Adaptive Approximate Computing for Enhanced Quality Assurance

Mahmoud Saleh Masadeh

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

August 2020

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Mahmoud Saleh Masadeh

Entitled: Adaptive Approximate Computing for Enhanced Quality Assurance

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy Electrical and Computer Engineering

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Chun Wang	Chair
Dr. Jie Han	External Examiner
Dr. Rajagopalan Jayakumar	External to Program
Dr. Mohammed Reza Soleymani	Examiner
Dr. Otmane Ait Mohamed	Examiner
Dr. Sofiène Tahar	Thesis Supervisor (s)

Approved by

Dr. Yousef R. Shayan — Chair of Department or Graduate Program Director

August 6, 2020
Date of Defence

Dr. Mourad Debbabi — Dean, Gina Cody School of Engineering & Computer Science

ABSTRACT

Adaptive Approximate Computing for Enhanced Quality Assurance

Mahmoud Saleh Masadeh, Ph.D.

Concordia University, 2020

Approximate Computing (AC) has been widely advocated for energy-efficiency in error-tolerant applications as it offers the opportunity to trade-off output quality for reduced power consumption and execution time. Approximate accelerators, which consist of a large number of functional units, have been proposed in error-resilient applications, to speed-up regularly executed code elements while assuring defined quality constraints. However, with an approximate static design, while the average output quality constraint is satisfied, the quality of individual outputs varies significantly with dynamically changing inputs. Thus, quality assurance is an essential and non-trivial problem. State-of-the-art approaches in approximate computing address this problem by precisely re-evaluating those quality-violating accelerator invocations. However, such methods can significantly diminish or even cancel the benefits of approximation, especially when the rate of input data variations is high and approximate errors are considerably above a user given threshold, i.e., target output quality (TOQ).

As a general solution to this problem, in this thesis, we propose a novel methodology to enhance the quality of approximation by two approaches: 1) design adaptation by predicting the most suitable settings of the approximate design to execute the inputs; and/or 2) error compensation by predicting the error magnitude to use in adjusting the output results. The proposed method predicts the design settings, or the error magnitude based on the applied input data and user preferences, without losing the gains of approximations. We mostly consider the case of approximate accelerators built with approximate functional units such as approximate multipliers, where we design a library of approximate accelerators with 20 different settings of 8 and 16-bit approximate multipliers.

For the adaptive approximate computing, we use machine learning (ML) algorithms to build an efficient and lightweight design selector to adapt the approximate accelerators to meet a user-defined quality constraint. Compared with contemporary techniques, our approach is a fine-grained input-dependent approximation approach, with no missed approximation opportunities or rollback recovery overhead. The proposed method applies to any approximate accelerator with error-tolerant components, and it is flexible in adapting various error metrics. We fully automate the proposed methodology of quality assurance of approximate accelerators using ML-based models, for both software and hardware implementations. The obtained analysis results of image processing and audio applications showed that it is possible to satisfy the TOQ with an accuracy ranging from 80% to 85.7%. The hardware implementation is based on Field Programmable Gate Arrays (FPGA) approximate adaptive accelerator with constraints on size, cost, and power consumption, which rely on dynamic partial reconfiguration to assist in satisfying these requirements.

To ensure the quality of results for a single approximate design rather than a library, we build a decision tree-based model for error compensation. The proposed model detects the magnitude of approximation error based on design inputs. Then, it enhances the accuracy of the approximate result by adding the error magnitude to it. The proposed methodology is able to enhance the quality of image processing applications with a negligible overhead.

In loving memory of my father,

To my mother, brothers and sister,

To my rock and lovely wife Huda,

To my kids Yamin, Nada and Jawad.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor, Prof. Sofiène Tahar, for his guidance, support, follow-up, and encouragement throughout the previous four years of my Ph.D. studies. He was always easily approachable, available and inspiring. I have learned so much from his deep insights about research and strong expertise. Moreover, I have learned many things from him regarding research, academia, dedication, and life in general. In short, I will always be indebted for the time and the effort he has spent to help me complete my Ph.D. Also, I am indebted to Dr. Osman Hasan, who not only initiated this thesis work but also followed the progress of my work and provided valuable feedback at various stages. I have learned a lot from his in-depth knowledge, hard work and immense experience.

I am deeply grateful to Yarmouk University for granting me a scholarship to conduct my doctoral studies at Concordia University. I would like to thank Prof. Jamal Abu-Doleh for his unrepayable help and support. Also, I would like to thank Dr. Ali Shatnawi for introducing me to Prof. Sofiène Tahar.

I would like to express my gratitude to Dr. Jie Han for taking time out of his busy schedule and accepting to be my external Ph.D. thesis examiner. I am also grateful to Dr. Rajagopalan Jayakumar, Dr. Reza Soleymani and Dr. Otmane Ait Mohamed for serving on my advisory thesis committee.

Many thanks to my friends and colleagues at the Hardware Verification Group (HVG), specially Mbarka Soualhia, Hassnaa El-derhalli, Yassmeen El-derhalli, Saif Najmeddin and Alain Aoun, for being kind, good listeners, supportive and helpful. Their company has made my journey at HVG fruitful and unforgettable.

