UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MARCOS TOMAZZOLI LEIPNITZ

Integrating Constraint Awareness and Multiple Approximation Techniques in High-Level Synthesis for FPGAs

Thesis presented in partial fulfillment of the requirements for the degree of Doctor of Computer Science

Advisor: Prof. Dr. Gabriel Luca Nazar

Porto Alegre
October 2022

CIP — CATALOGING-IN-PUBLICATION

Leipnitz, Marcos Tomazzoli

Integrating Constraint Awareness and Multiple Approximation Techniques in High-Level Synthesis for FPGAs / Marcos Tomazzoli Leipnitz. – Porto Alegre: PPGC da UFRGS, 2022.

136 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2022. Advisor: Gabriel Luca Nazar.

1. High-level synthesis. 2. Approximate computing. 3. Design space exploration. 4. Field-Programmable Gate Array. I. Nazar, Gabriel Luca. II. Título.

“It always seems impossible until it’s done.”

— NELSON MANDELA

ACKNOWLEDGEMENTS

The British philosopher Alfred North Whitehead once said: "No one who achieves success does so without acknowledging the help of others. The wise and confident acknowledge this help with gratitude." In that spirit, I dedicate this thesis to each person who has been involved in my academic life and helped me throughout this project. Special thanks to my parents, Roni Carlos Leipnitz and Larice Tomazzoli, for their support and affection, and to my advisor, Gabriel Nazar, for patiently guiding me in the right direction and for the words of motivation and inspiration.

ABSTRACT

The adoption of High-Level Synthesis (HLS) targeting Field-Programmable Gate Arrays (FPGAs) has increased as the latest HLS tools have evolved to provide high-quality results while increasing productivity and reducing time-to-market. Concurrently, numerous approximate computing (AC) techniques have been developed to reduce design costs in error-resilient application domains, such as signal and multimedia processing, data mining, machine learning, and computer vision, to trade-off computation accuracy with area and power savings or performance improvements. However, selecting adequate techniques for each application and optimization target is complex but crucial for high-quality results. In this context, many works have proposed incorporating AC techniques within HLS toolchains to alleviate the burden of hand-crafting approximate circuits, i.e., designers can resort to approximate HLS (AHLS) tools to automate the exploitation of AC techniques when attempting to make a design meet the specified requirements.

However, previous AHLS design methodologies do not allow specifying a set of design metrics constraints to guide the exploration of approximate circuits towards meeting the aimed optimizations. Moreover, they are typically tied to a single approximation technique or a difficult-to-extend set of techniques whose exploitation is not fully automated or steered by optimization targets. Therefore, available AHLS tools overlook the benefits of expanding the design space by mixing diverse approximation techniques toward meeting specific design objectives with minimum error. This thesis proposes that a constraint-aware AHLS methodology for FPGAs, able to automatically identify efficient combinations of multiple AC techniques for different applications and design optimizations, would be a promising option to manage the design effort of adopting the AC design paradigm while optimizing the quality of results. Experimental results over a set of signal and image processing benchmarks show that, on average, a reduction of about 30% in error measure, ranging from 9.5% to 52% depending on the target constraints (resources, worst-case execution time, or both), can be obtained when compared to constraint-oblivious approaches relying on unconstrained or error-constrained design methodologies. Moreover, additional improvements varying from 5% to 30% (about 18% on average) can be attained when constraint awareness is exploited with multiple AC techniques.

Keywords: High-level synthesis. Approximate computing. Design space exploration. Field-Programmable Gate Array.

Integração de Consciência de Restrições e Múltiplas Técnicas de Aproximação em Síntese de Alto Nível para FPGAs

RESUMO

A adoção de Síntese da Alto Nível (HLS do Inglês *High-Level Synthesis*) visando *Field-Programmable Gate Arrays* (FPGAs) aumentou à medida que as ferramentas mais recentes de HLS evoluíram para fornecer resultados de alta qualidade enquanto aumentam a produtividade e reduzem o *time-to-market*. Simultaneamente, inúmeras técnicas de computação aproximativa (AC do Inglês *Approximate Computing*) foram desenvolvidas para reduzir os custos de projeto em domínios de aplicação resilientes a erros, tais como processamento de sinais e multimídia, mineração de dados, aprendizado de máquina e visão computacional, para trocar a precisão da computação por economia de área e energia ou melhorias de desempenho. Entretanto, a seleção de técnicas adequadas para cada aplicação e otimização alvo é complexa, porém crucial para resultados de alta qualidade. Neste contexto, muitos trabalhos propuseram incorporar técnicas de AC dentro do fluxo de ferramentas HLS para aliviar a carga de explorar manualmente circuitos aproximados, ou seja, os projetistas podem recorrer a ferramentas de HLS aproximativas (AHLS do Inglês *Approximate High-Level Synthesis*) para automatizar a exploração das técnicas de AC quando tentarem fazer um projeto atender os requisitos especificados.

Entretanto, as metodologias prévias de AHLS não permitem especificar um conjunto de métricas de projeto para orientar a exploração de circuitos aproximados para atender às otimizações pretendidas. Além disso, esses métodos normalmente estão vinculados à uma única técnica de aproximação ou à um conjunto de técnicas de difícil extensão, cuja exploração não é totalmente automatizada ou orientada por objetivos de otimização. Portanto, as ferramentas AHLS disponíveis ignoram os benefícios de expandir o espaço de projeto, misturando diversas técnicas de aproximação para atingir objetivos específicos de projeto com o mínimo de erro. Esta tese propõe que uma metodologia AHLS consciente das restrições para FPGAs capaz de identificar automaticamente combinações eficientes de múltiplas técnicas de AC para diferentes aplicações e otimizações de projeto seria uma opção promissora para gerenciar o esforço de projeto para adoção do paradigma de projeto AC enquanto otimiza a qualidade dos resultados. Resultados experimentais sobre um conjunto de aplicações de processamento de sinais e imagem mostram que, em média, uma redução de cerca de 30% na medida do erro, variando de 9,5% a 52% dependendo das restrições alvo (recursos, tempo de execução de pior caso, ou ambos), pode ser obtida quando comparada a abordagens que não são conscientes das restrições e que dependem de metodologias de projeto sem restrições ou com restrições de erro. Além disso, melhorias adicionais variando de 5% a 30% (cerca de 18% em média) podem ser alcançadas quando a consciência das restrições é explorada com múltiplas técnicas de AC.

Palavras-chave: Síntese de alto nível. Computação aproximativa. Exploração do espaço de projeto. Field-Programmable Gate Array.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AC	Approximate Computing
ADPCM	Adaptive Differential Pulse-Code Modulation
AHLS	Approximate High-Level Synthesis
ALAP	As-Late-As-Possible
ALS	Approximate Logic Synthesis
ASAP	As-Soon-As-Possible
ASIC	Application Specific Integrated Circuit
BRAM	Block Random Access Memory
BWR	Bitwidth Reduction
CDFG	Control and Data Flow Graph
CFG	Control Flow Graph
CGP	Cartesian Genetic Programming
DAG	Directed Acyclic Graph
DFG	Data Flow Graph
DSE	Design Space Exploration
DSP	Digital Signal Processor
EM	Error Metric
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
IoT	Internet-of-Things
HLS	High-Level Synthesis
ICFG	Interprocedural Control Flow Graph
ILP	Integer Linear Programming

IPET	Implicit Path Enumeration Technique
IR	Intermediate Representation
LLVM	Low-Level Virtual Machine
LP	loop Perforation
LUT	Look-Up Table
MSE	Mean-Squared Error
NSGA	Non-Dominated Sorting Genetic Algorithm
PA	Percentage Accuracy
PSNR	Peak Signal-to-Noise Ratio
QoR	Quality of Results
V2C	Variable to Constant
V2M	Variable to Mean
V2P	Variable to Power
V2Z	Variable to Zero
WCET	Worst-Case Execution Time

CONTENTS

1 INTRODUCTION

The technology scaling of Complementary Metal Oxide Semiconductor (CMOS) devices has boosted advances in the processing capacity and functionality of modern computing systems. The exponential growth in the number of transistors on-chip, popularly known as Moore's law (MOORE, 1965), leveraged the integration of complex systems in a single package (Systems-on-Chip - SoCs) with concurrent advancements in area, power, and delay without jeopardizing costs. For example, smaller process node technologies allow adding more memory or application-specific accelerators to build systems requiring higher and heterogeneous processing capabilities. However, the innovation opportunities enabled by the technology scaling introduced new challenges as the physical limitations of transistor devices have diminished the performance scaling trend. With the end of the Dennard scaling (DENNARD et al., 1974) due to voltage scaling limits and increasing sub-threshold leakage current, integrating more transistors on-chip results in higher power density and hence in power dissipation issues, such as overheating and reliability degradation due to accelerated aging. Moreover, sustained chip performance becomes limited by the Thermal Density Power (TDP) constraint, which leveraged the adoption of multi-core architectures with heterogeneous computing resources to preserve performance scaling through parallel processing (BORKAR, 2007). Nevertheless, as transistors continue to shrink, the power wall may limit the concurrent activation of on-chip devices (ESMAEILZADEH et al., 2011). Therefore, power dissipation and energy consumption become primary concerns in the design and deployment of modern computing systems since they critically determine operating costs, cooling requirements, and battery autonomy of mobile devices.

In face of these technological hurdles, the growth in market demand for real-time performance and energy-bound systems for a wide range of applications involving multimedia streaming, computer graphics, computer vision, natural language processing, data mining, and virtual and augmented reality has been driving substantial research and engineering efforts. A typical example is the rise of edge computing applications based on the Internet-of-Things (IoT) paradigm, where computation and data storage are brought closer to data sources, such as smartphones, wearables, unmanned aerial vehicles, and sensor networks (VARGHESE et al., 2016; KHAN et al., 2019). The main idea is to enable applications requiring low response times by providing low latency communication and real-time processing capabilities. With the growing adoption of the 5G technology, mobile IoT will reach 4.4 billion connections by 2023, a four-fold growth from 2018, crossing over a third (34%) of all mobile devices connected to the internet (CISCO, 2020). However, the limited computing capacity of edge devices due to non-functional constraints like manufacturing cost, power dissipation, and energy consumption may hamper the viability and quality of compute-intensive networked services based on graphics processing and deep learning, for example. Moving such computations to the cloud to take advantage of the high processing capacity of modern data centers offering high-performance and heterogeneous computing resources may not be an option due to latency, scalability, and privacy concerns (CHEN; RAN, 2019). In that case, the user experience may be harmed by the lack of latency and bandwidth guarantees. Therefore, deploying edge computing applications often demands energy-efficient software and hardware implementations to meet specific performance requirements and become feasible.

On the other hand, cloud providers must deal with the ever-increasing demand for computing and storage resources in worldwide data centers. Public cloud services providers, such as Amazon, Google, and Microsoft, take in more than 250 billion dollars a year to maintain massive data centers around the world, all loaded with high-core-count CPUs, sporting terabytes of RAM and petabytes of storage. In such a scenario, global data centre electricity use reached 200-250 TWh in 2020, or around 1% of global final electricity demand, not to mention the energy needs for cryptocurrency mining, which were about 100 TWh in that same period (IEA, 2021). With augmented-reality spectacles streaming real-time video over the internet and the widespread adoption of smart digital currencies, the cloud infrastructure will provide the foundation for nearly every financial transaction and user interaction with data over the next decade (PESCE, 2021). Consequently, the cloud's energy consumption is poised to grow unsustainably, bringing the challenge of designing high-efficiency computing systems able to process huge amounts of data timely without busting power and energy budgets and raising capital and operational expenditures to impractical figures (ARMBRUST et al., 2010).

Another typical application in this context are the new video encoding standards with higher compression capabilities, such as the High Efficiency Video Coding (HEVC) and its successor, the Versatile Video Coding (VVC), commonly adopted by live and on-demand streaming platforms to manage bandwidth requirements (BROSS et al., 2021). The strong demand for streaming services has boosted the growth of network traffic. Globally, video streaming and gaming currently make up 87% of total consumer Internet traffic, projected to reach 332.7 exabytes per month by 2022 (CISCO, 2020). Considering only mobile data traffic, which reached around 84 exabytes per month by the end of 2021 and is projected to grow by a factor of about 4.3 to reach 368 exabytes per month in 2027, video traffic is estimated to account for 69%, a share that is forecast to increase to 79% in 2027 (ERICSSON, 2022). However, more complex encoding techniques usually come with higher computational costs. For example, when real-time encoding is necessary, the encoding system must handle computing needs that grow substantially with increasing resolutions while still meeting the required frame rate. Efficient implementations of such standards usually leverage dedicated hardware accelerators to handle the stringent latency and throughput requirements of applications such as high-resolution video streaming (4K and beyond) and cloud gaming without sacrificing user experience (SJöVALL et al., 2018; FAN et al., 2018; Bey Ahmed Khernache et al., 2021; GOGOI; PEESAPATI, 2021). Therefore, deploying compute-intensive networked services and applications while meeting performance requirements with stringent resources, power, and energy constraints can be very challenging.

As mentioned, among the strategies commonly used to meet the specific requirements of each application is the development of dedicated hardware accelerators, designed and adjusted manually through a hardware description language (HDL), such as VHDL or Verilog, according to the availability of resources and the restrictions imposed by the environment where the services will be implemented. Such strategy, however, increases the time-to-market and demands specialized designers, making its adoption more costly and difficult in environments where services and requirements constantly change with the emergence of new technologies, platforms, and applications. The High-Level Synthesis (HLS) design methodology has emerged in this scenario as an option to synthesize hardware targeting Field-Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs) from a behavioral description in a high-level language, such as C/C++ or SystemC, increasing productivity and reducing development cycles (COUSSY et al., 2009; Cong et al., 2011). However, compiling untimed algorithmic descriptions to low-level and cycle-accurate hardware specifications can be very challenging due to the inherent interdependence between the basic tasks involved in the synthesis process (resources allocation, scheduling, and binding), which may lead to sub-optimal results. Nevertheless, due to its advantages, the adoption of HLS is expanding as the available tools (e.g., Cadence Stratus, Xilinx Vivado, and Intel HLS Compiler) have evolved to provide high-quality results often comparable to hand-coded designs (Koch; Hannig; Ziener, 2016; LAHTI et al., 2019; Nane et al., 2016).

One of the great advantages of the HLS approach is the possibility of reusing and redirecting high-level IP cores to different technological substrates, allowing quick implementation of variants with unique characteristics regarding resources usage, performance, energy, and power, from a series of directives specified directly in the tool (number of functional units, resource sharing, clock frequency, loop unrolling, loop pipelining, etc.) or through pragmas in the source code. However, given that the design space exploration is carried out by steering the synthesis with a relatively large set of configurations not explicitly related to the target optimizations, obtaining satisfactory results requires designers to have familiarity with the specific application at hand and the chosen HLS tool.

Concurrently, the Approximate Computing (AC) design paradigm has become a powerful tool for aiding designers in implementing highly efficient software and hardware accelerators for a wide range of compute-intensive applications in domains like signal and image processing, computer graphics, data mining, machine learning, and computer vision (Palem et al., 2009; Han; Orshansky, 2013; LIU; LOMBARDI; SHULTE, 2020). Many applications in such domains are inherently tolerant to approximations in some of their computations due to redundant or noisy input data, probabilistic and statistical calculations, or perceptual limitations. Thus, delivering a less-than-optimal outcome with a controlled and occasional deviation from the exact implementation is often sufficient, making precise and costly computations unnecessary. For such cases, a plethora of AC techniques have been proposed to overcome the challenges arising from real-time requirements or stringent resources, power, and energy constraints, allowing the exploration of unique trade-offs between computation precision and design objectives (XU; MYTKOWICZ; KIM, 2016; Mittal, 2016). Mobile applications, for example, can benefit from AC techniques to reduce energy consumption without sacrificing user experience Pejović (2019). However, harnessing the full potential of AC can be very challenging, as it is an application- and workload-dependent task that relies on carefully selecting appropriate techniques and assessing where they should be applied to achieve the target optimizations without unacceptable quality loss (Venkataramani et al., 2015; Shafique et al., 2016). Therefore, the AC design paradigm has the drawback of increasing design complexity, impacting time-to-market and costs.

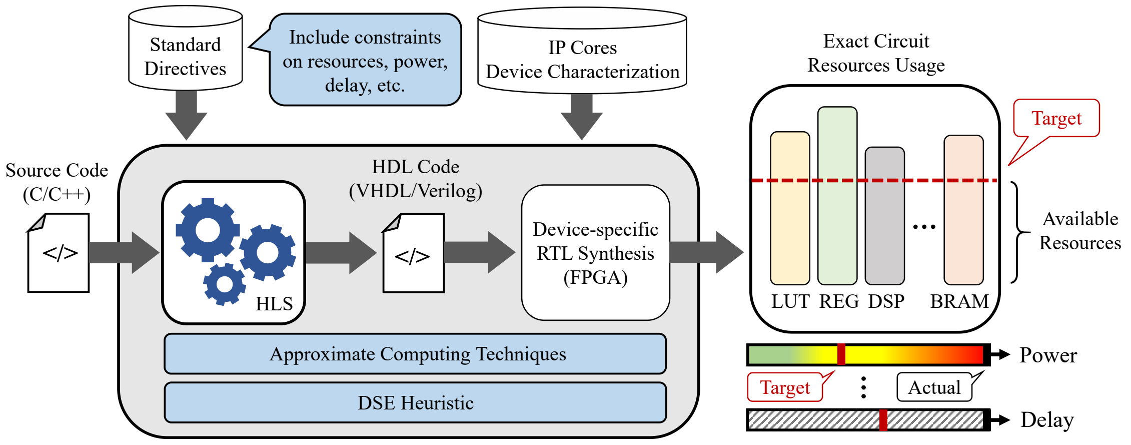
The hurdles for adopting the AC design paradigm have driven the proposal of numerous approximate logic synthesis (ALS) and approximate HLS (AHLS) design methodologies incorporating approximate computing techniques within HLS toolchains to deal with the Design Space Exploration (DSE) of approximate circuits and to relieve the burden of hand-crafting application-specific hardware accelerators (Vaverka; Hrbacek; Sekanina, 2016; Li et al., 2015; Lee; John; Gerstlauer, 2017; Schafer, 2017; Lee; Gerstlauer, 2017; Nepal et al., 2014; Nepal et al., 2019; Xu; Schafer, 2017; XU; SCHAFER, 2019c; CASTRO-GODíNEZ et al., 2020b). In a typical (i.e., not approximate) HLS design flow, designers steer the synthesis tool to produce the desired trade-off regarding area, power, and timing, allowing automatic exploration of diverse hardware implementations from high-level algorithmic descriptions. The wider flexibility offered by the HLS approach compared to traditional design flows, where the HDL code must be hand-crafted to achieve specific optimizations, is thus a promising option to manage the design effort of adopting AC as well. Including the exploitation of AC techniques within the HLS design space enables a comprehensive exploration of approximation opportunities at higher abstraction levels, enabling designers to trade off output precision with diverse optimization targets and generate approximate hardware quickly without requiring in-deep knowledge of approximation methodologies.

1.1 Motivation and Challenges

Reconfigurable devices, such as FPGAs, are widely used to implement computing systems that can benefit from fine-grained parallelism acceleration and in-field reconfiguration to add new features or improve their capacity to satisfy new requirements timely, which is the case for many applications exploiting the AC design paradigm. Compiling for reconfigurable computing architectures through HLS imposes specific challenges for efficiently mapping high-level algorithmic descriptions to device-specific hardware resources (Cardoso; Diniz; Weinhardt, 2010). Thus, state-of-the-art HLS tools rely on target-oriented resource allocation, scheduling, and binding optimizations to provide high-quality results. However, most built-in approximation methodologies proposed so far are target-oblivious, i.e., they disregard the hardware substrate when defining the methodology for choosing where approximations should be applied, which may result in sub-optimal designs, as the benefits of approximations may be very different when targeting FPGAs instead of ASICs.

Figure 1.1 illustrates a traditional HLS design flow augmented (blue boxes) with constraints on specific design metrics, a set of AC techniques, and a heuristic able to explore the design space of approximate designs to meet all constraints with minimum error. As can be observed, FPGAs comprise a wide variety of heterogeneous resources that can

Figure 1.1 – Constraint-aware exploration of AC techniques within HLS design flows


Standard Directives
Include constraints on resources, power, delay, etc.
IP Cores Device Characterization
Exact Circuit Resources Usage
Target
Source Code (C/C++)
</>
HDL Code (VHDL/Verilog)
HLS
Device-specific RTL Synthesis (FPGA)
Available Resources
LUT REG DSP BRAM
Approximate Computing Techniques
DSE Heuristic
Power
Target
Actual
Delay

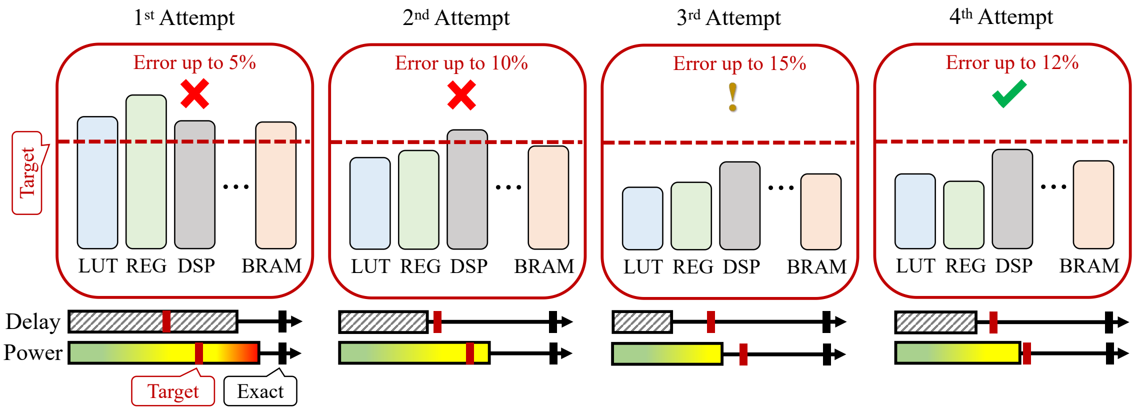
Source: The author

be exploited to fine-tune approximation strategies and improve the quality of results. Depending on design constraints, arithmetic operations and data storage can be mapped to built-in hard blocks such as Digital Signal Processors (DSPs) and Block RAMs (BRAMs) or carried out via fine-grained resources such as Look-Up Tables (LUTs) and Registers (REGs). Consequently, AHLS design methodologies that aim to reduce resource usage, power consumption, or delay in FPGAs should be aware that specific operations may have a more efficient mapping to dedicated resources, providing different gains when compared to ASIC-oriented synthesis.

Besides being target-oblivious, most existing approximation methodologies for HLS are also error-constrained, i.e., designers specify a maximum target error, and DSE is performed to minimize area, power, or delay, subject to the maximum error. As the concept of acceptable error is application- and workload-dependent, defining error bounds beforehand often requires deep knowledge of the application and the environment where it will be used, making such an approach unpractical for many scenarios. As exemplified in Figure 1.2, the available resources are frequently known a priori, which can be the entire device or a certain subset if it is shared with other components, and thus designers may need multiple attempts to find the target error that produces a circuit meeting all design requirements. In the third and fourth attempts shown in Figure 1.2, note how an error-constrained DSE is unable to focus on approximations with more impact on those design metrics that still have not been met in the second attempt, i.e., the number of DSP blocks and the circuit's power consumption. Consequently, designers may need to try multiple error thresholds until an acceptable result is found.

There are also unconstrained methodologies that produce Pareto-optimal designs

Figure 1.2 – Multiple attempts to meet design objectives with error-constrained AHLS

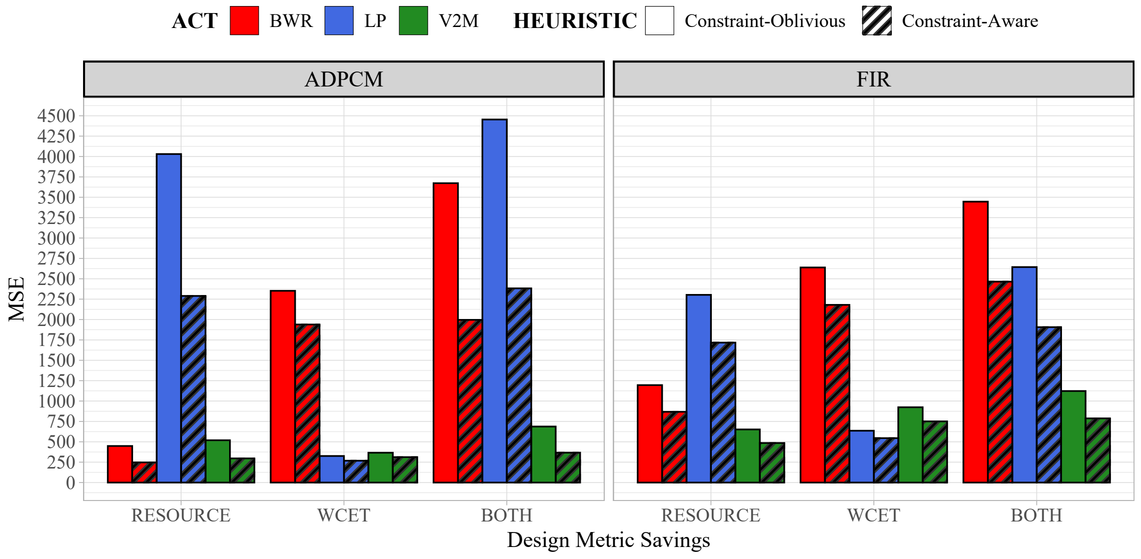


Source: The author

regarding the error and specific design metrics, with designers responsible for choosing the approximate design with a trade-off that best fits their needs. However, more suitable trade-offs may be lost in those cases due to less focused optimizations. Additionally, these methodologies usually restrict optimizations to specific design metrics, not being adequate if designers need to meet multiple constraints simultaneously. As a result, the tool may apply more approximations than the strictly necessary to meet the designer's objectives, jeopardizing results. Therefore, a promising option would be a target-oriented approach with which designers can directly control design objectives by steering the DSE with multiple constraints, such as resource usage, energy consumption, or performance.

Another limitation of AHLS methodologies proposed so far is relying on a limited, difficult-to-extend set of AC techniques tightly coupled to the design flow or on pre-built libraries of approximate functional units found in the literature. The drawback of such approaches is narrowing the design space by hampering the addition of new AC techniques that may enable the exploration of unique trade-offs. Variable to constant substitutions, for example, may provide both area and performance improvements by replacing costly operations with constant values. Conversely, loop perforation techniques are primarily used to reduce execution times by skipping some iterations of loop-based computations, while bit-width reduction is a fine-grain technique suitable for reducing the data-path size by simplifying costly operations. Figure 1.3 illustrates a motivational example where the bit-width reduction (BWR), loop perforation (LP), and variable-to-mean (V2M) AC techniques are applied to two different applications (ADPCM encoder and FIR filter), targeting savings of 20% in resources (LUTs, REGs, and DSPs), worst-case execution time (WCET), and both design metrics. The metric considered for quality measurement is the

Figure 1.3 – Motivational example: trade-offs provided by different AC techniques for different applications and target optimizations with savings of 20%

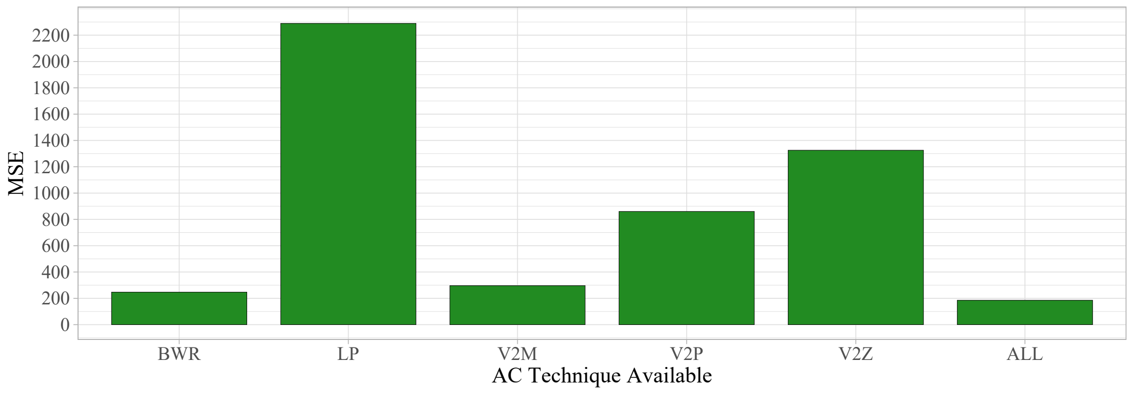


Source: The author

mean squared error (MSE). Additionally, for each scenario, it is considered two different approaches for DSE: constraint-oblivious, where approximations are performed until the aimed savings are attained based only on error-metric trade-offs, and constraint-aware, where the target savings of 20% is leveraged to select approximate design candidates based not just on error-metric trade-offs, but also on how distant they are from meeting the imposed constraints.

As can be observed in Figure 1.3, BWR and LP are more suitable for resource and WCET savings, respectively, being very inefficient for other scenarios, while V2M provides an intermediate option that performs well for most cases, even though it is not the best option for some cases where BWR and LP stand out. Nevertheless, depending on the application, the best technique for a given optimization may vary. For example, while BWR is the best choice for resource savings in the ADPCM application, it was overtaken by the V2M technique in the FIR application. Regarding the heuristic employed, it becomes clear that a constraint-aware approach is able to improve results by reducing the MSE, on average, by about 25% and 22% for the ADPCM and FIR applications, respectively. Such results represent the experiments presented in Chapter 6, where more applications and design scenarios are explored and discussed in detail. Therefore, although some techniques may impact multiple design metrics, other techniques were designed to focus on specific optimizations to avoid unnecessary quality loss. However, current efforts on AHLS concentrate on heuristically evaluating where to apply the AC techniques to opti-

Figure 1.4 – Motivational example: comparison between single-technique and multi-technique approaches for the ADPCM application with resource savings of 20%



Source: The author

mize results, i.e., which operations or functional units to approximate, not assessing the most suitable techniques for each particular application and target objective.

Moreover, as can be seen in the example in Figure 1.4, which explores all implemented AC techniques detailed in Section 4.2 (BWR, LP, V2M, V2P, and V2Z), a combination of multiple AC techniques (ALL), applied to different circuit operations in the ADPCM application, leads to about 19% in MSE reduction for resource savings of 20%, when compared to the single technique providing the least error (BWR). As will be presented in Chapter 7, this pattern is observed, in different degrees, for all applications and constraints considered for evaluation. Therefore, aside from exploiting different techniques for different scenarios, there is the possibility of obtaining improved results by carefully selecting a combination of techniques, especially when facing multiple constraints, a possibility currently overlooked by existing approaches. As a result, either the task of finding which techniques to apply is left as an additional burden for designers, or the DSE becomes limited, which may produce substantially sub-optimal designs.

As the number of available AC techniques increases, applicable with varying trade-offs to different applications and substrates, so increases the size and complexity of the associated design space. Efficient approaches to incorporate new AC techniques and to effortlessly identify the optimal approximations for each scenario become necessary to allow seamless integration of the AC design paradigm in design flows aimed at short development time. Thus, to optimize results, the exploration of approximate circuits within HLS design flows requires dedicated heuristics to quickly identify which computations can be approximated, which AC techniques are adequate to use, and to what extent they should be applied to avoid increasing output errors beyond the strictly necessary. For this purpose, it is crucial to define a systematic way of estimating the impact of approximations on the design metrics of interest and the resulting error. Therefore, the challenge to be dealt with is providing an AHLS design methodology able to minimize output errors by automatically fine-tuning a combination of approximation strategies on a per-application basis and according to their suitability in optimizing specific design metrics toward meeting the specified constraints, assuming that they cannot be met by simply adjusting the HLS tool knobs or redesigning the system.

1.2 Thesis Contributions

The main contribution of this thesis is a novel methodology for automated synthesis of approximate circuits within HLS-based design flows for FPGAs. More specifically, we propose a constraint-aware AHLS methodology that automatically combines multiple AC techniques to produce approximate hardware, exploring a wide range of optimization opportunities toward meeting multiple constraints with minimum error. The proposed methodology focuses on extensibility, i.e., the ability to broaden the design space by easily incorporating additional AC techniques and considering their potential benefits in conjunction with those already available when trying to meet the target constraints. For that purpose, the AC techniques are implemented as code transformations within the compiler infrastructure of HLS tools, enabling the synthesis of approximate hardware from an easy-to-extend set of software-level approximations. **In a nutshell, this thesis proposes that an AHLS design methodology for FPGAs which is directly steered by one or more constraints on design metrics, and that combines multiple AC techniques, is able to synthesize approximate hardware with reduced error, when compared to single-technique or constraint-oblivious approaches**. Therefore, we highlight the following contributions:

- We show that a constraint-aware design methodology where the DSE is directly steered by constraints on the design metrics of interest can meet said constraints with a reduced error.
- We show that different AC techniques offer different trade-offs depending on the application at hand and the target optimizations, demonstrating the advantages of a constraint-aware approach when exploring multiple AC techniques.
- We show that no single AC technique is able to outperform an adequate combination

of techniques whose selection is steered by constraints on the design metrics of interest, with all available AC techniques being used to improve results in different scenarios.

These contributions are experimentally validated through an implementation within the LegUp HLS tool (Canis et al., 2013). A set of five AC techniques were implemented as transformation passes within the LLVM compiler framework (Lattner; Adve, 2004) to reduce the WCET and the use of FPGA resources like LUTs, REGs, and DSP blocks. The design heuristic leverages the Greedy Randomized Adaptive Search Procedure (GRASP) (FEO; RESENDE, 1995; RESENDE; RIBEIRO, 2019) to identify suitable schedules, i.e., sequences of AC transformations, that meet constraints with minimum error. As will be shown, the adaptive nature of GRASP makes it a promising option for exploring multiple AC techniques, especially when aiming to meet multiple design objectives differently impacted by each. The results over a set of image, video, signal processing, and machine learning benchmarks show that we can decrease the MSE by up to 52% and achieve an absolute increase of percentage accuracy (PA) by up to 20.3% when compared to approaches disregarding the benefits of steering the DSE with a constraint-aware heuristic targeting the design objectives. Moreover, additional reductions of up to 30% in MSE and absolute increases of up to 6.5% in PA can be attained when such an approach is combined with the exploration of multiple AC techniques.

The remaining of this thesis is structured as follows. Chapter 2 presents a theoretical background on HLS and AC, while Chapter 3 discusses how AC techniques are currently exploited within HLS toolchains, highlighting the contributions of this thesis. Chapter 4 details the design methodology proposed to address some of the challenges faced by current AHLS works and how it can be incorporated in state-of-the-art HLS tools. Chapters 5, 6 and 7 present the experimental setup and results for different constraints and AC techniques, respectively, to support the contributions herein presented. Finally, Chapter 8 presents conclusion remarks and future improvements.

2 BACKGROUND

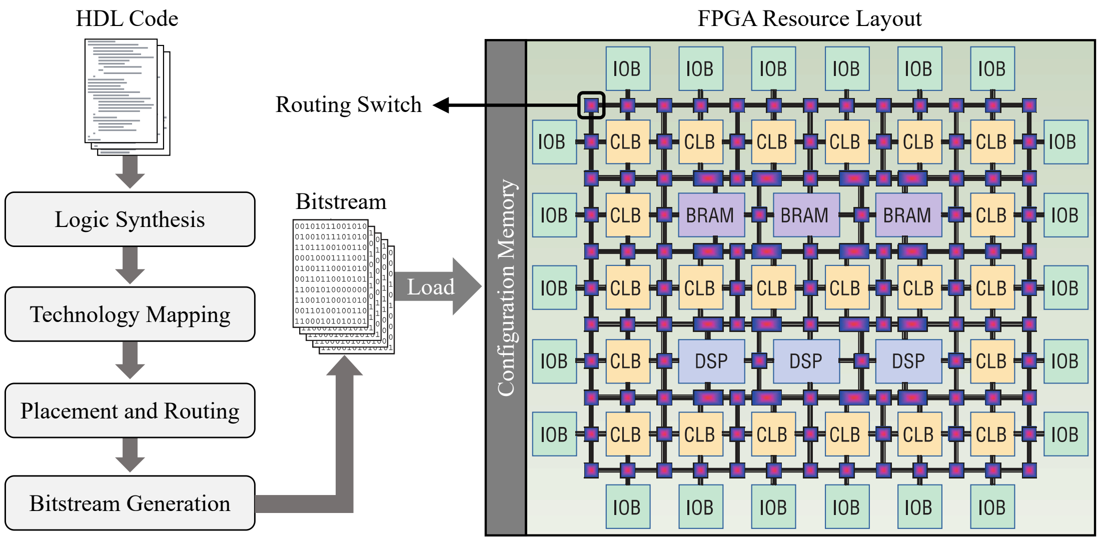
This chapter presents a theoretical background on High-Level Synthesis (HLS), Approximate Computing (AC), Approximate Logic Synthesis (ALS), and Approximate HLS (AHLS). We detail (i) the basic steps involved in generating a cycle-accurate RTL model from a high-level source code describing the design's behavior, (ii) the applications, principles, and techniques involved in adopting the AC design paradigm, and (iii) the methods and tools proposed in the literature to efficiently automate the exploration of AC techniques to yield near-optimal approximate hardware. Before that, it is important to have an overview of the typical structure of Field-Programmable Gate Arrays (FPGAs) and the basic steps involved in synthesizing hardware for such devices, as they become widely used to prototype, deploy, and adapt applications quickly, especially in the context of HLS design flows, and are the platform chosen to demonstrate the applicability of the contributions proposed in this thesis.

2.1 Field-Programmable Gate Arrays

FPGAs are reconfigurable devices widely used in computing systems to offer a compromise between the flexibility of software-based implementations and the performance per watt of ASICs, i.e., reconfigurability coupled with a high throughput processing of data streams and relatively low development time and costs. Such devices comprise a wide variety of heterogeneous resources. The typical layout of modern FPGAs is an array of interconnected blocks, including interconnecting resources, clock-management resources, Configurable Logic Blocks (CLBs), Input/Output Blocks (IOBs), and embedded blocks such as Digital Signal Processors (DSPs), general-purpose processors, high-speed IOBs, and memories (BRAMs). CLBs are used to perform simple combinational and sequential logic. These blocks typically consist of LUTs, multiplexers, flip-flops, and carry logic. Programmable interconnect resources, such as routing switches, allow interconnecting CLBs, IOBs, and embedded blocks to implement complex systems. The logic and routing resources in an FPGA are configured by the bits of a configuration memory, which may be based on anti-fuse, flash, or SRAM technology. More details about FPGAs architecture and usage can be found in (HAUCK; DEHON, 2007).

The design flow of FPGA-based systems involves the creation of a bitstream to load into the device, as illustrated in Figure 2.1. Typically, the process starts with the

Figure 2.1 – Typical FPGA structure and design flow



Source: adapted from Hauck and DeHon (2007)

system design written in a Hardware Description Language (HDL), such as VHDL or Verilog. Next, the design is optimized and mapped into the FPGA's available resources through logical synthesis, technology mapping, placement, and routing. Finally, the bitstream is generated, and the device can be programmed. FPGA vendors such as Xilinx, Intel, Lattice Semiconductor, and Atmel provide implementation tools to perform the design flow. Those characteristics make FPGAs a great option for accelerating applications that can take advantage of HLS and approximate computing, due to the fine-grained parallelism offered and the possibility of in-field reprogramming to correct faulty behaviors, add new features, or adapt their behavior timely, without the performance and power penalties introduced by the use of general purpose processors or rigid ASIC implementations with high non-recurring engineering costs. Several works are done on AC for specific system layers, like ASIC-oriented arithmetic circuit blocks, processors, and programming languages. However, the exploitation of AC on reconfigurable devices is more recent and the subject of intense research (PERRI et al., 2020; TOAN; LEE, 2020; PRABAKARAN et al., 2020; TIAN et al., 2017).

