromorphic Computing: From Devices to Circuits and Systems". In: *53rd Annual Design Automation Conference (DAC)*. DAC '16. Austin, Texas: Association for Computing Machinery, 2016. DOI: 10.1145/2897937.2905009 (cit. on p. 97).

[RR18] A. Raha and V. Raghunathan. "Approximating Beyond the Processor: Exploring Full-System Energy-Accuracy Tradeoffs in a Smart Camera System". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.12 (2018), pp. 2884–2897. DOI: 10.1109/TVLSI.2018.2864269 (cit. on p. 98).

[Reh+16] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel. "Architectural-Space Exploration of Approximate Multipliers". In: *Proceedings of the 35th International Conference on Computer-Aided Design (ICCAD)*. 2016. DOI: 10.1145/2966986.2967005 (cit. on p. 10).

[Sam+13] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke. "SAGE: Self-tuning Approximation for Graphics Engines". In: *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-46. Davis, California: ACM, 2013, pp. 13–24. DOI: 10.1145/2540708.2540711 (cit. on p. 75).

[SBC15] A. Sampson, J. Bornholt, and L. Ceze. "Hardware-Software Co-Design: Not Just a Cliché". In: *1st Summit on Advances in Programming Languages (SNAPL 2015)*. Ed. by T. Ball, R. Bodik, S. Krishnamurthi, B. S. Lerner, and G. Morrisett. Vol. 32. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 262–273. DOI: 10.4230/LIPIcs.SNAPL.2015.262 (cit. on p. 97).

[San+15] J. San Miguel, J. Albericio, A. Moshovos, and N. E. Jerger. "Doppelgänger: A Cache for Approximate Computing". In: *48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2015, pp. 50–61. DOI: 10.1145/2830772.2830790 (cit. on p. 98).

[Sca+19] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, and L. Pozzi. "Partition and Propagate: An Error Derivation Algorithm for the Design of Approximate Circuits". In: *56th Design Automation Conference (DAC)*. Las Vegas, NV, USA, 2019, 40:1–40:6. DOI: 10.1145/3316781.3317878 (cit. on p. 29).

[Sca+20] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda. "Approximate Logic Synthesis: A Survey". In: *Proceedings of the IEEE* (2020), pp. 1–19. DOI: 10.1109/JPROC.2020.3014430 (cit. on pp. 27, 37).

[SAP18] I. Scarabottolo, G. Ansaloni, and L. Pozzi. "Circuit Carving: A Methodology for the Design of Approximate Hardware". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2018, pp. 545–550. DOI: 10.23919/DATE.2018.8342067 (cit. on pp. 7, 27).

[Sch+17] J. Schlachter, V. Camus, K. V. Palem, and C. Enz. "Design and Applications of Approximate Circuits by Gate-Level Pruning". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.5 (2017), pp. 1694–1702. DOI: 10.1109/TVLSI.2017.2657799 (cit. on pp. 27, 28).

[SW11] R. Sedgewick and K. Wayne. *Algorithms*. Addison-Wesley, 2011 (cit. on p. 44).

[Sen+19] D. Sengupta, F. S. Snigdha, J. Hu, and S. S. Sapatnekar. "An Analytical Approach for Error PMF Characterization in Approximate Circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.1 (2019), pp. 70–83. DOI: 10.1109/TCAD.2018.2803626 (cit. on p. 53).

[Sha+15] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. "A Low Latency Generic Accuracy Configurable Adder". In: *52nd Design Automation Conference (DAC)*. 2015, pp. 1–6. DOI: 10.1145/2744769.2744778 (cit. on pp. 2, 7, 9–11, 25, 55, 64, 76, 86).

[Sha+16] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel. "Invited - Cross-Layer Approximate Computing: From Logic to Architectures". In: *53nd Design Automation Conference (DAC)*. 2016. DOI: 10.1145/2897937.2906199 (cit. on pp. 2, 3, 15, 97).

[Sta+20] P. Stanley-Marbell, A. Alaghi, M. Carbin, E. Darulova, L. Dolecek, A. Gerstlauer, G. Gillani, D. Jevdjic, T. Moreau, M. Cacciotti, A. Daglis, N. D. E. Jerger, B. Falsafi, S. Misailovic, A. Sampson, and D. Zufferey. "Exploiting Errors for Efficiency: A Survey from Circuits to Algorithms". In: *ACM Comput. Surv.* 53.3 (2020). DOI: 10.1145/3394898 (cit. on p. 1).

[TW17] T. N. Theis and H.-S. P. Wong. "The End of Moore's Law: A New Beginning for Information Technology". In: *Computing in Science Engineering* 19.2 (2017), pp. 41–50. DOI: 10.1109/MCSE.2017.29 (cit. on p. 1).

[Tic98] W. F. Tichy. “Should Computer Scientists Experiment More?” In: *Computer* 31.5 (1998), pp. 32–40. DOI: `10.1109/2.675631` (cit. on pp. 40, 55).

[Vas19] Z. Vasicek. “Formal Methods for Exact Analysis of Approximate Circuits”. In: *IEEE Access* 7 (2019), pp. 177309–177331. DOI: `10.1109/ACCESS.2019.2958605` (cit. on p. 12).