Last but by no mean least, it gives me immense pleasure to thank my family, i.e., my mother and my brothers, for their endless love and support. Nothing I say can do justice to how I feel about their support. To my father, whom I lost recently, your advising words (even a few) are striking my thoughts and lighting the long road. My wife, who has been with me in every moment of my Ph.D. tenure, is my source of strength. Without her support, this thesis would never start nor finish. Her consistent love and encouragement kept me going till this point, and even farther. Her sacrifice by quitting her job and MSc studies are not repayable nor forgettable. I would like to mention my kids Yamin, Nada and Jawad, for being the source of my motivation, and bringing happiness and joy to my life. Really, there's never a dull moment when these three are around.

TABLE OF CONTENTS

List of Tables

List of Figures

LIST OF ACRONYMS

AC	Approximate Computing
AMA	Approximate Mirror Adder
ANN	Artificial Neural Network
ASIC	Application-Specific Integrated Circuit
ATCM	Approximate Tree Compressor Multiplier
AXA	Approximate XOR/XNOR Adder
AxCLib	Approximate Library
AXI	Advanced eXtensible Interface
AxMAC	Approximate Multiply Accumulate Unit
BIN	Binary
BIT	Bitstream
BPI	Byte Peripheral Interface
BRAM	Block Random Access Memory
CB	Conditional Block
CFPU	Configurable Floating-Point Multiplier
COE	Coefficient
CPU	Central Processing Unit
DCP	Design CheckPoint
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DPR	Dynamic Partial Reconfiguration
DSP	Digital Signal Processing
DT	Decision Tree

ED	Error Distance
ER	Error Rate
ETM	Error Tolerant Multipliers
FA	Full Adder
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IC	Integrated Circuits
ICAP	Internal Reconfiguration Access Port
InXA	Inexact Adder
IoT	Internet of Things
JPEG	Joint Photographic Experts Group
LSB	Least Significant Bit
LUT	LookUp Table
MAC	Multiply Accumulate Unit
MED	Mean Error Distance
ML	Machine Learning
MRED	Mean Relative Error Distance
MSB	Most Significant Bit
MSE	Mean Square Error
NED	Normalized Error Distance
NMED	Normalized Mean Error Distance
NN	Neural Network
PC	Personal Computer
PDP	Power Delay Product

PR	Partial Reconfiguration
PRR	Partial Reconfigurable Region
PSNR	Peak Signal-to-Noise Ratio
QC	Quality Control
RC	Reconfigurable Core
RED	Relative Error Distance
RM	Reconfigurable Module
RMAC	Runtime Configurable Floating Point Multiplier for AC
RR	Reconfigurable Region
RTL	Register Transfer Level
SoC	System on Chip
SRAM	Static Random Access Memory
SVM	Support Vector Machine
TOQ	Target Output Quality
V2C	Variable-to-Constant
V2V	Variable-to-Variable
VOS	Voltage Overscaling
VS	Virtual Socket
VSM	Virtual Socket Manager

Chapter 1

Introduction

In this chapter, we first introduce the motivation behind this thesis and the problem statement. Then, we present the most relevant related work, followed by our proposed methodology to achieve the primary goal of this thesis. Finally, we outline the main contributions and the organization of this thesis.

1.1 Motivation

The continuing scaling in feature size (process node) has made integrated circuit behavior increasingly vulnerable to soft errors as well as process, voltage, and temperature variations. Thus, the challenge of ensuring strictly exact computing is increasing [3]. On the other hand, present age computing systems are pervasive, portable, embedded, and mobile, which led to an ever-increasing demand for ultra-low power consumption, small footprint, and high-performance systems. Such battery-powered systems are the main pillars in the area of the internet of things (IoT), which do not necessarily need entirely accurate results. *Approximate Computing* (AC), known as best-effort computing, is a nascent computing paradigm that allows us to achieve these objectives by compromising the arithmetic accuracy [4]. Nowadays, many applications, such as image processing, multimedia, recognition, machine learning, communication, big data analysis, and data mining are error-tolerant, and thus can benefit from approximate computing. These applications exhibit *intrinsic error-resilience* due to the following factors [5]: (i) redundant and noisy input data; (ii) lack of golden or single output; (iii) imperfect perception in human sense; and (iv) implementation algorithms with self-healing and error attenuation patterns.

Various approximation techniques, which fall under the umbrella of approximate computing, e.g., voltage over scaling [6], algorithmic approximations [7] and the approximation of basic arithmetic operations [8], have gained a significant research interest, in both academia and industry, such as IBM [9], Intel [10] and Microsoft [11]. Nevertheless, approximate computing is still immature and does not have standards yet, which poses severe bottlenecks and main challenges. Thus, future work should be guided by the following general principles to achieve the best efficiency [5]:

1- Significance-driven approximation: Identifying the approximable parts of an application or circuit design is a great challenge. Therefore, it is critical to distinguish the approximable parts with their approximation settings.

2- Measurable notion of approximation quality: Quality specification and verification of approximate design are still open challenges, where quality metrics are application and user-dependent. In the sequel of this chapter, we will explain a list of quality metrics that are used to quantify the approximation errors.