2.2 High-Level Synthesis

The HLS design methodology, sometimes called C synthesis, algorithmic synthesis, or behavioral synthesis, promises to reduce hardware design complexity by generating production-quality Register-Transfer Level (RTL) implementations from high-level specifications. It is a single process synthesis method mainly used to develop hardware accelerators targetting ASICs or FPGAs that transforms a design expressed in a High-Level Language (HLL), usually ANSI C, C++, or SystemC, into a Verilog or VHDL RTL description (COUSSY et al., 2009; Cong et al., 2011; Koch; Hannig; Ziener, 2016). In other words, HLS aims to automate an otherwise manual process, eliminating the source of many design errors and accelerating the very long and iterative part of the development cycle. Moreover, it can be used to accelerate applications initially implemented in software, bringing speed and energy advantages over performing computations in software running on a general-purpose processor due to the overheads of fetching and decoding instructions or loading and storing to the memory hierarchy. If designers can exploit hardware parallelism, drastic speed advantages are possible by executing computations in hardware concurrently that would otherwise need to be done sequentially.

HLS is especially appealing to reduce the complexity of designing hardware accelerators using design flows starting from a hand-coded RTL description. In a traditional RTL development flow, the early stage of the design specification is essentially functional, i.e., it contains little to no hardware implementation details, and its primary purpose is to validate and fine-tune the desired behavior. Once tested, the behavioral model undergoes a process made of several steps until it takes the form of the actual hardware implementation. The first step is to define an optimal architecture to implement the desired functionality, with direct consequences on performance, area, and power consumption. After defining the architecture, the design team hand-codes it in the form of an RTL description in Verilog or VHDL. However, while designer productivity has grown over the past decades, the rate of improvement has not kept pace with the demand for more complex applications, driven by the technological advances leveraged by increasing chip capacities. Finding an optimal architecture can be very challenging as the manual nature of this approach makes it error-prone, and thus many development cycles may be needed, especially when designing complex systems. Designers must test, report errors, and spend time finding and fixing them individually, a long and continuous process that has to end at some point to meet deadlines.

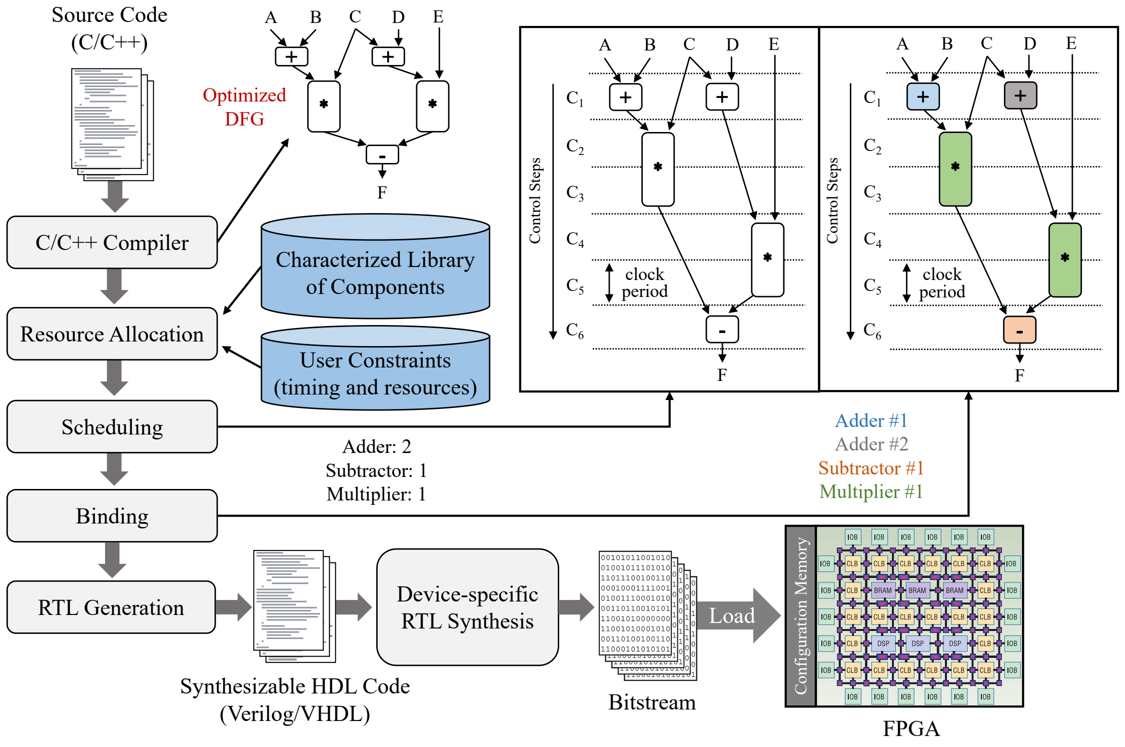
Moreover, as mentioned, the decisions that most impact the design quality in metrics like performance and power efficiency are made early in the design process when the system architecture and micro-architecture are defined. For example, decisions about the pipeline depth, datapath structure, and register allocation are usually part of the implementation stage, whether writing the RTL by hand or via HLS. While such decisions have the most impact, the extent of their impact is not well known until platform-specific RTL tools fully evaluate the implementation for the target device. As a result, designers have no choice but to make decisions based on previous experience. For experienced designers, that is often good enough, as any suboptimal decisions still have a good chance of meeting the overall design objectives. Even for them, however, there is the risk of missing significant optimizations.

Therefore, the HLS design methodology has become widely used to increase productivity by quickly generating RTL designs with diverse trade-offs, a result of its many benefits, e.g., faster design and verification, fewer bugs, and easier source code maintenance. As raising the abstraction level results in fewer lines of code compared to RTL descriptions, the adoption of HLS design flows can reduce the number of lines of code by an average of 10 times, leading to a shorter design cycle and fewer bugs and making it easier to verify and maintain (WAKABAYASHI; SCHAFER, 2008). Additionally, as the micro-architecture details are defined during HLS, designs described at higher abstraction levels are easier to write and re-target for new technology platforms than traditional hand-coded RTL descriptions, reducing time-to-market without jeopardizing design quality.

Current targets for HLS are mostly heterogeneous Multi-Processor Systems on Chip (MPSoCs), which include multiple processors, memories, interfaces, and application-specific hardware accelerators. Traditional FPGA vendors like Altera and Xilinx have released their own configurable MPSoCs devices, e.g., Altera's Cyclone V SoC and Xilinx's Zynq Ultrascale. More recently, Intel and AMD have acquired Altera and Xilinx, respectively, mainly based on the benefits of tight integration of x86 processors with FPGAs to design automotive, industrial, video, and communications applications. Therefore, it is not surprising that AMD and Intel have their own HLS tools to support designers in implementing complex systems for their devices. Such tools can be used, for example, to deal with the complexity of efficiently distributing the execution of application kernels across the heterogeneous resources offered by cloud infrastructures (LIGNATI et al., 2021).

Figure 2.2 shows an overview of the complete HLS design flow. It takes as inputs a behavioral description, a set of design directives and constraints, and a characterized library of hardware components for the target platform. The basic steps for HLS to produce an RTL design representing the given algorithmic description are resource allocation, scheduling, and binding. After parsing and optimizing the source code through a compiler infrastructure and assembling the Control Data Flow Graph (CDFG), the **resource**

Figure 2.2 – Typical HLS design flow for FPGAs



Source: The author

allocation task defines the necessary hardware resources, i.e., it defines the instantiated hardware resources according to the operations found in the system's Data Flow Graphs (DFGs). Such implementations are annotated with both timing and area information to be used later during scheduling. Any given operator may have multiple hardware implementations with different area, delay, and power trade-offs that can be selected from a technology-specific pre-characterized library containing sufficient components for a wide range of bit widths and clock frequencies. Therefore, the operations described in the source code can potentially be allocated to many different resources.

By default, the HLS tool will maximize the reuse of resources as much as possible to reduce the allocation of functional units (FUs) and thus avoid increasing the area. Nevertheless, designers can explicitly control resource allocation to insert pipeline registers or define the number of available resources by typically setting the maximum number of FUs the synthesizer can instantiate. For example, in Figure 2.2, the HLS tool must implement the design with at most two adders, one subtractor, and one multiplier.

Next, the **scheduling** task analyses the operations in the optimized DFG and decides the exact time step in which they will be executed, such that data dependencies and resource and timing constraints are not violated. The task of scheduling is dividing the design's CDFG into states, also referred to as control steps, so that it can be directly synthesized into a finite state machine model. Registers are added between operations based on the target clock frequency, similar to what is done in manual RTL designs with pipelining, by which registers are inserted to reduce combinational delays. Conversely, the allocation of registers can be reduced by chaining a sequence of operations connected by data dependencies in the same control step if the total delay is smaller than the target clock period. As exemplified in Figure 2.2, considering the given constraints and that multipliers have a delay of two clock cycles, the design's DFG is scheduled to execute in six control steps. Note that providing just one multiplier prevents the DFG execution in four clock cycles because the two multiplications cannot be placed in the same control step.

The amount of loop pipelining used in the design also affects scheduling by allowing a new iteration of a loop to be started before the current iteration has finished, i.e., it allows the execution of loop iterations to be overlapped, increasing the design performance by running them in parallel. The amount of overlap is controlled by the Initiation Interval (II), which determines how many clock cycles are needed before starting the next loop iteration, defining the number of pipeline stages. The II for loops is usually set either as a design constraint in the HLS design environment or through compiler pragmas. Additionally, loop unrolling can add parallelism to the design by enabling the scheduling of multiple loop iterations in parallel whenever possible, improving performance at the cost of increasing resource usage.

The resulting hardware generated from the schedule varies depending on how the design was constrained regarding clock frequency, resource allocation, and the amount of loop unrolling and loop pipelining used. As the DFGs expose the design's parallelism, each operation has a range of control steps that can be assigned. Regardless of resource constraints, the earliest and latest bounds within which operations in the DFG can be scheduled are determined by the As-Soon-As-Possible (ASAP) and the As-Late-as-Possible (ALAP) scheduling algorithms. These algorithms are especially useful to identify the operations mobility, widely used by more advanced scheduling algorithms considering time and resource constraints (HWANG; LEE; HSU, 1991; SLLAME; DRABEK, 2002; KUNDU; CHANDRAKAR; ROY, 2014).

After scheduling operations, the **binding** stage is the last step in HLS before the RTL description generation. Binding maps each operation to a FU and each variable used in more than one control step (loads and stores) to a register. More specifically, it assigns the design's operations and variables to specific hardware resources, given the resources available in the technology library, the resource constraints, and the design schedule. In short, the binding step of HLS is associating each computation in the behavioral description with a specific unit in the hardware such that performance is optimized without violating resource constraints.

Once the binding task is finished, an optimized RTL design described in Verilog or VHDL is generated, ready to be synthesized down to the basic components of the target substrate by vendor-specific synthesis tools, such as the Intel Quartus or the Xilinx Vivado. Note that there is an evident interdependence between the resource allocation, scheduling, and binding tasks, which makes generating high-efficient cycle-accurate hardware implementations from untimed algorithmic descriptions very challenging. Nevertheless, as HLS compilation has been the subject of research and engineering efforts for over three decades, current state-of-the-art tools can provide high-quality results quickly without requiring experienced RTL designers (LAHTI et al., 2019; Nane et al., 2016).

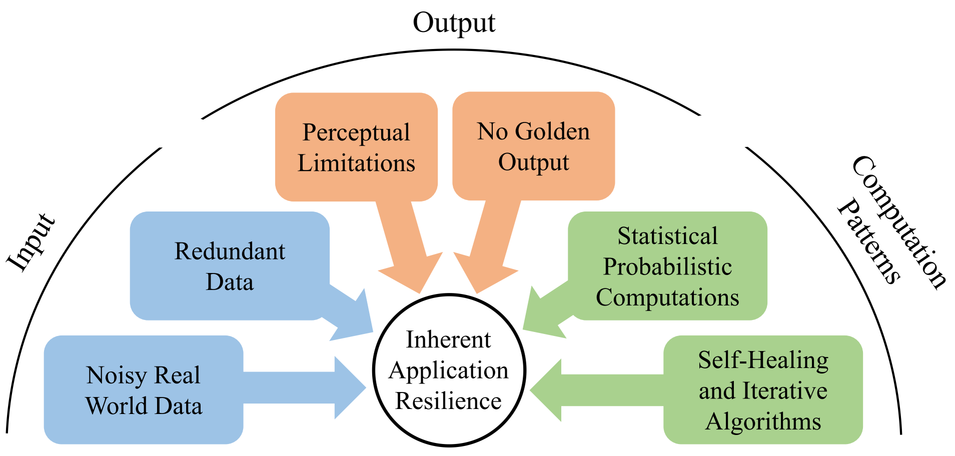
2.3 Approximate Computing

2.3.1 Applications and Principles

The AC design paradigm aims to reduce unnecessary costs (area, power, delay, and other design metrics) in error-tolerant applications by relaxing the traditional requirement of exact results. However, leveraging AC techniques to reduce design costs strongly depends on the nature of the applications. Applications well-suited to AC can be implemented in software or hardware that does not guarantee the execution of computations in a 100% correct manner. A large class of computing workloads, including digital signal processing, multimedia processing (image, video, audio), network processing, wireless communications, web search and recognition, data mining and big data analytics, computer vision, and machine learning, possesses characteristics that enable them to execute well on approximate software and hardware platforms. Such workloads exhibit inherent application resilience to approximations or the ability to produce acceptable outputs even when some of their computations are approximate. An analysis of a benchmark suite of 12 recognition, mining, and search applications showed that, on average, 83% of the runtime is spent in computations that can tolerate at least some degree of approximation (Chippa et al., 2013).

As illustrated in Figure 2.3, the source of inherent imprecision resilience of ap-

Figure 2.3 – Sources of Inherent application resilience


Output
Input
Computation Patterns
Perceptual Limitations
No Golden Output
Redundant Data
Statistical Probabilistic Computations
Noisy Real World Data
Inherent Application Resilience
Self-Healing and Iterative Algorithms

Source: adapted from Chippa et al. (2013)

plications usually relates to redundant or noisy input data (signal processing), the limited perceptual ability of users (multimedia processing), the inexistence or inability to offer perfect or golden outputs, i.e., several different outputs are equally acceptable (search or recommendation systems and machine learning), or computation patterns, such as statistical computations that result in attenuation/cancellation of errors (data analytics) and the iterative-refinement nature of some computations wherein errors due to approximations may be healed in the iteration process (scientific computations) (Han; Orshansky, 2013; Chippa et al., 2013). Therefore, given that many applications are error-resilient to some extent, the central challenge in AC is developing abstractions that make imprecision controlled and predictable without sacrificing the quality of results, allowing designers to attain accuracy-efficient trade-offs regarding the target optimizations (e.g., performance, area, power, and energy). In this direction, the literature has established some important principles to guide the development of AC design methodologies (Venkataramani et al., 2015).

First, **the notion of acceptable quality must be measurable**, and methods to ensure its sustainability when AC techniques are employed should be part of the design methodology. For example, some designers may wish to evaluate the worst-case error, the average-case error, or the error probability. Additionally, the error distribution for all possible inputs may be of interest in some cases. Therefore, the quality analysis should ideally be flexible enough to support multiple error metrics. Moreover, although the acceptable quality is an application-specific property and quality metrics vary across them (e.g., recognition accuracy or visual quality of images), the abstraction and methodology used to specify and validate quality should still be general, using functional verification concepts as much as possible.

Second, **approximate hardware or software components must be quality configurable** to be modulated according to the application usage, as resilience to approximations is not a static property of an application and may depend on both the input data processed and the context in which the outputs are used. For example, a machine learning algorithm for health-critical medical diagnosis may have much more stringent quality constraints than when used in a product recommendation system.

Third, **AC methodologies must be significance-driven**, i.e., it is paramount to separate resilient and sensitive computations and approximate the resilient ones based on how significantly they impact the output quality. For example, computations that involve pointer arithmetic or affect the control flow should be avoided, as they may lead to catastrophic effects when approximated. Even among resilient computations, the extent to which they impact application quality when approximated varies considerably.

Fourth, **AC methodologies should provide disproportionate benefits**, i.e., large improvements in efficiency for little to no impact on quality. For example, it is often beneficial to target bottleneck operations such as global synchronization and communication in software, or critical paths in hardware, to achieve performance improvements. Also, the sources of disproportionate benefits are usually spread across different computing stack layers, making a cross-layer approach where hardware and software techniques are co-designed more likely to produce better overall trade-offs. In short, the objective of AC methodologies and platforms should be to provide a range of quality vs. efficiency trade-offs according to user-defined optimizations, quality metrics, and error bounds.

With those principles in mind, the exploitation of AC in software and hardware has shown promising results (LIU; LOMBARDI; SHULTE, 2020). Software techniques typically improve performance by skipping tasks or reducing costly operations, such as inter-thread synchronizations or high-precision floating-point computations. In contrast, hardware techniques modify the design at various levels of abstraction (behavioral level, register transfer level, and physical level) through operations on inexact or faulty hardware. As discussed, approximate computing techniques should be targeted toward resilient computations while avoiding sensitive ones, as even the most resilient applications contain both resilient and sensitive computations. Once variables and operations that can be approximated are identified, a variety of strategies can be employed at different abstraction layers (software, architecture, and circuit) to explore unique trade-offs (Mittal, 2016; Shafique et al., 2016; XU; MYTKOWICZ; KIM, 2016; STANLEY-MARBELL et al., 2020).

2.3.2 Software-level Techniques

Software-level AC techniques leverage code transformations to reduce code size and simplify its execution on general-purpose or specialized processors through fewer and lower-cost instructions, optimizing performance and energy consumption. Although such techniques are primarily used to approximate software-based implementations, their scope can be extended to generate approximate hardware by incorporating them in HLS design flows. Therefore, some of the presented techniques will be incorporated into our experimental flow, detailed in Chapter 5.

Precision scaling is one of the most effective software-level AC techniques. It is a fine-grained approximation strategy that relies on changing the precision of inputs or intermediate operands (bit-width reduction) to reduce storage requirements or processing demands (Yeh et al., 2007; RUBIO-GONZáLEZ et al., 2013; ANAM; WHATMOUGH; ANDREOPOULOS, 2013; RAHA et al., 2015; PARK; CHOI; ROY, 2010; YESIL; AKTURK; KARPUZCU, 2018). For example, reducing data width in some computations from 64 to 16 bits may provide significant performance and energy improvements through data path simplification.

On the other hand, **loop perforation** can be used to skip some iterations of loop-based computations to reduce the computational effort needed and trade-off accuracy with performance (Sidiroglou-Douskos et al., 2011; LI; PARK; MAHLKE, 2018). It has been shown that the so-called hot loops in error-tolerant applications can be perforated by up to 50% with a similar reduction in execution time while still producing acceptable results. Note that the decision of which loops and iterations can be perforated must be carefully evaluated to avoid excessive quality degradation as it is application-dependent.

Variable-to-constant substitutions (V2C) replaces an operation's output with a constant value, possibly propagating optimizations to other operations and variables due to control and data dependencies (Nepal et al., 2014; Nepal et al., 2019). A common approach for V2C is replacing operations with data statistics, such as the mean values obtained through code instrumentation and simulation with a set of training inputs. In this case, the decision if the operation should be replaced can be based on a given threshold for the standard deviation (Xu; Schafer, 2017; XU; SCHAFER, 2018; CHOWDHURY; SCHAFER, 2021). Another option to leverage V2C substitutions is using **probabilistic**

pruning, which allows designers to achieve area, power, and performance improvements by replacing costly operations with constant values based on their execution probability and significance in affecting primary outputs, usually defined according to error propagation analysis and data statistics (Lingamneni et al., 2013; BARBARESCHI; IANNUCCI; MAZZEO, 2016). A more aggressive form of V2C substitution is replacing the operation's output with a constant zero (V2Z) (Lee; John; Gerstlauer, 2017). Although this approach may introduce more errors than the mean value if the standard deviation is small, the constant zero can, for example, eliminate adders ($X + 0 = X$) and propagate through multipliers ($X \times 0 = 0$), enabling further optimizations throughout the code.

Instead of replacing operations with constant values, some works employ a less aggressive form of operation pruning called **variable-to-variable substitutions** (V2V) (Xu; Schafer, 2017; XU; SCHAFER, 2018; CHOWDHURY; SCHAFER, 2021). With V2V, a variable's computation is replaced by another variable's output if their output values are similar. The similarities are previously identified by code instrumentation and simulation to calculate statistics such as the mean value (μ) and the standard deviation (σ) of each variable's output. Then, for example, if two variables are within each other's $\mu \pm \sigma$, it can substitute one variable with the other's output. Consequently, the code needed to calculate one of the variables will be fully pruned away.

Besides eliminating operations, **operation transformations** can be employed to simplify arithmetic operations, for example, by replacing additions with bitwise ORs and multiplications with shifts and additions. A similar technique is **arithmetic expression transformations**, where the computations of near similar arithmetic structures are simplified by sharing common or similar operands through V2V substitutions. For example, we can approximate the expression $(a \times b + c \times d)$ by substituting d by b if they are similar, leading to $(a + c) \times b$, thus saving one multiplier (Nepal et al., 2014; AWAIS; MOHAMMADI; PLATZNER, 2018; Nepal et al., 2019).

With **load value approximations**, load values are estimated when a load miss occurs in cache memory, avoiding the latency associated with fetching the data from the next-level cache or main memory. Therefore, the cache miss latency can be hidden by allowing the processor to progress without stalling for a response (Miguel; Badr; Jerger, 2014; YAZDANBAKHSH et al., 2015).

Conversely, **memoization** works by storing the results of previous computations of functions for later reuse when an identical input arrives for the same function, skipping repeated computations (Rahimi; Benini; Gupta, 2013). Therefore, memoization can potentially improve performance and save energy by trading costly computations involving elementary functions, for example, for a few memory operations (MULLER, 2020). In error-tolerant applications, the results of specific operations can often be reused for similar inputs to increase the memorization scope at the cost of introducing more errors. Other methods to achieve efficiency with bounded quality loss are **selectively skipping tasks and memory accesses** in multi-core architectures or processing a portion of the input data through data sampling (Samadi et al., 2014; VASSILIADIS et al., 2015; GOIRI et al., 2015).

There are also accelerators employing **neural networks to substitute compute-intensive kernels** and execute code regions more efficiently (ESMAEILZADEH et al., 2012b; GRIGORIAN; REINMAN, 2015; GRIGORIAN; FARAHPOUR; REINMAN, 2015). The neural network is trained to approximate the code regions annotated by the programmers. Also, there are AC frameworks dedicated to **approximate Artificial Neural Networks (ANNs)** and applications based on iterative methods for solution convergence (ZHANG et al., 2014; ZHANG et al., 2015). With a more general approach, in Oliveira et al. (2018), the authors propose using a tree-based classification algorithm as an approximation tool for general-purpose applications to improve performance and reduce the energy-delay product without hardware support.

2.3.3 Architecture-Level Techniques

Architecture-level AC techniques usually apply to processors, memory, and storage subsystems. Ideally, the system architecture is defined to balance performance with energy efficiency under various constraints imposed by a given technology, such as chip area and power. It also attempts to balance density (or the cost per bit) with performance for memory and storage. For example, with **data storage approximations**, the performance, lifetime, and density of solid-state memories can be improved by enabling applications to store data approximately (SAMPSON et al., 2014; LI et al., 2019a; FROEHLICH; GROSSE; DRECHSLER, 2020).

Adopting memory architectures with **unreliable memory arrays** through aggressive voltage over-scaling or reduced refresh rates (for DRAMs) can also provide power efficiency for error-resilient applications like video processing and deep learning (CHANG; MOHAPATRA; ROY, 2011; NGUYEN et al., 2020). Similarly, the properties of such applications can be leveraged to approximate data and reduce energy consumption in cache architectures by shutting down error correction mechanisms or reducing the data size through similarity-based encoding schemes (SAMPAIO et al., 2015; MIGUEL et al., 2015; RANJAN et al., 2020; ZHAO et al., 2021).

More coarse-grained techniques, such as **selective approximations**, work by executing some chosen instructions or code segments on approximate hardware to improve performance (VENKATARAMANI et al., 2013). In that direction, architectural support for approximate programming can be offered by extending the Instruction Set Architecture (ISA) of general-purpose processors to efficiently map approximate operations and storage to hardware (ESMAEILZADEH et al., 2012a). Domain-specific knowledge can also be used to design application-specific accelerators using specialized processors that leverage the approximate nature of some computations to improve performance and reduce power consumption (CHIPPA et al., 2014; NDOUR et al., 2019; JOE; KIM, 2019; DU et al., 2021; KUNDI et al., 2022).

2.3.4 Circuit-Level Techniques

At the circuit level, **approximate arithmetic units** (adders, multipliers, and dividers) or entire datapaths have been broadly studied based on the general principle of significance-guided design, i.e., fewer resources should be provided for insignificant computations with lower complexity (ZHU et al., 2010; KULKARNI; GUPTA; ERCEGOVAC, 2011; Kahng; Kang, 2012; FARSHCHI; ABRISHAMI; FAKHRAIE, 2013; DRANE; ROSE; CONSTANTINIDES, 2014; Aksoy; Flores; Monteiro, 2015; Becher et al., 2016; HASHEMI; BAHAR; REDA, 2016; LIU et al., 2016; YIN et al., 2016; SCHLACHTER; CAMUS; ENZ, 2016; REHMAN et al., 2016; ULLAH et al., 2018; PRABAKARAN et al., 2018; SAADAT; JAVAID; PARAMESWARAN, 2019). For example, an approximate adder structure for LUT-based FPGA technology is proposed by Becher et al. (2016). The critical path is significantly shortened when compared with an accurate carry-ripple adder, which enables increasing the clock frequency. Therefore, the throughput of an FPGA-based system using this adder may be significantly improved. Moreover, the authors claim that the average error can be reduced, when compared to similar ASIC implementation approaches.

Focusing on more specific applications, Aksoy, Flores and Monteiro (2015) present an approximation of Multiple Constant Multiplications (MCM) for FPGAs, aiming to reduce the implementation costs associated with this operation in hardware. An exact algorithm is introduced to find the minimum number of distinct LUTs required to realize the partial products of constant multiplications without violating the defined error constraint. For example, MCM is commonly used in many DSP systems to perform Fourier transforms and implement Finite Impulse Response (FIR) filters. In that direction, many types of **reconfigurable approximate arithmetic units** that allow adjusting power and energy consumption by mixing approximate and accurate operations at the register transfer level were proposed (GUPTA et al., 2011; YE et al., 2013; LIU; HAN; LOMBARDI, 2014; HU; QIAN, 2015; SHAFIQUE et al., 2015; HASHEMI; BAHAR; REDA, 2015; MAZAHIR; HASAN; SHAFIQUE, 2018; HASSANI; REZAALIPOUR; DEHYADEGARI, 2018).

Regarding **memory-based approximations in hardware**, the work proposed by Sinha and Zhang (2016) concentrates on developing an automatic synthesis flow to exploit the benefits of memoization for AC on FPGAs. Minimizing the magnitude of errors in faulty hardware according to a constrained quality, such as shifting the least significant bits of data in the faulty cells of a memory, are common approaches too (Ganapathy et al., 2015). Roldao-Lopes et al. (2009), for example, propose using AC to accelerate the solution of a system of linear equations on FPGAs based on iterative solvers. To achieve the desired quality of results (QoR), the user can lower the precision of intermediate computations and run more iterations. For a fixed area constraint, lowering the precision of the iterative solvers increases the parallelism. Therefore, by balancing the operation precision and iteration count, they achieve a significant performance improvement (26 times on average) with controllable quality degradation.

More recently, the use of AC to reduce the costs of fault-tolerance architectures based on **partial hardware redundancy** has become the subject of research (SáNCHEZ-CLEMENTE; ENTRENA; GARCíA-VALDERAS, 2016; RODRIGUES; KASTENSMIDT; BOSIO, 2021; NAZAR et al., 2021a). For example, Sánchez-Clemente, Entrena and García-Valderas (2016) propose using approximate logic circuits to build a partial Triple Modular Redundancy (TMR) scheme on FPGAs. Although TMR is a very effective mitigation technique, it is often expensive in terms of FPGA resource usage and power consumption. Therefore, partial TMR can be used to trade off the reliability with the design costs for applications that can tolerate some temporary misbehavior. Similarly, Nazar et al. (2021a) present a methodology to exploit approximations in both modules of a dual modular redundancy scheme to accept a degree of imprecision even in the absence of faults, reducing the area overhead of duplicating the system while optimizing throughput.

Additionally, the proposed technique is integrated into an HLS tool to automate the DSE of circuits with varying costs and degrees of approximation.

Designers can also explore **approximations to reduce the system's supply voltage** to save energy at the cost of possible errors (MOHAPATRA et al., 2011; KRAUSE; POLIAN, 2011; MIAO et al., 2012; RAMASUBRAMANIAN et al., 2013; MOONS; VERHELST, 2015; LI et al., 2019b; ZERVAKIS et al., 2019). For instance, erroneous computations may occur in critical paths due to the increased delay. Moreover, the probability of wrong memory operations increases, such as writing a wrong value or flipping a bit during read (Sampson et al., 2011).

Considering that ANNs can be efficiently accelerated by dedicated hardware due to their inherent parallelism, an ANN model can be trained to offer performance and energy gains by mimicking exact computations in hardware (St. Amant et al., 2014). Instead of executing the original code, the neural network model can be invoked to execute on a Neural Processing Unit (NPU) accelerator implemented in reconfigurable logic, which leads to better efficiency as neural networks are amenable to efficient hardware implementations, accelerating a broad range of applications with bounded quality degradation (Moreau et al., 2015). Moreover, the flexibility of reconfigurable devices allows modifying the neural network topology according to the functionality implemented to manage the acceptable error bound. Since ANNs are typically used in error-tolerant applications, they can be approximated in hardware as well (Venkataramani et al., 2014). In this direction, Castro-Godínez et al. (2020a) present a method to explore the resilience of applications based on convolutional neural networks (CNNs) for IoT edge devices. That work proposes an approximate accelerator to speed up the execution of the convolutional layer, which is the most time-consuming component of CNNs. The CNNs are trained and deployed on an SoC with reconfigurable hardware resources.

2.3.5 Current Challenges

The most efficient AC technique and where it can be applied to avoid excessive quality degradation or correctness issues heavily depends on the application at hand and the target optimizations. Moreover, more than one technique can be used simultaneously in different parts of the system to optimize results. It is also crucial to consider the platform where the application will be implemented when choosing the most suitable techniques since different memory technologies, processor components, and processing units offer different trade-offs, which requires careful evaluation. As design metrics such as performance, power, and resource usage are usually known after device-specific synthesis, exploring the design space of approximate hardware at higher abstraction layers also imposes the challenge of early estimating the impact of approximations on design metrics and performing error-bound analysis and management to optimize decisions, not to mention reliability and security issues as possible affected domains. Therefore, design methodologies and tools for automating, at least in part, the exploration of approximate designs across the software and hardware stacks have become an important research topic. Although current works have shown great promises for specific domains and optimizations, significant innovations and research efforts are still needed to enable AC as a practical mainstream computing paradigm.

3 RELATED WORK

The automated synthesis of approximate hardware to reduce the development time and effort required for adopting the AC design paradigm has been the subject of numerous recent works. As this area of research is relatively new, these works usually differ in many important aspects: (i) the target platforms (e.g., ASICs, FPGAs, or both), (ii) the AC techniques explored, (iii) the software and hardware abstraction layers where the AC techniques are implemented in the synthesis process (e.g., source code, HLS, RTL, and gate-level), (iv) the metrics and methodologies used for evaluating the quality of results, (v) the optimization targets (e.g., area, delay, or power), and (vi) the developed optimization algorithms.

These works can also be divided into ALS methodologies, where the exploration of AC techniques is limited to the RTL representation and below (usually the gate-level netlist), and AHLS methodologies that expand the design space by leveraging approximations at higher abstraction layers, including the high-level source code, the compiler intermediate representation, and the HLS basic tasks to produce the RTL design (resource allocation, scheduling, and binding). Mostly, the trend is to provide error-constrained design methods for specific optimizations capable of generating approximate circuits that never exceed a predefined error threshold. However, approaches employing unconstrained design methodologies using greedy and evolutionary algorithms to produce Pareto-optimal designs with varying trade-offs can also be found.

3.1 Approximate Logic Synthesis

Many works on ALS leverage structural netlist transformations as a general approach to exploring approximate circuits. These works usually rely on greedy heuristics for netlist pruning and manipulation or stochastic transformations.

Shin and Gupta (2011), for example, employ a greedy iterative heuristic for generic circuit simplification, applied to adders used in image compression and decompression. The circuit area is used as the cost metric for minimization. A test generation algorithm is responsible for identifying multiple Stuck-At-Faults (SAFs) that can be injected (static 0 or 1) to introduce errors of low severities while providing significant area reductions due to operations elimination. The circuit netlist is simplified forward and backward, and the process is repeated until the error constraint is violated. A parallel fault simulator with a set of test vectors is used to evaluate the error on the final output at each SAF simplification.

Schlachter et al. (2017) presents another greedy iterative algorithm for netlist simplification. The proposed framework represents the exact circuit as a Direct Acyclic Graph (DAG). Then, its nodes are pruned according to two metrics: the node significance, which represents the impact of that node on the final output, and the node activity, expressed as the toggle count. Depending on the application characteristics, nodes can be pruned starting from those with lower significance, lower activity, or a combination of the two using the Significance–Activity Product (SAP). Node activity is obtained through gate-level hardware simulation, while the significance is computed in a reverse topological graph traversal. After nodes have been ranked according to the desired metric, the framework iteratively removes a node from the original circuit, setting its output to a constant. Next, the transformed netlist is synthesized and simulated with a Monte Carlo process to verify if the error constraints (error rate and mean relative error) have not been violated, recomputing the SAP for node ranking. When the error threshold is reached, the algorithm stops. As computing node activity is time-consuming due to gate-level simulations, using only the node sensitivity is preferred for large designs, at least for a first evaluation.

As opposed to other works performing netlist pruning heuristically, Scarabottolo, Ansaloni and Pozzi (2018) resort to an exhaustive netlist exploration to identify the maximum portion of the exact circuit that can be eliminated through gate-level pruning to derive an inexact version bounded by a given error threshold. They propose an algorithm based on binary tree exploration that identifies the exact influence of each circuit gate on the output correctness by employing back-tracking. The best candidate sub-circuit is the largest one (in terms of the number of gates) that does not overcome the given error threshold. The key idea of this approach is to consider the combined effect of multiple pruning choices, eliminating the risk of getting stuck in local minima. However, the algorithm's efficiency strongly relies on accurate estimation of node significance, either through exhaustive simulation (if the circuit size allows it) or by exploiting the circuit regularity to derive node significance through induction. Even when the exploration is stopped after reaching a time limit, this method has shown to perform better than the greedy-based heuristic proposed by Schlachter et al. (2017), to which it is compared in terms of energy, delay, and area reduction for the same constrained error. Although that work does not focus on generating libraries of approximate arithmetic operators, it can be used for this purpose as the approach applies only to combinatorial circuits.

Focusing on netlist manipulation, Venkataramani, Roy and Raghunathan (2013) propose automating the design and synthesis of quality configurable approximate circuits by systematically identifying signal pairs in the circuit's netlist that carry the same value with high probability and substitute one for the other. These substitutions are carefully performed to introduce functional approximations that result in logic elimination from the circuit while also enabling the downsizing of gates on critical paths to boost power savings. When the target signal is replaced, the gates belonging exclusively to its generating cone of logic are removed from the circuit. The proposed framework, named SASIMI, performs substitutions and simplifications through a greedy optimization algorithm that takes as input the original circuit and a target error and then iteratively selects the best candidate signal pair, the substitution, and consequent circuit simplification, followed by quality evaluation. Once the target error constraint is reached, the iterative algorithm stops. Moreover, based on the desired quality, the discrepancy between two signals can either be ignored or recovered using an additional clock cycle to recompute the logic that generates the substituted signals.

The error behavior of approximated circuits (e.g., error rate or error magnitude) depends heavily on the specific synthesis technique and the input vectors, often hindering designers from confidently adopting AC design methodologies. Tracking this problem, Liu and Zhang (2017) propose a statistically certified approximate logic synthesis (SCALS) framework using techniques from stochastic optimization. The authors apply statistical hypothesis testing to estimate the errors obtained on the circuit outputs after approximations to guarantee that the population behavior is a reliable representation of the actual data distribution. First, the original circuit is mapped to the desired technology in order to work directly on what will be the actual implementation. The mapped netlist is then partitioned into sub-netlists, which will be independently approximated in parallel. That work implements three gate-level approximations: REDUCE, FLIP, and ADD. REDUCE randomly pick a logic gate from the netlist and removes one of its fanins, whereas FLIP inverts its output. An ADD transformation, instead, inserts a two-input logic gate to the circuit, connecting it to existing signals at random. For each logic transformation, the approximate sub-netlist is mapped to the desired technology to assess its quality. Statistical inference techniques are then used to estimate the errors introduced in this phase since hypothesis testing for each transformation would be computationally infeasible. If the quality improves, the sub-netlists are updated, and the process starts again until a fixed point is reached. The optimized sub-netlists are then recombined, and the resulting netlist error is evaluated through statistical hypothesis testing. This sequence of operations represents a single trial, and the process continues until the specified error constraint is reached. Therefore, they can provide a confidence level for the analyzed error metrics, namely error rate, and average relative error magnitude. Note that the transformation space is explored stochastically, which means considering possible transformations randomly to maximize the number of design points tackled.

In a similar direction, Vasicek and Sekanina (2015) propose a genetic algorithm to mutate the circuit into approximate versions by swapping gates with wire connections to optimize area. Circuits are represented as DFGs, whose nodes can be Boolean gates or more complex components according to the technology library. The nodes are arranged in a 2-D grid representing the chromosome, which is randomly modified to explore new design points. This mutation evolves using a fitness function to lead to better approximation choices. After computing the area and error of the initial population, the algorithm iteratively selects the best-scored circuit, generates offspring from the parent through mutation, and evaluates the new population. Full simulation is employed for small circuits to evaluate the resulting error of each mutation. In contrast, the authors resort to more complex error analysis techniques based on Boolean Satisfiability (SAT) or Binary Decision Diagrams (BDDs) for larger circuits. Moreover, the design process can be repeated many times in order to obtain various trade-offs between accuracy and area. Vasicek and Sekanina (2016) use this same approach to deal with the approximation of general combinational logic (e.g., pattern matching circuits and complex encoders) in which no additional information is available to establish a suitable error metric. For that cases, the error of approximations is expressed in terms of the Hamming distance between the output values produced by the approximate candidate and the accurate circuit., the error of approximations is expressed in terms of the Hamming distance between the output values produced by the approximate candidate and the accurate circuit.

Rather than directly performing transformations on the design's netlist, numerous works on ALS propose logic rewriting using Boolean representations. Such ALS methodologies capture the circuit's logic in a formal Boolean representation and then transform it to yield an approximate Boolean representation, which is synthesized to a gate-based netlist.