[VMS19] Z. Vasicek, Z. Mrazek, and L. Sekanina. “Automated Circuit Approximation Method Driven by Data Distribution”. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, pp. 96–101. DOI: `10.23919/DATE.2019.8714977` (cit. on p. 97).

[Ven+15] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan. “Approximate Computing and the Quest for Computing Efficiency”. In: *52nd Design Automation Conference (DAC)*. 2015, pp. 1–6. DOI: `10.1145/2744769.2751163` (cit. on p. 1).

[Ven+12] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. “SALSA: Systematic Logic Synthesis of Approximate Circuits”. In: *Design Automation Conference (DAC)*. 2012, pp. 796–801. DOI: `10.1145/2228360.2228504` (cit. on p. 7).

[VBI08] A. K. Verma, P. Brisk, and P. Ienne. “Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design”. In: *2008 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2008, pp. 1250–1255. DOI: `10.1109/DATE.2008.4484850` (cit. on pp. 7, 8, 10, 76).

[Wan+16] T. Wang, Q. Zhang, N. S. Kim, and Q. Xu. “On Effective and Efficient Quality Management for Approximate Computing”. In: *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ISLPED ’16. San Francisco Airport, CA, USA: ACM, 2016, pp. 156–161. DOI: `10.1145/2934583.2934608` (cit. on pp. 72, 75, 87, 88, 90).

[WB09] Z. Wang and A. C. Bovik. “Mean Squared Error: Love It or Leave It? A new look at signal fidelity measures”. In: *IEEE Signal Processing Magazine* 26.1 (2009), pp. 98–117. DOI: `10.1109/MSP.2008.930649` (cit. on pp. 13, 59).

[Wan+04] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. “Image Quality Assessment: From Error Visibility to Structural Similarity”. In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: `10.1109/TIP.2003.819861` (cit. on p. 13).

[Wil] S. Williams. *Icarus Verilog*. `http://iverilog.icarus.com/` (cit. on p. 31).

[Wol] C. Wolf. *Yosys Open SYnthesis Suite*. `http://www.clifford.at/yosys/` (cit. on p. 31).

[Wu+19] Y. Wu, Y. Li, X. Ge, Y. Gao, and W. Qian. “An Efficient Method for Calculating the Error Statistics of Block-Based Approximate Adders”. In: *IEEE Transactions on Computers* 68.1 (2019), pp. 21–38. DOI: `10.1109/TC.2018.2859960` (cit. on pp. 11, 19, 53).

[WQ16] Y. Wu and W. Qian. “An Efficient Method for Multi-Level Approximate Logic Synthesis under Error Rate Constraint”. In: *53nd Design Automation Conference (DAC)*. 2016, pp. 1–6. DOI: `10.1145/2897937.2897982` (cit. on p. 27).

[XMK16] Q. Xu, T. Mytkowicz, and N. S. Kim. “Approximate Computing: A Survey”. In: *IEEE Design Test* 33.1 (2016), pp. 8–22. DOI: `10.1109/MDAT.2015.2505723` (cit. on pp. 1, 2, 26).

[XS17] S. Xu and B. C. Schafer. “Exposing Approximate Computing Optimizations at Different Levels: From Behavioral to Gate-Level”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.11 (2017), pp. 3077–3088. DOI: `10.1109/TVLSI.2017.2735299` (cit. on pp. 78, 98).

[Yan+13] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi. “Approximate XOR/XNOR-based Adders for Inexact Computing”. In: *13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013)*. 2013, pp. 690–693. DOI: `10.1109/NANO.2013.6720793` (cit. on pp. 8, 21).

[Ye+13] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu. “On reconfiguration-oriented approximate adder design and its application”. In: *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2013, pp. 48–54. DOI: `10.1109/ICCAD.2013.6691096` (cit. on pp. 10, 25, 76).

[Zen+17] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram. “RoBA Multiplier: A Rounding-Based Approximate Multiplier for High-Speed yet Energy-Efficient Digital Signal Processing”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.2 (2017), pp. 393–401. DOI: `10.1109/TVLSI.2016.2587696` (cit. on pp. 2, 10, 26).

[Zer+16] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. 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` (cit. on pp. 7, 10, 54).

[Zer+19] G. Zervakis, S. Xydis, D. Soudris, and K. Pekmestzi. “Multi-Level Approximate Accelerator Synthesis Under Voltage Island Constraints”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 66.4 (2019), pp. 607–611. DOI: 10.1109/TCSII.2018.2869025 (cit. on p. 53).

[ZGY09] N. Zhu, W. L. Goh, and K. S. Yeo. “An Enhanced Low-Power High-Speed Adder For Error-Tolerant Application”. In: *Proceedings of the 2009 12th International Symposium on Integrated Circuits*. 2009, pp. 69–72 (cit. on pp. 10, 76).

List of Figures

List of Tables

Jorge Castro-Godínez

Automated Design of Approximate Accelerators

With the emergence of the approximate computing paradigm, many approximate functional units have been reported in the literature, particularly approximate adders and multipliers. For a plethora of such approximate circuits, and considering their usage as building blocks for the design of approximate accelerators for error-tolerant applications, a challenge arises: selecting those approximate circuits for a given application that minimize the required resources while satisfying a defined accuracy.

This work proposes automated methods for designing and implementing approximate accelerators built with approximate arithmetic circuits.