3- Quality configuration: Error resiliency of applications depends on the applied inputs and the context in which the outputs are consumed.

4- Asymmetric approximation benefits: It is essential to identify the approximable components of the design, which reduces the quality insignificantly while improving efficiency considerably.

For a static approximate design, the approximation error persists during its operational-life time. It restricts approximation versatility and results in under- or over-approximated systems for dynamic input data, causing excessive power usage and insufficient accuracy, respectively. Given the dynamic nature of the applied inputs into static approximate designs, errors are the norm rather than the exception in approximate computing, where the error magnitude depends on the user inputs [12] [13]. On the other hand, the defined tolerable error threshold, i.e., Target Output Quality (TOQ), can be dynamically changed. In both cases, errors with a high magnitude value produced by approximate components in an approximate accelerator, even with a low error rate, have a more significant impact on the quality than those caused by approximate parts with small magnitude. This is in line with the notion of fail-small, fail-rare, or fail-moderate approaches, [14] [15], where error magnitudes and rates should be restricted to avoid high loss in the output-quality. The fail-small technique allows approximations with low error magnitudes and high error rates, while the fail-rare technique allows approximations with low error rates and high error magnitudes [14]. On the other hand, the fail-moderate technique allows approximations with moderate error magnitude and moderate error rate [15]. Thus, the approaches mentioned above limit the design space to prevent approximations with high error rates and high error magnitudes, where such combination degrades the quality loss significantly.

Approximate computing is an immature computing paradigm where its quality assurance is still missing a mathematical model for the impact of approximation on the output quality [5]. Towards this goal, in this thesis, we design a set of energy-efficient approximate multipliers suitable for approximate accelerators. Then, we utilize them to develop a runtime adaptive approximate accelerators based on the fine-grained input data to satisfy a user-defined target output quality (TOQ) constraint. Design adaptation uses a machine learning-based design selector to choose the most suitable approximate design for runtime data dynamically. The target approximate accelerator is implemented with configurable levels and types of approximate multipliers.

1.2 State-of-the-Art

In this section, we briefly review the most relevant literature in the area of approximate accelerators design, as well as quality assurance of their approximated results either by design adaptation or by error compensation, which are closely related to this thesis.

1.2.1 Approximate Accelerators

Hardware accelerators are a special hardware, which are devoted to execute frequently-called functions. Accelerators are more efficient than software running on general-purpose processors. Generally, they are constructed by connecting multiple simple arithmetic modules, e.g., discrete Fourier transform (DFT) and discrete cosine transform (DCT) modules are used in signal and image processing.

The existing literature has proposed the design of approximate accelerators using neural networks [16] or approximate functional units, particularly approximate adders [17] and multipliers [18]. Moreover, several functionally approximate designs for basic arithmetic modules, including adders [8] [19] [20], dividers [21] [22] [23] and multipliers [24] [25] [26], have been investigated for their pivotal role in various applications. These individually designed components are seldom used alone, especially in the computationally intensive error-tolerant applications, which are amenable to approximation. The optimization of accuracy performance at the accelerator level has received little or no attention in the previous literature. Next, we review relevant literature in the area of approximate multipliers, which are intensively used to build approximate accelerators.

Approximate Multipliers

Multipliers are one of the most foundational components for most functions and algorithms in classical computing. As avowed by [27], multiplier components utilize 46% of chip area in most Multiply-accumulate (MAC) modules. Thus, an energy-efficient multiplier design can play a significance role in low-power VLSI system design. However, they are the most energy-costly units compared to other essential central processing unit (CPU) functions, such as register shifts or binary logical operators. Thus, their approximation would introduce an enhancement in their performance and energy, which automatically induces crucial benefits for the whole application. Approximate multipliers have been mainly designed using three techniques, i) approximation in partial product generation: e.g., Kulkarni et al. [28] proposed an approximate 2x2 binary multiplier at the gate level by changing a single entry in the Karnaugh-map with an error rate of 1/16; ii) approximation in partial product tree: e.g., Error Tolerant Multipliers (ETM) [29] divide the input operands into two parts, i.e., the multiplication part for the most significant bits (MSBs) and the non-multiplication part for the least significant bits (LSBs), and thus omitting the generation of some partial products [24]; iii) approximation in partial product summation: Approximate full adder (FA) cells are used to form an array multiplier, e.g., in [20], an approximate mirror adder has been used to develop a multiplier. Similarly, Momeni et al. [25] proposed an approximate compressor for building approximate multipliers, but this multiplier is known to give a non-zero result for zero inputs. Jiang et al. [30] compared the characteristics of different approximate multipliers implemented in VHDL based on the three different techniques mentioned previously.