The work presented by Venkataramani et al. (2012) is one of the earliest ALS methodologies to propose logic rewriting-based approximations for combinational circuits using Boolean simplifications. Given the circuit's RTL specification and a quality constraint, the SALSA methodology synthesizes an approximate version that meets a predefined quality bound. The authors map the approximate synthesis problem into an equivalent traditional logic synthesis problem, allowing the capabilities of existing synthesis tools to be leveraged for approximate logic synthesis. Specifically, SALSA encodes the quality constraints using logic functions called Q-functions, constructed by comparing the outputs of the exact circuit and the approximate circuit. Depending on the error metric, the Q-function can compute the arithmetic difference or the Hamming distance between the exact and the approximate outputs. Then, to simplify the logic of the approximate circuit, SALSA computes the observability of don't cares for each approximate output relative to the primary outputs of the Q-function. These don't cares represent the primary input combinations for which the outputs of the Q-function are insensitive to the approximate circuit outputs. Therefore, they can be used for circuit simplification using traditional don't care based minimization techniques.

As SALSA targets only combinational circuits, it has been extended by Ranjan et al. (2014) to handle sequential circuits as well, where errors may arise over multiple clock cycles. Given a sequential circuit and an output quality constraint, their method, named ASLAN, creates an approximate version of the circuit that consumes lower energy while meeting the specified quality bound. It uses a circuit block exploration method to evaluate the impact of approximating the combinational blocks in the sequential circuit, generating local quality-energy trade-off curves for them. Then, a gradient-descent approach is employed to find good approximation trade-offs for the entire circuit.

In contrast to SALSA's and ASLAN's global minimization approach, Wu and Qian (2016) propose a local minimization approach for multilevel ALS under error rate constraints. The basic idea is to pick nodes in a Boolean network and shrink them by approximating their factored-form expressions. Each of these Boolean approximations impacts the error and circuit complexity differently, as measured by the design area. Aiming to identify the best expressions to apply the simplifications and the particular form of simplification that leads to minimal area, each Boolean expression is given both a value, defined as the reduction in the area it realizes when simplified, and a weight, defined as the introduced error. Then, a knapsack formulation is constructed and solved to identify the best set of nodes to approximate to maximize value (total area reduction) given weight constraints (maximum error).

Complementary to previous works, Hashemi, Tann and Reda (2018) introduce a new formal method for logic rewriting approximations by synthesizing approximate circuits using Boolean Matrix Factorization (BMF). Such methodology approximates the truth table of the exact design's sub-circuits using BMF to control the approximation degree and the factorization results to synthesize a less complex sub-circuit. A method for circuit decomposition is also employed to scale the methodology to large circuits, and a sub-circuit design-space exploration technique is used to identify the best order for sub-circuit approximations. The goal is to provide smooth trade-offs between accuracy and area and power consumption.

Some works rely on BDDs to approach approximate synthesis (SOEKEN et al., 2016; FROEHLICH; GROßE; DRECHSLER, 2017). Reduced-order BDDs are a canonical representation for Boolean circuits that can be minimized to reduce the synthesized circuit's size. The general idea of such approaches is first to represent an input circuit as a BDD and then transform it to produce an approximate and smaller BDD representation that adheres to an error threshold. If the error between the original BDD and approximate BDD does not exceed the given error threshold, the approximate BDD is accepted and synthesized to a circuit. For example, Soeken et al. (2016) present operators to derive approximated functions and algorithms to precisely compute the error metrics directly on the BDD representation. One of the techniques employed is replacing a node with one of its children, i.e., co-factors. Another possibility is rounding, where a child of a node is either replaced by terminal 1 or terminal 0. Since each internal node has two children, the child with a fewer number of assignments that lead to terminal 1 is chosen. This approach, however, does not guarantee a minimum BDD size for a target error bound. Nonetheless, Froehlich, Große and Drechsler (2017) face this problem by devising an error-constrained exact algorithm to solve the Error Bounded Exact BDD Minimization (EBEBM) problem when trying to find a functional approximation with a minimal representation in terms of BDD size for a single output function.

Using AND-Inverter Graphs (AIGs) as the Boolean representation for netlist simplification is another approach used in ALS. An AIG is a representation of circuits where nodes correspond to two-input AND gates, and edges can be either inverted or not. Although AIGs provide a scalable graphical representation for circuit synthesis, they are not canonical as BDDs. Nevertheless, Chandrasekharan et al. (2016) propose an algorithm for approximate AIG rewriting that can guarantee error bounds for the generated approximate circuits. To accomplish that result, first, the critical paths are identified in the AIG, where the critical paths are the paths from the primary inputs to the primary outputs with the largest number of nodes. A SAT solver is then used to compare the original AIG and the approximate AIGs to check whether the error constraint is violated, hence guaranteeing the error bound.

Besides power optimizations, performance improvements are also sought by Alan and Henkel (2018), wherein a general-purpose systematic logic synthesis methodology is proposed to assess potential delay improvements in noncritical paths. As most input combinations do not invoke the critical path, they can be executed in a shorter time by introducing timing speculations. Given the RTL code, the main idea is to synthesize circuits with tight constraints toward minimizing the number of near-critical paths, reducing the probability of introducing errors due to timing violations when frequency or voltage are scaled, opening the opportunity for performance and energy optimizations.

In face of vast approximation techniques targetting netlist optimizations, Castro-Godínez et al. (2021) present a framework for ALS techniques based on netlist transformations, where the gate-level representation of a circuit is simplified to reduce the circuit area and power consumption while respecting a user-provided accuracy threshold. As the implementation of netlist-based approximations is challenging due to the complexity of transforming low-level circuit representations, the authors propose a systematic approach to facilitate the implementation and evaluation of existing netlist-level approximations, usually based on pruning nodes of the netlist's DFG or replacing gates and connections, and encourage the proposal of novel methodologies.

Although significant advances have been made in ALS, some limitations prevent a wider adoption for exploring approximate hardware implementations. In general, scalability is one of the major concerns. ALS methodologies that rely on BDDs and BMF for Boolean representation and optimization, for example, cannot handle large circuits and, as a result, need circuit partitioning methods to break down the circuit into manageable sub-circuits. However, this partitioning often reduces the quality of results attained from these methods. Conversely, approaches exploring gate-level transformations, whether using exhaustive, greedy iterative, stochastic, or evolutionary algorithms, also suffer from scalability due to time-consuming optimization and error analysis arising from operating at complex low-level circuit representations. Indeed, most ALS proposals are better suited for approximating small combinatorial circuits or exploring approximate arithmetic operators, being impractical for more general applications with large designs comprising complex data and control paths.

3.2 Approximate High-Level Synthesis

The ALS design methodologies discussed in Section 3.1 operate on either the RTL code, the gate-level netlist, or a Boolean representation of the circuit. In contrast, AHLS aims to integrate software-level AC techniques and inexact operators as building blocks for the RTL design assembled by the resource allocation, scheduling, and binding HLS tasks to efficiently implement approximate circuits from the design's algorithmic specification, usually in C/C++ or SystemC. Similar to ALS approaches, the input for AHLS flows is the original design specification to be analyzed and modified by the HLS tool to produce approximate hardware for the target platform.

In this direction, Nepal et al. (2014) raise the level of abstraction where approximation techniques are implemented to the behavioral HDL code and present a tool for automatic DSE of approximate circuits that can be incorporated in standard design flows for ASICs and FPGAs to generate Pareto-optimal approximate circuits adhering to a given error threshold. That tool, named ABACUS, uses a greedy iterative heuristic to randomly apply precision scaling (PS), operation transformations (OT) (e.g., sum to logical disjunction and multiplication to logical conjunction), arithmetic expressions transformations (AET), loop transformations (unrolling and perforation), and V2C substitutions on the circuit's Abstract Syntax Tree (AST) generated from the HDL code, aiming to achieve area and power savings. The approximate variants of the AST are written back to Verilog code, simulated for error evaluation, and synthesized for the area and power evaluation. A similar solution is presented by Nepal et al. (2019), wherein the authors merge the iterative greedy heuristic with a Non-dominated Sorting Genetic Algorithm (NSGA) to prioritize approximations on the critical path, creating a positive slack that can be exploited to reduce the voltage while keeping the frequency intact. Therefore, this approach leads to additional power savings beyond those provided by approximate logic by enabling aggressive voltage scaling. As a result, it can yield low-power approximate circuits for ASICs with varying accuracy trade-offs.

Following a similar approach, Awais, Mohammadi and Platzner (2018) introduce a framework for automated synthesis of the approximate circuits using as input the behavioral HDL code, which is parsed to determine the potential operands and operations to approximate. A notable difference is that the authors employ an adapted version of the Monte Carlo Tree Search (MCTS) technique rather than a greedy iterative algorithm to deal with the large design space. Given an error threshold, the potential approximations using precision scaling, operation transformations (e.g., partially or completely replacing additions with bitwise OR or XOR operations), and loop unrolling are explored by the MCTS algorithm to generate Pareto-optimal designs in terms of accuracy, power, and area.

The work presented by Li et al. (2015), on the other hand, proposes to join precision optimizations and the HLS basic tasks to minimize total leakage energy consumption in ASICs. Given precision constraints (user-defined), the system DFG, and a library of approximate adders and multipliers with varying bit widths, that work explores the design space of approximate designs through a knapsack-based optimization algorithm using Integer Linear Programming (ILP) with a statistical precision scaling model integrated into traditional scheduling and binding HLS tasks. The objective is to find an optimal combination of exact and approximate operators to trade off errors with energy consumption. The drawback of this approach is related to the use of linear quality and energy models supporting only combinatorial circuits, which limits its use for general applications. Lee, John and Gerstlauer (2017) go further and exploit joint precision and voltage scaling to increase energy savings even more. Given the accurate high-level C description, output quality constraints, and a testbench, the authors collect data statistics and branch probabilities from simulations of the accurate design. Additionally, they obtain the operations mobility and total latency through ASAP and ALAP pre-schedulings. Next, given the data and operations to approximate (decision variables in the C code) and an approximate library of adders and multipliers, their tool performs precision scaling (data rounding) and operation pruning (substitution by zero) through a quality-energy optimization solver. To estimate quality degradation and energy savings, the solver uses the profiled information and a semi-analytical model based on the system CDFG. Next, the precision-optimized solution is scheduled to minimize latency. Finally, precision and timing slack are rebalanced across clock cycles to minimize the critical path and maximize voltage scaling.

Targeting performance optimizations, Lee and Gerstlauer (2017) extends previous work (Lee; John; Gerstlauer, 2017) to exploit data dependencies between loop iterations for performance optimizations through operations elimination with V2Z substitutions. That work focuses on eliminating execution cycles as much as possible to improve overall performance while respecting user-defined error bounds. Specifically, loop clustering is performed using a distance metric computed with iteration-wise data statistics, weighted by the estimated quality and error sensitivity of approximation points. Then, a hierarchical clustering algorithm determines the optimal mappings of iterations to clusters. In this process, iterations are reordered whenever possible to maximize approximation benefits.

Rather than exploring loop-based computations directly to achieve performance optimizations, Xu and Schafer (2020) propose to generate faster approximate circuits for ASICs by individually approximating the program operations scheduled in each control step, exploiting the slack between them to increase the performance and reduce the latency of the resultant circuit. Specifically, given a maximum error boundary, multiple control steps are approximated by replacing exact operations with approximate ones from a component library and applying software-level V2C and V2V substitutions. Then, to further optimize the resultant circuit, the final scheduling is generated by merging different control steps based on their timing slack without violating the target frequency. In short, to optimize performance, the authors explore that the code operations assigned to each control step mainly depend on the target synthesis frequency and technology.

Considering specifically the FPGA substrate, Schafer (2017) exploits the bitwidth optimization capabilities of commercial HLS tools to limit the size of some FUs through bitwidth adjustments on output variables of loops. Thus, resource-sharing-based DSE for FPGAs can be enabled to reduce area and delay at the cost of introducing output errors, as the area savings from using smaller FUs outweigh the area needed for multiplexers, which are very costly in terms of LUTs. Although that work does not focus on finding area-error trade-offs nor reaching area constraints, it borrows concepts of AC, as bit-width reduction (precision scaling) is one of the most used AC techniques. Therefore, it can be considered a complementary approach to AHLS design methodologies, further expanding the design space of bit-width reduction techniques.

Approximate computing can also be explored at different levels of abstractions in the HLS flow to expose unique optimization opportunities. In this direction, the work presented by Xu and Schafer (2017) proposes a multilevel approach, i.e., applying different approximation techniques at each abstraction layer to generate Pareto-dominant approximate designs for different input data distributions (random, normal, positive skew, and negative skew), aiming to optimize results. At the software level, source-code pruning using V2C and V2V substitutions is employed based on data statistics obtained through code profiling. Next, at the HLS level, each operation is individually characterized (area, latency, and error) according to a technology library of approximate FUs (adders, subtractors, and multipliers). The resulting designs are evaluated through a cycle-accurate model provided by a commercial HLS tool targeting ASICs and then merged through a greedy algorithm, with a cost function guiding the process of obtaining Pareto-dominant designs. After generating the RTL designs, the tool applies internal signal substitutions (V2C and V2V) and bit-level optimizations. Finally, stability analysis is performed at the gate netlist level to fully observe the result of all the optimizations, providing best and worst-case scenarios. Note that each abstraction layer explores a different set of AC techniques. Therefore, they are applied in a fixed order as the design goes through from the algorithmic representation to the hardware implementation, which may prevent the exploration of better trade-offs.

The design space of approximate circuits within HLS design flows using pre-built approximate FUs as building blocks for circuit optimizations can be very large as many approximate arithmetic circuits with different trade-offs have been reported in the literature (mostly adders and multipliers). Therefore, the problem of selecting those that minimize the required resources for designing and generating an approximate accelerator while satisfying a given accuracy constraint can be very challenging. In this context, Castro-Godínez et al. (2020b) propose a framework for HLS of approximate accelerators using a given library of approximate FU's that comprises a set of analytical models for estimating the required computational resources when using approximate adders and multipliers, called AxME, and a general-purpose DSE methodology, called DSEwam, that uses such analytical models to estimate the resources needed and the accuracy of approximated designs. The authors employ an optimization solver based on tabu search, placed between the HLS resource allocation and scheduling, that evaluates optimizations through a fitness function comprising area, power, and delay design metrics. As a result, Pareto-optimal approximate accelerators are generated for a given error threshold and minimization goal.

Xu and Schafer (2018) propose an AHLS methodology specifically for Coarse-Grain Runtime Reconfigurable Arrays (CGRRA). Applications mapped for these architectures pose unique challenges when approximated, e.g., partitioning an application into a large number of contexts can limit the depth of the approximation to the context itself, while having a smaller number of contexts increases the overall CGRRA area but maximizes the effect of the different approximation optimizations. Therefore, the authors study the effect of different context granularities when mapping an application to be approximated onto a CGRRA, exposing the different trade-offs, and propose a method to create the context sizes of different granularities to minimize the energy consumption under a maximum error constraint while leading to a good balance between other circuit parameters such as area and latency. The effectiveness of the proposed method is explored with software-level V2C and V2V substitutions and a library of approximate operators (adders and multipliers), as defined in previous work (Xu; Schafer, 2017).

Proposing a more general approach relying on machine learning models, Xu and Schafer (2019a) present a method to selectively extract portions of a high-level behavioral description (ANSI C or SystemC) to be synthesized as a hardware accelerator using HLS onto different predictive models (PMs) to trade off the accuracy of the accelerators' outputs with area and power. Because the main aim of that work is to synthesize the newly approximated behavioral description, the authors investigate the use of different predictive models, particularly Linear Regression (LR) and Multi-Layer Perceptron (MLP), highlighting the trade-offs of using one over the other. In addition, the search space is further extended by reducing the precision (bit-width) of the predictive models' coefficients, thus leading to a wider range of solutions. The source code transformed with such predictive models is validated for a given error threshold through HLS simulations. Going further, Xu and Schafer (2019b) propose substituting the CGRRA contexts in their previous work (XU; SCHAFER, 2018) with approximate expressions using the LR and MLP machine learning models to reduce the CGRRA area and energy at the cost of introducing different levels of output errors. Moreover, the authors present a technique to merge these approximated contexts with exact ones, leading to a set of Pareto-optimal configurations.

Chowdhury and Schafer (2021) present a method to unlock approximations that would have been ignored with traditional approaches. Notably, the authors focus on V2C and V2V approximations at the source code (e.g., ANSI C) for either HLS or embedded software. The main idea is to use V2C and V2V to substitute the computation of a variable by either another variable computed previously in the code or a constant, respectively, to trade off circuit size or code size with output error. An automatic source code refactoring method is proposed to enable such approximations, combined with a selective substitution of portions of the code with predictive models. An automated search method is also used to find the smallest possible predictive model for a given error threshold, aiming to minimize code size, area, or power.

A major challenge for adopting approximate computing is that the resulting approximated circuit is highly dependent on the training data. If the workload is dynamic or changes over time, output errors may reach unacceptable levels. Therefore, dynamic control methods are needed to solve this problem. To address this challenge, Xu and Schafer (2019c) propose an approximate self-adaptive architecture for HLS-generated designs that tunes itself at runtime based on the workload. Two control mechanisms are proposed, one based on a regular heartbeat (HB), which resets the approximate circuits at regular intervals, and the other based on internal lightweight checkers (LWCs) to detect a change in the workload before resetting approximations. The proposed framework is divided into two phases. The first phase analyzes the input code (ANSI C) to extract a set of approximation points amenable to self-tuning, where V2V and V2V substitutions are considered for approximations. The second phase automatically inserts the self-tunable circuitry at those points to enable or disable these approximations. The goal is to contain the output error within a given threshold for dynamic workloads and minimize the total energy consumption, including the energy due to the controller logic.

Many AHLS methodologies rely, at least partially, on pre-built technology libraries of approximate components to explore the design space. Thus, synthesizing such libraries is a crucial requirement for the broad adoption of AHLS design flows using this approach, as the operations' characterization regarding the introduced error and design metrics such as area, delay, and power are design variables considered when exploring approximation options, i.e., many compromise implementations should exist for each component. Focusing on this problem for ASIC-oriented AHLS, Vaverka, Hrbacek and Sekanina (2016) propose a Cartesian Genetic Programming (CGP) algorithm to generate a comprehensive component library containing hundreds of Pareto optimal implementations of approximate adders and multipliers, where the error, area, and delay are simultaneously optimized. The authors also investigate the impact of such libraries on the quality of approximate circuits produced by HLS by employing another multi-objective evolutionary algorithm to solve the problem of binding suitable approximate components to the DFG nodes describing the circuit. Nevertheless, except for very few works (LIU et al., 2016; YIN et al., 2016; SAADAT; JAVAID; PARAMESWARAN, 2019), approximate arithmetic studies have been focused on integer arithmetic disregarding fixed- and floating-point operators. Therefore, building general-purpose robust AHLS tools relying only on technology libraries requires the development of extensive libraries of approximate components, which may limit their application scope.

3.3 Open Challenges Addressed by this Thesis

As discussed, AHLS design methodologies raise the exploration of AC techniques to higher abstraction layers (software - SW, HLS, and behavioral HDL) to expand the de-

Table 3.1 – Comparison between AHLS works found in the literature and this thesis

Reference	Abstraction Layer	AC Technique	Constraint	Goal	Target
Nepal et al. (2014)	HDL	PS OT AET V2C LP	Error	Pareto-optimal trade-offs (error and power)	Any
Awais, Mohammadi and Platzner (2018)	HDL	PS OT LP	Error	Pareto-optimal trade-offs (error, area, and power)	Any
Nepal et al. (2019)	HDL	PS OT AET V2C LP	Error	Pareto-optimal trade-offs (error and power)	ASIC
Li et al. (2015)	HLS	TechLib	Error	Minimize energy	Any
Lee, John and Gerstlauer (2017)	SW HLS	TechLib V2Z	Error	Minimize energy	Any
Lee and Gerstlauer (2017)	SW	V2Z	Error	Maximize performance	Any
Xu and Schafer (2020)	SW HLS	TechLib V2C V2V PM	Error	Maximize performance	ASIC
Schafer (2017)	SW	BWR	No	Pareto-optimal trade-offs (error, area, and latency)	FPGA
Xu and Schafer (2017)	SW HLS RTL	TechLib V2C V2V BWR	No	Pareto-optimal trade-offs (error, area, power, and latency)	Any
Castro-Godínez et al. (2020b)	HLS	TechLib	Error	Minimize area, power, or latency	Any
Xu and Schafer (2018)	SW HLS	TechLib V2C V2V	Error	Minimize energy	CGRRA
Xu and Schafer (2019a)	SW	PM	Error	Pareto-optimal trade-offs (error, area, and power)	Any
Xu and Schafer (2019b)	SW	PM	Error	Pareto-optimal trade-offs (error, area, latency, and energy)	CGRRA
Chowdhury and Schafer (2021)	SW	V2C V2V	Error	Minimize area or power	Any
Xu and Schafer (2019c)	SW	V2C V2V	Error	Minimize energy	ASIC
This Thesis	SW	V2M V2Z V2P BWR LP	Design Metrics	Minimize error	FPGA

Source: The author

sign space and address the complexity of exploring approximations at RTL and gate-level circuit representations, either manually or via ALS design flows, leveraging the tight connection between the design's algorithmic description, typically in C/C++ or SystemC, and the HLS tasks involved in generating the RTL code. However, despite the significant advances in automating the design of approximate hardware accelerators, several hurdles still have to be faced for AHLS design methodologies to become mainstream in approximate hardware design.

Table 3.1 summarizes the main characteristics of current AHLS design methodologies proposed in the literature and the modifications addressed by this thesis. As can be observed, some proposals entirely rely on pre-characterized technology libraries (TechLib) of approximate arithmetic operators (adders and multipliers) (Li et al., 2015; CASTRO-GODíNEZ et al., 2020b). Such works explore the design space of approximate hardware accelerators by evaluating the composition of multiple such operators. However, by picking the proper building blocks among the ones available in a library, such strategies propose designing approximate hardware as a selection problem, limiting solutions to the type and bit-width of the available approximate arithmetic components. Moreover, selecting appropriate libraries requires previous knowledge of the design's functionality or semantics, narrowing the design space and limiting the application scope. On the other hand, works that do not restrict approximation options to pre-built libraries do so by employing a limited set of AC techniques implemented either at the HDL layer, i.e., the set of AC techniques explored is difficult to extend due to the complexity of implementing approximation strategies at this level of abstraction (Nepal et al., 2014; AWAIS; MOHAMMADI; PLATZNER, 2018; Nepal et al., 2019), or at higher abstraction layers but in a fixed order tightly coupled to a specific implementation flow. As a result, such works hamper the possibility of integrating new AC techniques to enable other optimization opportunities arising from exploring a combination of diverse techniques, and thus relevant solutions may be missed due to the limited design space.

Another drawback of most current AHLS frameworks is that they propose error-constrained approximation methodologies to minimize or maximize specific design metrics by either using design parameters not typically available to designers of applications for reconfigurable platforms (e.g., the internal multiplier architecture) and thus are not directly applicable to standard FPGA design flows, or are not optimized to the proportional costs of operations in the FPGA fabric (Li et al., 2015; Lee; John; Gerstlauer, 2017; Lee; Gerstlauer, 2017; XU; SCHAFER, 2020; CASTRO-GODíNEZ et al., 2020b; XU; SCHAFER, 2018; CHOWDHURY; SCHAFER, 2021; XU; SCHAFER, 2019c). Unlike ASICs, FPGAs require managing a predefined number of heterogeneous resources, either for more efficient implementations or for fitting multiple components in a single device. Therefore, by disregarding such device-specific optimization opportunities, error-constrained design methodologies may impose the need for multiple attempts to find the target error that meets the design metrics of interest, jeopardizing productivity. Moreover, suppose multiple design metrics must be met. In that case, the quality of results may be significantly impacted, as error-constrained approaches are not able to dynamically change the optimization focus to the design metrics that still have not been met or are far from being met. Therefore, besides being counter-intuitive from a design perspective, steering the DSE with error thresholds may produce largely sub-optimal designs as the suitability of each available AC technique towards meeting specific design objectives is not exploited directly.

On the other hand, methods that provide compromise circuit implementations along a Pareto front, showing different trade-offs between circuit parameters (delay, area, and power consumption) and error by leveraging either unconstrained (Schafer, 2017; Xu; Schafer, 2017) or error-bounded (Nepal et al., 2014; AWAIS; MOHAMMADI; PLATZNER, 2018; Nepal et al., 2019; XU; SCHAFER, 2019a; XU; SCHAFER, 2019b) optimization algorithms make designers responsible for choosing the solution that best fits their needs a posteriori. In those cases, although designers are relieved from specifying tight error bounds for each target application, better solutions may be lost during the optimization process for that same reason, i.e., the DSE is not directly steered by the designer's specific objectives, usually defined in advance. Therefore, an AHLS design methodology that leverages the architecture of FPGAs to optimize approximation choices toward minimizing errors while exploring a combination of diverse AC techniques according to constraints on the design metrics of interest rather than error bounds would be a promising option to manage the design effort required for adopting AHLS for such devices. Note that, although Schafer (2017) targets FPGAs, the specific goal is to explore software-level bit-width adjustments to reduce the FUs size to the point where the area savings overweights the cost of using multiplexers for resource sharing, i.e., the author does not provide a general approach to exploring optimal approximation options according to target-specific features. Similarly, works targetting CGRRAs (XU; SCHAFER, 2018; XU; SCHAFER, 2019b) focus on studying the effect of using different context granularities when mapping approximated applications to expose different trade-offs. Therefore, the architectural features of such devices are not directly explored to guide approximation choices.

Another important observation is that, although other AHLS works implement multiple AC techniques to expand the design space, such techniques are used to optimize specific design metrics through tightly designed methodologies focusing on error thresholds, overlooking the unique trade-offs offered by different techniques according to the target optimizations. Therefore, automatically exploring a proper combination of diverse techniques to minimize errors is a task that should be directly steered by the designer's specific objectives through a flexible constraint-aware design methodology. This enables optimization heuristics to dynamically adapt the selection of a wide range of approximation options as the design evolves to meet the specified design objectives. In such a direction, it is also important to offer a flexible method to allow adding new or different

AC techniques to expand or adapt the design space according to the target application and optimizations. As can be observed in Table 3.1, we have implemented five software-level AC techniques that offer different trade-offs: Variable-to-Mean (V2M), Variable-to-Zero (V2Z), Variable-to-Power (V2P), Bit-Width Reduction (BWR), and Loop Perforation (LP), which are detailed in Section 4.2. Nevertheless, this set of techniques can be seamlessly extended using the methodology described in that section.

In face of those gaps in the AHLS literature, this thesis addresses the challenges of defining a constraint-aware AHLS design methodology for FPGAs to automatically and adaptively identify application- and constraint-specific compositions of multiple AC transformations from an easy-to-extend set of software-level AC techniques toward meeting multiple design constraints with minimum error. As a result, the integration of previous efforts for efficient AC and AHLS can be achieved without imposing an additional burden on developers while paving the way for future integration of additional techniques that may benefit a wider range of application scenarios. Supporting the value of such a methodology, which is detailed in Chapter 4, experimental results in Chapters 6 and 7 show that the optimal AC transformations greatly depend on the application considered and the target optimizations and that combining multiple AC techniques outperforms any single technique for all scenarios.

4 DESIGN METHODOLOGY

This chapter presents the design methodology proposed to deal with the AHLS challenges addressed by this thesis and discussed in Section 3.3. Specifically, Section 4.1 provides a general overview of the proposed constraint-aware multi-technique optimization heuristic. Next, Sections 4.2, 4.3, and 4.4 detail respectively: which AC techniques were implemented and how new techniques can be added; how optimizations are evaluated in terms of the considered design metrics (resources and WCET); and the GRASP-based heuristic employed to explore the design space of approximated designs.

4.1 Methodology Overview

The methodology overview is divided in two sections. First, the optimization problem faced by this thesis is defined in Section 4.1.1. Next, Section 4.1.2 overviews the optimization heuristic employed to deal with the design space of a constraint-aware multi-technique approach, which will be presented in greater depth in Section 4.4.

4.1.1 Optimization Problem

This thesis proposes that a constraint-aware HLS-based design methodology for FPGAs that automatically explores multiple AC techniques to yield error-minimized designs meeting multiple constraints can overcome some of the drawbacks of existing AHLS approaches. Without loss of generality, the optimization problem involved in such a proposal will be defined in terms of common FPGA resources (LUTs, REGs, and DSPs) and real-time constraints, which are the design metrics considered for presenting the experimental results detailed in Chapters 6 and 7. More precisely, given a design $\mathcal{D}$ described in a high-level language and a set of AC techniques $\mathcal{A}$, we define the set of AC transformations $\Gamma = \mathcal{A} \times \mathcal{D}$ as the set of operators $\gamma = (a, d)$ representing the application of an AC technique $a \in \mathcal{A}$ on a design entity $d \in \mathcal{D}$, which can be a single operation (arithmetic, logic, or memory), a basic block, a loop, or a function, depending on the granularity with which a can be applied. It follows that an approximate design s can be defined by a sequence of AC transformations over the exact design $\mathcal{D}$, i.e.,

$$s = (\gamma_k)_{k=1}^{n} : \gamma_k \in \Gamma \text{ and } 1 \leq n \leq |\Gamma|. \tag{4.1}$$

Thus, given the set S of all possible functional[1] approximate designs resulting from applying a sequence of AC transformation over $\mathcal{D}$, as defined by Equation (4.1), we define $\mathcal{F} \subseteq S$ as the set of feasible designs such that, $\forall s \in \mathcal{F}$, s satisfies

$$\forall t \in \mathcal{T} : \mathcal{R}_t(s) \leq C_{\mathcal{R}_t} \tag{4.2}$$

$$W(s) \leq C_W \tag{4.3}$$

where Equations (4.2) and (4.3) define resources and real-time constraints, respectively. In Equation (4.2), $\mathcal{T}$ is the set of all constrained resource types in the target FPGA (e.g., LUTs, REGs, and DSPs), $\mathcal{R}_t(s)$ is the number of resources of type t required by the approximated design s, and $C_{\mathcal{R}_t}$ is the respective resource constraint. Similarly, in Equation (4.3), $W(s)$ is the WCET of approximate design s and C_W is the real-time constraint. Thus, designers can specify a usage limit $C_{\mathcal{R}_t}$ for each resource type and a limit C_W for execution time. In short, a design is considered feasible if it is functional and meets all specified constraints.

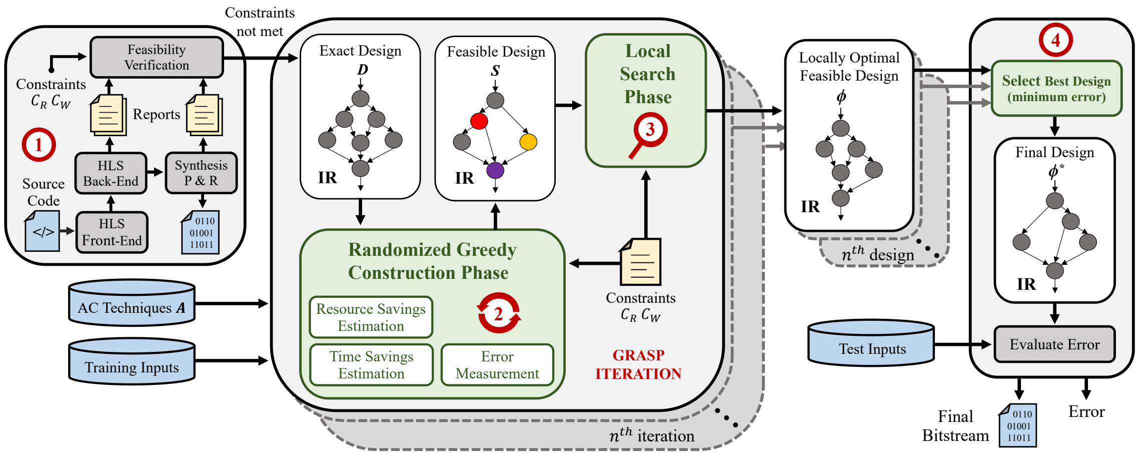
Therefore, given an error function $E : S \rightarrow \Re+$, the optimization problem involved in our methodology is finding a feasible approximate design s that minimizes E, i.e.,

$$\text{minimize } E(s) \text{ s.t. } s \in \mathcal{F} \tag{4.4}$$

where the error function in Equation (4.4) is user-provided and can be defined to calculate any Error Metric (EM) developers see fit for the application at hand. As the design space available for multiple AC techniques, arbitrarily applied to complex designs, cannot be exhaustively explored in a timely manner due to combinatorial explosion, an efficient heuristic for this purpose must be developed.

[1]Regardless of error measures, a design is considered functional if it can be compiled by the HLS backend and does not hang or crash when executed.

Figure 4.1 – GRASP-based design flow to meet multiple constraints with minimum error



Source: The author

4.1.2 Design Space Exploration Overview

The heuristic devised to explore the design space is based on GRASP, a widely used metaheuristic for combinatorial optimization problems (FEO; RESENDE, 1995; RESENDE; RIBEIRO, 2019), adapted to address the multi-constrained optimization problem described in Section 4.1.1. GRASP is a multi-start iterative process in which each iteration consists of a *Randomized Greedy Construction Phase* (RGC) and a *Local Search Phase* (LS). For our purposes, the construction phase must build a feasible design satisfying Equations (4.2) and (4.3). Its neighborhood is then explored in the local search phase to find a locally optimal feasible design, i.e., a feasible design with minimum error. According to the provided EM, the best solution (less error) over all GRASP iterations is kept as the final design. Figure 4.1 shows an overview of the GRASP-based design methodology. The main inputs are the design specification in a high-level language, such as C/C++, a set of AC techniques $\mathcal{A}$, a set of representative inputs for error measurement, divided into training inputs and test inputs, and the resources and real-time constraints $C_{\mathcal{R}}$ and C_W respectively. The outputs are the approximated bitstream, ready to be downloaded to the target FPGA through vendor-specific tools, and the error measure according to the chosen EM.

As can be observed, the design flow is divided into four major steps. In **step 1**, the HLS front-end is used to generate the design's Intermediate Representation (IR), $\mathcal{D}_{IR}$, which is a high-level assembly-like code suitable to be optimized by transformations provided by the HLS compiler toolchain, including the application of software-level AC techniques. Next, the IR code is fully synthesized to the target FPGA to obtain the synthesis reports used to verify feasibility, i.e., whether the exact design already meets all constraints. If it does not, it is forwarded to **step 2**, where the GRASP construction phase iteratively builds a feasible design s by applying a sequence of approximation transformations over the exact IR code $\mathcal{D}_{IR}$, one at a time, until the resources and real-time constraints are met. At each construction iteration, a *Restricted Candidate List* (RCL) is built with Pareto-optimal designs resulting from applying each transformation $\gamma \in \Gamma$ on the design selected at the previous RGC iteration (the exact one if it is the first iteration). Next, one Pareto-optimal design, considering the introduced error and a measure estimating the distance to meeting all constraints, is randomly selected from the RCL. The selection probability for each design in the RCL is defined by an adaptive fitness function measuring the cost-benefit of each approximated design. Sections 4.3 and 4.4.1 detail respectively how design metrics are evaluated to build the RCL and how the construction phase is implemented to iteratively select approximate design candidates.

Given the feasible approximate design s built in the construction phase, in **step 3** the GRASP local search phase tries to find a better solution in its neighborhood $\mathcal{N}_s$, i.e., a feasible design with lower error than s. In our design flow, all Pareto-optimal designs derived from the same parent design after applying a single approximation step are considered neighbors. A neighbor of s is, thus, any design in the RCL from which s was chosen in the last iteration of the construction phase. To search for a local optima, we check the feasibility of each neighbor $\phi \in \mathcal{N}_s$ such that $E(\phi) < E(s)$. If ϕ is feasible, it replaces the initial solution s. Additionally, the neighbors' feasibility is evaluated in ascending order of error. Therefore, the local search stops when the first feasible solution is found. More details regarding the LS phase implementation can be found in Section 4.4.2.

Starting from the exact design, the GRASP iteration comprising steps 2 and 3 is repeated until a stop criterion is reached, such as a fixed number of iterations or a given running time. The idea behind performing multiple GRASP iterations is to broaden the design space by selecting the optimal design among all locally optimal feasible designs generated. In this case, each GRASP iteration uses a different seed to perform the biased random selection from the RCL at each iteration of the construction phase. Moreover, the multiple GRASP iterations can be executed in parallel using multi-core and cluster architectures to shorten development time, as is done in our experimental setup. Specifically, designers can provide the number of GRASP iterations to be processed, and one thread is assigned for each of them. Finally, in **step 4**, the feasible approximate designs generated after multiple GRASP iterations are compared, and the best one with minimum error is kept as the final design ϕ^*. The bitstream for the target FPGA, provided by the synthesis tool, is then delivered along with the final error measure.

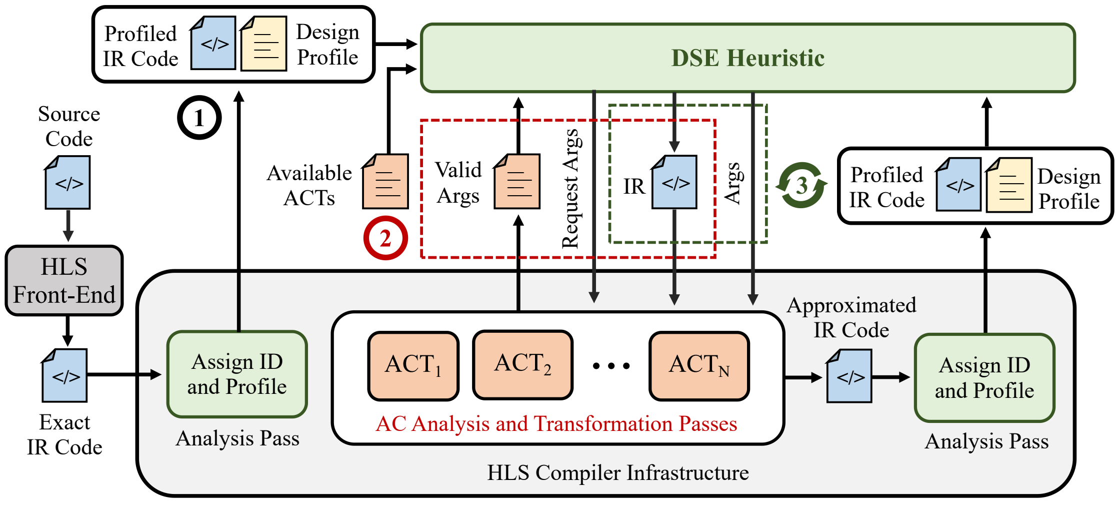
Regardless of the heuristic employed to explore the design space, three important challenges must be faced: (i) how to provide a multi-technique approach to explore the unique optimization opportunities offered by the numerous AC techniques proposed in the literature by allowing a seamless integration of new techniques in the design flow, (ii) how to obtain a reasonable estimation of the impact of each approximation option on the design metrics of interest without relying on fully synthesizing each approximate circuit candidate, which is impractical even for moderately-sized designs due the vast design space that arises from exploring multiple techniques, and (iii) how to offer scalability for error measurement, as characterizing the error of an approximate circuit candidate for the whole input range is difficult and time-consuming in general.