In [31], Hashemi et al. designed an approximate multiplier, called DRUM, which finds the first leading one from the most significant position in both multiplication operands, and then prunes the size of the required multiplier to reduce the approximation error. It has been demonstrated for producing a near-to-zero mean error for uniformly distributed input. However, applications with other input distributions (e.g., Gaussian) cannot utilize DRUM. Moreover, this design cannot be integrated on general-purpose processors as it needs to set the multiplier size for each application offline. The work in [32] proposed a configurable floating-point multiplier (CFPU), which multiplies input operands depending on the input mantissa. However, such CFPU gives poor accuracy with coarse grain tuning capability. Therefore, the number of applications that could benefit from the approximation is limited. The work in [33] proposed a runtime configurable floating-point multiplier (RMAC) by approximating the mantissa multiplication to a simple addition between the mantissa of the input operands. Despite being runtime configurable with low energy consumption, both designs in [32] and [33] are still unable to completely remove the need for exact multiplication from their designs.

In this thesis, we focus on array multipliers, which are neither the fastest, nor the smallest. Their short wiring, however, gives them a periodic structure with a compact hardware layout. Thus, they are one of the most used in embedded System on Chip (SoC). We therefore designed various 8 and 16-bit approximate array multipliers based on the approximation in partial product summation.

1.2.2 Quality Control of Approximate Accelerators

Managing the quality of approximate hardware designs for dynamically changing inputs has a significant importance to ensure that the obtained results satisfy the required target output quality (TOQ). To the best of our knowledge, there are very few works targeting the assurance of the accuracy of approximate systems compared to designing approximate components. While most prior works focus on *error prediction*, we propose to overcome the approximation error through two different techniques: 1) an input-dependent *self adaptation of design*, and 2) input-dependent *self compensation* of errors. Design adaptation could be implemented in software-based systems by having different versions of the approximate code, while hardware-based systems rely on having various implementations for the functional units. However, concurrently having such functional units diminishes approximation benefits. Thus, dynamic partial reconfiguration could be used to have only a single implementation of the design at any instance of time. In the sequel, we review pertinent literature related to design adaptation, runtime partial reconfiguration and error compensation of approximate computing.

Design Adaptation

There are mainly two approaches for monitoring and controlling the accuracy of the results of approximate accelerators at run time. The first approach suggests to periodically, through *sampling techniques*, measure the error of an accelerator through comparing its outcome with the exact computation performed by the host processor. Then, a re-calibration and adjustment process is performed to improve the quality in subsequent invocations of the accelerator if the error is above a defined range, e.g., Green [34] and SAGE [35]. However, the quality of unchecked invocations cannot be ensured, and the previous quality violations cannot be compensated. The second approach relies on implementing lightweight pre-trained error predictors to expect if the invocation of an approximate accelerator would produce an unacceptable error for a particular input data set [36] [37] [38] [39]. However, the quality checker proposed in [36] is application-specific, and [37] shows a low prediction accuracy for large applications. [38] proposes a quality management framework for approximate computing, by using *multiple lightweight predictors*, that achieve a better energy efficiency than [36] and [37] through avoiding unnecessary rollback recoveries. The authors of [39] proposed a framework supporting an iterative training process, to coordinate the training of the classifier and the accelerator with an intelligent selection of training data.

In general, the work [34] –[39], mainly target controlling software approximation, i.e., loops and functions approximation, through program re-execution, and thus are not applicable for hardware designs. Moreover, they ignore input dependencies and do not consider choosing an adequate design from a set of design choices.

Raha et al. [40] designed a dual-mode, reconfigurable adder block. Similarly, four designs of 4-to-2 compressors are proposed in [41]. Moreover, the deployment of a self-adaptive image filter, which can choose among different degrees of binary adder approximations at run time, is also demonstrated in [42]. However, the approaches, reported in [40] - [42], either have minimal configuration options (e.g., [40]), area overhead (e.g., [41]) or latency overhead (e.g. [42]).

Recently, Xu et al. [43] proposed a runtime reconfigurable manager to select the most suitable approximate design based on the detected input data distribution. However, they utilize approximate accelerators designed at the behavioral level for various coarse-grained expected input data distributions. Furthermore, the proposed approximate circuits heavily depend on the training data used during the approximation process, where not all possible workload distributions can be pre-characterized. Thus, the real workload may differ completely from the training one. Xu et al. [44] also presented a self-tunable runtime adaptive approximate architecture that is suitable for application-specific integrated circuit (ASIC) designs. However, the used approximation techniques are variable-to-variable (V2V) and variable-to-constant (V2C) optimization only. Overall, none of these state-of-the-art techniques, i.e., [34] –[44], exploits the potential of different settings of approximate computing and their adaptations based on a user-specified quality constraint to ensure the accuracy of the individual outputs, which is the main idea proposed in this thesis. Our proposed work is complementary to [43] and [44] in the sense that our designed approximate multipliers are approximated independently of the applied inputs (unlike [43]) and encompass various simplifications (unlike [44]).

Runtime Partial Reconfiguration

Runtime partial reconfiguration is a special feature offered by modern FPGAs that allows designers to reconfigure particular parts of the FPGA during runtime without affecting other parts of the design. This feature allows the hardware to be adaptive to a changing environment, with reduced area, power and configuration time. Contributions on runtime partially reconfigurable systems become practically feasible only recently, as FPGA vendors announced the technical support. For example, an adaptive edge detection filter using dynamic partial reconfiguration has been proposed in [45]. The effectiveness of the DPR feature for edge detection applications is evaluated on the filter with different scenarios varying in size, complexity and intensity of computation, where the resource utilization and timing are evaluated. In [46], design flow for image and signal processing IP (Intellectual Property) cores based on FPGA dynamic partial reconfiguration have been proposed. The benefits of dynamic partial reconfiguration for embedded vision applications have been quantified in [47]. These benefits include area, power, delay and energy reduction. Thus, integrating "runtime partial reconfiguration" with "approximate computing" will significantly ameliorate the efficiency of design approximation.

Error Compensation

Error compensation relies on adjusting the output results of an arithmetic circuit, to reduce the difference between the approximate and the exact values. Such difference would depend on the applied inputs. A fault recovery method utilizing machine learning to ameliorate the effect of permanent faults has been proposed in [48], assuming that the number of unique values of error distance (ED) is meager, i.e., less than 5. However, such an assumption is unrealistic, where the amount of the ED may range from 1 to 2^n, based on fault location, where n is the number of circuit inputs. A different self-compensating accelerator has been proposed in [49] by integrating approximate components with their complementary designs, i.e., having the same error magnitude with opposite polarity. However, obtaining such integral parts is not always guaranteed. Moreover, the approximate design and its complementary may have different characteristics, i.e., area, power, delay and energy. Recently, [50] proposed a scheme for error compensation of signed truncated adders, multipliers and dividers. For that, a padding is added to reduce the average error based on statistical information, under normal distribution, of the targeted arithmetic operations. However, such technique only targets a truncation-based approximate arithmetic circuits.

1.3 Proposed Methodology

The main objective of this thesis is to assure the quality of approximation by two approaches: 1) design adaptation by predicting the most suitable settings of the approximate design to execute the inputs; and/or 2) error compensation by predicting the error magnitude to use in adjusting the output results. The proposed method predicts the *design settings*, or the *error magnitude* based on the applied input data and user preferences, without losing the gains of approximations. We mostly consider the case of approximate accelerators built with approximate functional units such as approximate multipliers, where we develop a library of approximate accelerators with 20 different settings of 8 and 16-bit approximate multipliers.

We propose a comprehensive methodology that addresses the limitations of the current state-of-the-art in terms of fine-grained input dependency, suitability for various approximate modules (e.g., adders, dividers and multipliers) and applicability to both hardware and software implementations. Figure 1.1 provides a general overview of our proposed methodology for design adaptation and/or error compensation. As shown in the figure, the methodology encompasses two phases: (1) an offline phase,

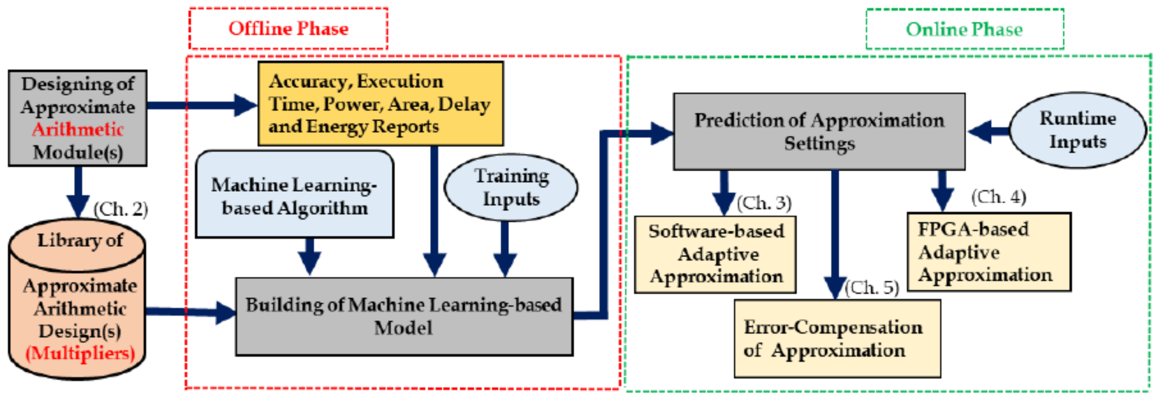

Offline Phase
Online Phase
Designing of Approximate Arithmetic Module(s)
(Ch. 2)
Library of Approximate Arithmetic Design(s) (Multipliers)
Accuracy, Execution Time, Power, Area, Delay and Energy Reports
Machine Learning-based Algorithm
Training Inputs
Building of Machine Learning-based Model
Prediction of Approximation Settings
Runtime Inputs
(Ch. 3)
(Ch. 4)
(Ch. 5)
Software-based Adaptive Approximation
FPGA-based Adaptive Approximation
Error-Compensation of Approximation

Figure 1.1: General Overview of the Proposed Methodology

which is executed once for building a machine learning-based model. Such model predicts the design settings (for design adaptation) or predict the error magnitude (for self compensation); and (2) an online stage, where the machine learning-based model continuously accepts inputs and predicts accordingly based on the runtime inputs. Overall, the proposed methodology encompasses the following main steps:

(1) Building a library of approximate designs: The first step is designing the library of basic functional units, such as adders, multipliers and dividers with different settings, which will be integrated into a quality assured approximate design. For each of these designs, we need to evaluate their characteristics including accuracy, area, power, delay and energy consumption.