In the proposed GRASP-based heuristic, AC techniques are applied from an extensible library of compiler-based IR code transformations, which are detailed in Section 4.2. As design metrics estimation and error measurement play a central role in building the RCL and selecting the approximate circuit candidate within each iteration of the construction phase, efficient methods to perform such estimations are needed. Details on how resources and time savings are estimated can be found in Sections 4.3.1 and 4.3.2 respectively, while the error measurement approach is described in Section 4.3.3.

4.2 Approximate Computing Techniques

A flexible AHLS design methodology aimed at exploring diverse optimization opportunities through exploiting a combination of AC techniques should provide a systematic way of integrating new techniques into the standard design flow. To offer such flexibility, we propose implementing software-based AC techniques as a library of transformation passes within the compiler infrastructure of HLS design flows, decoupling the set of exploited techniques from the design heuristic employed to explore the design space. Figure 4.2 illustrates such an approach to integrate new techniques. Before exploiting any AC transformation pass, in **step 1** the design entities in the target IR code are identified, which can be the exact one or other already approximated if using an iterative process. Specifically, we developed an analysis pass to assign a unique identifier (ID) as metadata to each IR entity, including operations, basic blocks, loops, and functions, and profile its attributes, such as the operations type (arithmetic, logic, memory, or con-

Figure 4.2 – Software-level AC techniques implemented as code transformations



Source: The author

trol) and bit-width, the operations sources and destinations, the operands type (variable or constant), the basic blocks' operations, and the loops' trip counts. Additionally, this analysis pass then exports the identified entities and their attributes to a file in the YAML format (YAML, 2022). Note that the only modification made by this step is including metadata information in the provided IR code.

Next, in **step 2**, the DSE heuristic can explore approximation options by selecting the entities to approximate from the IR code profile and using the library of AC techniques (ACTs) made available through a file containing the list of callable transformations passes with their definitions (names and parameters). To use an approximation pass, the heuristic must know the valid arguments for the target IR, as different techniques may apply to different entities (e.g., loops or arithmetic operations) and to different extents, such as the perforation ratio for loop perforation or the constant value for variable-to-constant substitution. Therefore, the AC technique implementation must provide an additional pass that receives the profiled IR code with the metadata IDs previously added for each entity to generate a list of those entities that can be approximated by the implemented transformation, along with their respective valid arguments. This list must be saved as a file in the YAML format with the pass name and key-value pairs corresponding to the entity's ID and the list of valid arguments. Then, to use one of the provided AC transformations, the heuristic first requests the list of valid entities and arguments pairs by calling the technique-specific analysis pass for the target IR code. Finally, in **step 3**, the heuristic can exploit the techniques available by calling the respective compiler optimization pass for the IR code to be transformed along with the entity's ID and one of the valid arguments provided. Note that such an approach offers great flexibility for designers, as it allows the inclusion of AC techniques applicable only to specific entities and to limited extents without the need to modify or replace the heuristic. For example, one may provide an AC transformation that exposes only resilient operations for approximations to reduce the design space based on a previous analysis of the IR code.

Although Figure 4.2 exemplifies an iterative process for DSE, multiple AC transformations can be applied in sequence before updating and profiling the approximated IR code. Therefore, given the updated and profiled IR code before and after one or more AC transformations, the DSE heuristic can estimate the optimizations achieved regarding the design metrics of interest using appropriate methodologies. For our experiments, approximation techniques were chosen to cover a range of distinct trade-offs in terms of introduced error and gains in resource usage and execution time. Nevertheless, this set of AC techniques can be easily extended by developers by following the approach herein defined, especially to include new AC transformations that are suitable for the targeted application domains. The techniques detailed in the next subsections were implemented as transformation passes within the LLVM compiler framework.

4.2.1 Variable-to-Constant Substitutions

As discussed in Section 2.3.2, variable-to-constant (V2C) substitutions allow designers to achieve area, power, and performance improvements by replacing costly operations with constant values. This technique was implemented through a transformation pass that, given the IR code, the operation ID, and the substitute value as a parameter, transforms the IR code by replacing the given operation with the given constant value. For this thesis, we have implemented three different types of V2C substitution: variable-to-mean (V2M), variable-to-zero (V2Z), and variable-to-power (V2P).

The V2M approximation option replaces an operation's output with its mean value, previously obtained with code instrumentation and simulation with a set of user-provided training inputs. The code instrumentation is done with another LLVM transformation pass that inserts a function call after each operation in the IR code to get the value computed after its execution. This function uses the unique identifiers saved as metadata to differentiate operations and account for multiple executions, saving the computed values in a list. Before returning from the main function, the instrumented IR code saves the computed statistics for each operation to a file in the YAML format with key-value pairs, i.e., the operation ID and its mean value, which can be used later by the DSE heuristic to feed the V2C transformation pass with valid arguments for the target operation. Note that the mean value can minimize the error introduced by approximations using a fixed value regardless of the operation, assuming the training inputs are representative of the inputs found during the system's actual operation. This technique is especially suitable when the standard deviation is relatively small, providing excellent trade-offs and adhering to the AC principle of disproportionate benefits. However, since the output can be any value, V2M is unlikely to completely eliminate other operations (such as additions or multiplications) that depend on the approximated value. The V2Z option, on the other hand, is a more aggressive optimization strategy that consists in replacing an operation's output with the constant zero. In this case, the only valid argument for substituting operations is a constant zero. Although this may introduce more error than V2M, the constant zero can propagate throughout the circuit, completely eliminating other arithmetic operations and thus enabling further optimizations, as exemplified in Section 2.3.2.

The V2P approximation option aims to be an intermediate strategy when compared to V2Z and V2M, and it is a novel strategy used in this thesis. With V2P, the output of an operation is replaced by a constant value that is the nearest power of two to its mean value. Consequently, multiplications using this value can be replaced by constant shifts, and adders are also likely to be simplified depending on the number of trailing zeros in the value's binary representation. Therefore, V2P can provide even better trade-offs than V2M if the power-of-two value is within the operation standard deviation. This technique uses the same analysis pass developed to generate the application's statistical profile for V2M substitutions, except that the valid arguments provided for the V2C transformation pass are computed as the nearest power of two to the operations' mean values.

4.2.2 Bit-width Reduction

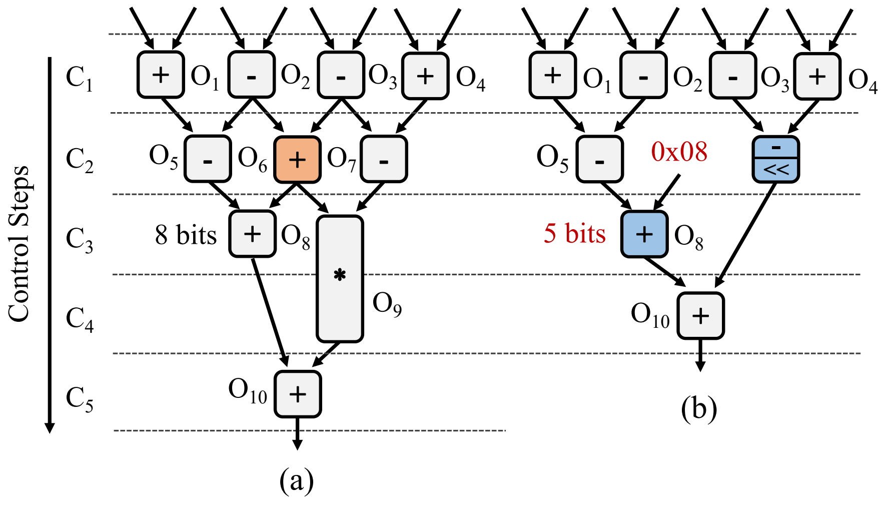
Bit-width reduction (BWR) consists in reducing the data width of variables, below the minimum amount of bits specified in the source code or which would be necessary to represent all possible values during the system's regular operation. It does not completely eliminate operations but allows some variability according to the current input values, providing a wider design space with less error. We consider the entire design space associated with BWR, i.e., from removing one single bit up to removing all but the most significant bit. Specifically, we provide an approximation pass that receives as inputs the IR code, the ID of the operation to be approximated, and the number of bits to be reduced as an argument, starting from the least significant. Similarly to the V2C substitution passes, an analysis pass was developed to generate a file in the YAML format containing the transformation pass name and a list of valid arguments for each operation eligible to be approximated by this technique. For example, the ID associated with an 8-bit adder will have a list of valid arguments containing integer values ranging from one to seven, i.e., the bitwidth of this operation can be reduce from one to seven bits.

This technique is an important alternative to more coarse-grained operation-level approximations, such as V2C substitutions, especially for yielding approximate hardware, where the data path bit-width is fully customizable. Indeed, many variables declared at the high-level source code use more bits than necessary, so state-of-the-art HLS tools perform bit-width optimizations before generating the RTL code. Therefore, applying BWR to approximate a single operation may propagate further bit-width optimizations to other operations, extending its benefits.

4.2.3 Loop Perforation

Loop perforation (LP) skips entire iterations of loops, and the frequency of skipped iterations allows exploring the trade-offs in terms of error and gains. We consider the range of skipping one every two iterations up to one every 100 iterations, i.e., the implemented transformation pass provide the valid arguments as a list of integers ranging from 2 to 100. Then, to be used, it must receives as inputs the IR code, the loop ID, and the perforation ratio between 2 and 100 as an argument. It should be noted that LP operates at a different granularity when compared to the other considered AC techniques. During the heuristic search, operations that are the first in a loop body are used as handlers for the loop, while other operations are ignored by this technique. Similar approaches may be employed when integrating other techniques that operate on different granularities. Moreover, as detailed in Section 4.3.2, we restrict LP to loops in the time-critical execution path provided by WCET analysis, which reduces the searching space to the most promising candidates.

Figure 4.3 – HLS optimizations on area and delay (a) before and (b) after a V2P substitution



Source: The author

4.2.4 Exploring Further Optimizations

Besides exploring the AC techniques described above, we also explore traditional compiler optimizations that may have been enabled by each AC transformation, such as dead code elimination, constant propagation, and strength reduction. When an operation is approximated, other operations may be affected due to data or control dependencies. Consequently, further resource and time savings may be achieved as a result of a single approximation step without introducing additional errors. Therefore, such optimizations must be considered when evaluating the impact of each AC transformation on design metrics. More coarse-grained AC techniques, such as V2Z and V2P, will likely enable more aggressive compiler optimizations at the cost of increased error. Moreover, each technique produces different trade-offs for different design parameters. Although some techniques may impact resources and execution time by eliminating operations and control steps (e.g., V2Z, V2M, and V2P), others were developed to optimize specific design metrics. LP, for example, is more suitable for reducing execution times, while BWR is a fine-grained approach primarily used to reduce the data path size.

Therefore, selecting an optimal sequence of AC transformations to meet multiple constraints while minimizing output errors for different scenarios requires carefully examining the effects of each approximation step throughout the code. Figure 4.3 exemplifies one case where V2P yields further savings for FPGAs. It shows a data flow graph with ten operations scheduled across five control steps (ASAP), assuming that adders and subtractors have a latency of one clock cycle while multipliers need two clock cycles. Registers between control steps are omitted for the sake of clarity. The adder highlighted in Figure 4.3(a) is replaced by a power-of-two constant with three zeros in the least significant bits (0x08). Thus, in Figure 4.3(b), the following adder, O_8, can be reduced from eight to five bits, saving three LUTs. Additionally, the multiplier O_9 in Figure 4.3(a) can be replaced by a constant shift, which is implemented by appropriately connecting the subtractor's input without consuming any logic resources. Assuming multipliers are mapped to DSPs, one or more DSP blocks may be saved depending on the multiplier bit-width and the device's specific DSP block architecture. Moreover, the two registers needed to use the outputs of O_6 and O_9 in control steps C_3 and C_5 respectively can be eliminated. Thus, besides reducing the circuit delay in one clock cycle, eliminating O_6 results in further resource savings enabled by optimizations without additional error. The AC transformations implemented for this thesis leverage the LLVM's standard optimization passes to account for such additional optimizations before updating the design entities' identifiers and generating the approximated IR code profile.

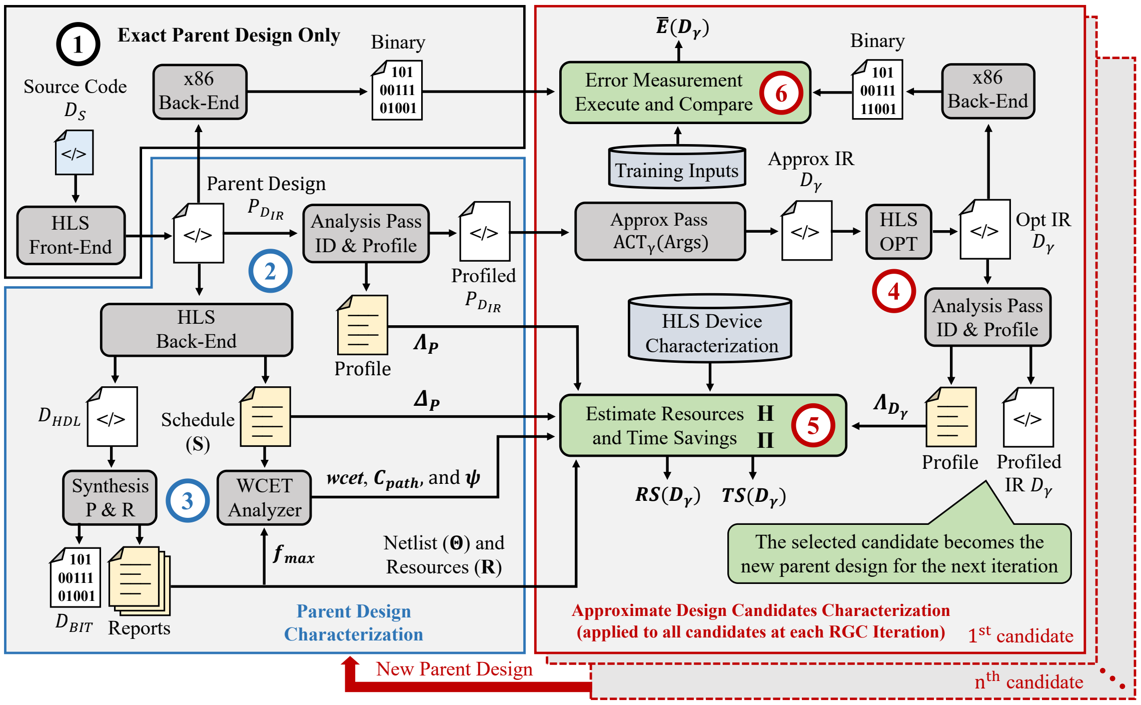
4.3 Design Metrics Evaluation

The results achieved by the proposed methodology highly depend on the quality of the locally optimal feasible designs generated by the Randomized Greedy Construction (RGC) phase of each GRASP iteration. The RGC phase iteratively applies approximations to designs, producing new approximate designs from a parent design, which is initially the exact implementation. Therefore, it is crucial to adequately estimate the optimizations achieved with each transformation applied at the RGC phase without excessively extending running times or producing rough estimates that may degenerate results. Figure 4.4 shows the sequence of steps and tools used to estimated resource savings, time savings, and error, as will be detailed in the next subsections.

4.3.1 Resource Savings

The resource savings achieved with each approximation step in the RGC phase can be precisely measured by synthesizing the circuit down to the basic components of the target FPGA. However, this method is time-consuming, being impractical for most cases. Alternatively, one can estimate the resource use with the device characterization

Figure 4.4 – Sequence of steps to measure error and evaluate resources and time savings



Source: The author

file provided by the HLS tool and used to guide the synthesis process. The LegUp tool, for example, is implemented over the LLVM framework and includes a pre-characterization of required resources for every operation in the LLVM's IR considering the target FPGA. Although this approach can offer a reasonable estimation of the resource usage of each operation, it does not account for target-specific optimizations performed by synthesis tools. Depending on how operations were scheduled, registers may be unnecessary due to operation chaining. Afterward, resources may be eliminated, duplicated, or merged as a result of netlist optimizations. Therefore, an analytical model based only on the HLS pre-characterization may produce significant over-estimations (HSIAO; ANDERSON, 2018).

For this thesis, such limitations are circumvented by using both the HLS pre-characterization and the full synthesis outcomes to estimate resource savings. As can be observed in Figure 4.4, **step 1** is generating the exact parent's design IR code $\mathcal{P}_{\mathcal{D}_{IR}}$ with the HLS front-end and the exact binary code with the x86 back-end, which is used later for error measurement (details in Section 4.3.3). The parent design (the exact one in the first RGC iteration or an already approximated from the previous iteration) is then characterized as follows: in **step 2**, the scheduling report $\mathcal{S}$ and the HDL code $\mathcal{D}_{HDL}$ are generated by the HLS back-end, and the IR design entities are identified and profiled through the analysis pass discussed in Section 4.2. Next, in **step 3**, the circuit's netlist

Θ and the exact resources usage report $\mathcal{R}$ are produced by fully synthesizing (placement and routing) the HDL code with the device-specific synthesis tool. With the parent design characterization at hand (profile, schedule, netlist, and resources), in **step 4**, an optimized IR code $\mathcal{D}_\gamma$ and its profile are generated by applying each AC transformation $\gamma \in \Gamma$, i.e., by exploiting the available AC techniques for each design entity, followed by standard compiler optimizations, as described in Section 4.2.

Following to **step 5**, the resulting profiles before and after approximations are parsed and compared to identify the IR entities that were eliminated, added, or transformed, allowing the estimation of resource savings for each approximate design candidate. Note that at this point the time savings are estimated as well, which will be detailed in Section 4.3.2. To accomplish the resource savings estimation task, the analysis pass also includes the design entities' IDs in their naming schema as a prefix, such that the set of resources associated with each IR entity in the parent's design can be identified in the circuit's netlist for mapping the IR-level operations to the design entities that are actually synthesized for the target FPGA. As will be detailed next, by parsing the design reports, resources that were eliminated, duplicated, or merged during the synthesis process are also tracked to avoid significant under- or over-estimations. Note that operations may be present only in the approximated design profile, i.e., transformed or added operations that cannot be directly mapped to the parent's design netlist. The resource usage estimation is made using the HLS pre-characterization for these cases, with the control steps of operations not affected, taken from the parent's design scheduling, being used to estimate the insertion of registers according to the sources and destinations of the new operations. Therefore, we can obtain a good estimation of the total resource savings without performing a full synthesis for each approximate design candidate. Finally, in **step 6**, the error measurement approach detailed in Section 4.3.3 is performed, ending the approximate design candidate characterization.

Formally, let $\mathcal{O}_\mathcal{P}$ and $\mathcal{O}_{\mathcal{D}_\gamma}$ denote respectively the set of IR operations in the parent's design $\mathcal{P}_{\mathcal{D}_{IR}}$ and in the optimized design $\mathcal{D}_\gamma$ resulting from approximating $\mathcal{P}_{\mathcal{D}_{IR}}$ with an AC transformation $\gamma \in \Gamma$. It follows that $\mathcal{E}_\gamma = \mathcal{O}_\mathcal{P} \setminus \mathcal{O}_{\mathcal{D}_\gamma}$ is the set of eliminated operations and $\mathcal{W}_\gamma = \mathcal{O}_{\mathcal{D}_\gamma} \setminus \mathcal{O}_\mathcal{P}$ is the set of new operations. Additionally, let $\Lambda_\mathcal{P} : \mathcal{O}_\mathcal{P} \rightarrow \mathcal{V}_\mathcal{P}$ and $\Lambda_{\mathcal{D}_\gamma} : \mathcal{O}_{\mathcal{D}_\gamma} \rightarrow \mathcal{V}_{\mathcal{D}_\gamma}$ be functions mapping the set of operations in $\mathcal{P}_{\mathcal{D}_{IR}}$ and $\mathcal{D}_\gamma$ respectively to their sets of attributes. Then, $\mathcal{X}_\gamma = \{x : x \in \mathcal{O}_\mathcal{P} \cap \mathcal{O}_{\mathcal{D}_\gamma} \text{ and } \Lambda_\mathcal{P}(x) \neq \Lambda_{\mathcal{D}_\gamma}(x)\}$ is the set of transformed IR operations, i.e., operations whose attributes changed after approximating $\mathcal{P}_{\mathcal{D}_{IR}}$ with AC transformation γ, such as the bit-width or the operands type (variable to constant, for example). Therefore, the savings for resource type $t \in \mathcal{T}$ obtained with the transformed and optimized IR code $\mathcal{D}_\gamma$ can be estimated as

$$\begin{aligned} RS_t(\mathcal{D}_\gamma) = &\sum_{x \in \mathcal{X}_\gamma} \max\left(0, \Pi_t(\Lambda_{\mathcal{P}}(x)) - \mathcal{H}_t(\Lambda_{\mathcal{D}_\gamma}(x))\right) \\ &+ \sum_{e \in \mathcal{E}_\gamma} \Pi_t(\Lambda_{\mathcal{P}}(e)) - \sum_{w \in \mathcal{W}_\gamma} \mathcal{H}_t(\Lambda_{\mathcal{D}_\gamma}(w)) \end{aligned} \tag{4.5}$$

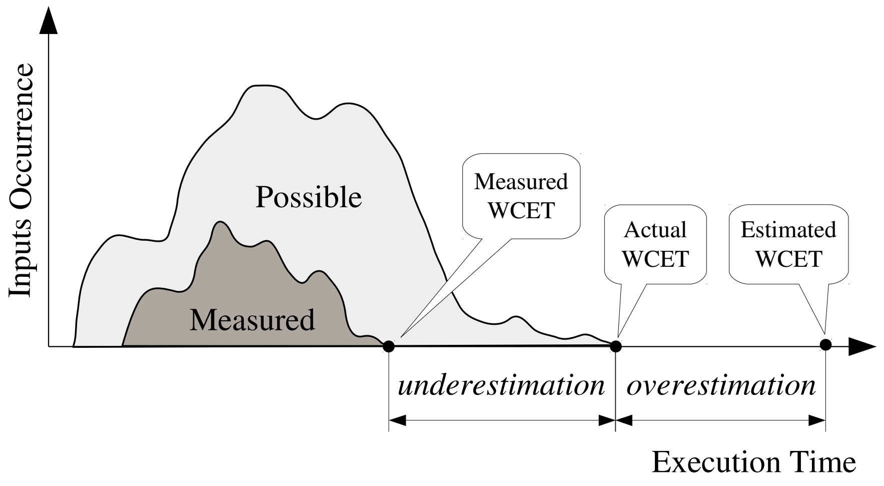
where $\Pi_t : \mathcal{V}_{\mathcal{P}} \to \mathbb{N}$ and $\mathcal{H}_t : \mathcal{V}_{\mathcal{D}_\gamma} \to \mathbb{N}$ are functions mapping the IR operation attributes in the parent and approximated designs respectively to their number of resources of type t, derived from the netlist Θ and the device's HLS pre-characterization respectively. Note that, after approximating an operation in $\mathcal{O}_{\mathcal{P}}$, it is either part of $\mathcal{E}_\gamma$ or $\mathcal{X}_\gamma$ depending on the AC technique applied. Moreover, due to possibly over-estimations for the resource usage of transformed operations derived from the HLS pre-characterization, the difference in resources usage before and after such approximations are floored at zero.

After estimating the resource savings, we can calculate the *Relative Resource Savings* (RRS), which measures the resources of each type we have eliminated relative to the total number of resources we still have to eliminate to meet each resource constraint at the current RGC iteration, given the parent's design resources usage. For each resource type $t \in \mathcal{T}$, the RRS of the approximated design $\mathcal{D}_\gamma$ can be calculated as

$$RRS_t(\mathcal{D}_\gamma) = \begin{cases} \min\left(\dfrac{RS_t(\mathcal{D}_\gamma)}{\mathcal{P}_{\mathcal{R}_t} - C_{\mathcal{R}_t}}, 1\right) & \text{if } \mathcal{P}_{\mathcal{R}_t} > C_{\mathcal{R}_t} \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

where $\mathcal{P}_{\mathcal{R}_t}$ is the exact number of resources of type t used by the parent design $\mathcal{P}$, and $C_{\mathcal{R}_t}$ is the resource constraint. Note that $RRS_t(\mathcal{D}_\gamma)$ is clamped within $[0, 1]$, where $RRS_t(\mathcal{D}_\gamma) = 0$ means that approximating $\mathcal{P}_{\mathcal{D}_{IR}}$ with the γ does not result in any benefit regarding the resource type t because either resources of this type are not saved ($RS_t(\mathcal{D}_\gamma) = 0$) or the constraint is already met ($\mathcal{P}_{\mathcal{R}_t} \leq C_{\mathcal{R}_t}$). Conversely, $RRS_t(\mathcal{D}_\gamma) = 1$ means that the constraint would be met if this approximation step is kept. Avoiding values greater than one prevents overvaluing approximations that provide resources elimination that exceed the strictly necessary to meet constraints. Therefore, this metric dynamically adapts as the RGC phase iteratively converges to meet all resource constraints. Still, note that full synthesis compilation is performed for the chosen design (which becomes the parent in the next iteration) to get the exact resources usage and and thus avoid accumulating estimation errors across the RGC iterations.

Figure 4.5 – Execution times distribution and timing analysis for real-time systems



Source: The author

4.3.2 Execution Time Savings

The design of real-time systems requires dealing with the problem of estimating the WCET to guarantee that the system execution time is strictly bounded to the specified real-time constraint (Lokuciejewski; Marwedel, 2011). In general, the WCET analysis must be safe and accurate, i.e., the actual execution times should not exceed the estimated WCET and it must be as close as possible to the actual maximal execution time to avoid over-design. For AHLS purposes, for example, overestimating the system WCET may steer the tool to introduce more error than the necessary as approximations are performed until the constraint is met. Therefore, the quality of results is highly dependent of the WCET analysis methodology.

Figure 4.5 illustrates the WCET analysis problem. As can be observed, measurement-based methodologies allow designers to estimate the WCET by stimulating the synthesized hardware with a set of representative inputs chosen according to some criterion, such as the probability of occurrence. Ideally, those inputs should cover scenarios where the maximal execution time manifests. This approach, however, is not safe for hard real-time systems, as the inputs that lead to the worst-case scenario are generally not known. Given that measuring the execution times for all possible inputs is not feasible for most cases, designers usually add a safety margin, which often leads to highly overestimated results. Thus, measurement-based methods are not adequate to ensure hard real-time requirements.

To overcome the limitations of measurement-based approaches, designers can resort to static WCET analysis, which can guarantee that the system's execution time is bounded by the timing constraint specified. With this approach, the system is not executed on a real hardware or simulator, but instead it is statically analyzed for determining the time-critical path, i.e., the sequence of operations performed in the extreme running time case, and how much time the system will take to execute that sequence. Due to runtime dependencies, however, the path analysis problem is undecidable in general (Puschner; Koza, 1989). Loop bounds, for example, cannot be statically determined if they depend on input values. Thus, designers must provide insights about the system functionality regarding runtime properties, which often produce over-estimations.

Additionally, static analysis methodologies require modeling the hardware where the system will be executed to estimate the number of clock cycles spent by each basic block. The number of cycles and the clock period are mandatory to compute how much time the system will take to execute the time-critical path. For processor-based implementations, modeling the microarchitecture behavior is much more challenging due to unpredictable runtime behaviors, such as cache hits/misses, which dynamically impact the execution times of basic blocks (Shaw, 1989). Thus, it is a common source of imprecision when estimating the WCET. On the other hand, for hardware implementations (ASICs or FPGAs) without processor support this step is simplified, as the execution time of basic blocks without blocking function calls is constant and can be derived from the clock frequency and the number of control steps produced by the synthesis process.

As discussed, static WCET analysis is a traditional method for computing safe execution time bounds, which is mandatory for hard real-time systems. Thus, this method plays an important role for an AHLS methodology aimed to perform approximations to systematically reduce the system WCET until a given time constraint is met. Specifically, static methods offer more flexibility, as they operate on the same code and data representations used by HLS tools to perform resource, scheduling, and binding optimizations, and thus can be easily integrated with them. For this thesis, we have implemented an optimized version of the Implicit Path Enumeration Technique (IPET) proposed by Li and Malik (1997), which is a widely used method for calculating the WCET in state-of-the-art timing analysis tools.

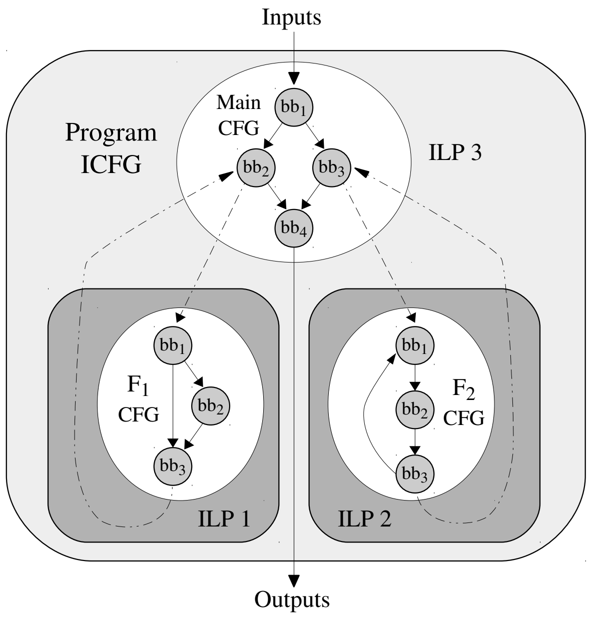
The IPET estimates the system WCET by formulating an ILP problem that implicitly evaluates all possible execution paths. It defines a linear objective function representing the program execution time, a set of non-negative integer decision variables representing the execution frequency of edges in the Interprocedural Control Flow Graph (ICFG), and a set of linear constraints. The goal is to maximize the objective function such that all constraints are satisfied. Structural constraints (flow properties) are fully derived from the program ICFG, while functional constraints that depend on runtime properties must be provided by designers, being mandatory for at least loop bounds that cannot be statically computed. Optionally, other path information that depends on the system functionality can be provided to eliminate unfeasible paths from consideration, which may reduce the estimation pessimism at the cost of increasing the ILP problem size. Therefore, the WCET analysis accuracy depends on properly steering the analysis tool. More details can be found in the work proposed by Li and Malik (1997).

Our IPET-based WCET analysis tool was implemented as an analysis pass within the LLVM compiler framework and exploits the fact that the programs designed for HLS compilation do not have recursion, i.e., their call graphs are always DAGs. Thus, instead of solving the ILP problem derived from the entire program, we can solve smaller ILP problems derived from the Control Flow Graph (CFG) of each function in the reversed order produced by topological sort. Moreover, independent ILP problems can be solved in parallel. In a nutshell, we divide the original ILP problem proposed by Li and Malik (1997) in smaller ILP problems and solve them in a bottom-up fashion, parallelizing whenever possible. In general, this approach can offer a sensible reduction in the time spent with WCET analysis, as ILP problems are NP-complete with respect to the number of constraints. It is specially beneficial when analyzing complex programs devised with several functions. Therefore, in the context of HLS-based designs, the WCET analysis problem can be translated into a set of ILP problems by defining an objective function for each function F in program P as

$$\forall F \in P : WCET(F) = max \sum_{B \in F} WCET(B) \cdot C_B, \tag{4.7}$$

where $WCET(B)$ and C_B are respectively the estimated WCET and the execution count of basic block B. In Equation (4.7), the execution time of basic blocks without function calls is constant, defined by the minimum clock period informed by the device-specific synthesis tool times the number of cycles spent by the basic block, which can be obtained from the scheduling report produced by the HLS back-end. However, if a basic block calls a function, then its WCET can be estimated as the sum of the constant part and the WCET estimated for the called function, assuming blocking function calls. Thus, the ILP problem for the main function is the last one to be solved, as the WCET of each function is estimated in the order defined by topological sort. Regarding structural and functional

Figure 4.6 – IPET-based static WCET analysis for HLS



Source: The author

constraints for each objective function defined in Equation (4.7), they are formulated in the same way as defined by Li and Malik (1997), except that we have a set of constraints for each function instead of for the whole program.

Figure 4.6 exemplifies the IPET-based WCET analyzer for a simple program with three functions. The basic blocks $bb2$ and $bb3$ in the main function call functions F_1 and F_2 respectively. As F_1 and F_2 do not call any function and are independent, their ILP problems are solved in parallel. Next, the estimated WCETs are passed to the caller function (main) to estimate the $bb2$ and the $bb3$ WCETs. Finally, the program WCET is estimated by solving the ILP problem associated with the main function. An important observation is that the presented WCET analyzer is used as a case study for our methodology, as it leverages specific characteristics of HLS-based designs to minimize the estimation complexity and runtime. It has the same limitations as other static analyzers implementing the IPET formulation, i.e., the estimation accuracy depends on properly steering the tool, as previously discussed. However, any other WCET tool can be used. The only requirement is that it must provide the estimated critical path and the execution count of basic blocks in it, so that approximation strategies can be fine-tuned to optimize the WCET reduction.

Time savings are obtained depending on whether the AC transformations are able to increase the circuit's maximum frequency or reduce the number of control steps in the time-critical execution path. Thus, given the parent's design scheduling report $\mathcal{S}$ provided by the HLS back-end and the basic blocks in the time-critical path $\mathcal{C}_{path}$ provided by the

WCET analysis pass, as presented in Figure 4.4, we can estimate how many cycles can be saved by calculating the difference between the number of control steps in those basic blocks' data flow graph before and after each approximation step according to the ASAP scheduling, times their execution count in the worst-case scenario ($\Psi_{\mathcal{P}}$), which is provided by the WCET analyzer as well. The actual time savings are then obtained by dividing the cycle savings by the circuit's maximum frequency f_{max} obtained from the timing report for the target FPGA. Note that the maximum frequency is from the parent's design characterization, i.e., possible changes in frequency due to single AC transformations are overlooked. Nevertheless, we have observed that those changes are negligible when they occur, as the HLS back-end tries to chain as many operations as possible at each control step following the provided timing directives. Thus, the number of control steps is much more likely to be affected. Moreover, the maximum frequency is precisely evaluated for each new parent design to avoid accumulating such variations across the RGC iterations.

Formally, let $\mathcal{B}_{\mathcal{P}}$ and $\mathcal{B}_{\mathcal{D}_\gamma}$ denote respectively the set of basic blocks in the parent's IR code time-critical execution path $\mathcal{C}_{path}$ and in the optimized IR code $\mathcal{D}_\gamma$ resulting from applying the AC transformation $\gamma \in \Gamma$. It follows that $\mathcal{E}_\gamma = \mathcal{B}_{\mathcal{P}} \setminus \mathcal{B}_{\mathcal{D}_\gamma}$ is the set of eliminated basic blocks in $\mathcal{C}_{path}$ and $\mathcal{M}_\gamma = \mathcal{B}_{\mathcal{P}} \cap \mathcal{B}_{\mathcal{D}_\gamma}$ is the set of basic blocks in $\mathcal{C}_{path}$ kept in the approximated design. Additionally, let $\Delta_{\mathcal{P}} : \mathcal{B}_{\mathcal{P}} \to \mathbb{N}$ and $\Delta_{\mathcal{D}_\gamma} : \mathcal{B}_{\mathcal{D}_\gamma} \to \mathbb{N}$ be functions mapping the set of basic blocks in $\mathcal{P}_{\mathcal{D}_{IR}}$ and $\mathcal{D}_\gamma$ to their control step counts respectively, as provided by the parent's scheduling report and the approximated basic block's new scheduling derived from the set of eliminated operations. Therefore, the WCET reduction obtained by approximating the parent design $\mathcal{P}_{\mathcal{D}_{IR}}$ with an AC transformation $\gamma \in \Gamma$ can be estimated as

$$\begin{aligned} TS(\mathcal{D}_\gamma) = \frac{1}{f_{max}} \Big(&\sum_{b \in \mathcal{M}_\gamma} (\Psi_{\mathcal{P}}(b) \cdot \Delta_{\mathcal{P}}(b) - \Psi_{\mathcal{D}_\gamma}(b) \cdot \Delta_{\mathcal{D}_\gamma}(b)) \\ &+ \sum_{b \in \mathcal{E}_\gamma} \Psi_{\mathcal{P}}(b) \cdot \Delta_{\mathcal{P}}(b) \Big) \end{aligned} \tag{4.8}$$

where $\Psi_{\mathcal{P}}(b)$ and $\Psi_{\mathcal{D}_\gamma}(b)$ are respectively the execution count of basic block b in the parent's design worst-case scenario provided by the WCET analysis tool and in the transformed design $\mathcal{D}_\gamma$. Note that $\Psi_{\mathcal{P}}(b) \neq \Psi_{\mathcal{D}_\gamma}(b)$ only if the basic block b belongs to a loop in which the number of iterations was changed by the the AC transformation γ, a common situation when loop perforation techniques are employed. For those cases, the estimation $\Psi_{\mathcal{D}_\gamma}(b) = \lceil \Psi_{\mathcal{P}}(b) - \Psi_{\mathcal{P}}(b)/r \rceil$ is considered, where r is the perforation ratio used as the

LP transformation parameter defined in Section 4.2.3.

After estimating the time savings, we are able to calculate the *Relative Time Savings* (RTS), which measure the time we have saved relative to the total time we still have to save to meet the real-time constraint at the current RGC iteration, given the parent's design WCET. It can be calculated as

$$RTS(\mathcal{D}_\gamma) = \begin{cases} \min\left(\dfrac{TS(\mathcal{D}_\gamma)}{\mathcal{P}_W - C_t}, 1\right) & \text{if } \mathcal{P}_W > C_t \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

where $\mathcal{P}_W$ is the parent's design WCET, and C_W is the real-time constraint. Note that $RTS(\mathcal{D}_\gamma)$ is clamped within $[0, 1]$, where $RTS(\mathcal{D}_\gamma) = 0$ means that approximating $\mathcal{P}_{\mathcal{D}_{IR}}$ with the γ does not result in WCET reduction because either no control step in $\mathcal{C}_{path}$ was eliminated ($TS(\mathcal{D}_\gamma) = 0$) or the constraint is already met ($\mathcal{P}_W \leq C_t$). Conversely, $RTS(\mathcal{D}_\gamma) = 1$ means that the constraint would be met if this approximation step is kept. Avoiding values greater than one prevents overvaluing approximations that provide WCET reductions that exceed the strictly necessary to meet the constraint. Therefore, similarly to the RRS calculation in Equation (4.6), this metric dynamically adapts as the RGC phase iteratively converges to meet the real-time constraint. Nevertheless, full WCET analysis is performed for parent designs to avoid accumulating time savings estimation errors across the RGC iterations.

4.3.3 Error Measurement

The error introduced by approximations can be evaluated through simulation-based techniques or formal methods based on symbolic computations. Simulation approaches are commonly used to produce error statistics derived from Monte Carlo simulations, which can be performed over the entire circuit or smaller sub-blocks. With a divide-and-conquer strategy, the circuit can be considered a network of approximate building blocks, which can be individually analyzed by Monte Carlo simulations. Then, the error statistics are propagated through the network using analytical methods for error composition. Using pre-characterized error statistics of smaller sub-blocks usually reduces simulation times by limiting input combinations. However, as this approach requires the use of analytical methods modeling the error composition from the approximated sub-blocks to the system primary outputs, which are currently restricted to composing errors across arithmetic operators like adders and multipliers, the application scope becomes limited. Moreover, it has the drawback of depending on a library previously characterized operators (Chakrapani et al., 2008; Palem et al., 2009; Chan et al., 2013; Li et al., 2015; SENGUPTA et al., 2017; CASTRO-GODíNEZ et al., 2018). Also, as the error is input-dependent, providing significant confidence for simulation-based methodologies depends on stimulating the system with typical workloads and defining adequate EMs for each application scenario (Chan et al., 2013).

In contrast, formal methods based on symbolic computations on Boolean formulas can provide formal error guarantees for a wide range of EMs. BDD-based error analysis, for example, scales very well on non-pathological classes of circuits to provide an error distribution offering information about the probability of occurrence of errors of different magnitudes. However, it cannot be applied to calculate arbitrary error metrics and the BDDs can be challenging to construct for common circuits comprising larger bit-width operators like multipliers, division, remainder, and reciprocal, as they exhibit exponential memory requirements for any variable ordering (VASICEK, 2017). SAT-based approaches may offer scalability as well, but they apply to the worst-case error analysis only, which may not be an adequate error metric to explore approximation options in general because in most applications the knowledge of the worst-case error alone will not suffice, since its probability of occurrence is very low in most operating scenarios (VENKATESAN et al., 2011). Although formal methods relying on Boolean satisfiability solvers can provide error bound guarantees on the solution, scalability is often the main limitation when establishing other EMs like the error rate or the average-case error. Therefore, formal methods may jeopardize generality and scalability depending on the underlying model checking algorithm (RANJAN et al., 2014; CHANDRASEKHARAN et al., 2016), being impractical for arbitrary designs with complex data and control paths.

For this thesis, a simulation-based methodology was adopted to characterize the error statistics for the entire circuit, wherein the evaluated EMs are the Mean-Squared Error (MSE) and the Percentage Accuracy (PA), averaged over a set of training inputs for DSE and another set of test inputs to evaluate the final design. The MSE and PA metrics for each input are defined as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(O_i - \hat{O}_i\right)^2 \tag{4.10}$$

$$PA = 100 \times \left(1 - \frac{1}{N}\sum_{i=1}^{N} diff\left(O_i, \hat{O}_i\right)\right) \tag{4.11}$$

where N is the number of values in a training or test input and O_i and $\hat{O}_i$ are the golden and approximated values respectively. In Equation 4.11, $diff(O_i, \hat{O}_i) = 1$ if $O_i \neq \hat{O}_i$, otherwise $diff(O_i, \hat{O}_i) = 0$. Note that Equation 4.11 calculates the percentage number of approximated output values that match with the golden values. Those metrics are commonly used to assess results when evaluating approximation techniques for a wide range of multimedia processing and machine learning applications (Li et al., 2015; Lee; John; Gerstlauer, 2017; Nepal et al., 2019; NAZAR et al., 2021a). They are used as representative examples in this work and can be replaced by any metric the designers see fit for each specific application. Although simulating entire circuits is time-consuming due to the number of inputs needed for sufficient confidence, in this work such limitation is circumvented by performing functional simulations at software-level, with the IR code that is functionally equivalent to the implemented circuit. In the last step for characterizing approximate design candidates, as presented in Figure 4.4 (step 6), the exact outputs generated by executing the exact IR code compiled with the compiler's x86 back-end are compared with the approximate outputs generated by executing each approximated design generated across the RGC iterations. Therefore, as detailed in chapters 6 and 7, this approach can produce error statistics in reasonable run-times without losing generality, even for complex designs.

4.4 GRASP-based Optimization Heuristic

In this section, we detail the optimization heuristic developed to explore the design space by exploiting the set AC techniques implemented as compiler transformations, as detailed in Section 4.2, and using the methods described in Section 4.3 to estimate resource and time savings and to obtain error statistics of approximate design candidates. Algorithm 1 shows the GRASP-based approximation heuristic with its pseudo-code. The inputs are the design source code $\mathcal{D}_S$, the set of AC techniques $\mathcal{A}$, the set of training in-

Algorithm 1 GRASP-based Approximation Heuristic

```
Inputs: Design source code ($\mathcal{D}_S$), set of training inputs ($\Omega$), set of AC techniques ($\mathcal{A}$),
    set of resource types ($\mathcal{T}$), set of resource constraints ($C_{\mathcal{R}}$), real-time constraint ($C_W$),
    and number of iterations ($K$)
Output: The approximated design ($\mathcal{D}_{BIT}$)
 1: $\mathcal{D}_{IR} \leftarrow HLS\_FRONT\_END(\mathcal{D}_S)$
 2: $(\mathcal{D}_{HDL}, \mathcal{S}) \leftarrow HLS\_BACK\_END(\mathcal{D}_{IR})$
 3: $(\mathcal{D}_{BIT}, \Theta, f_{max}, \mathcal{R}) \leftarrow SYNTH\_P\&R(\mathcal{D}_{HDL})$
 4: $(\mathcal{C}_{path}, w, \Psi) \leftarrow WCET(D_{IR}, f_{max}, \mathcal{S})$
 5: $\Xi \leftarrow (\mathcal{D}_{IR}, \mathcal{D}_{BIT}, \mathcal{R}, \mathcal{S}, \Theta, w, \Psi, \mathcal{C}_{path}, f_{max}, \epsilon = 0)$
 6: if $\Xi_{\mathcal{R}_t} \leq C_{\mathcal{R}_t}\ \forall t \in \mathcal{T}$ and $\Xi_w \leq C_W$ then
 7:     return $(\Xi_{\mathcal{D}_{BIT}}, \Xi_\epsilon)$
 8: else
 9:     $\Phi \leftarrow \emptyset$
10:     for $k = 1$ to $K$ do
11:         $seed \leftarrow NEW\_SEED()$
12:         $(s, \mathcal{N}_s) \leftarrow RGC(\Xi, seed)$
13:         $\phi \leftarrow LS(s, \mathcal{N}_s)$
14:         $\Phi \leftarrow \Phi \cup \{\phi\}$
15:     end for
16:     $\phi^* \leftarrow \min_\epsilon(\Phi)$
17:     return $(\phi^*_{\mathcal{D}_{BIT}}, \phi^*_\epsilon)$
18: end if
```

puts Ω, the set of resource constraints $C_{\mathcal{R}}$, the real-time constraint C_W, and the number of GRASP iterations K. First, the source code is compiled with the HLS front-end to generate the design's IR code $\mathcal{D}_{IR}$. Next, the exact IR code is fully synthesized to obtain the scheduling report $\mathcal{S}$, the netlist Θ, the maximum clock frequency f_{max}, the resources usage report $\mathcal{R}$ for the target FPGA, and the design's bitstream $\mathcal{D}_{BIT}$ (lines 1-3). With the scheduling and the maximum operating frequency at hand, the circuit's WCET w, the time-critical execution path $\mathcal{C}_{path}$, and the execution count of basic blocks Ψ are obtained through the WCET analyzer (line 4).

After saving the exact IR code $\mathcal{D}_{IR}$ and its characterization regarding $\mathcal{R}$, $\mathcal{S}$, Θ, w, Ψ, $\mathcal{C}_{path}$, f_{max}, and error ($\epsilon = 0$) as the exact design Ξ to be transformed by the GRASP iterations (line 5), we verify if it already meets constraints, i.e., if $\Xi_{\mathcal{R}_t} \leq C_{\mathcal{R}_t}$, $\forall t \in \mathcal{T}$, and $\Xi_w \leq C_W$, where $\Xi_{\mathcal{R}_t}$ and Ξ_w are the exact design resource usage for resource type t and WCET respectively (line 6). If it does, the exact bitstream $\Xi_{\mathcal{D}_{BIT}}$ and its error Ξ_ϵ (no error at all) are returned as the final solution (line 7). Otherwise, it is forwarded to the GRASP heuristic to be approximated by K independent iterations (lines 10-15), executed in parallel using K processor cores or threads. At each GRASP iteration, the exact design is forwarded to the Randomized Greedy Construction (RGC) procedure with a unique seed (lines 11-12), where a feasible design s meeting all constraints is returned, along with a list of its neighbor designs $\mathcal{N}_s$. The RGC procedure is detailed in Section 4.4.1.

Next, the feasible design s built in the RGC procedure is forwarded to the Local Search (LS) procedure along with it's neighborhood list $\mathcal{N}_s$ (line 13) to search for a locally optimal feasible design ϕ with minimum error. The LS procedure is detailed in Section 4.4.2. Such design is then returned and inserted into the set Φ of locally optimal feasible designs built by each GRASP iteration (line 14). After all GRASP iterations have finished, the design $\phi^* \in \Phi$ with minimum error, i.e, the optimal one among all GRASP iterations, is selected as the final solution (line 16). Finally, its bitstream $\phi^*_{\mathcal{D}_{BIT}}$ and error ϕ^*_ϵ are returned as the heuristic outcomes. Note, however, that due to the complexity of the design space in the optimization problem defined in Section 4.1.1, no global optimality guarantee can be offered. Still, sufficient GRASP iterations can provide sufficiently good solutions, as demonstrated in sections 6 and 7 with experimental results.

4.4.1 Randomized Greedy Construction Procedure

Algorithm 2 shows the RGC procedure pseudo-code, where the inputs are the exact design Ξ and a random seed. First, the exact design is saved as the parent design $\mathcal{P}$ to be iteratively transformed by the AC techniques available until all constraints are met. At each iteration of the main loop (lines 2-19), a set Υ of approximate designs is built by applying each approximation transformation $\gamma \in \Gamma$, one at a time, where $\Gamma = \mathcal{P}_{\mathcal{D}_{IR}} \times \mathcal{A}$ is the set of all possible approximation transformations for the parent's IR code giving the set of AC techniques available, which may include transformations on operations (arithmetic, logic, and memory), basic blocks, loops, and functions (lines 5-12).

After transforming the parent's design IR code $\mathcal{P}_{\mathcal{D}_{IR}}$ with an AC transformation γ and applying the standard compiler optimizations provided by the HLS compiler infrastructure (e.g., dead code elimination, constant propagation, and strength reduction) (lines 6-7), we estimate the resulting resources and time savings for the transformed and optimized IR code $\mathcal{D}_\gamma$, as defined by Equations (4.5) and (4.8) respectively. These estimations are then used to calculate the relative resources and time savings (RRS and RTS), measuring the savings relative to the total resources and time we still need to save to meet all constraints at the current iteration, given the parent's design resources usage and WCET, as defined by Equations (4.6) and (4.9) respectively. With those metrics at hand (line 8), we can calculate the *Distance to Constrain* (DTC), δ, which is a function measuring how

Algorithm 2 GRASP Randomized Greedy Construction Procedure

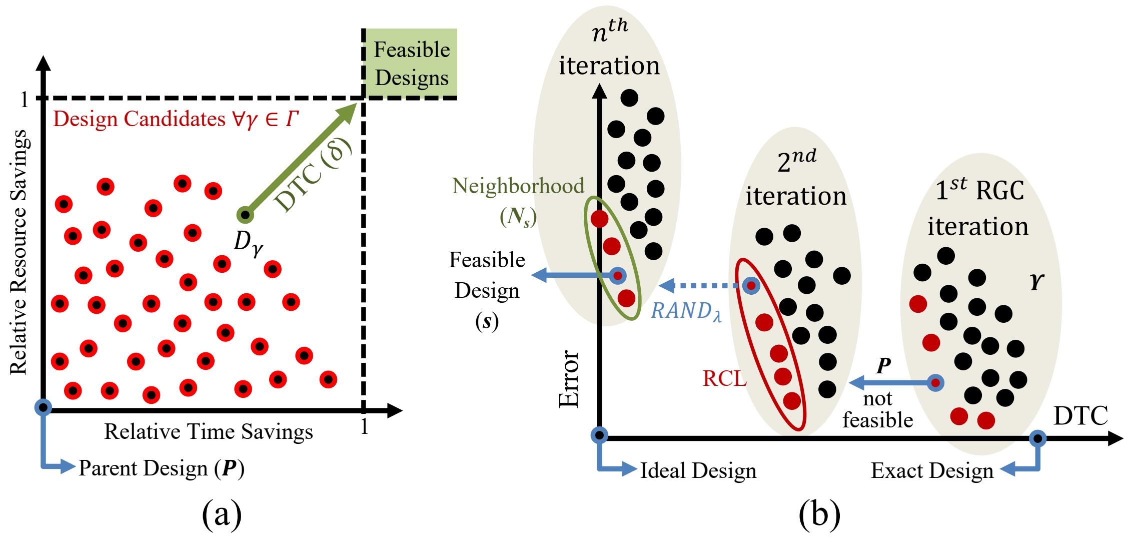
```
procedure RGC($\Xi$, $seed$)
 1: $\mathcal{P} \leftarrow \Xi$
 2: repeat
 3:     $\Upsilon \leftarrow \emptyset$
 4:     $\Gamma \leftarrow \mathcal{P}_{\mathcal{D}_{IR}} \times \mathcal{A}$
 5:     for all approximation transformations $\gamma \in \Gamma$ do
 6:         $\mathcal{D}_\gamma \leftarrow ACT_\gamma(\mathcal{P}_{\mathcal{D}_{IR}})$
 7:         $\mathcal{D}_\gamma \leftarrow HLS\_OPT(\mathcal{D}_\gamma)$
 8:         $\delta \leftarrow DTC(\mathcal{D}_\gamma)$
 9:         $\epsilon \leftarrow \bar{E}(\mathcal{D}_\gamma, \Omega)$
10:         $\lambda \leftarrow fit(\delta, \epsilon)$
11:         $\Upsilon \leftarrow \Upsilon \cup (\mathcal{D}_\gamma, \delta, \epsilon, \lambda)$
12:     end for
13:     $RCL \leftarrow PARETO_{\delta,\epsilon}(\Upsilon)$
14:     $(\mathcal{D}_{IR}, \epsilon) \leftarrow RAND_\lambda(RCL, seed)$
15:     $(\mathcal{D}_{HDL}, \mathcal{S}) \leftarrow HLS\_BACK\_END(\mathcal{D}_{IR})$
16:     $(\mathcal{D}_{BIT}, \Theta, f_{max}, \mathcal{R}) \leftarrow SYNTH\_P\&R(\mathcal{D}_{HDL})$
17:     $(\mathcal{C}_{path}, w, \Psi) \leftarrow WCET(D_{IR}, f_{max}, \mathcal{S})$
18:     $\mathcal{P} \leftarrow (\mathcal{D}_{IR}, \mathcal{D}_{BIT}, \mathcal{R}, \mathcal{S}, \Theta, w, \mathcal{C}_{path}, \Psi, f_{max}, \epsilon)$
19: until $\mathcal{P}_{\mathcal{R}_t} \leq C_{\mathcal{R}_t}\ \forall t \in \mathcal{T}$ and $\mathcal{P}_w \leq C_W$
20: return $(\mathcal{P}, RCL)$
end RGC
```

distant an approximated design is from meeting constraints and thus being feasible, as illustrated in Figure 4.7(a) with $|\mathcal{T}| = 1$ for the sake of clarity. Note, however, that the DTC metric is an estimation used to avoid fully synthesizing all approximate candidates, since it is obtained from the estimated values for resource and time savings. Nevertheless, as the DTC calculation is based on the parent's design precise resources and WCET metrics, estimation errors are not accumulated across the RGC iterations.

Next, the set of training inputs, Ω, is used to evaluate the design's average error ϵ using the appropriate EM (line 9), as described in Section 4.3.3. At this point, the design's fitness λ towards meeting all constraints with minimum error at the current iteration is evaluated according to δ and ϵ (line 10). Finally, the approximated IR code $\mathcal{D}_\gamma$ and its characterization regarding δ, ϵ, and λ are inserted into the set Υ of approximated designs candidates. Section 4.4.3 details how the approximate designs' DTC and fitness are evaluated at a given RGC iteration.

Figure 4.7(b) illustrates the DSE of approximated designs in the RGC procedure. When all approximation transformations available at a given iteration have been evaluated, we build the RCL with all Pareto-optimal approximated designs from Υ regarding the distance to constrain δ and the error ϵ (line 13), shown in red in Figure 4.7(b). Then,

Figure 4.7 – DSE of approximate designs: (a) DSE at each RGC iteration and (b) DSE on subsequent RGC iterations until a feasible design is found



Source: The author

one approximated design, along with its error, is randomly selected from the RCL using the fitness value λ as a selection bias, i.e., designs with higher fitness are more likely to be selected (line 14). Next, the selected IR code is fully synthesized and evaluated for WCET (lines 15-17), and its characterization regarding the IR code ($\mathcal{D}_{IR}$), bitstream ($\mathcal{D}_{BIT}$), resources ($\mathcal{R}$), scheduling ($\mathcal{S}$), netlist (Θ), WCET (w), time-critical execution path ($\mathcal{C}_{path}$), basic blocks' execution count (Ψ), maximum frequency (f_{max}), and error (ϵ) is saved as the new parent design for the next iteration (line 18).

At this point, the new parent design feasibility is verified against the constraints $\mathcal{C}_{\mathcal{R}}$ and C_W by analyzing the resources report $\mathcal{P}_{\mathcal{R}}$ and the WCET $\mathcal{P}_w$ respectively (line 19). If constraints are not met (design not feasible), it is forwarded to the next iteration. Otherwise, it is returned as the feasible design s to the main GRASP iteration, along with the neighbors list $\mathcal{N}_s$, i.e., the RCL from which s was selected. An important observation is that the actual feasibility of a given design is known only after being selected from the RCL and fully synthesized, due to possible estimation errors in the DTC computation.

4.4.2 Local Search Procedure

Algorithm 3 shows the Local Search (LS) procedure pseudo-code, where the inputs are the feasible design s and its neighbors list $\mathcal{N}_s$ built by the RGC procedure in the main GRASP iteration. First, the neighborhood of s is sorted in ascending order of error to obtain the sorted list $\mathcal{N}_s^*$. Next, we search for a locally optimal feasible design with

Algorithm 3 GRASP Local Search Procedure

```
procedure LS($s, \mathcal{N}_s$)
 1: $\mathcal{N}_s^* \leftarrow SORT\_ASCENDING_\epsilon(\mathcal{N}_s)$
 2: repeat
 3:     $(\mathcal{D}_{IR}, \epsilon) \leftarrow REMOVE\_FIRST(\mathcal{N}_s^*)$
 4:     if $\epsilon \geq s_\epsilon$ then
 5:         return $s$
 6:     else
 7:         $(\mathcal{D}_{HDL}, \mathcal{S}) \leftarrow HLS\_BE(\mathcal{D}_{IR})$
 8:         $(\mathcal{D}_{BIT}, \Theta, f_{max}, \mathcal{R}) \leftarrow SYNTH\_P\&R(\mathcal{D}_{HDL})$
 9:         $(\mathcal{C}_{path}, w, \Psi) \leftarrow WCET(D_{IR}, f_{max}, \mathcal{S})$
10:         $\mathcal{L} \leftarrow (\mathcal{D}_{IR}, \mathcal{D}_{BIT}, \mathcal{R}, \mathcal{S}, \Theta, w, \mathcal{C}_{path}, \Psi, f_{max}, \epsilon)$
11:         if $\mathcal{L}_{\mathcal{R}_t} \leq C_{\mathcal{R}_t}\ \forall t \in \mathcal{T}$ and $\mathcal{L}_w \leq C_W$ then
12:             return $\mathcal{L}$
13:         end if
14:     end if
15: until $\mathcal{N}_s^* = \emptyset$
16: return $s$
end LS
```

lower error than s by iteratively removing the first design in $\mathcal{N}_s^*$ (line 3), along with its error, to verify the optimality and feasibility against s_ϵ and the constraints $C_\mathcal{R}$ and C_W respectively. The search proceeds until the first design in $\mathcal{N}_s^*$ has an equal or higher error than the current feasible design s we already have, a feasible design $\mathcal{L}$ with a lower error than s is found, or $\mathcal{N}_s^*$ becomes empty (lines 2-15).

Specifically, at each iteration, we first verify if the error ϵ of the design $\mathcal{D}_{IR}$ removed from the sorted neighbors list $\mathcal{N}_s^*$ is lower than s_ϵ (line 4). If it is not, the search stops, and the current design s is returned, i.e., the input s already is the locally optimal feasible design (line 5). Otherwise, $\mathcal{D}_{IR}$ is fully synthesized and characterized to verify its feasibility, i.e., if all constraints are met (lines 7-10). If constraints are met, the design $\mathcal{L}$, characterized by the IR code ($\mathcal{D}_{IR}$), bitstream ($\mathcal{D}_{BIT}$), resources ($\mathcal{R}$), scheduling ($\mathcal{S}$), netlist (Θ), WCET (w), time-critical execution path ($\mathcal{C}_{path}$), basic blocks' execution count (Ψ), maximum frequency (f_{max}), and error (ϵ) is returned as the locally optimal feasible solution (line 12). Otherwise, the search proceeds to verify the next design in $\mathcal{N}_s^*$. At this point, if $\mathcal{N}_s^*$ is empty, the search stops, and the solution s is returned.

4.4.3 Fitness Function

As discussed in Section 4.4.1, the fitness of each approximate design candidate is a function of their DTC and error at a given RGC iteration. After evaluating Equations (4.6) and (4.9) as a result of approximating $\mathcal{P}_{\mathcal{D}_{IR}}$ with the AC transformation $\gamma \in \Gamma$ and applying the standard compiler optimizations, we calculate the DTC of design $\mathcal{D}_\gamma$ as

$$DTC(\mathcal{D}_\gamma) = \sqrt{(1 - RTS(\mathcal{D}_\gamma))^2 + \sum_{t \in \mathcal{T}} (1 - RRS_t(\mathcal{D}_\gamma))^2} \qquad (4.12)$$

Therefore, $DTC(\mathcal{D}_\gamma)$ measures the Euclidean distance of approximated design $\mathcal{D}_\gamma$ to meet all constraints and thus become feasible. Finally, after evaluating Equation (4.12) and the EM for each approximation step, we can rank the approximate design candidates according to their fitness, defined as

$$fit(\mathcal{D}_\gamma) = \frac{1}{\bar{E}(\mathcal{D}_\gamma, \Omega) \cdot DTC(\mathcal{D}_\gamma)} \qquad (4.13)$$

where $\bar{E}(\mathcal{D}_\gamma, \Omega)$ is the EM measured for the approximated design at a given RGC iteration, averaged over the set of training inputs. Not to lose generality, we define $E = f$ or $E = 1/f$ depending if the EM is a function f that should be maximized or minimized, respectively, such as the PA and the MSE. Thus, $fit(\mathcal{D}_\gamma)$ is a measure of the cost-benefit of approximating $\mathcal{P}_{\mathcal{D}_{IR}}$ with the AC transformation γ at a given RGC iteration that can be fine-tuned according to any adequate EM for the application at hand.

4.4.4 Generality and Scalability

As detailed in the previous sections, full synthesis and WCET analysis are performed for the exact design before the first RGC iteration and for the approximate design candidate chosen from the RCL at the end of each RGC iteration, aiming to verify their feasibility regarding the specified constraints. The designs' full characterization obtained with such an approach is used not just to stop the RGC iterative process if constraints are met but also to guide the heuristic decisions in the next iteration, if necessary, by using the actual netlist, clock period, resources usage, WCET, execution counts of basic blocks in the time-critical path, and scheduling information of the parent design used for further approximations to avoid accumulating possible resources and time savings estimation errors as new approximations are applied across iterations. Moreover, by knowing the parent's design resources usage and WCET, the heuristic is able to make wiser decisions when exploring approximation options to optimize the target design metrics, choosing coarse-grained or more fine-grained AC transformations depending on how far each constraint is from being met. Note that there are no restrictions for the input source code, except the ones already defined by the chosen HLS tool.

Regarding the heuristic analysis runtime for each GRASP iteration (lines 10-15 in Algorithm 1), it is dominated by the characterization of the selected approximate design in the RGC and LS procedures (lines 15-17 in Algorithm 2 and 7-9 in Algorithm 3), which depends on the full synthesis and the WCET analysis, and the Monte Carlo simulations for evaluating the error statistics of each approximate design candidate generated by the RGC inner loop (line 5-12 in Algorithm 2). Note that the circuit size and the number of iterations needed to meet all constraints determine the total execution time of the RGC procedure, i.e., more stringent constraints tend to increase the heuristic runtime. As shown in Chapter 7, however, the number of RGC iterations needed to meet all constraints (outer loop in lines 2-19) is usually much lower than the number of approximate design candidates to explore (hundreds even for small circuits). Therefore, the benefits of characterizing parent designs with full synthesis and WCET analysis can be exploited without excessive runtime penalty, as most of the DSE runtime is spent processing the RGC procedure inner loop.

More specifically, the execution time of the RGC inner loop (lines 5-12) is directly proportional to the number of available design entities to approximate, which tends to grow with circuit size, times the number of approximation options for such entities made available by the set of implemented AC techniques (line 4). Thus, the RGC inner loop runtime is determined by $|\Gamma|$ (line 5) times the evaluation time of each approximate design candidate (lines 6-10), which is heavily dominated by the Monte Carlo simulations needed for error statistics evaluation (line 9). Nevertheless, as simulations are performed at software-level, reasonable runtimes are expected even for relatively large datasets. Moreover, note that the characterization of each approximate design candidate resulting from applying each available AC transformation can be parallelized, as the evaluation of resources savings, time savings, and error can be performed independently. Therefore, besides executing the GRASP iterations in parallel by assigning one processor thread for each, as discussed in Section 4.1.2, the DSE runtime of each GRASP iteration can be drastically reduced by parallelizing the execution of the RGC procedure's inner loop as well, especially if there are enough threads available. An analysis of the runtime and quality of results trade-off with varying GRASP iteration counts (K) will be presented in Chapter 7.

5 EXPERIMENTAL SETUP

This chapter presents the experimental setup used to produce the experimental results discussed in Chapters 6 and 7, which includes the design flow implementation and the set of applications considered for experimental evaluation.

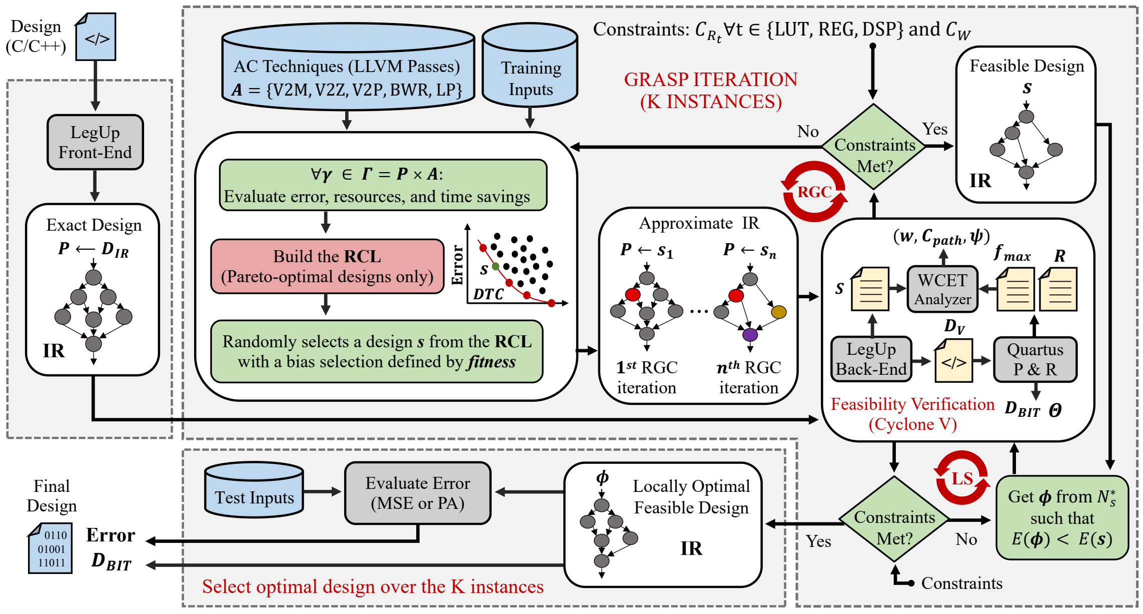
5.1 Design Flow Implementation

Figure 5.1 shows the proposed AHLS methodology implemented within the LegUp HLS tool and targeting the Intel Cyclone V FPGA (SoCKit). The inputs are the design source code in C/C++, the set of AC techniques defined in Section 4.2, i.e., V2Z, V2M, V2P, BWR, and LP, the real-time constraint (C_W), the set of resource constraints ($C_{\mathcal{R}}$) regarding LUT's, REG's, and DSP's, the number of GRASP iterations (K), and disjoint sets of training and test inputs for error measuring. First, the exact source code is compiled by the LegUp's front-end into LLVM IR ($\mathcal{D}_{IR}$). Next, the IR code is delivered to LegUp's back-end to generate a synthesizable Verilog code ($\mathcal{D}_V$) and the scheduling report ($\mathcal{S}$) targeting the Intel Cyclone V FPGA. The Verilog code is then fully synthesized using the Intel Quartus 15.0 synthesis tool to obtain the circuit's netlist (Θ), the maximum clock frequency (f_{max}), and the precise number of LUTs, REGs, and DSPs required ($\mathcal{R}$) for the target FPGA.

Given the IR code, the scheduling report, and the maximum clock frequency, we use our IPET-based WCET analyzer implemented as an LLVM analysis pass to estimate the WCET of the exact design (w) and to provide the set of basic blocks in the worst-case execution path ($\mathcal{C}_{path}$) and their execution counts (Ψ). Next, the exact design feasibility regarding constraints is verified. If the constraints are not met, the IR code and its characterization are forwarded as the parent design $\mathcal{P}$ to K instances of our GRASP-based heuristic, executed independently as K threads. At the RGC iteration, the tool automatically explores the set of LLVM approximation transformations available for the parent's design IR code, as discussed in Section 4.4.1.

The approximated design candidates are then characterized regarding the relative resources and time savings, as presented in Sections 4.3.1 and 4.3.2. We also calculate the error by executing the approximate designs with LLVM's x86 back-end and the provided set of training inputs, as described in Section 4.3.3. Then, their outputs are compared with the golden outputs previously generated to calculate the EMs herein considered (average

Figure 5.1 – Experimental setup and tool flow



Source: The author

MSE and PA). If the application hangs or crashes, we consider that the solution is not functional, and thus it is discarded as it is not feasible. Next, the RCL is built with all Pareto-optimal designs regarding the error and the distance to constrain, and one of them is randomly selected with a selection bias defined by fitness, as detailed in Section 4.4.3. The selected design (s) is forwarded to the next RGC iteration as the new parent design only if it is not feasible yet. Therefore, this process is repeated until a functional design meeting constraints is selected from the RCL. The feasible design is then forwarded to the LS iteration, where it is replaced by a locally optimal feasible design, as described in Section 4.4.2. Finally, the optimal design over the K GRASP instances is delivered as the final solution. Specifically, our design flow returns the design's bitstream for the target FPGA and the average EM calculated over the test inputs (MSE or PA).

5.2 Benchmarking Applications

The results are assessed for different applications and scenarios by exploring the available set of AC techniques both separately and in conjunction to show that (i) a constraint-aware approach for DSE can provide significant improvements over constraint-oblivious methodologies for scenarios where designers must meet single or multiple system requirements and suit to the available hardware substrate, which is a typical case

Table 5.1 – Benchmark kernels (synthesized for Intel Cyclone V)

Kernel	Resource Usage			WCET (ms)	Avg. AHLS Running Time (min)	RGC Iterations (min-max)	Quality Metric	Benchmark Suite
	LUT	REG	DSP					
ADPCM	3478	5706	64	860	123	6-24	MSE	CHStone
FIR	569	684	2	1180	49	5-23	MSE	AxBench
FFT	978	782	8	470	84	5-22	MSE	AxBench
SOBEL	1269	1042	0	1420	111	9-24	MSE	AxBench
JPEG	24148	18078	87	2630	438	5-23	MSE	CHStone
3DR	8987	12522	11	12	149	8-22	MSE	na Rosetta
MOTION	8033	7948	0	290	190	5-21	MSE	CHStone
DIGIT	42527	35136	1	23	75	9-24	PA	Rosetta

Source: The author

for FPGA-oriented synthesis (Chapter 6), and (ii) no single AC technique can outperform a proper mix of techniques, and thus the benefits of a methodology allowing seamless integration and exploitation of AC techniques within the standard design flow are demonstrated (Chapter 7). For those purposes, we have synthesized a set of eight kernels taken from the CHStone (Hara et al., 2009) (ADPCM encoder, JPEG decoder, and MOTION Estimation), the AxBench (Yazdanbakhsh et al., 2017) (FIR filter, FFT, and SOBEL), and the Rosetta (ZHOU et al., 2018) (3D Rendering and DIGIT Recognition) benchmark suites. Additionally, as discussed in greater detail in Chapter 7, we have used $K \in \{1, 2, 4, 8, 16\}$ to evaluate how the number of GRASP iterations may impact the results' quality and the heuristic running time.

Table 5.1 lists the kernels, along with the implementation results (resources and WCET) for the exact designs, the average AHLS running time and the range of RGC iterations considering all evaluated scenarios, and the quality metric used for each benchmark (PA for DIGIT and MSE for the remaining kernels). Note that the MSE measures were normalized (8-bit outputs) to ease comparing applications with distinct output value ranges. Each kernel was compiled with the LegUp default settings for the target FPGA, targeting an operating frequency of 50 MHz, and fully synthesized with the Intel Quartus 15.0 for evaluating design metrics. The test data are from the ITU-T (ITU-T, 2022), the USC-SIPI (USC-SIP, 2022), and the MNIST (DENG, 2012) databases, divided into training inputs (80%) and testing inputs (20%). The experiments were executed on an Intel Core i5-7500 at 3.4 GHz, with 16 GB of RAM, running Ubuntu 14.04 LTS.

Observe that the ADPCM and JPEG applications use substantially more DSP blocks than the other kernels. These applications make extensive use of multiplications inside loops whose iterations can be parallelized through loop unrolling optimizations, and

Figure 5.2 – Code excerpt from the ADPCM encoder application

```c
int encode (int xin1, int xin2)
{
  ...
  /* main multiply accumulate loop for samples and coefficients */
  for (i = 0; i < 10; i++)
  {
     xa += (long) (*tqmf_ptr++) * (*h_ptr++);
     xb += (long) (*tqmf_ptr++) * (*h_ptr++);
  }
  ...
}
```

Source: The author

Figure 5.3 – Code excerpt from the JPEG decoder application

```c
void ChenIDct (int *x, int *y)
{
  ...
  /* Loop over columns */
  for (i = 0; i < 8; i++)
  {
     ...
     c0 = MSCALE ((c7d16 * a0) - (c1d16 * a3));
     c1 = MSCALE ((c3d16 * a2) - (c5d16 * a1));
     c2 = MSCALE ((c3d16 * a1) + (c5d16 * a2));
     c3 = MSCALE ((c1d16 * a0) + (c7d16 * a3));
     ...
  }
  /* Loop over rows */
  ...
}
```

Source: The author

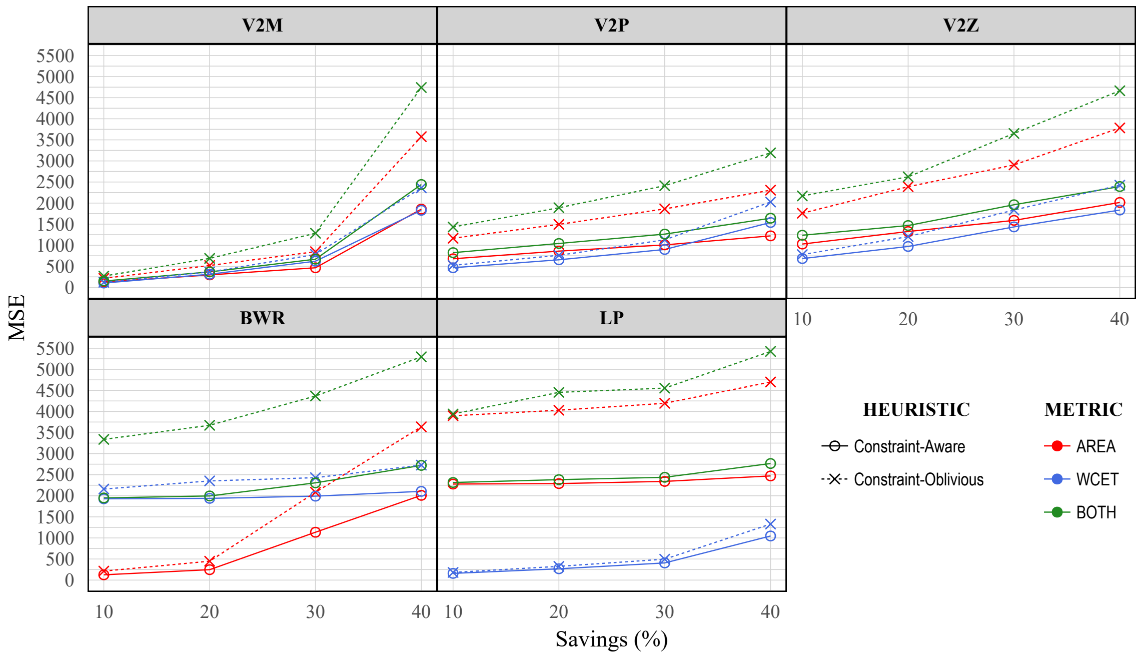
thus the use of dedicated resources like DSPs can be explored more widely. As shown in the code excerpt in Figure 5.2, the ADPCM encoder performs multiply-accumulate operations inside a loop to implement the quadrature mirror filters for the input samples. Similarly, as shown in Figure 5.3, the JPEG decoder uses many multiplications inside loops to perform the inverse discrete cosine transform, i.e., such operations can be easily parallelized and optimized using DSPs. As will be seen in Chapter 6, heavy use of heterogeneous resources makes constraint-oblivious approaches inefficient in meeting usage constraints of specific resources, as it is not able to focus on optimizing the operations using them.

6 CONSTRAINT-AWARE HEURISTIC RESULTS

This chapter presents experimental results for each benchmark defined in Table 5.1 to show that different AC techniques may provide diverse trade-offs depending on the application at hand, the optimization targets, and the stringency of the imposed constraints. For that purpose, each design was approximated to meet resources and WCET constraints separately and in conjunction. Resource constraints were defined to save a fraction of the resources used by the exact implementation to simulate varied target areas, ranging from 10% to 40% in steps of 10%. For the sake of comparison, we have defined the same proportional reduction for each resource type (LUTs, REGs, and DSPs). Likewise, WCET targets were defined considering this same range of savings over the WCET of exact designs. The objective is to evaluate the quality of the approximate designs generated by the proposed constraint-aware heuristic in terms of MSE (ADPCM, FIR, FFT, JPEG, SOBEL, 3DR, and MOTION) and percentage accuracy (DIGIT) as more stringent constraints are imposed, considering single-technique approaches using each of the implemented AC techniques (V2M, V2P, V2Z, BWR, and V2Z) separately. The number of GRASP instances considered for the experiments herein presented is four ($K = 4$). The reasoning for choosing this value for K is detailed in Section 7.2.

The advantages of a constraint-aware design methodology are also demonstrated by evaluating the improvements that can be achieved with the proposed heuristic, where approximation decisions are guided by the design metrics constraints, compared to a constraint-oblivious approach representing error-oriented methodologies concerning how they evaluate the cost-benefit of approximations, i.e., without considering the proportional costs of operations mapped to heterogeneous resources in the FPGA fabric and how close the approximate design candidates are from meeting the design objectives, as discussed in Section 3.3 for AHLS methodologies proposed so far. Figures 6.1 to 6.8 shows the results for each application considering different scenarios regarding the AC technique employed, the target optimizations (AREA, WCET, or BOTH), and the heuristic (constraint-aware or constraint-oblivious). The results regarding using different AC techniques for different target constraints and applications and the impact of constraint awareness are discussed in detail in Sections 6.1 and 6.2 respectively. Section 6.3 closes this chapter with a summary of the discussed results.