(2) Designing of machine learning-based model: On the offline phase, we use supervised learning where both input and desired output data are provided. We employ decision trees and neural networks algorithms to build a model which is used to predict the unseen data, e.g., the design settings for design adaptation, and the error magnitude for self compensation. This step includes generating the training data, pre-processing of the training data, such as, quantization, sampling and reduction. The training inputs are applied exhaustively to an approximate design to create the training data. For n-bit designs with two inputs, the size of the input combinations is 2^{2n}.

(3) Predicting of the approximation settings: In the online phase, the user-specified runtime inputs, i.e., the target output quality and the inputs of the approximate design, are given to the ML-based models to predict approximation-related output, i.e., setting of the adaptive design or error magnitude for self-compensation. The implemented ML-based model should lightweight, i.e., have a high prediction accuracy with fast execution.

(4) Integrating the approximate accelerator into error tolerant applications: For adaptive design, the approximate accelerator, which has been selected by the ML-based model, is adapted within an error-resilient application. Such design could be implemented in software (off-FPGA, as explained in Chapter 3) or in hardware (on-FPGA, as described in Chapter 4). For error compensation, the predicted error magnitude is added to the approximate result to partially reduce the approximation error, as explained in Chapter 5.

We compare the proposed methodology with the state-of-the-art [34] – [44] in terms of several requirements related to hardware and software applicability, data dependency, quality assurance and methodology overhead. Table 1.1 summarizes the comparison between the different approaches for approximate computing quality control. Generally, all methods can control the average quality of the approximation results.

(a) *Hardware/Software*: Existing work [34] – [39] exhibit limited applicability for software approaches only. The approaches in [40] - [44] apply to hardware with limited configurations. The methodology we propose in this thesis applies to both hardware and software approximate applications with approximable components.

(b) *Input Data Dependency*: A distinctive characteristic of our proposed methodology is its data dependency. To the best of our knowledge, none of the previous works targeted fine-grained input-dependency of approximate designs to control the output quality. The proposed approach quantizes the input data, then applies it to a library of approximate designs to generate training data. Another distinctive feature of our approach is the development of a lightweight decision tree and neural network-based models for design selection with a satisfying output accuracy.

(c) *Quality Assurance and Overhead*: The approaches [34] - [39] control the quality of results through rollback and program re-execution. However, they require extra time and out-of-order-execution. The approach in [43] depends on the training data used during the approximation process, which may differ significantly from the real workload, while the method in [44] relies on variable-to-variable (V2V) and variable-to-constant (V2C) approximation techniques only. On the other hand, the proposed approach needs a one-time data generation, training and model building. Then, the design is adapted, with negligible overhead.

1.4 Thesis Contributions

The primary contribution of this thesis is the development of a methodology to guarantee the quality of approximate accelerators. It meets a user-given quality constraint for dynamically changing inputs. It also presents new approaches for utilizing machine learning techniques to build a design selector, i.e., quality controller, to choose the most suitable approximate design based on fine-grained applied inputs and a user-defined quality threshold. In the sequel, we list the main contributions of this work along with references to related publications provided in the Biography at the end of the thesis document.

- A library of 8-bit and 16-bit approximate multipliers based on a large set of approximate full adders (FA) implemented at the transistor level (TSMC65nm). We have evaluated these multipliers based on their power, area, delay and error, and identified the best designs using an image processing benchmark application. Then, we validated them on an image blending application, where the best designs are identified [Bio-Cf5].

Table 1.1: Comparison between Approximate Computing Quality Control Approaches

Characteristics	**Software-based Design ([34] - [39])**	**Hardware-based Design ([40] - [44])**	**ML-based Adaptive Design**
Hardware/Software	Applicable to software approximation only, i.e., loop unrolling, at a very coarse granularity	Applicable to hardware approximation with limited configuration options	Applicable to both hardware and software approximate techniques with multiple approximate components
Input Data Dependency	Completely ignore differences between input data	[43] Workload distribution used during training may not be characterized [44] Use only V2V and V2C input dependent optimization	Significantly relies on the input data for design adaptation
Quality Assurance	Through execution rollback and program re-execution	Quality can not be assured if the real workload differs from the one used for training	Utilizes ML-based models to select the most suitable approximate component
Overhead	Rollback recovery time and cost	[43] Runtime overhead is negligible [44] introduces 8.5% area and 8.1% delay overhead	Overhead of generating training data and build the models Energy and delay overheads are negligible

- A general machine learning-based methodology to assure the quality of approximate accelerators based on the applied inputs and user preferences. The principal advantages of the proposed approach include: (1) fine-grained input-dependent approximation; (2) no missed approximation opportunities; (3) no rollback recovery overhead; (4) applicable to any approximate application with error-tolerant components; and (5) flexibility in adapting various error metrics [Bio-Cf3].