Figure 6.1 – Constraint-aware heuristic results for the ADPCM benchmark

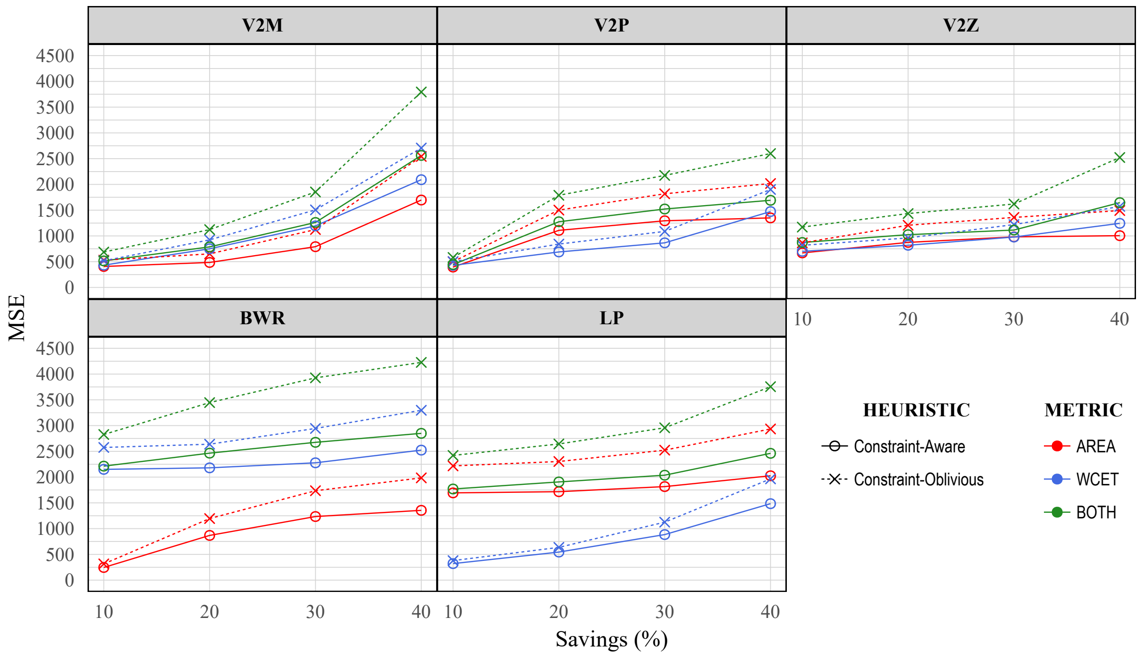


Source: The author

6.1 Techniques Performance for Different Constraints and Applications

Figures 6.1 to 6.8 show that, for each target constraint considered for optimization, there is a significant variation in error measures for both heuristics depending on the application and the AC technique employed. Analyzing the constraint-aware approach, the ADPCM application evaluated in Figure 6.1, for example, presents a better response in general with the V2M technique for area and WCET savings of up to 30%, showing a smooth quality degradation with reasonable MSE values ranging from 120 to 465. For savings of 40%, error figures increase significantly, and the V2P technique becomes a better choice. Still, the resulting MSE of 1222 is beyond what would be considered reasonable for an application with 8-bit outputs (Lee; Gerstlauer, 2017; CASTRO-GODíNEZ et al., 2020b). Such a result is expected, as stringent constraints steer the heuristic to select approximate candidates with higher relative costs that often produce more errors. This behavior depends not just on the application being approximated but also on the approximation technique employed. Although the V2M technique seems to be the best choice in a scenario of area savings of up to 30%, the BWR technique stands out for savings of 10% and 20%, mainly because it offers more fine-grained approximations that are adequate for less comprehensive optimizations. However, the need for more iterations for higher savings jeopardizes the gains achieved with BWR. An interesting observation is

Figure 6.2 – Constraint-aware heuristic results for the FIR benchmark

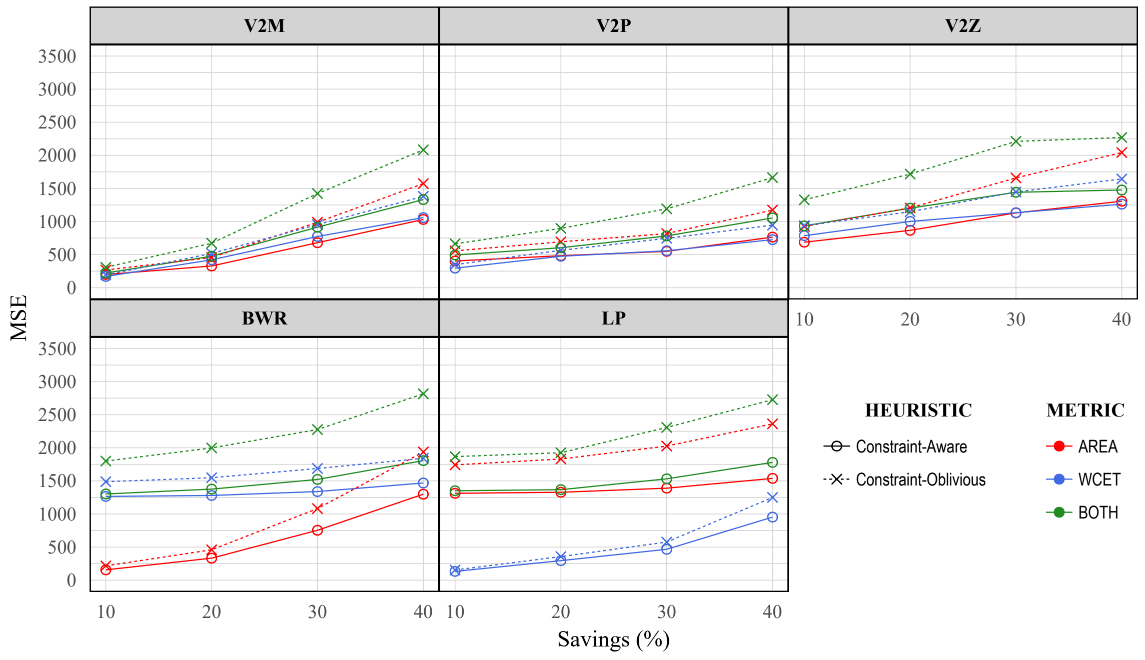


Source: The author

that the LP technique is very inefficient for area savings, showing high and almost steady error figures for all constraint scenarios. The reason is that eliminating loop iterations is unlikely to provide area savings, except if loops are entirely eliminated as a result of successive perforations, i.e., the LP technique cannot reduce area in smaller steps. A similar situation can be observed with the V2Z and V2P techniques to a minor degree, i.e., they produce higher quality degradation for savings of up to 30% if compared to the V2M and BWR techniques.

Regarding WCET savings, the quality degradation for the ADPCM benchmark presents a similar trend. However, although V2M is still a good option, especially for savings of 10%, with an MSE of 102, the LP technique becomes the best choice for savings between 20% and 40%, with MSE values of 268, 406, and 1046 respectively, as it can offer fine-grained timing optimizations. In such a scenario, the BWR technique is the one that becomes inefficient, as it reduces the cost of operations without eliminating them. Thus the number of control steps is hardly affected. When both area and resource savings are considered, it is clear that BWR and LP techniques are not good options to be considered in isolation. In these cases, the best technique for savings of up to 30% is the V2M, with an MSE ranging from 150 to 668, while the V2P is best for savings of 40%, but with an unacceptable MSE of 1640. As expected, even though the best AC technique may change when area and WCET savings are sought concurrently, such constraints produce

Figure 6.3 – Constraint-aware heuristic results for the FFT benchmark



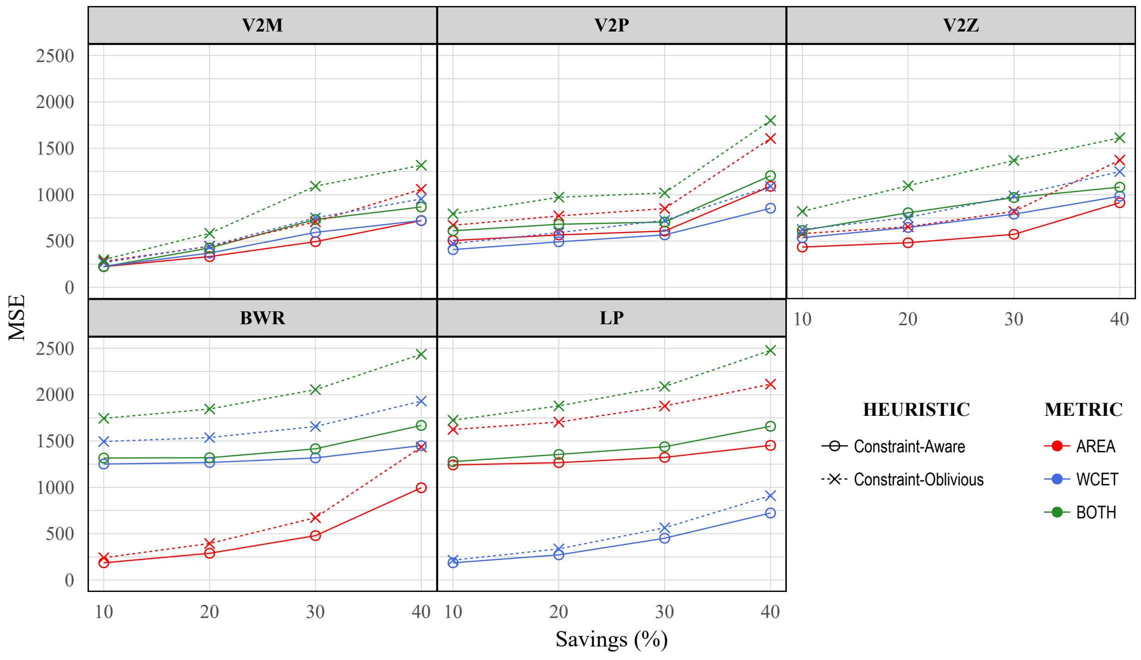
Source: the author.

higher MSE figures for all scenarios.

The results produced with the BWR and LP techniques for the ADPCM benchmark are representative across all remaining applications. Unlike other techniques that can optimize both area and WCET, these techniques are reasonable choices to optimize specific design metrics, but still to a limited extent depending on the application at hand. Thus they must be used carefully not to degrade the quality of results. For example, the results presented in Figures 6.2, 6.3, and 6.7 for the FIR, FFT, and MOTION benchmarks show that BWR is the best option only for area savings of 10%, with MSE values of 246, 156, and 86 respectively. In contrast, for the SOBEL and DIGIT benchmarks presented in Figures 6.4 and 6.8, the BWR technique is the best option for area savings up to 30%, with MSE and accuracy measures ranging from 186 and 91.8% to 479 and 80.6% respectively. Note that the exact design for DIGIT offers an accuracy of 94.2%, i.e., BWR can provide significant area reductions without reducing the percentage accuracy to unacceptable figures. Regarding the 3DR and JPEG benchmarks presented in Figures 6.5 and 6.6, BWR is able to produce lower MSE values of 65 and 81 for area savings of 10%, and 158 and 177 for area savings of 20% respectively.

Conversely, the trade-offs offered by the LP technique are steadier across applications when constraining the WCET only. It stands out for savings of up 20% for FIR, with MSE values of 320 for 10% and 544 for 20%, up to 30% for FFT, SOBEL, and MOTION,

Figure 6.4 – Constraint-aware heuristic results for the SOBEL benchmark



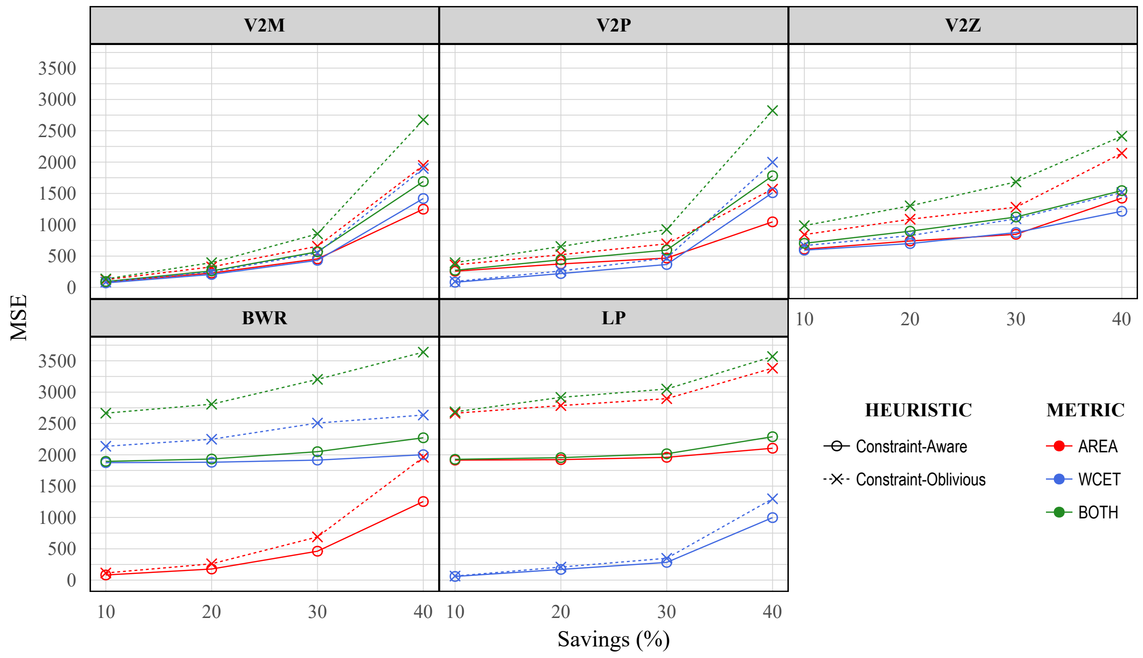
Source: The author

with MSE values ranging from 86 to 577, and in all scenarios for JPEG, 3DR, and DIGIT, with MSE values ranging from 69 and 58 to 891 and 997 for JPEG and 3DR respectively, and accuracy ranging from about 92.2% to 70.3% for DIGIT.

Analyzing the trade-offs offered by the V2M, V2P, and V2Z techniques, each of them may be a good choice for any combination of area and WCET constraints, as such techniques are commonly used to optimize area and performance. Nevertheless, as shown by the experiments, there are significant differences among applications. For example, the most aggressive technique, V2Z, is a reasonable choice only for higher savings of 40%, especially for the FIR benchmark, where it is the best option for area, WCET, or both constraints, with MSE values of 1005, 1244, and 1646 respectively. On the other hand, V2Z is never the best option for ADPCM and FFT. Except for JPEG, where V2Z is the best technique for 40% of area and both area and WCET savings, with MSE values of 831 and 1306 respectively, for the remaining applications it is the best option only for 40% of both area and WCET savings. Still, with MSE figures ranging from 1080 (SOBEL) to 1745 (MOTION) and an accuracy of about 66% for DIGIT, this technique is hardly a good choice for a single-technique methodology intended to be a general solution, as its worth only appears when stringent constraints are applied.

As for the V2M and V2P techniques, they are intermediate alternatives that present similar results for SOBEL, 3DR, MOTION, and DIGIT, especially for savings up to 30%.

Figure 6.5 – Constraint-aware heuristic results for the 3DR benchmark



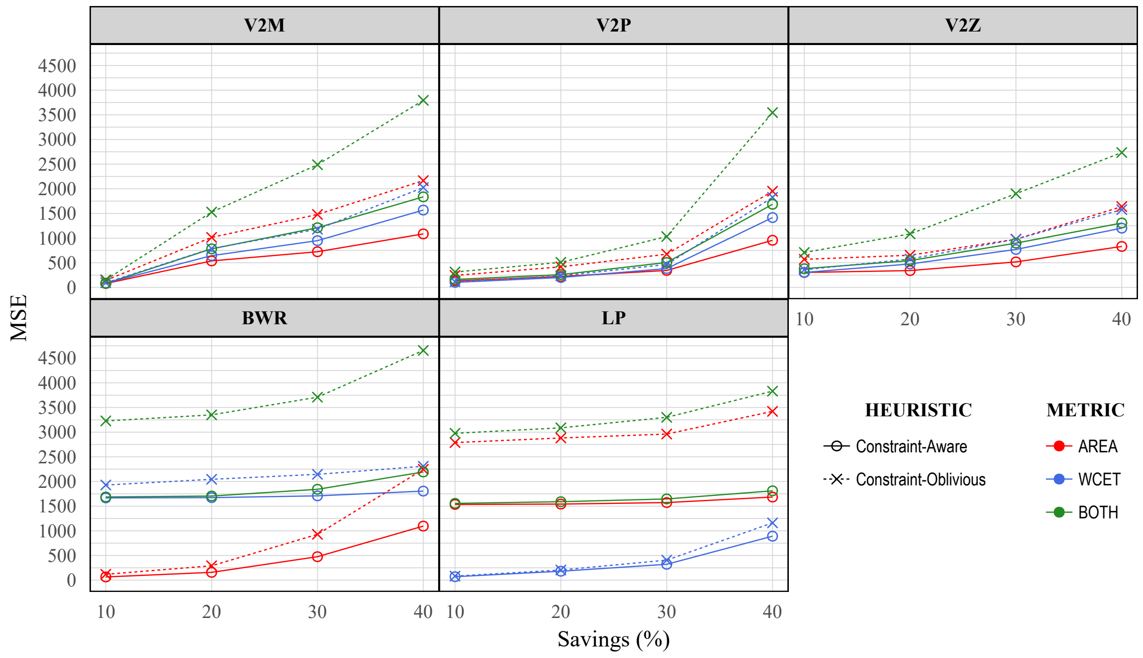
Source: The author

Specifically, V2M stands out when both area and WCET savings of 10% and 20% are sought. It is the best option in these scenarios for ADPCM, FFT, SOBEL, 3DR, MOTION, and DIGIT, with MSE values ranging from 82 (JPEG) to 223 (SOBEL) and from 262 (3DR) to 496 (MOTION) for savings of 10% and 20% respectively. The accuracy for DIGIT is about 90.2% and 82.6% for these same savings, showing that V2M can provide good trade-offs for a wide range of applications. As a general trend, the V2P technique provides better results than V2M when more stringent constraints of 30% and 40% are applied. It is the best option, for example, when both area and WCET constraints are set to 30% for FFT, SOBEL, JPEG, 3DR, and MOTION, with MSE figures ranging from 505 (JPEG) to 848 (MOTION). Note that, although V2M and V2P provided similar results for many scenarios, there are cases where the differences are significant, as for JPEG, where the V2P technique is a better option in general.

6.2 Impact of Constraint Awareness

The constraint-oblivious approach is evaluated by using the same GRASP-based heuristic described in Section 4.4, but with a modified version of the fitness function defined by Equation (4.13). More specifically, in the constraint oblivious approach, the

Figure 6.6 – Constraint-aware heuristic results for the JPEG benchmark



Source: The author

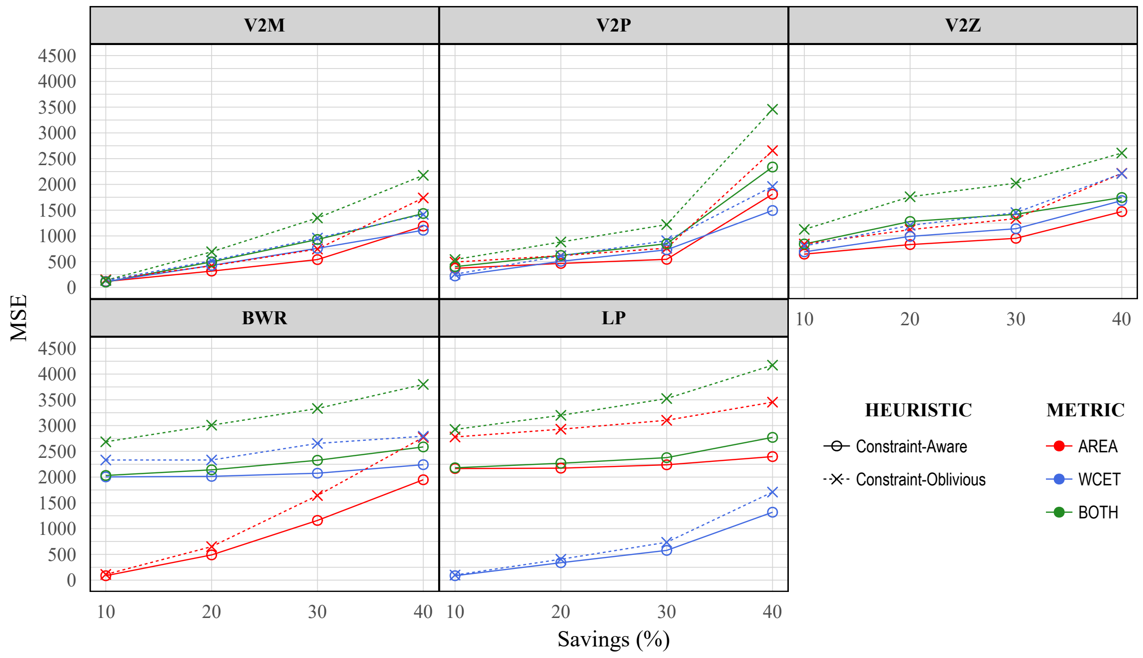
fitness of each approximate design candidate $\mathcal{D}_\gamma$ generated by applying an AC transformation $\gamma \in \Gamma$ is calculated as

$$fit_{obl}(\mathcal{D}_\gamma) = \frac{\alpha \cdot \sum_{t \in \mathcal{T}} RS_t(\mathcal{D}_\gamma) + \beta \cdot TS(\mathcal{D}_\gamma)}{\bar{E}(\mathcal{D}_\gamma, \Omega)} \tag{6.1}$$

where $RS_t(\mathcal{D}_\gamma)$ and $TS(\mathcal{D}_\gamma)$ are the resource and time savings defined by Equations (4.5) and (4.8), respectively, and $\bar{E}(\mathcal{D}_\gamma, \Omega)$ is the average error calculated over the set Ω of training inputs. For the DIGIT benchmark, we take the inverse of the error function, as it calculates the percentage accuracy, which should be maximized. The parameters α and β are set according to the target savings. For resource savings only, $\alpha = 1$ and $\beta = 0$, while for WCET savings only we have $\alpha = 0$ and $\beta = 1$. When both resource and WCET savings are sought, $\alpha = \beta = 1$. Although design constraints are disregarded for evaluating approximate design candidates at each RCG iteration, the constraint-oblivious heuristic still verifies if the aimed resource and time savings are met as part of the feasibility verification performed by the RGC and LS procedures and also allows a direct comparison between the resulting error of both approaches for the same optimization thresholds.

Note that the changes made in the proposed methodology are restricted to how the approximate design candidates are evaluated, i.e., the improvements achieved with the constraint-aware heuristic are isolated from other variables that may affect the comparison

Figure 6.7 – Constraint-aware heuristic results for the MOTION benchmark

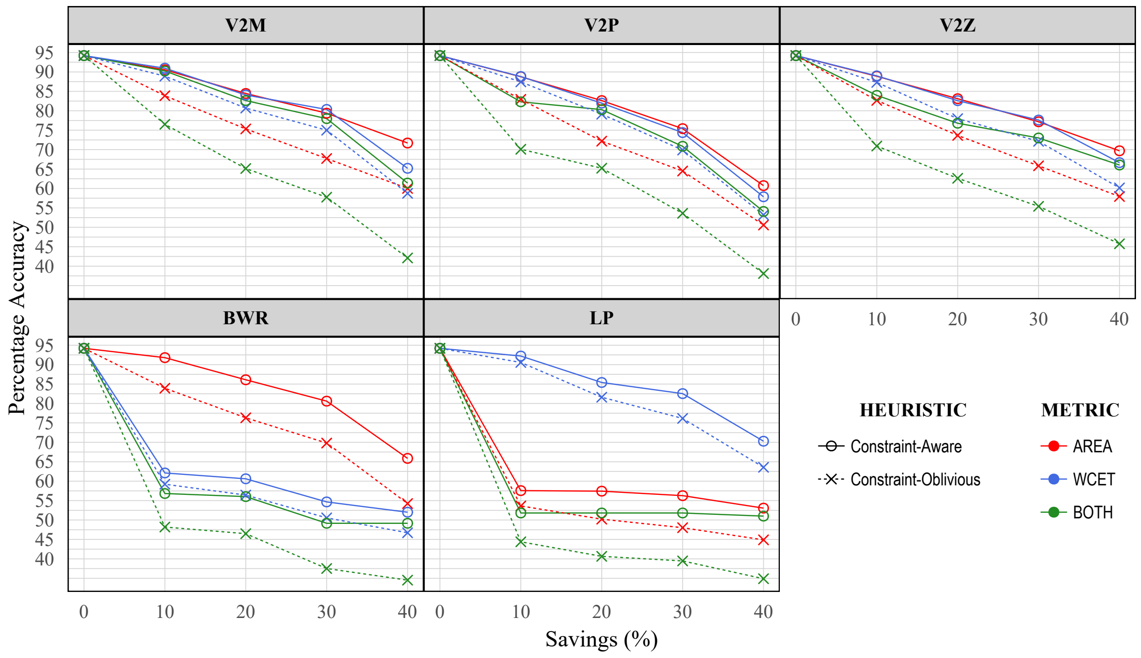


Source: The author

fairness. Therefore, although the constraint-oblivious AHLS methodologies found in the literature and discussed in Section 3.2, such as the ones proposed by Nepal et al. (2014), Xu and Schafer (2017), Lee and Gerstlauer (2017), Nepal et al. (2019), Castro-Godínez et al. (2020b), use different heuristics, tools, and benchmarks to evaluate area and performance optimizations, the results herein presented are representative to demonstrate the benefits of a constraint-aware approach, as the fitness function defined in Equation (6.1) reflects the methods used in such works to characterize approximate design candidates.

Analyzing the results shown in Figures 6.1 to 6.8, it can be observed that the constraint-oblivious results, indicated by dashed lines, always display higher error when compared to their resource-aware counterparts, indicated by continuous lines. The benefits of the former over the latter are significant, decreasing the MSE and increasing the accuracy in all scenarios, ranging from 9.54% (10% of WCET savings for 3DR) to 52.23% (40% of both area and WCET savings for JPEG), with an average of 29%, and from 1.65% to 20.28% (10% and 40% of WCET and both area and WCET savings respectively for DIGIT), with an average of 10.87%. Note that, for DIGIT, the accuracy improvements are expressed in absolute values. Moreover, the differences tend to grow as more stringent constraints are applied when considering each constraint scenario separately (AREA, WCET, or BOTH). For example, the SOBEL benchmark showed MSE reductions ranging from 22.81% to 31.93% for area savings only, from 13.28% to 24.40% for WCET sav-

Figure 6.8 – Constraint-aware heuristic results for the DIGIT benchmark



Source: The author

ings only, and from 25.36% to 33% for both savings. With some variations, this pattern is observed for all evaluated applications.

Another important observation is that the differences are more significant when area savings are sought. The reason is that there is more than one constraint involved for area (LUTs, REGs, and DSPs), which allows the constraint-aware heuristic to focus on approximation choices according to the proportional costs of each operation on the target FPGA regarding the available resources. As the constraint-oblivious approach is not guided towards the resource types that still need savings according to the specified constraints, the RGC phase of the GRASP-based heuristic typically needs more iterations until all constraints are met. In contrast, there is just one constraint for timing, and thus the improvements achieved with a constraint-aware heuristic tend to be less aggressive. Nevertheless, as the constraint-oblivious approach is not steered to select approximate design candidates with adequate trade-offs according to the time that still needs to be saved at each RGC iteration, it typically performs more approximations than necessary, increasing the error.

6.3 Summary of Results

Table 6.1 summarizes the main results herein presented, i.e., the best AC technique (ACT), the resulting MSE or percentage accuracy, and the improvement of the constraint-aware heuristic over the constraint-oblivious approach for all constraint scenarios. As can be observed, the fine-grained techniques BWR and LP tend to provide better results when exploited for optimizing specific design metrics. When resources must be saved, for example, BWR can smooth the quality degradation, especially for less stringent constraints of up to 20%, by enabling smaller approximation steps, i.e., the heuristic can make better decisions when evaluating operations for fitness due to a broader design space. Beyond that threshold, the more aggressive techniques V2M, V2P, and V2Z become better options depending on the considered application. Conversely, for WCET savings, LP is clearly the best option. It provides the best results in most cases, even when considering more stringent constraints of up to 30% or even 40% for some applications. However, when both resource and WCET savings are specified, the BWR and LP techniques become the worst choices, as such techniques are very inefficient for WCET and resource savings, respectively. The V2M, V2P, and V2Z techniques become better options in these cases, where V2M and V2P provide better results for constraints of up to 30%, while V2Z is a good option for stringent constraints of 40%.

Regarding the comparison between the constraint-aware and the constraint-oblivious approaches, the former always provides the best results, with an average improvement of about 29%, ranging from 9.54% to 52.23% across all applications and constraint scenarios. Moreover, for all applications, such an approach leads to greater improvements when more constraints are involved. For area savings, which comprises three different constraints (LUTs, REGs, and DSPs), and both area and WCET savings, it was able to reduce the MSE, on average, by about 33.6% and 35.8%, respectively. As can be observed, in these scenarios the constraint-aware approach was especially beneficial for the ADPCM and JPEG applications, with average MSE reductions of 45.65% and 48.48%, respectively. In contrast, for only WCET savings, an average reduction of 18% was observed, which still is a notable improvement as other design metrics are not jeopardized.

The conclusion from the obtained results is that the best AC technique varies according to the target application, the target optimization, and the constraints stringency. Therefore, a general methodology to explore the design space of approximate designs should be aware of the specific optimizations sought by designers by allowing the speci-

Table 6.1 – Summary of results for the constraint-aware GRASP-based heuristic

Benchmark	Result	Design Metrics Constraints											
		AREA Savings (%)				WCET Savings (%)				AREA and WCET Savings (%)			
		10	20	30	40	10	20	30	40	10	20	30	40
ADPCM	Best ACT	BWR	BWR	V2M	V2P	V2M	LP	LP	LP	V2M	V2M	V2M	V2P
	MSE	125	247	465	1222	102	268	406	1046	150	366	668	1640
	Improv. (%)	41.28	44.90	45.09	47.01	14.63	17.73	17.96	21.27	43.74	46.62	47.95	48.59
FIR	Best ACT	BWR	V2M	V2M	V2Z	LP	LP	V2P	V2Z	V2P	V2M	V2M	V2Z
	MSE	246	485	791	1005	320	544	865	1244	443	787	1261	1646
	Improv. (%)	21.97	25.57	29.61	32.67	15.40	14.32	20.41	20.86	23.69	29.88	31.95	34.74
FFT	Best ACT	BWR	V2M	V2P	V2P	LP	LP	LP	V2P	V2M	V2M	V2P	V2P
	MSE	156	328	549	764	132	295	466	725	222	475	782	1054
	Improv. (%)	27.85	28.17	32.87	34.87	13.74	17.22	19.01	23.14	27.88	29.21	34.32	36.61
SOBEL	Best ACT	BWR	BWR	BWR	V2M	LP	LP	LP	V2M	V2M	V2M	V2P	V2Z
	MSE	186	289	479	720	186	272	451	720	223	425	704	1080
	Improv. (%)	22.81	26.71	28.83	31.93	13.28	19.16	19.97	24.40	25.36	27.15	30.81	33.00
JPEG	Best ACT	BWR	BWR	V2P	V2Z	LP	LP	LP	LP	V2M	V2P	V2P	V2Z
	MSE	65	158	345	831	69	180	321	891	82	259	505	1306
	Improv. (%)	45.02	45.73	48.85	49.30	14.32	12.13	20.85	23.24	46.74	48.97	50.99	52.23
3DR	Best ACT	BWR	BWR	V2M	V2P	LP	LP	LP	LP	V2M	V2M	V2P	V2Z
	MSE	81	177	454	1046	58	170	282	997	93	262	596	1544
	Improv. (%)	28.14	31.89	30.75	33.29	9.54	18.17	18.79	23.17	30.84	33.57	35.48	35.98
MOTION	Best ACT	BWR	V2M	V2M	V2M	LP	LP	LP	V2M	V2M	V2M	V2P	V2Z
	MSE	86	317	541	1190	86	338	577	1110	103	496	848	1745
	Improv. (%)	22.61	24.48	27.41	31.49	12.88	16.55	21.38	21.61	25.12	28.19	30.47	33.09
DIGIT	Best ACT	BWR	BWR	BWR	V2M	LP	LP	LP	LP	V2M	V2M	V2M	V2Z
	Accuracy (%)	91.80	86.10	80.60	71.70	92.21	85.41	82.54	70.29	90.24	82.63	77.98	66.04
	Improv.	7.88	9.79	10.79	11.69	1.65	3.80	6.39	6.71	13.72	17.49	20.23	20.28

Source: The author

fication of multiple design constraints. Such a conclusion is supported by the substantial gains achieved with the constraint-aware approach over the constraint-oblivious one. Although improvements may vary across applications, as shown in Table 6.1, the constraint-aware approach is the best option for any evaluated scenario, with increasing gains as more design metrics are considered and more stringent are the target constraints, showing that such an approach is a promising option to optimize results, especially when optimizations in multiple design metrics are required. Moreover, as the best technique for different scenarios varies greatly, a general-purpose AHLS methodology should also be able to explore multiple techniques in an automated way to relieve designers from the task of choosing the best technique for each application and design objective, which is a complex and error-prone decision that depends on multiple factors.

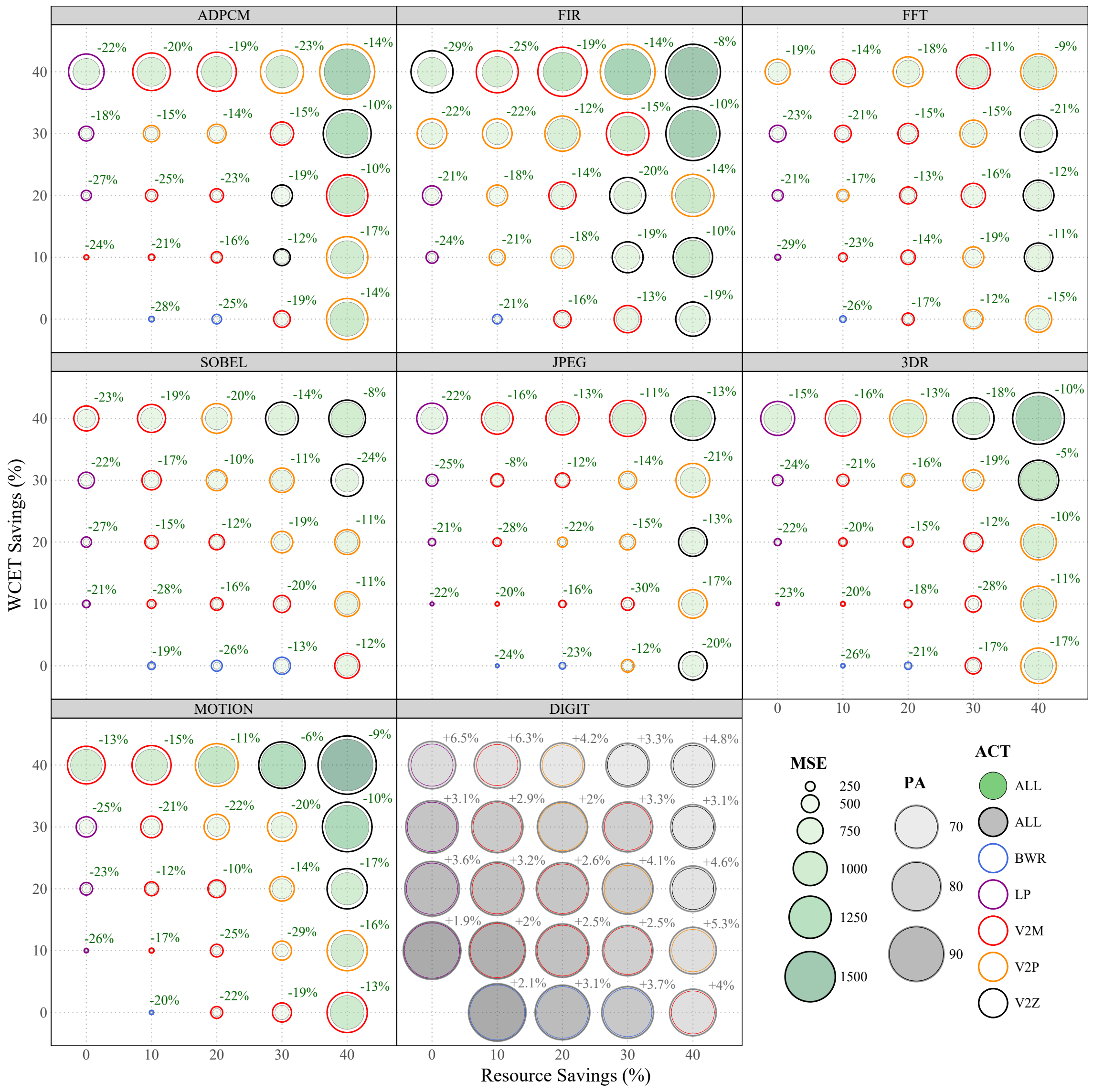
7 MULTI-TECHNIQUE HEURISTIC RESULTS

This chapter shows the experimental results for each application defined in Table 5.1 (ADPCM, FIR, FFT, SOBEL, JPEG, 3DR, MOTION, and DIGIT) regarding a comparison between our constraint-aware multi-technique approach and the best single technique, as discussed in Chapter 6, for different combinations of resources and WCET constraints. Note that the constraint-oblivious approach discussed in Section 6.2 is not considered for comparison in this chapter, as it displayed worst results for all scenarios. We used the same experimental setup for the experiments presented in Chapter 6, i.e., each design was approximated to save a fraction of the resources used by the exact implementation to simulate varied target areas, ranging from 10% to 40% in steps of 10%, with the same proportional reduction for each resource type (LUTs, REGs, and DSPs). Likewise, WCET targets were defined considering this same range of savings over the WCET of exact designs. However, in this chapter we consider a broader scenario, with both area and WCET constraints used for disproportional savings as well. The objective is to evaluate the quality of the approximate designs generated by our heuristic as more stringent constraints are imposed and to verify the advantages of using a constraint-aware composition of different AC techniques over a single-technique approach. The results regarding the multi-technique heuristic, the error and runtime variation for different values of K (number of GRASP instances), and the contribution of each individual technique for achieving the obtained results are discussed in Sections 7.1, 7.2, and 7.3 respectively. Section 7.4 closes this chapter with a summary of the discussed results.

7.1 Comparing the Single-Technique and Multi-Technique Approaches

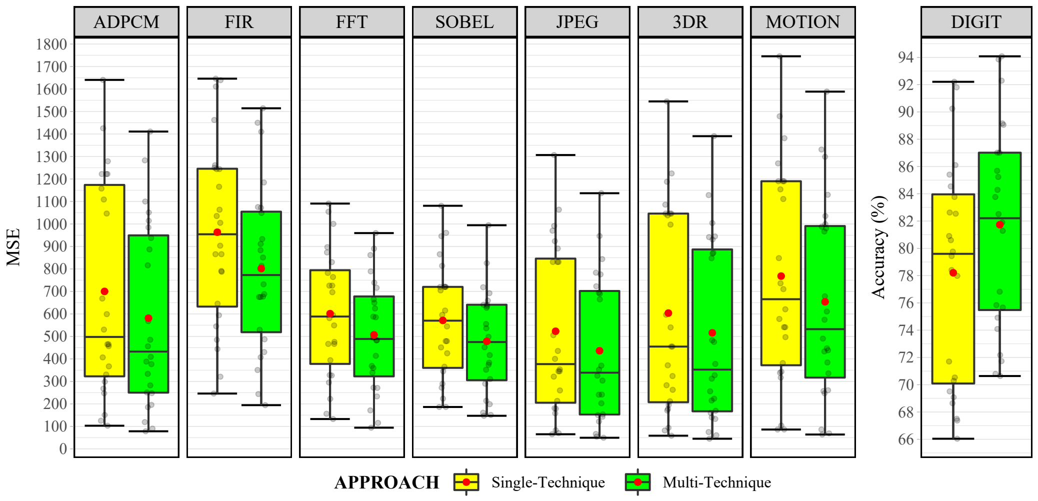
Figure 7.1 shows the improvements achieved with the multi-technique approach for each application, where the AC techniques made available (BWR, LP, V2M, V2P, and V2Z) are exploited in conjunction, over the single-technique approach that provided the best result, as presented in Table 6.1. The error measures in Figure 7.1 are proportional to the circles' radius. The circles filled in green (MSE) and grey (PA) represent the proposed multi-technique approach (ALL), with which the heuristic can choose any available AC transformation across the GRASP RGC iterations, as discussed in Section 4.4.1. In contrast, the empty circles with circumferences in varied colors represent the single AC technique (BWR, LP, V2M, V2P, or V2Z) that provided the best result for each combina-

Figure 7.1 – Comparison between the multi-technique and the best single-technique approaches



Source: The author

Figure 7.2 – Error variation for each application considering all constraint scenarios



Source: The author

tion of constraints, i.e., with a heuristic that uses a single technique for all RGC iterations. Additionally, the percentage gain (relative for MSE and absolute for PA) of the multi-technique approach over the best single technique is highlighted next to the circles.

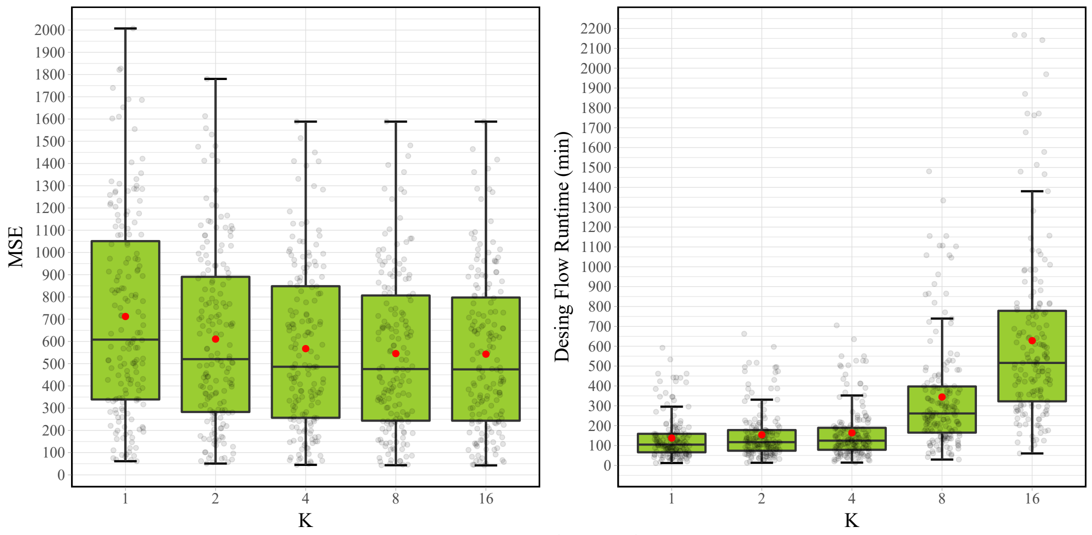
The results presented in Figure 7.1 indicate sensible reductions in MSE, ranging from 5% for the 3DR application with resource and WCET constraints of 40% and 30%, respectively, to 30% for the JPEG application with resource and WCET constraints of 30% and 20% respectively. Regarding the DIGIT benchmark, the absolute gains in accuracy varies from 1.9% to 6.5% for WCET savings of 10% and 40% respectively. Moreover, note that the multi-technique approach outperforms single techniques for all scenarios. Still, there is a slight tendency of higher improvements where BWR and LP offer the best single-technique results. Although such techniques may provide good results when resource and WCET savings are sought separately, as demonstrated in Section 6.1, they seem to work better when combined with other techniques, with improvements ranging from 21% to 29% for LP and 19% to 28% for BWR. Figure 7.2 provides a more comprehensive view of the attained improvements with the multi-technique approach for each application by showing the MSE and accuracy variations across all evaluated constraint scenarios. As can be observed, the decrease in average MSE (red dots) ranges from 17.13% (FFT) to 18.75% (ADPCM), with an average of about 17.7%. As for DIGIT, there is an absolute increase of 3.5% in average accuracy.

7.2 Heuristic Running Time and Error Variation for Different Values of K

The results shown in Figures 7.1 and 7.2 were obtained with $K = 4$, i.e., we used four processor threads to execute four GRASP instance in parallel. Nevertheless, experiments were run for different values of K to show how the heuristic running time and results may vary according to the number of performed GRASP iterations. Recall from Section 4.4 that our GRASP-based heuristic selects the best feasible design (meets constraints with minimum error) among all locally optimal feasible designs generated by each GRASP iteration, i.e., the more iterations are used, the more likely the heuristic is to converge to the globally optimal solution, considering the design space provided by the available AC transformations. However, depending on the level of parallelism allowed by the available setup, the drawback is increasing the design flow running time without guaranteeing improved results.

Figure 7.3 shows the MSE and running time results over all applications (ex-

Figure 7.3 – Error and running time variation over all scenarios for different values of K



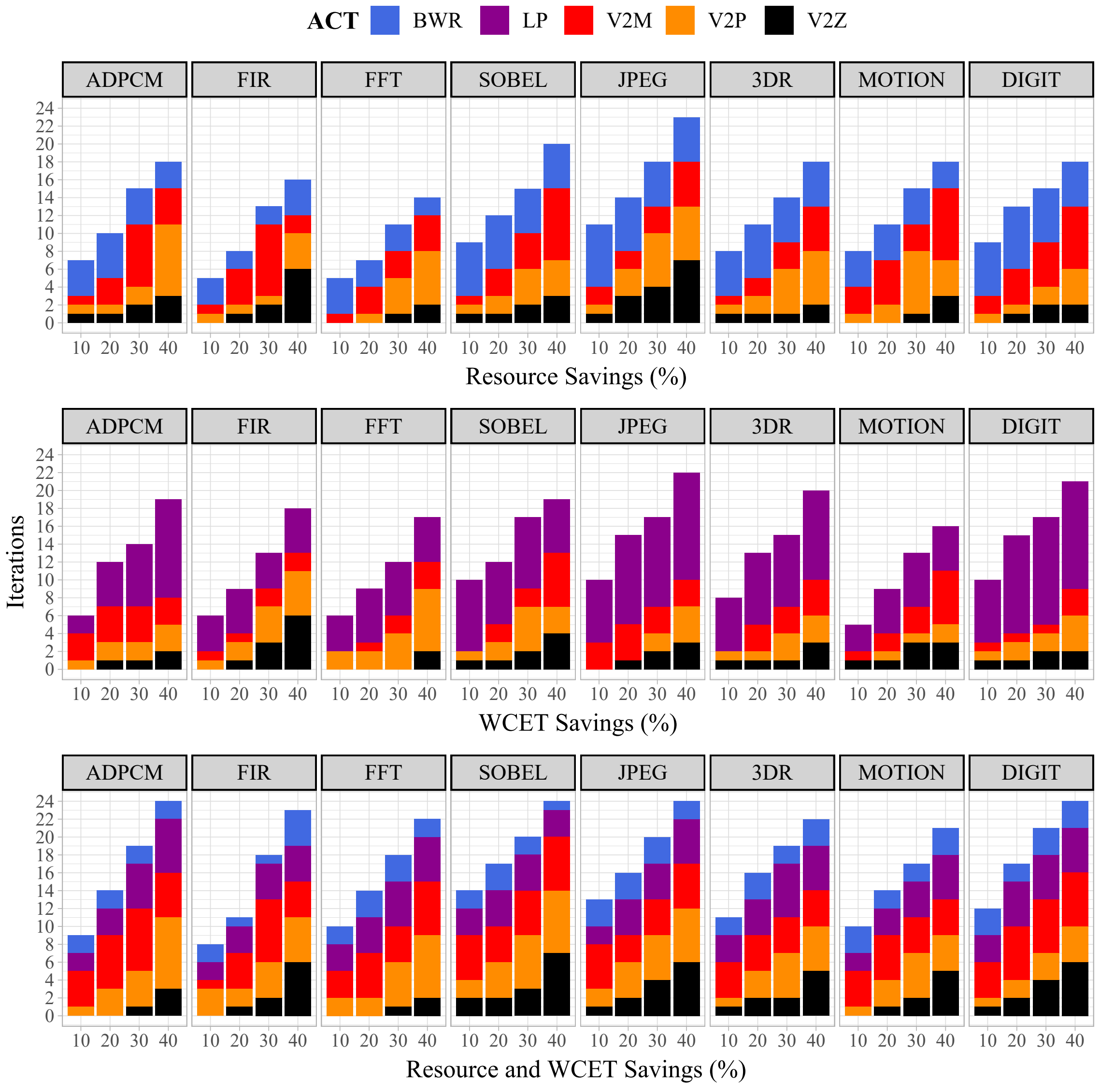
Source: The author

cept DIGIT, which uses a different error metric) and constraint combinations for $K \in \{1, 2, 4, 8, 16\}$. As can be seen, there are respective reductions of about 15%, 22%, 24%, and 25% in average MSE (red dots) for $K = 2$ to $K = 16$ when compared to using a single instance ($K = 1$). Conversely, the average design flow running time increased by 7%, 18%, 242%, and 450%, respectively. Therefore, $K = 4$ is a reasonable choice, as the small reductions in MSE for $K = 8$ and $K = 16$, relative to $K = 4$, do not justify the substantial running time increase, especially for large designs. The almost steady running time for $K \leq 4$ was expected, as our experimental setup uses a quad-core processor with four threads. Nevertheless, designers with more cores or threads available are likely to slightly improve results while managing similar running times by defining greater values for K. However, at least for the set of applications and AC techniques herein considered, there is little room for error minimization with $K > 4$.

7.3 Techniques Contribution for Different Constraints and Applications

Figure 7.4 presents which AC techniques were chosen across the RGC iterations, showing that a heuristic able to select an appropriate combination of techniques is crucial to optimize results. In Figure 7.4, we consider resources and WCET constraints in isolation and simultaneously (same proportional savings) to highlight that the techniques chosen highly depend on the constraints. The BWR technique, for example, is only considered when resource savings are involved, especially if less stringent constraints of 10% and 20% are imposed, and the primary goal is reducing area, where BWR is selected for

Figure 7.4 – Contribution of each AC technique over the total number of RGC iterations



Source: The author

most iterations. In such scenarios, fine-grained AC techniques may help reach the design goals with lower error through more focused transformations. Regardless of the application, if more stringent constraints of 30% or 40% are specified, BWR is no longer the most chosen technique, even though it is still used. In contrast, the LP technique is never chosen for resource savings, as it mainly impacts execution times. However, if WCET reductions are the primary goal, the LP technique dominates in most scenarios, while BWR is no longer selected. Moreover, note that the LP technique usage seems more regular for different WCET savings than BWR for different resource savings.

If both resource and WCET reductions are needed, the heuristic automatically balances using a varied combination of techniques. As more stringent constraints are imposed for all applications, more aggressive techniques like V2M, V2P, and V2Z come into play, with V2Z being used more extensively only for savings of 30% or 40% for most applications. Although such techniques may introduce more errors, their usage may produce more comprehensive optimizations within a single iteration, thus reducing the distance to constraint more quickly as the RGC phase converges to provide a feasible design. As a result, the number of iterations needed to meet constraints does not necessarily increase proportionally to the resources or WCET to save, varying from 4 iterations for FIR with 10% of resources savings to 24 iterations for ADPCM, SOBEL, JPEG, and DIGIT with 40% of resources and WCET savings. Moreover, the most used technique for each scenario matches the best single technique presented in Figure 7.1. However, it should be noted that a different mix of techniques was used for each scenario across all heuristic iterations, indicating that no single approximation technique dominates the design solution. Therefore, combining multiple techniques applied over different applications contributes to broader the design space with improved design choices.

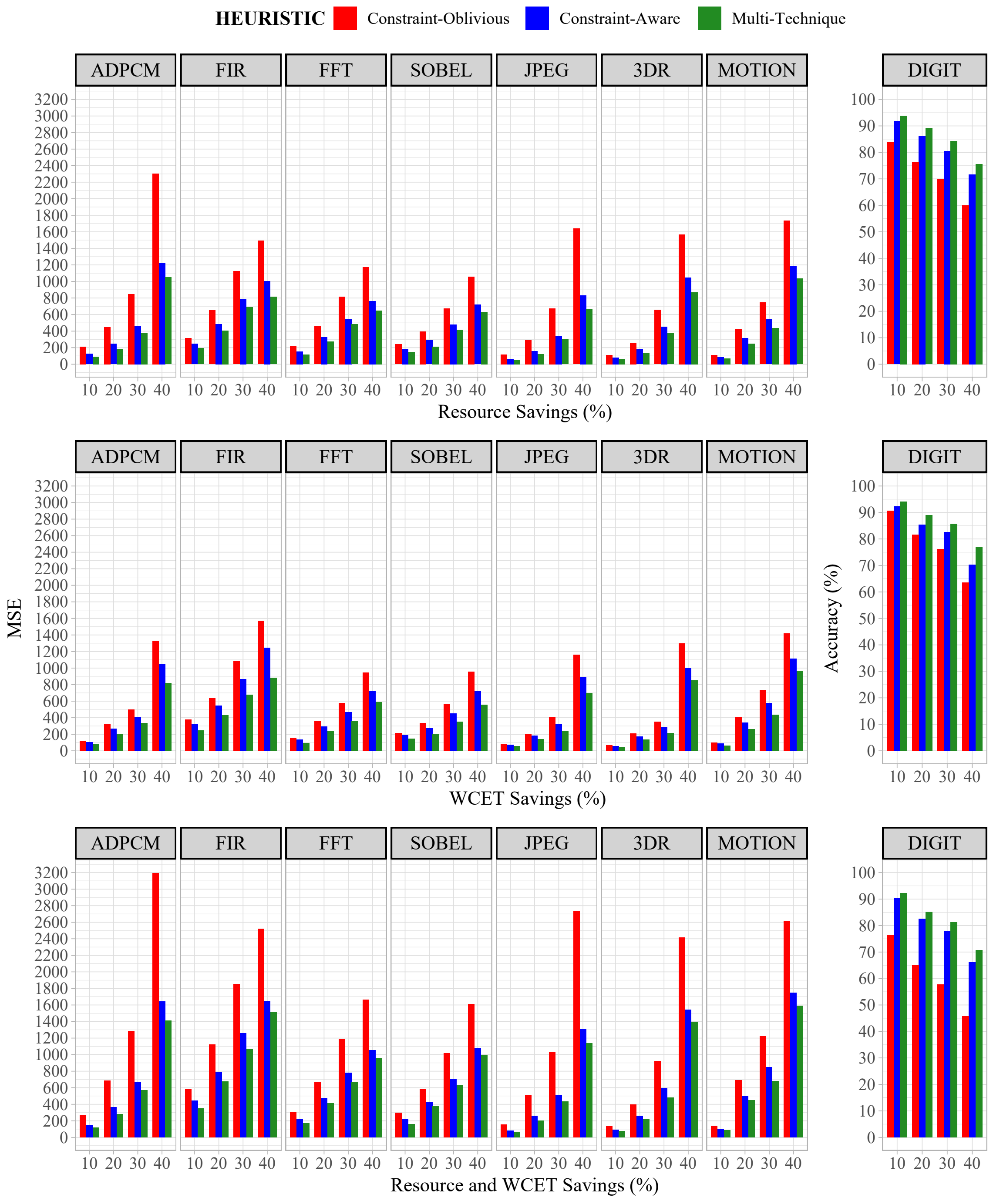
7.4 Summary of Results

The results presented in this chapter demonstrate that, besides adopting a constraint-aware methodology to steer the DSE towards meeting specific target optimizations, as shown in Chapter 6, an automated exploration of multiple techniques may provide even more improvements. As different AC techniques provide different gains depending on the constraints stringency for each design metric sought for optimization, the employed heuristics should be able to focus on exploiting the best combination of AC techniques for each specific scenario on a per-application basis. Although designers can manage the design space by defining the AC techniques to be exploited, it becomes clear that just selecting the best technique or using AHLS tools that implement such techniques for a given scenario is not enough to optimize results. In that direction, providing a systematic way for seamlessly integrating new techniques into the standard AHLS flow is crucial to offer an automated exploration of unique trade-offs for the target constraints and applications. On average, the multi-technique approach attained an MSE reduction of 17.7%, varying from 5% to 30%, and an absolute increase in accuracy of 3.5%, from 1.9% to 6.5%.

Figure 7.5 summarizes the results obtained for each application and target constraint by comparing the constraint-oblivious and constraint-aware heuristics, considering the best single-technique for each scenario, and the constraint-aware multi-technique methodology presented in this chapter. On average, reductions of 29% in MSE, ranging from 9.54% to 52.23% depending on the target application and constraint, were obtained with the single-technique constraint-aware approach compared to the constraint-oblivious one. For the DIGIT application, an absolute increase in accuracy ranging from 1.65% to 20.28% is achieved, with an average of 10.87%. When comparing the single-technique constraint-aware approach with the multi-technique one, average decreases of 17.7% in MSE were achieved, varying from 5% to 30%, while an absolute increase between 1.9% and 6.5% was attained for DIGIT, with an average of 3.5%. Finally, when adopting the constraint-aware multi-technique approach over the constraint-oblivious single-technique one, reductions of about 43% on average can be attained in MSE, ranging from 30% to 60%. As for DIGIT, an absolute increase in accuracy ranging from 3.52% to 25.03% is achieved, with an average of 14.26%. Note this is the compound effect of constraint awareness and the joint use of multiple techniques. As a general trend, more significant improvements can be observed as more stringent constraints are imposed.

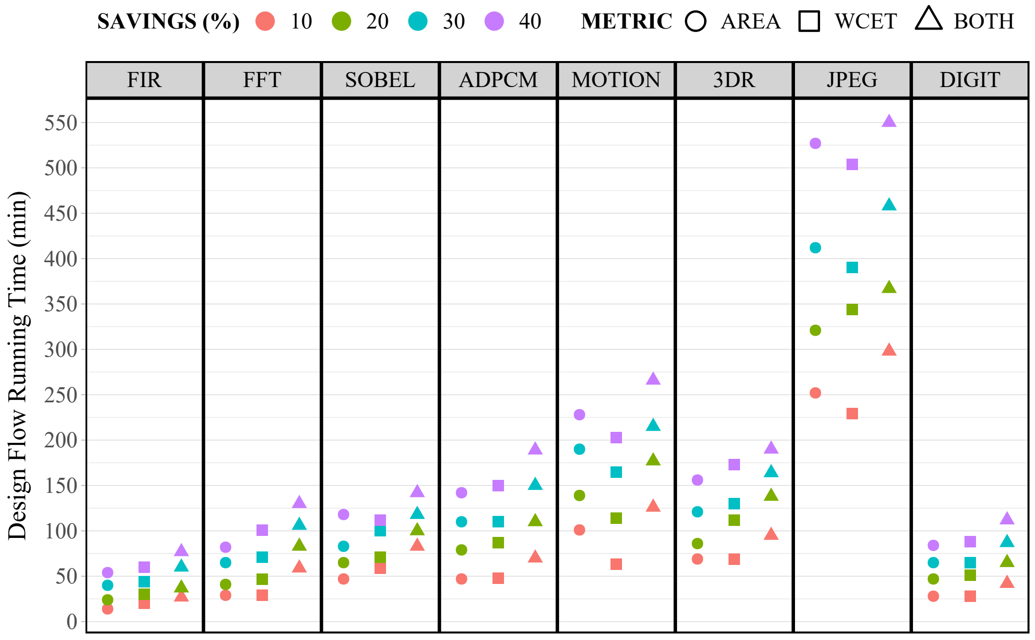
Moreover, as discussed in Chapter 6, greater improvements are displayed when resource savings are involved, especially for the ADPCM and JPEG applications, where average reductions of about 50% were obtained. These two applications are notably different from the others for using many more DSPs, as shown in Table 5.1. As the constraint-oblivious approach is not able to focus on the operation types that use specific resources that still need to be saved, such as DSPs, it typically approximates more operations until all resource constraints are met, i.e., even though the constraint-aware approach considerably reduces the error metrics in all scenarios, it is especially suitable for designs with heavier use of heterogeneous resources. Therefore, as demonstrated by experiments, a constraint-aware AHLS design methodology able to automatically ex-

Figure 7.5 – Summary of results for constraint-oblivious, constraint-aware, and multi-technique design methodologies



Source: The author

Figure 7.6 – AHLS running time for different applications, constraints, and target design metrics



Source: The author

plore multiple AC techniques is a promising option to optimize approximate hardware synthesized from high-level design specifications, especially when dealing with multiple optimization targets.

Regarding the scalability of the proposed methodology, Figure 7.6 shows the AHLS running time for each application, considering area, WCET, and both design metrics for savings ranging from 10% to 40%. Note that the deployed heuristic scales very well concerning the constraint stringency for all applications and design metrics, mainly because it is able to apply more aggressive approximations when more savings are needed. Consequently, the number of RGC iterations to meet all constraints does not necessarily increase proportionally to the constraint stringency, as discussed in Section 7.3. The Pearson correlation coefficient between the size of each application and the average AHLS running time, as detailed in Table 5.1, is 0.28, i.e., even though the correlation is positive, the relationship is weak. On the other hand, if we consider only the applications' average execution time, the correlation coefficient is 0.68, i.e., there is a tendency for high execution times going with high AHLS running times. Note that the average execution time of each application is calculated by executing the binary code generated with the LLVM's x86 backend for all provided input vectors. This result shows that the time needed to evaluate the error measure for each approximate design candidate, which was done through Monte Carlo simulations, has more impact on the heuristic running time than the number of design candidates to evaluate, which increases with circuit size.

Nevertheless, as discussed in Section 4.4.4, the heuristic's scalability is better explained by considering both the circuit size and the average execution time of each application. In that case, the correlation coefficient is 0.94, showing that the heuristic running time scales with the product between the circuit's size and average execution time. For example, even though the DIGIT application is the largest one, its average execution time is also the lowest among all considered applications. As a result, the heuristic running time for DIGIT is relatively low. On the other hand, the JPEG application has the higher AHLS running times precisely because it is not just the second largest design but also by far the slowest one.

8 CONCLUSION

This thesis proposed that a multi-technique AHLS methodology able to automatically explore multiple approximation techniques to meet different design constraints, separately or simultaneously, with minimum error, would provide better results than unconstrained or error-constrained methodologies. More specifically, by prioritizing approximations with better trade-offs in terms of the estimated distance to constraints and introduced error at each iteration, the proposed GRASP-based heuristic exploits the combined effects of diverse approximation techniques towards meeting multiple constraints, focusing on minimizing error metrics instead of adhering to a user-provided error threshold, which typically results in sub-optimal designs and many attempts to attain the desired trade-offs for each application at hand.

In that direction, the effectiveness of using software-level approximations to generate approximate hardware through a constraint-aware AHLS approach was first demonstrated by allowing the specification of resource constraints for FPGAs regarding the heterogeneous resources offered by FPGA architectures. Rather than evaluating area as a single measure, such an approach considers the fact that specific operations may be mapped to different resources in the FPGA fabric (e.g., LUTs, REGs, DSPs, and BRAMs) and thus their proportional costs regarding the target constraints can be leveraged to optimize approximation decisions. These conclusions are based on the resource-constrained results discussed in Chapter 6 and were initially published in (Leipnitz; Nazar, 2019a). Following a similar approach, real-time constraints were studied to optimize the generation of approximate designs meeting WCET requirements. In this case, WCET analysis is leveraged to steer the AHLS design space exploration towards focusing approximations on wort-case execution paths, producing better trade-offs than a constraint-oblivious approach would provide. Similarly, these conclusions are based on the performance-constrained results discussed in Chapter 6 and were initially published in (LEIPNITZ; NAZAR, 2019b).

Although these results support the first contribution of this thesis, i.e., that a constraint-aware AHLS design methodology where the DSE is directly steered by constraints on the design metrics of interest can optimize results, their scope is limited as they focus on optimizing a single design metric (resources or WCET) through a single-technique heuristic. However, a multi-technique approach allowing the specification of multiple constraints can open the door for the exploration of multiple optimizations while providing significant reductions in error measures. In that direction, the experimental results presented in Chapter 6 also support the second contribution of this thesis, i.e., that different AC techniques offer different trade-offs depending on the application at hand and the target optimizations. Thus, a constraint-aware approach should also explore different AC techniques to optimize results. Experiments realized over a set of eight image, video, signal processing, and machine learning benchmarks showed that output error can be reduced, compared to a constraint-oblivious approach where the evaluation of approximation options targeting the aimed optimizations disregards the specified constraints. The proposed methodology attained, on average, a reduction of about 25% in MSE for the ADPCM, FIR, FFT, JPEG, SOBEL, 3DR, and MOTION benchmarks, ranging from 9.54% to 34.74%, and an absolute increase of 10.87% in accuracy for the DIGIT benchmark, ranging from 1.65% to 20.28%. A significant error variation was displayed depending on the considered application, the employed AC technique (BWR, LP, V2M, V2P, or V2Z), and the target constraints (resources, WCET, or both), demonstrating that defining the best technique for a given scenario is crucial to optimize results, but it is also a challenging task that must be carried carefully by experienced designers, as it depends on multiple factors.

In a complementary way, Chapter 7 supports the third contribution of this thesis, showing that no single AC technique can outperform an adequate combination of techniques which selection is steered by constraints on one or more design metrics of interest, with all available AC techniques being used to improve results in different scenarios. When compared to the best constraint-aware single-technique approach exploiting one of the implemented techniques, the exploration of multiple techniques in conjunction can reduce the average MSE by 17.7%, ranging from 5% to 30% for the ADPCM, FIR, FFT, JPEG, SOBEL, 3DR, and MOTION benchmarks, and increase the average accuracy for DIGIT by 3.5% (absolute), ranging from 1.9% to 6.5% depending on the application and imposed constraints. Therefore, the obtained results show that a constraint-aware multi-technique methodology is a promising option to optimize the generation of approximate hardware through AHLS design flows.

8.1 Published Works

The work developed to support the constraint-aware multi-technique AHLS design methodology presented in this thesis resulted in the publication of five papers:

- LEIPNITZ, M. T.; NAZAR, G. L. High-level synthesis of resource-oriented approximate designs for fpgas. In: **Proceedings of the 56th Annual Design Automation Conference 2019**. New York, NY, USA: ACM, 2019a. (DAC '19), p. 126:1–126:6. ISBN 978-1-4503-6725-7.
- LEIPNITZ, M. T.; NAZAR, G. L. High-level synthesis of approximate designs under real-time constraints. **ACM Trans. Embed. Comput. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 18, n. 5s, oct 2019b. ISSN 1539-9087. Presented at the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES).
- LEIPNITZ, M. T.; NAZAR, G. L. High-level synthesis of throughput-optimized and energy-efficient approximate designs. In: **Proceedings of the 17th ACM International Conference on Computing Frontiers**. New York, NY, USA: Association for Computing Machinery, 2020. (CF '20), p. 221–224. ISBN 9781450379564.
- LEIPNITZ, M. T.; PERLEBERG, M. R.; PORTO, M. S.; NAZAR, G. L. Enhancing real-time motion estimation through approximate high-level synthesis. In: **2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [S.l.: s.n.], 2020. p. 30–35.
- LEIPNITZ, M. T.; NAZAR, G. L. Throughput-oriented spatio-temporal optimization in approximate high-level synthesis. In: **2020 IEEE 38th International Conference on Computer Design (ICCD)**. [S.l.: s.n.], 2020. p. 316–323.

Additionally, the following papers were published with the contribution of the author of this thesis:

- NAZAR, G. L.; KOPPER, P. H.; LEIPNITZ, M. T.; JUURLINK, B. Precep: Automatic insertion of partial redundancy based on critical error probability. **Microelectronics Reliability**, v. 126, p. 114226, 2021. ISSN 0026-2714. Proceedings of ESREF 2021, 32nd European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.
- NAZAR, G. L.; KOPPER, P. H.; LEIPNITZ, M. T.; JUURLINK, B. Lightweight dual modular redundancy through approximate computing. In: **2021 XI Brazilian Symposium on Computing Systems Engineering (SBESC)**.[S.l.: s.n.], 2021. p. 1–8.

The ideas presented in two of these works (Leipnitz; Nazar, 2019a; LEIPNITZ; NAZAR, 2019b) are directly derived from the proposed methodology, while the other five works were published as a consequence of the thesis evolving process (LEIPNITZ; NAZAR, 2020a; LEIPNITZ et al., 2020; LEIPNITZ; NAZAR, 2020b; NAZAR et al., 2021b; NAZAR et al., 2021a). Another work, titled "Constraint-Aware Multi-Technique Approximate High-Level Synthesis for FPGAs", was submitted to the IEEE Transactions on Computers as a direct result of this thesis. It is currently under revision.

8.2 Future Works

The main future works derived from the conclusions of this thesis include adding more AC techniques to the proposed design flow and investigating how other constraints and design goals may steer approximation choices in different directions. Moreover, other heuristics for evaluating the trade-offs offered by approximate design candidates may be considered to explore the design space, i.e., the scope of the proposed AHLS methodology can be expanded to explore diverse design opportunities not yet considered in the context of a constraint-aware multi-technique design flow aimed at selecting approximations toward meeting specific design objectives.

For example, energy consumption and power dissipation budgets are typical requirements faced by designers, especially in the context of embedded systems and edge computing applications. Although many works have proposed AHLS methodologies targeting energy or power optimizations (Nepal et al., 2014; Li et al., 2015; Nepal et al., 2019), none of them leverages a constraint-aware approach by allowing the specification of energy or power constraints to steer the design space exploration. However, measuring energy and power using RTL or gate-level simulations to profile the switching activity of signals is a complex challenge as it is very time-consuming, thus it is unlikely to be a viable general solution for characterizing approximation options. Therefore, to include such optimizations in the proposed design flow, it is necessary to provide a method for reasonably estimating savings quickly, as discussed in Sections 4.3.1 and 4.3.2 for resources and WCET, respectively. For example, machine learning models can be used for fast power inference, as proposed by Zhou et al. (2019). Nevertheless, even though energy savings is not the focus of this thesis, it was demonstrated in the previous works using gate-level simulations and a more limited scope (Leipnitz; Nazar, 2019a; LEIPNITZ; NAZAR, 2019b), that energy reduction is roughly proportional to the area and time savings product. Therefore, given a method for quickly estimating savings for each approximation option, other design metrics, such as energy and power, can be included as new dimensions to be evaluated by the fitness function presented in Section 4.4.3.

Moreover, the proposed constraint-aware approach is not tied to the GRASP-based heuristic detailed in Section 4.4. Although GRASP is a well-known approach for combinatorial optimization problems due to its flexibility, other metaheuristics may be explored for constraint-aware AHLS, such as Monte Carlo Tree Search (AWAIS; MOHAMMADI; PLATZNER, 2018), Non-dominated Sorting Genetic Algorithms (Nepal et al., 2019), or Tabu Search (CASTRO-GODíNEZ et al., 2020b). As long as the implemented heuristic does not constrain the design space to specific AC techniques, expanding approximations at the software or other abstraction layers is another possibility that can be explored. For example, techniques such as variable-to-variable substitutions (Xu; Schafer, 2017), memoization (Rahimi; Benini; Gupta, 2013), predictive models using linear regression, multi-layer perceptrons, or artificial neural networks to replace and mimic the execution of compute-intensive blocks (XU; SCHAFER, 2019b), can be included to broaden the design space toward optimizing results even more. Libraries of approximate components can also be exploited at the HLS abstraction level, allowing the selection of pre-characterized approximate operators and functions (CASTRO-GODíNEZ et al., 2020b). As shown in this thesis, additional techniques expand the design space and are likely to enable more efficient implementations. However, this expanded solution set warrants ever more sophisticated heuristics to be explored in a timely manner, creating a challenging superposition of future contributions.

REFERENCES

Aksoy, L.; Flores, P.; Monteiro, J. Approximation of multiple constant multiplications using minimum look-up tables on fpga. In: **2015 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2015. p. 2884–2887. ISSN 0271-4302.

ALAN, T.; HENKEL, J. Slackhammer: Logic synthesis for graceful errors under frequency scaling. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 37, n. 11, p. 2802–2811, 2018.

ANAM, M. A.; WHATMOUGH, P. N.; ANDREOPOULOS, Y. Precision-energy-throughput scaling of generic matrix multiplication and discrete convolution kernels via linear projections. In: **The 11th IEEE Symposium on Embedded Systems for Real-time Multimedia**. [S.l.: s.n.], 2013. p. 21–30.

ARMBRUST, M. et al. A view of cloud computing. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 4, p. 50–58, apr 2010. ISSN 0001-0782. Available from Internet: <https://doi.org/10.1145/1721654.1721672>.

AWAIS, M.; MOHAMMADI, H. G.; PLATZNER, M. An mcts-based framework for synthesis of approximate circuits. In: **2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)**. [S.l.: s.n.], 2018. p. 219–224.

BARBARESCHI, M.; IANNUCCI, F.; MAZZEO, A. A pruning technique for b&b based design exploration of approximate computing variants. In: **2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [S.l.: s.n.], 2016. p. 707–712.

Becher, A. et al. A lut-based approximate adder. In: **2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)**. [S.l.: s.n.], 2016. p. 27–27.

Bey Ahmed Khernache, M. et al. Hevc hardware vs software decoding: An objective energy consumption analysis and comparison. **Journal of Systems Architecture**, v. 115, p. 102004, 2021. ISSN 1383-7621. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1383762121000199>.

BORKAR, S. Thousand core chips: A technology perspective. In: **Proceedings of the 44th Annual Design Automation Conference**. New York, NY, USA: Association for Computing Machinery, 2007. (DAC '07), p. 746–749. ISBN 9781595936271. Available from Internet: <https://doi.org/10.1145/1278480.1278667>.

BROSS, B. et al. Developments in international video coding standardization after avc, with an overview of versatile video coding (vvc). **Proceedings of the IEEE**, v. 109, n. 9, p. 1463–1493, 2021.

Canis, A. et al. Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems. **ACM Trans. Embed. Comput. Syst.**, v. 13, n. 2, p. 24:1–24:27, sep. 2013.

Cardoso, J. a. M. P.; Diniz, P. C.; Weinhardt, M. Compiling for reconfigurable computing: A survey. **ACM Comput. Surv.**, v. 42, n. 4, p. 13:1–13:65, jun. 2010.

CASTRO-GODíNEZ, J. et al. Axls: A framework for approximate logic synthesis based on netlist transformations. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 68, n. 8, p. 2845–2849, 2021.

CASTRO-GODíNEZ, J. et al. Compiler-driven error analysis for designing approximate accelerators. In: **2018 Design, Automation Test in Europe Conference Exhibition (DATE)**. [S.l.: s.n.], 2018. p. 1027–1032.

CASTRO-GODíNEZ, J. et al. Approximate acceleration for cnn-based applications on iot edge devices. In: **2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)**. [S.l.: s.n.], 2020. p. 1–4.

CASTRO-GODíNEZ, J. et al. Axhls: Design space exploration and high-level synthesis of approximate accelerators using approximate functional units and analytical models. In: **2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)**. [S.l.: s.n.], 2020. p. 1–9.

Chakrapani, L. N. et al. Highly energy and performance efficient embedded computing through approximately correct arithmetic: A mathematical foundation and preliminary experimental validation. In: **Proc. Int. Conf. Compilers, Architectures and Synthesis for Embedded Systems**. [S.l.: s.n.], 2008. (CASES ’08), p. 187–196.

Chan, W. T. J. et al. Statistical analysis and modeling for error composition in approximate computation circuits. In: **IEEE 31st Int. Conf. Computer Design (ICCD)**. [S.l.: s.n.], 2013. p. 47–53. ISSN 1063-6404.

CHANDRASEKHARAN, A. et al. Approximation-aware rewriting of aigs for error tolerant applications. In: **2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2016. p. 1–8.

CHANG, I. J.; MOHAPATRA, D.; ROY, K. A priority-based 6t/8t hybrid sram architecture for aggressive voltage scaling in video applications. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 21, n. 2, p. 101–112, 2011.

CHEN, J.; RAN, X. Deep learning with edge computing: A review. **Proceedings of the IEEE**, v. 107, n. 8, p. 1655–1674, 2019.

Chippa, V. K. et al. Analysis and characterization of inherent application resilience for approximate computing. In: **2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2013. p. 1–9. ISSN 0738-100X.

CHIPPA, V. K. et al. Storm: A stochastic recognition and mining processor. In: **2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)**. [S.l.: s.n.], 2014. p. 39–44.

CHOWDHURY, P.; SCHAFER, B. C. Unlocking approximations through selective source code transformations. In: **Proceedings of the 2021 on Great Lakes Symposium on VLSI**. New York, NY, USA: Association for Computing Machinery, 2021. (GLSVLSI ’21), p. 359–364. ISBN 9781450383936. Available from Internet: <https://doi.org/10.1145/3453688.3461498>.

CISCO. **Cisco Annual Internet Report (2018–2023) White Paper**. 2020. Available from Internet: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>.

Cong, J. et al. High-level synthesis for fpgas: From prototyping to deployment. **IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.**, v. 30, n. 4, p. 473–491, April 2011.

COUSSY, P. et al. An introduction to high-level synthesis. **IEEE Design & Test of Computers**, v. 26, n. 4, p. 8–17, 2009.

DENG, L. The mnist database of handwritten digit images for machine learning research. **IEEE Signal Processing Magazine**, IEEE, v. 29, n. 6, p. 141–142, 2012.

DENNARD, R. et al. Design of ion-implanted mosfet's with very small physical dimensions. **IEEE Journal of Solid-State Circuits**, v. 9, n. 5, p. 256–268, 1974.

DRANE, T. A.; ROSE, T. M.; CONSTANTINIDES, G. A. On the systematic creation of faithfully rounded truncated multipliers and arrays. **IEEE Transactions on Computers**, v. 63, n. 10, p. 2513–2525, 2014.

DU, J. et al. Design of an approximate fft processor based on approximate complex multipliers. In: **2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [S.l.: s.n.], 2021. p. 308–313.

ERICSSON. **Mobile data traffic outlook**. 2022. Available from Internet: <https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/mobile-traffic-forecast>.

ESMAEILZADEH, H. et al. Dark silicon and the end of multicore scaling. In: **2011 38th Annual International Symposium on Computer Architecture (ISCA)**. [S.l.: s.n.], 2011. p. 365–376.

ESMAEILZADEH, H. et al. Architecture support for disciplined approximate programming. In: **Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems**. New York, NY, USA: Association for Computing Machinery, 2012. (ASPLOS XVII), p. 301–312. ISBN 9781450307598. Available from Internet: <https://doi.org/10.1145/2150976.2151008>.

ESMAEILZADEH, H. et al. Neural acceleration for general-purpose approximate programs. In: **2012 45th Annual IEEE/ACM International Symposium on Microarchitecture**. [S.l.: s.n.], 2012. p. 449–460.

FAN, Y. et al. A hardware-oriented ime algorithm for hevc and its hardware implementation. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 28, n. 8, p. 2048–2057, 2018.

FARSHCHI, F.; ABRISHAMI, M. S.; FAKHRAIE, S. M. New approximate multiplier for low power digital signal processing. In: **The 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS 2013)**. [S.l.: s.n.], 2013. p. 25–30.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. **Journal of Global Optimization**, Springer, v. 6, n. 2, p. 109–133, 1995.

FROEHLICH, S.; GROSSE, D.; DRECHSLER, R. Approximate memory: Data storage in the context of approximate computing. In: **Information Storage: A Multidisciplinary Perspective**. Springer International Publishing, 2020. p. 111–133. ISBN 978-3-030-19262-4. Available from Internet: <https://doi.org/10.1007/978-3-030-19262-4_4>.

FROEHLICH, S.; GROßE, D.; DRECHSLER, R. Error bounded exact bdd minimization in approximate computing. In: **2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)**. [S.l.: s.n.], 2017. p. 254–259.

Ganapathy, S. et al. Mitigating the impact of faults in unreliable memories for error-resilient applications. In: **2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2015. p. 1–6. ISSN 0738-100X.

GOGOI, S.; PEESAPATI, R. A hybrid hardware oriented motion estimation algorithm for hevc/h.265. **J. Real-Time Image Process.**, Springer-Verlag, Berlin, Heidelberg, v. 18, n. 3, p. 953–966, jun 2021. ISSN 1861-8200. Available from Internet: <https://doi.org/10.1007/s11554-020-01056-w>.

GOIRI, I. et al. Approxhadoop: Bringing approximations to mapreduce frameworks. **SIGPLAN Not.**, Association for Computing Machinery, New York, NY, USA, v. 50, n. 4, p. 383–397, mar 2015. ISSN 0362-1340. Available from Internet: <https://doi.org/10.1145/2775054.2694351>.

GRIGORIAN, B.; FARAHPOUR, N.; REINMAN, G. Brainiac: Bringing reliable accuracy into neurally-implemented approximate computing. In: **2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)**. [S.l.: s.n.], 2015. p. 615–626.

GRIGORIAN, B.; REINMAN, G. Accelerating divergent applications on simd architectures using neural networks. **ACM Trans. Archit. Code Optim.**, Association for Computing Machinery, New York, NY, USA, v. 12, n. 1, mar 2015. ISSN 1544-3566. Available from Internet: <https://doi.org/10.1145/2717311>.

GUPTA, V. et al. Impact: Imprecise adders for low-power approximate computing. In: **IEEE/ACM International Symposium on Low Power Electronics and Design**. [S.l.: s.n.], 2011. p. 409–414.

Han, J.; Orshansky, M. Approximate computing: An emerging paradigm for energy-efficient design. In: **18th IEEE European Test Symp. (ETS)**. [S.l.: s.n.], 2013. p. 1–6.

Hara, Y. et al. Proposal and quantitative analysis of the chstone benchmark program suite for practical c-based high-level synthesis. **Journal of Information Processing**, v. 17, p. 242–254, 2009.

HASHEMI, S.; BAHAR, R. I.; REDA, S. Drum: A dynamic range unbiased multiplier for approximate applications. In: **2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2015. p. 418–425.

HASHEMI, S.; BAHAR, R. I.; REDA, S. A low-power dynamic divider for approximate applications. In: **2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2016. p. 1–6.

HASHEMI, S.; TANN, H.; REDA, S. Blasys: Approximate logic synthesis using boolean matrix factorization. In: **2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2018. p. 1–6.

HASSANI, A. M.; REZAALIPOUR, M.; DEHYADEGARI, M. A novel ultra low power accuracy configurable adder at transistor level. In: **2018 8th International Conference on Computer and Knowledge Engineering (ICCKE)**. [S.l.: s.n.], 2018. p. 165–170.

HAUCK, S.; DEHON, A. **Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007. ISBN 9780080556017.

HSIAO, H.; ANDERSON, J. H. Sensei: An area-reduction advisor for fpga high-level synthesis. In: **2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2018. p. 25–30.

HU, J.; QIAN, W. A new approximate adder with low relative error and correct sign calculation. In: **2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2015. p. 1449–1454.

HWANG, C.-T.; LEE, J.-H.; HSU, Y.-C. A formal approach to the scheduling problem in high level synthesis. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 10, n. 4, p. 464–475, 1991.

IEA. **Data Centres and Data Transmission Networks Report**. 2021. Available from Internet: <https://www.iea.org/reports/data-centres-and-data-transmission-networks>.

ITU-T. **ITU-T Test Signals for Telecommunication Systems**. 2022. Available from Internet: <https://www.itu.int/net/itu-t/sigdb/menu.aspx>.

JOE, H.; KIM, Y. Efficient approximate image processor with low-part stochastic computing. In: **2019 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)**. [S.l.: s.n.], 2019. p. 29–32.

Kahng, A. B.; Kang, S. Accuracy-configurable adder for approximate arithmetic designs. In: **Proceedings of the 49th Annual Design Automation Conference**. New York, NY, USA: ACM, 2012. (DAC '12), p. 820–825. ISBN 978-1-4503-1199-1.

KHAN, W. Z. et al. Edge computing: A survey. **Future Gener. Comput. Syst.**, Elsevier Science Publishers B. V., NLD, v. 97, n. C, p. 219–235, aug 2019. ISSN 0167-739X. Available from Internet: <https://doi.org/10.1016/j.future.2019.02.050>.

Koch, D.; Hannig, F.; Ziener, D. **FPGAs for Software Programmers**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2016. ISBN 3319264060, 9783319264066.

KRAUSE, P. K.; POLIAN, I. Adaptive voltage over-scaling for resilient applications. In: **2011 Design, Automation & Test in Europe**. [S.l.: s.n.], 2011. p. 1–6.

KULKARNI, P.; GUPTA, P.; ERCEGOVAC, M. Trading accuracy for power with an underdesigned multiplier architecture. In: **2011 24th Internatioal Conference on VLSI Design**. [S.l.: s.n.], 2011. p. 346–351.

KUNDI, D.-E.-S. et al. Axrlwe: A multilevel approximate ring-lwe co-processor for lightweight iot applications. **IEEE Internet of Things Journal**, v. 9, n. 13, p. 10492–10501, 2022.

KUNDU, S.; CHANDRAKAR, K.; ROY, S. Sat based scheduling in high level synthesis. In: KUNDU, M. K. et al. (Ed.). **Advanced Computing, Networking and Informatics-Volume 2**. Cham: Springer International Publishing, 2014. p. 533–542. ISBN 978-3-319-07350-7.

LAHTI, S. et al. Are we there yet? a study on the state of high-level synthesis. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 38, n. 5, p. 898–911, 2019.

Lattner, C.; Adve, V. Llvm: a compilation framework for lifelong program analysis transformation. In: **Int. Symp. Code Generation and Optimization (CGO)**. [S.l.: s.n.], 2004. p. 75–86.

Lee, S.; Gerstlauer, A. Data-dependent loop approximations for performance-quality driven high-level synthesis. **IEEE Embedded Syst. Lett.**, v. 10, n. 1, p. 18–21, March 2017.

Lee, S.; John, L. K.; Gerstlauer, A. High-level synthesis of approximate hardware under joint precision and voltage scaling. In: **Proc. Conf. Design, Automation & Test in Europe**. [S.l.: s.n.], 2017. (DATE '17), p. 187–192.

Leipnitz, M. T.; Nazar, G. L. High-level synthesis of resource-oriented approximate designs for fpgas. In: **Proceedings of the 56th Annual Design Automation Conference 2019**. New York, NY, USA: ACM, 2019a. (DAC '19), p. 126:1–126:6. ISBN 978-1-4503-6725-7. Available from Internet: <http://doi.acm.org/10.1145/3316781.3317839>.

LEIPNITZ, M. T.; NAZAR, G. L. High-level synthesis of approximate designs under real-time constraints. **ACM Trans. Embed. Comput. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 18, n. 5s, oct 2019b. ISSN 1539-9087. Available from Internet: <https://doi.org/10.1145/3358182>.

LEIPNITZ, M. T.; NAZAR, G. L. High-level synthesis of throughput-optimized and energy-efficient approximate designs. In: **Proceedings of the 17th ACM International Conference on Computing Frontiers**. New York, NY, USA: Association for Computing Machinery, 2020. (CF '20), p. 221–224. ISBN 9781450379564. Available from Internet: <https://doi.org/10.1145/3387902.3394039>.

LEIPNITZ, M. T.; NAZAR, G. L. Throughput-oriented spatio-temporal optimization in approximate high-level synthesis. In: **2020 IEEE 38th International Conference on Computer Design (ICCD)**. [S.l.: s.n.], 2020. p. 316–323.

LEIPNITZ, M. T. et al. Enhancing real-time motion estimation through approximate high-level synthesis. In: **2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [S.l.: s.n.], 2020. p. 30–35.

Li, C. et al. Joint precision optimization and high level synthesis for approximate computing. In: **Proc. 52Nd Annu. Design Automation Conference**. [S.l.: s.n.], 2015. (DAC '15), p. 104:1–104:6.

LI, Q. et al. Leveraging approximate data for robust flash storage. In: **2019 56th ACM/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2019. p. 1–6.

LI, R. et al. Approximate computing with stochastic transistors' voltage over-scaling. **IEEE Access**, v. 7, p. 6373–6385, 2019.

LI, S.; PARK, S.; MAHLKE, S. Sculptor: Flexible approximation with selective dynamic loop perforation. In: **Proceedings of the 2018 International Conference on Supercomputing**. New York, NY, USA: Association for Computing Machinery, 2018. (ICS '18), p. 341–351. ISBN 9781450357838. Available from Internet: <https://doi.org/10.1145/3205289.3205317>.

Li, Y. . S.; Malik, S. Performance analysis of embedded software using implicit path enumeration. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 16, n. 12, p. 1477–1487, Dec 1997. ISSN 0278-0070.

LIGNATI, B. N. et al. Exploiting hls-generated multi-version kernels to improve cpu-fpga cloud systems. In: **2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)**. [S.l.: s.n.], 2021. p. 536–541.

Lingamneni, A. et al. Synthesizing parsimonious inexact circuits through probabilistic design techniques. **ACM Trans. Embed. Comput. Syst.**, v. 12, n. 2s, p. 93:1–93:26, may 2013.

LIU, C.; HAN, J.; LOMBARDI, F. A low-power, high-performance approximate multiplier with configurable partial error recovery. In: **2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2014. p. 1–4.

LIU, G.; ZHANG, Z. Statistically certified approximate logic synthesis. In: **2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2017. p. 344–351.

LIU, W. et al. Design and analysis of inexact floating-point adders. **IEEE Transactions on Computers**, v. 65, n. 1, p. 308–314, 2016.

LIU, W.; LOMBARDI, F.; SHULTE, M. A retrospective and prospective view of approximate computing [point of view]. **Proceedings of the IEEE**, v. 108, n. 3, p. 394–399, 2020.

Lokuciejewski, P.; Marwedel, P. **Worst-Case Execution Time Aware Compilation Techniques for Real-Time Systems**. [S.l.: s.n.], 2011. ISBN 978-90-481-9928-0.

MAZAHIR, S.; HASAN, O.; SHAFIQUE, M. Adaptive approximate computing in arithmetic datapaths. **IEEE Design & Test**, v. 35, n. 4, p. 65–74, 2018.

MIAO, J. et al. Modeling and synthesis of quality-energy optimal approximate adders. In: **2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2012. p. 728–735.

MIGUEL, J. S. et al. Doppelgänger: A cache for approximate computing. In: **Proceedings of the 48th International Symposium on Microarchitecture**. New York, NY, USA: Association for Computing Machinery, 2015. (MICRO-48), p. 50–61. ISBN 9781450340342. Available from Internet: <https://doi.org/10.1145/2830772.2830790>.

Miguel, J. S.; Badr, M.; Jerger, N. E. Load value approximation. In: **2014 47th Annual IEEE/ACM International Symposium on Microarchitecture**. [S.l.: s.n.], 2014. p. 127–139. ISSN 1072-4451.

Mittal, S. A survey of techniques for approximate computing. **ACM Comput. Surv.**, v. 48, n. 4, p. 62:1–62:33, mar. 2016.

MOHAPATRA, D. et al. Design of voltage-scalable meta-functions for approximate computing. In: **2011 Design, Automation & Test in Europe**. [S.l.: s.n.], 2011. p. 1–6.

MOONS, B.; VERHELST, M. Dvas: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing. In: **2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)**. [S.l.: s.n.], 2015. p. 237–242.

MOORE, G. E. Cramming more components onto integrated circuits. **Electronics**, v. 38, n. 8, April 1965.

Moreau, T. et al. Snnap: Approximate computing on programmable socs via neural acceleration. In: **2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)**. [S.l.: s.n.], 2015. p. 603–614. ISSN 1530-0897.

MULLER, J.-M. Elementary functions and approximate computing. **Proceedings of the IEEE**, v. 108, n. 12, p. 2136–2149, 2020.

Nane, R. et al. A survey and evaluation of fpga high-level synthesis tools. **IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.**, v. 35, n. 10, p. 1591–1604, Oct 2016.

NAZAR, G. L. et al. Lightweight dual modular redundancy through approximate computing. In: **2021 XI Brazilian Symposium on Computing Systems Engineering (SBESC)**. [S.l.: s.n.], 2021. p. 1–8.

NAZAR, G. L. et al. Precep: Automatic insertion of partial redundancy based on critical error probability. **Microelectronics Reliability**, v. 126, p. 114226, 2021. ISSN 0026-2714. Proceedings of ESREF 2021, 32nd European Symposium on Reliability of Electron Devices, Failure Physics and Analysis. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S002627142100192X>.

NDOUR, G. et al. Evaluation of variable bit-width units in a risc-v processor for approximate computing. In: **Proceedings of the 16th ACM International Conference on Computing Frontiers**. New York, NY, USA: Association for Computing Machinery, 2019. (CF ’19), p. 344–349. ISBN 9781450366854. Available from Internet: <https://doi.org/10.1145/3310273.3323159>.

Nepal, K. et al. Automated high-level generation of low-power approximate computing circuits. **IEEE Transactions on Emerging Topics in Computing**, v. 7, n. 1, p. 18–30, Jan 2019. ISSN 2168-6750.

Nepal, K. et al. Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In: **Proc. Conf. Design, Automation & Test in Europe**. [S.l.: s.n.], 2014. (DATE ’14), p. 361:1–361:6.

NGUYEN, D. T. et al. An approximate memory architecture for energy saving in deep learning applications. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 67, n. 5, p. 1588–1601, 2020.

OLIVEIRA, G. F. et al. Employing classification-based algorithms for general-purpose approximate computing. In: **Proceedings of the 55th Annual Design Automation Conference**. New York, NY, USA: Association for Computing Machinery, 2018. (DAC '18). ISBN 9781450357005. Available from Internet: <https://doi.org/10.1145/3195970.3196043>.

Palem, K. V. et al. Sustaining moore's law in embedded computing through probabilistic and approximate design: Retrospects and prospects. In: **Proc. Int. Conf. Compilers, Architecture, and Synthesis for Embedded Systems**. [S.l.: s.n.], 2009. (CASES '09), p. 1–10.

PARK, J.; CHOI, J. H.; ROY, K. Dynamic bit-width adaptation in dct: An approach to trade off image quality and computation energy. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 18, n. 5, p. 787–793, 2010.

PEJOVIć, V. Towards approximate mobile computing. **GetMobile: Mobile Comp. and Comm.**, Association for Computing Machinery, New York, NY, USA, v. 22, n. 4, p. 9–12, may 2019. ISSN 2375-0529. Available from Internet: <https://doi.org/10.1145/3325867.3325871>.

PERRI, S. et al. Efficient approximate adders for fpga-based data-paths. **Electronics**, v. 9, n. 9, 2020. ISSN 2079-9292. Available from Internet: <https://www.mdpi.com/2079-9292/9/9/1529>.

PESCE, M. **Cloud Computing's Coming Energy Crisis**. 2021. Available from Internet: <https://spectrum.ieee.org/cloud-computings-coming-energy-crisis>.

PRABAKARAN, B. S. et al. Approxfpgas: Embracing asic-based approximate arithmetic components for fpga-based systems. In: **2020 57th ACM/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2020. p. 1–6.

PRABAKARAN, B. S. et al. Demas: An efficient design methodology for building approximate adders for fpga-based systems. In: **2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2018. p. 917–920.

Puschner, P.; Koza, C. Calculating the maximum execution time of real-time programs. **Real-Time Syst.**, v. 1, n. 2, p. 159–176, sep. 1989. ISSN 0922-6443.

RAHA, A. et al. Quality configurable reduce-and-rank for energy efficient approximate computing. In: **2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2015. p. 665–670.

Rahimi, A.; Benini, L.; Gupta, R. K. Spatial memoization: Concurrent instruction reuse to correct timing errors in simd architectures. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 60, n. 12, p. 847–851, Dec 2013. ISSN 1549-7747.

RAMASUBRAMANIAN, S. G. et al. Relax-and-retime: A methodology for energy-efficient recovery based design. In: **2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2013. p. 1–6.

RANJAN, A. et al. Approximate memory compression. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 28, n. 4, p. 980–991, 2020.

RANJAN, A. et al. Aslan: Synthesis of approximate sequential circuits. In: **2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2014. p. 1–6.

REHMAN, S. et al. Architectural-space exploration of approximate multipliers. In: **2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2016. p. 1–8.

RESENDE, M. G. C.; RIBEIRO, C. C. Greedy randomized adaptive search procedures: Advances and extensions. In: **Handbook of Metaheuristics**. [S.l.]: Springer International Publishing, 2019. p. 169–220. ISBN 978-3-319-91086-4.

RODRIGUES, G. S.; KASTENSMIDT, F. L.; BOSIO, A. Approximate computing for safety-critical applications. In: **2021 IEEE 22nd Latin American Test Symposium (LATS)**. [S.l.: s.n.], 2021. p. 1–3.

ROLDAO-LOPES, A. et al. More flops or more precision? accuracy parameterizable linear equation solvers for model predictive control. In: **2009 17th IEEE Symposium on Field Programmable Custom Computing Machines**. [S.l.: s.n.], 2009. p. 209–216.

RUBIO-GONZáLEZ, C. et al. Precimonious: Tuning assistant for floating-point precision. In: **SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis**. [S.l.: s.n.], 2013. p. 1–12.

SAADAT, H.; JAVAID, H.; PARAMESWARAN, S. Approximate integer and floating-point dividers with near-zero error bias. In: **2019 56th ACM/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2019. p. 1–6.

Samadi, M. et al. Paraprox: Pattern-based approximation for data parallel applications. **SIGARCH Comput. Archit. News**, ACM, New York, NY, USA, v. 42, n. 1, p. 35–50, feb. 2014. ISSN 0163-5964.

SAMPAIO, F. et al. Approximation-aware multi-level cells stt-ram cache architecture. In: **2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)**. [S.l.: s.n.], 2015. p. 79–88.

Sampson, A. et al. Enerj: Approximate data types for safe and general low-power computation. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 46, n. 6, p. 164–174, jun. 2011. ISSN 0362-1340.

SAMPSON, A. et al. Approximate storage in solid-state memories. **ACM Trans. Comput. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 32, n. 3, sep 2014. ISSN 0734-2071. Available from Internet: <https://doi.org/10.1145/2644808>.

SáNCHEZ-CLEMENTE, A. J.; ENTRENA, L.; GARCíA-VALDERAS, M. Partial tmr in fpgas using approximate logic circuits. **IEEE Transactions on Nuclear Science**, v. 63, n. 4, p. 2233–2240, 2016.

SCARABOTTOLO, I.; ANSALONI, G.; POZZI, L. Circuit carving: A methodology for the design of approximate hardware. In: **2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2018. p. 545–550.

Schafer, B. C. Enabling high-level synthesis resource sharing design space exploration in fpgas through automatic internal bitwidth adjustments. **IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.**, v. 36, n. 1, p. 97–105, Jan 2017.

SCHLACHTER, J.; CAMUS, V.; ENZ, C. Design of energy-efficient discrete cosine transform using pruned arithmetic circuits. In: **2016 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2016. p. 341–344.

SCHLACHTER, J. et al. Design and applications of approximate circuits by gate-level pruning. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 25, n. 5, p. 1694–1702, 2017.

SENGUPTA, D. et al. Saber: Selection of approximate bits for the design of error tolerant circuits. In: **2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2017. p. 1–6.

SHAFIQUE, M. et al. A low latency generic accuracy configurable adder. In: **2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2015. p. 1–6.

Shafique, M. et al. Invited - cross-layer approximate computing: From logic to architectures. In: **Proceedings of the 53rd Annual Design Automation Conference**. New York, NY, USA: ACM, 2016. (DAC '16), p. 99:1–99:6. ISBN 978-1-4503-4236-0.

Shaw, A. C. Reasoning about time in higher-level language software. **IEEE Trans. Softw. Eng.**, IEEE Press, Piscataway, NJ, USA, v. 15, n. 7, p. 875–889, jul. 1989. ISSN 0098-5589.

SHIN, D.; GUPTA, S. K. A new circuit simplification method for error tolerant applications. In: **2011 Design, Automation & Test in Europe**. [S.l.: s.n.], 2011. p. 1–6.

Sidiroglou-Douskos, S. et al. Managing performance vs. accuracy trade-offs with loop perforation. In: **Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering**. New York, NY, USA: ACM, 2011. (ESEC/FSE '11), p. 124–134. ISBN 978-1-4503-0443-6.

Sinha, S.; Zhang, W. Low-power fpga design using memoization-based approximate computing. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 24, n. 8, p. 2665–2678, Aug 2016. ISSN 1063-8210.

SJöVALL, P. et al. Fpga-powered 4k120p hevc intra encoder. In: **2018 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2018. p. 1–5.

SLLAME, A.; DRABEK, V. An efficient list-based scheduling algorithm for high-level synthesis. In: **Proceedings Euromicro Symposium on Digital System Design. Architectures, Methods and Tools**. [S.l.: s.n.], 2002. p. 316–323.

SOEKEN, M. et al. Bdd minimization for approximate computing. In: **2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)**. [S.l.: s.n.], 2016. p. 474–479.

St. Amant, R. et al. General-purpose code acceleration with limited-precision analog computation. In: **Proceeding of the 41st Annual International Symposium on Computer Architecuture**. Piscataway, NJ, USA: IEEE Press, 2014. (ISCA ’14), p. 505–516. ISBN 978-1-4799-4394-4.

STANLEY-MARBELL, P. et al. Exploiting errors for efficiency: A survey from circuits to applications. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 3, jun 2020. ISSN 0360-0300. Available from Internet: <https://doi.org/10.1145/3394898>.

TIAN, Y. et al. Approxlut: A novel approximate lookup table-based accelerator. In: **2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2017. p. 438–443.

TOAN, N. V.; LEE, J.-G. Fpga-based multi-level approximate multipliers for high-performance error-resilient applications. **IEEE Access**, v. 8, p. 25481–25497, 2020.

ULLAH, S. et al. Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators. In: **Proceedings of the 55th Annual Design Automation Conference**. New York, NY, USA: Association for Computing Machinery, 2018. (DAC ’18). ISBN 9781450357005. Available from Internet: <https://doi.org/10.1145/3195970.3195996>.

USC-SIP. **The USC-SIPI Image Database**. 2022. Available from Internet: <http://sipi.usc.edu/database>.

VARGHESE, B. et al. Challenges and opportunities in edge computing. In: **2016 IEEE International Conference on Smart Cloud (SmartCloud)**. [S.l.: s.n.], 2016. p. 20–26.

VASICEK, Z. Relaxed equivalence checking: a new challenge in logic synthesis. In: **2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)**. [S.l.: s.n.], 2017. p. 1–6.

VASICEK, Z.; SEKANINA, L. Evolutionary approach to approximate digital circuits design. **IEEE Transactions on Evolutionary Computation**, v. 19, n. 3, p. 432–444, 2015.

VASICEK, Z.; SEKANINA, L. Evolutionary design of complex approximate combinational circuits. **Genetic Programming and Evolvable Machines**, Kluwer Academic Publishers, USA, v. 17, n. 2, p. 169–192, jun 2016. ISSN 1389-2576. Available from Internet: <https://doi.org/10.1007/s10710-015-9257-1>.

VASSILIADIS, V. et al. A programming model and runtime system for significance-aware energy-efficient computing. **SIGPLAN Not.**, Association for Computing Machinery, New York, NY, USA, v. 50, n. 8, p. 275–276, jan 2015. ISSN 0362-1340. Available from Internet: <https://doi.org/10.1145/2858788.2688546>.

Vaverka, F.; Hrbacek, R.; Sekanina, L. Evolving component library for approximate high level synthesis. In: **IEEE Symp. Series on Computational Intelligence (SSCI)**. [S.l.: s.n.], 2016. p. 1–8.

Venkataramani, S. et al. Approximate computing and the quest for computing efficiency. In: **Proc. 52Nd Annu. Design Automation Conference**. [S.l.: s.n.], 2015. (DAC '15), p. 120:1–120:6.

VENKATARAMANI, S. et al. Quality programmable vector processors for approximate computing. In: **2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)**. [S.l.: s.n.], 2013. p. 1–12.

Venkataramani, S. et al. Axnn: Energy-efficient neuromorphic systems using approximate computing. In: **2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)**. [S.l.: s.n.], 2014. p. 27–32.

VENKATARAMANI, S.; ROY, K.; RAGHUNATHAN, A. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In: **2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2013. p. 1367–1372.

VENKATARAMANI, S. et al. Salsa: Systematic logic synthesis of approximate circuits. In: **DAC Design Automation Conference 2012**. [S.l.: s.n.], 2012. p. 796–801.

VENKATESAN, R. et al. Macaco: Modeling and analysis of circuits for approximate computing. In: **2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2011. p. 667–673.

WAKABAYASHI, K.; SCHAFER, B. C. "all-in-c" behavioral synthesis and verification with cyberworkbench. In: **High-Level Synthesis: From Algorithm to Digital Circuit**. Dordrecht: Springer Netherlands, 2008. p. 113–127. ISBN 978-1-4020-8588-8. Available from Internet: <https://doi.org/10.1007/978-1-4020-8588-8_7>.

WU, Y.; QIAN, W. An efficient method for multi-level approximate logic synthesis under error rate constraint. In: **2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2016. p. 1–6.

XU, Q.; MYTKOWICZ, T.; KIM, N. S. Approximate computing: A survey. **IEEE Design & Test**, v. 33, n. 1, p. 8–22, 2016.

Xu, S.; Schafer, B. C. Exposing approximate computing optimizations at different levels: From behavioral to gate-level. **IEEE Trans. Very Large Scale Integr. (VLSI) Syst.**, v. 25, n. 11, p. 3077–3088, Nov 2017.

XU, S.; SCHAFER, B. C. Deep: Dedicated energy-efficient approximation for dynamically reconfigurable architectures. In: **2018 IEEE 36th International Conference on Computer Design (ICCD)**. [S.l.: s.n.], 2018. p. 587–594.

XU, S.; SCHAFER, B. C. Approximating behavioral hw accelerators through selective partial extractions onto synthesizable predictive models. In: **2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2019. p. 1–8.

XU, S.; SCHAFER, B. C. Low power design of runtime reconfigurable fpgas through contexts approximations. In: **2019 IEEE 37th International Conference on Computer Design (ICCD)**. [S.l.: s.n.], 2019. p. 524–531.

XU, S.; SCHAFER, B. C. Toward self-tunable approximate computing. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 27, n. 4, p. 778–789, 2019.

XU, S.; SCHAFER, B. C. On the design of high performance hw accelerator through high-level synthesis scheduling approximations. In: **2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2020. p. 1378–1383.

YAML. **YAML Ain't Markup Language**. 2022. Available from Internet: <https://yaml.org>.

Yazdanbakhsh, A. et al. Axbench: A multiplatform benchmark suite for approximate computing. **IEEE Design Test**, v. 34, n. 2, p. 60–68, April 2017. ISSN 2168-2356.

YAZDANBAKHSH, A. et al. Axilog: Language support for approximate hardware design. In: **2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2015. p. 812–817.

YE, R. et al. On reconfiguration-oriented approximate adder design and its application. In: **2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2013. p. 48–54.

Yeh, T. et al. The art of deception: Adaptive precision reduction for area efficient physics acceleration. In: **40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)**. [S.l.: s.n.], 2007. p. 394–406. ISSN 1072-4451.

YESIL, S.; AKTURK, I.; KARPUZCU, U. R. Toward dynamic precision scaling. **IEEE Micro**, v. 38, n. 4, p. 30–39, 2018.

YIN, P. et al. Design and performance evaluation of approximate floating-point multipliers. In: **2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [S.l.: s.n.], 2016. p. 296–301.

ZERVAKIS, G. et al. Multi-level approximate accelerator synthesis under voltage island constraints. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 66, n. 4, p. 607–611, 2019.

ZHANG, Q. et al. Approxann: An approximate computing framework for artificial neural network. In: **2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2015. p. 701–706.

ZHANG, Q. et al. Approxit: An approximate computing framework for iterative methods. In: **2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2014. p. 1–6.

ZHAO, W. et al. Improving the energy efficiency of stt-mram based approximate cache. In: **2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2021. p. 1104–1109.

ZHOU, Y. et al. Rosetta: A realistic high-level synthesis benchmark suite for software programmable fpgas. In: **Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays**. New York, NY, USA: Association for Computing Machinery, 2018. (FPGA '18), p. 269–278. ISBN 9781450356145.

ZHOU, Y. et al. Primal: Power inference using machine learning. In: **Proceedings of the 56th Annual Design Automation Conference 2019**. New York, NY, USA: Association for Computing Machinery, 2019. (DAC '19). ISBN 9781450367257. Available from Internet: <https://doi.org/10.1145/3316781.3317884>.

ZHU, N. et al. Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 18, n. 8, p. 1225–1229, 2010.

APPENDIX A — EXTENDED ABSTRACT IN PORTUGUESE (*RESUMO ESTENDIDO EM PORTUGUÊS*)

Este apêndice apresenta de forma resumida esta tese de doutorado, intitulada "Integração de Consciência de Restrições e Múltiplas Técnicas de Aproximação em Síntese de Alto Nível para FPGAs".

A.1 Introdução

A adoção de Síntese da Alto Nível (HLS do Inglês *High-Level Synthesis*) visando *Field-Programmable Gate Arrays* (FPGAs) aumentou à medida que as ferramentas mais recentes de HLS evoluíram para fornecer resultados de alta qualidade enquanto aumentam a produtividade e reduzem o *time-to-market*. Simultaneamente, inúmeras técnicas de computação aproximativa (AC do Inglês *Approximate Computing*) foram desenvolvidas para reduzir os custos de projeto em domínios de aplicação resilientes a erros, tais como processamento de sinais e multimídia, mineração de dados, aprendizado de máquina e visão computacional, para trocar a precisão da computação por economia de área e energia ou melhorias de desempenho. Entretanto, a seleção de técnicas adequadas para cada aplicação e otimização alvo é complexa, porém crucial para resultados de alta qualidade. Neste contexto, muitos trabalhos propuseram incorporar técnicas de AC dentro do fluxo de ferramentas HLS para aliviar a carga de explorar manualmente circuitos aproximados, ou seja, os projetistas podem recorrer a ferramentas de HLS aproximativas (AHLS do Inglês *Approximate High-Level Synthesis*) para automatizar a exploração das técnicas de AC quando tentarem fazer um projeto atender os requisitos especificados.

A.2 Motivação e Contribuições

As metodologias atualmente disponíveis na literatura de AHLS não permitem especificar um conjunto de métricas de projeto para orientar a exploração de circuitos aproximados no sentido de atender as otimizações pretendidas com o erro dentro dentro de um limite aceitável. Além disso, esses métodos normalmente estão vinculados à uma única técnica de aproximação ou à um conjunto de técnicas de difícil extensão, cuja exploração não é totalmente automatizada ou orientada por objetivos de otimização. Portanto, as ferramentas de AHLS disponíveis ignoram os benefícios de expandir o espaço de projeto, misturando diversas técnicas de aproximação para atingir objetivos específicos de projeto com o mínimo de erro. Neste contexto, a principal contribuição desta tese é uma nova metodologia de síntese automatizada de circuitos aproximados dentro de fluxos de projeto baseados em HLS para FPGAs. Mais especificamente, propomos uma metodologia de AHLS consciente das restrições impostas pelo desenvolvedor que combina automaticamente múltiplas técnicas de AC para produzir hardware aproximado, explorando uma vasta gama de oportunidades de otimização no sentido de satisfazer múltiplas restrições simultaneamente com o mínimo de erro. A metodologia proposta centra-se em extensibilidade, ou seja, a capacidade de expandir o espaço de projeto através da inclusão de novas técnicas de AC, considerando os seus potenciais benefícios em conjunto com os já disponíveis na tentativa de atender as restrições especificadas. Para esse objetivo, as técnicas de AC são implementadas como transformações de código dentro da infraestrutura de compilação de ferramentas HLS, permitindo a síntese de hardware aproximado a partir de um conjunto de aproximações de nível de software de fácil extensão.

Em resumo, esta tese propõe que uma metodologia de desenvolvimento AHLS para FPGAs diretamente orientada por uma ou mais restrições em métricas de projeto, e que combina múltiplas técnicas de AC, é capaz de sintetizar hardware aproximado com erro reduzido, quando comparada com abordagens de uma única técnica ou que não consideram as restrições impostas pelo projetista. Portanto, destacamos as seguintes contribuições:

- Mostramos que uma metodologia de projeto consciente das restrições, onde a exploração do espaço de projeto é diretamente dirigida por restrições nas métricas de projeto de interesse do projetista, pode atender essas restrições com erro reduzido.
- Mostramos que diferentes técnicas de AC oferecem diferentes compensações dependendo da aplicação considerada e das otimizações alvo, demonstrando as vantagens de uma abordagem consciente das restrições ao explorar múltiplas técnicas.
- Mostramos que nenhuma técnica de AC é capaz de oferecer melhores resultados que uma combinação adequada de técnicas cuja seleção é orientada por restrições nas métricas de projeto de interesse, com todas as técnicas de AC disponíveis sendo utilizadas para melhorar os resultados em diferentes cenários.

Estas contribuições foram validadas experimentalmente através de uma implementação dentro da ferramenta de HLS LegUp (Canis et al., 2013). Um conjunto de cinco técnicas de AC foram implementadas como passos de transformação dentro da estrutura de compilador LLVM (Lattner; Adve, 2004), visando reduzir o tempo de execução de pior caso (WCET do Inglês *Worst-Case Execution Time*) e o uso de recursos de FPGAs como *look-up tables* (LUTs), registradores (REGs), e blocos para processamento digital de sinais (DSPs do Inglês *Digital Signal Processing*). A heurística empregada baseia-se no *Greedy Randomized Adaptive Search Procedure* (GRASP) (FEO; RESENDE, 1995; RESENDE; RIBEIRO, 2019) para identificar seqüências de aproximações que atendam as restrições com o mínimo de erro. A natureza adaptativa do GRASP o torna uma opção promissora para explorar múltiplas técnicas de AC, especialmente quando o objetivo é atingir múltiplos objetivos de projeto que podem ser impactados de forma diferente por diferentes técnicas de aproximação.

A.3 Resultados Experimentais

Resultados experimentais foram obtidos utilizando o conjunto de aplicações listadas na Tabela 5.1, envolvendo processamento de sinais, imagem, vídeo, e aprendizagem de máquina. A Figura 7.5 resume os resultados obtidos para cada aplicação e restrição alvo, comparando a heurística não consciente das restrições com a consciente das restrições. Adicionalmente, são analisadas a melhor técnica isolada para cada cenário e a metodologia de multitécnica para o cenário consciente das restrições. Em média, reduções de 29% no erro quadrático médio (MSE do Inglês *Mean Squared Error*), variando de 9,54% a 52,23% dependendo da aplicação e da restrição alvo, foram obtidas com a abordagem de técnica única consciente das restrições, em comparação com a abordagem não consciente das restrições. Para a aplicação DIGIT, um aumento absoluto na precisão variando de 1,65% a 20,28% foi obtido, com uma média de 10,87%. Ao comparar a abordagem de técnica única com a multitécnica, foram obtidas reduções médias de 17,7% em MSE, variando de 5% a 30%, enquanto um aumento absoluto entre 1,9% e 6,5% foi alcançado para o DIGIT, com uma média de 3,5%. Finalmente, ao adotar a abordagem multitécnica consciente das restrições em relação à abordagem não consciente das restrições com técnica única, reduções de cerca de 43% em média podem ser alcançadas em MSE, variando de 30% a 60%. Quanto ao DIGIT, um aumento absoluto na precisão variando de 3,52% a 25,03% foi alcançado, com uma média de 14,26%. Observe que este é o efeito composto da consciência das restrições e o uso conjunto de múltiplas técnicas.

Como tendência geral, melhorias mais significativas podem ser observadas à medida que são impostas restrições mais rigorosas. Além disso, maiores melhorias são exibidas quando se trata de economia de recursos, especialmente para as aplicações ADPCM e JPEG, onde foram obtidas reduções médias de cerca de 50% no MSE. Estas duas aplicações são notavelmente diferentes das outras por utilizarem muito mais DSPs, como mostrado na Tabela 5.1. Como a abordagem não consciente das restrições não é capaz de se concentrar nos tipos de operação que utilizam recursos específicos que ainda precisam ser economizados, como os DSPs, ela normalmente aproxima mais operações até que todas as restrições de recursos sejam atendidas, ou seja, mesmo que a abordagem consciente das restrições reduza consideravelmente as métricas de erro em todos os cenários, ela é especialmente adequada para projetos com uso mais abrangente de recursos heterogêneos.

Os resultados demonstram que, além de adotar uma metodologia consciente das restrições para orientar a exploração do espaço de projeto no sentido de atingir otimizações específicas, uma exploração automatizada de múltiplas técnicas pode proporcionar resultados ainda melhores. Dado que diferentes técnicas de AC proporcionam ganhos diferentes dependendo do rigor das restrições para cada métrica de projeto considerada para otimização, a heurística empregada deve ser capaz de se concentrar na exploração da melhor combinação de técnicas de AC para cada cenário específico relacionado à aplicação alvo. Embora os projetistas possam gerenciar o espaço de projeto definindo as técnicas de AC a serem exploradas, torna-se claro que selecionar apenas a melhor técnica ou usar ferramentas AHLS que implementam tais técnicas para um cenário específico não é suficiente para otimizar os resultados. Nessa direção, fornecer uma maneira sistemática de integrar novas técnicas ao fluxo padrão de AHLS é crucial para oferecer uma exploração automatizada de otimizações únicas para as restrições e aplicações alvo. Portanto, como demonstrado nos experimentos, uma metodologia de projeto AHLS consciente das restrições capaz de explorar automaticamente múltiplas técnicas AC é uma opção promissora para otimizar hardware sintetizado a partir de especificações de projeto de alto nível, especialmente quando se busca otimizações em múltiplas métricas de projeto.