- A fully-automated toolchain is implemented to adapt the approximate accelerator to meet the target output quality (TOQ). It utilizes an input-dependent machine learning-based, i.e., decision tree and neural network, design selector [Bio-Jr2] [Bio-Cf1]. In particular, we extend the adaptive approximate design approach developed in [Bio-Cf3] to have a full software-based implementation.

- An adaptive approximate hardware architecture consisting of various designs implemented on the latest FPGA platforms while exploiting the features of dynamic partial reconfiguration (DPR) is realized [Bio-Jr1]. It represents a hardware-based implementation for the approach we developed in [Bio-Cf3].

- A novel approach to explore a decision tree-based quality assurance, in approximate accelerators, by *error compensation* without losing the gains of approximations. The method is lightweight in terms of its area, delay and power where complementary approximate modules are not required [Bio-Cf2].

1.5 Thesis Organization

The remainder of this thesis is organized as follows: In Chapter 2, we design and evaluate a set of approximate arithmetic modules including, full adders, compressors, 8 and 16-bit multipliers. Then, we adopt a library of 8 and 16-bit approximate multipliers, with 20 different settings of each, to be integrated into our proposed methodology.

In Chapter 3, we describe the proposed methodology of quality assurance by design adaptation. We explain how we use the input data and user preferences to build a machine learning-based design selector. Then, we utilize the design selector to come up with a runtime adaptive approximate design that assures the final output quality. The feasibility and efficiency of the *software* implementation of the proposed quality assurance methodology will be demonstrated on two applications of image processing, i.e., image blending and image filtering. In Chapter 4, we utilize dynamic partial reconfiguration of FPGA for the *hardware* implementation of the proposed methodology.

Based on our general methodology of quality assurance utilizing a set of approximate designs, in Chapter 5, we propose to assure the quality of a single approximate design, rather than a library, by error compensation. For that, we suggest and design an error compensation module, which is integrated within the architecture of approximate hardware accelerators, to efficiently reduce the accumulated error at its output. The proposed module utilizes lightweight decision trees to capture input dependency of the error. Finally, Chapter 6 summarizes this thesis and outlines potential future research directions.

Chapter 2

Library of Approximate Multipliers

This chapter explains the approximate library of multipliers that we need for the realization of the proposed thesis methodology.We first provide an overview of an approach for designing energy-efficient approximate multipliers. We then define a number of error metrics, which will be used through out the thesis. Next we develop several approximate multiplier building blocks and designs and evaluate their characteristics.

2.1 Introduction

Functional approximation [28], in hardware, mostly deals with the design of approximate arithmetic units, such as adders, dividers and multipliers, at different abstraction levels, i.e., transistor, gate, register transfer, and application. Some notable approximate adders include speculative adders, segmented adders, carry select adders and approximate full adders [51]. The transistor-level approximation provides the highest flexibility due to the ability to tweak most of the design parameters at this level. Various approximate full adders (FA) at the transistor level have been proposed, including the mirror adders [8], the XOR/XNOR based FA [19] and the inexact FA [52]. On the other hand, most of the approximate multipliers have been designed at the higher levels of abstractions, i.e., gate and application.

The design space for approximate multipliers based on different approximate FAs and compressors is quite huge. However, it is not easy to select the most suitable design for a specific application. Figure 2.1 presents an overview of our design approach to build different approximate multipliers and compare their design metrics to select the most suitable design. It consists of the following steps:

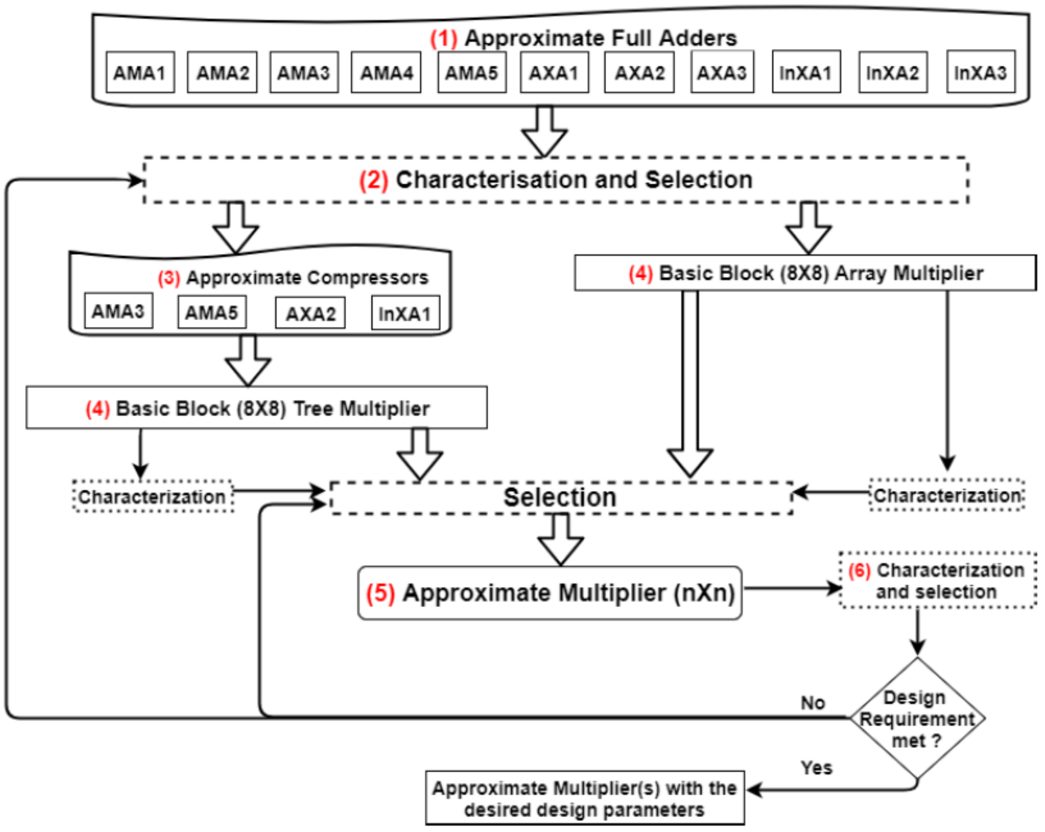

(1) Approximate Full Adders
AMA1
AMA2
AMA3
AMA4
AMA5
AXA1
AXA2
AXA3
InXA1
InXA2
InXA3
(2) Characterisation and Selection
(3) Approximate Compressors
AMA3
AMA5
AXA2
InXA1
(4) Basic Block (8X8) Array Multiplier
(4) Basic Block (8X8) Tree Multiplier
Characterization
Selection
Characterization
(5) Approximate Multiplier (nXn)
(6) Characterization and selection
Design Requirement met ?
No
Yes
Approximate Multiplier(s) with the desired design parameters

Figure 2.1: Proposed Approach for Designing Approximate Multipliers

1. Building a library of elementary approximate FAs using the TSMC65nm technology in Cadence Spectre: We use the default transistors of this technology to design 11 approximate FA designs comprising of 5 mirror FAs [8], 3 XOR/XNOR gate FAs [19] and 3 inexact FAs [52].

2. Characterization and early space reduction: We perform area, power, latency and quality characterizations of different approximate FAs to filter out non-Pareto designs.

3. Building a library of approximate compressors: We develop a Cadence library of approximate compressors using the optimal approximate FA, as recommended by [8].

4. Building approximate multipliers basic blocks: Based on the approximate FAs and compressors, we design various approximate 8-bit array and tree multipliers, respectively. These proposed designs are related to ripple-carry array multiplier architecture, which is the most power-efficient conventional architecture [53]. Moreover, the approximate FAs are applicable to sequential multipliers with a single n-bit adder for partial product accumulation.

5. Designing target approximate multipliers: Based on different configurations of 8-bit approximate multipliers, we design and charactersize the target multiplier modules, i.e, 16 and 32-bit multipliers.

6. Selection of design points: Considering the required quality constraints of a specific application, we select a subset of power-efficient design points.

To evaluate the efficiency of the proposed approximate designs, power consumption and area represented by the number of transistors used, are measured, and circuit performance is evaluated using the maximum delay between changing the inputs and observing the output(s). Besides these traditional design metrics, accuracy is also an important design constraint in approximate computing. We choose several commonly used error metrics in approximate computing, as explained in the next section, to quantify errors and measure accuracy in the proposed approach. As shown in

Figure 2.1, *the characterization and selection* process is applied at multiple steps to different components during the design flow. Characterization aims to find the design characteristics of the circuits, including area, power consumption, performance, error metrics, and other derived metrics such as Power-Delay-Product (PDP). The design selection process for the evaluated approximate designs depends on the application domain of the given circuit.

2.2 Error Metrics

In this section, we briefly present the main error metrics which emerged with approximate computing, to understand better how to quantify approximation errors.

Approximation introduces accuracy as a new design metric. Thus, several application dependent error metrics are used to quantify approximation errors and evaluate design accuracy [54] [52]. For example, considering an approximate design with two inputs, i.e., X and Y, of n-bit each, where the exact result is (P), and the approximate result is (P'), these error metrics include:

- Error Distance (ED): The arithmetic difference between the exact output and the approximate output for a given input. ED can be presented by:

$$ED = |P - P'| \tag{2.1}$$

- Error Rate (ER): Also called error probability, is the percentage of erroneous outputs among all outputs, and it is expressed as:

$$ER = \frac{Number\ of\ erroneous\ computations}{Total\ number\ of\ computations} \tag{2.2}$$

- Mean Error Distance (MED): The average of ED values for a set of outputs obtained by applying a set of inputs. MED is a useful metric for measuring the implementation accuracy of multiple-bit circuit design, and got as:

$$MED = \frac{1}{2^{2n}} \sum_{i=1}^{2^{2n}} |ED_i| \tag{2.3}$$

- Normalized Error Distance (NED): The normalization of MED by the the maximum value of error that an unreliable circuit can have (P_{Max}). NED is an invariant metric independent of the size of the circuit. Therefore, it is used for comparing circuits of different sizes, and expressed as:

$$NED = \frac{MED}{P_{Max}} \tag{2.4}$$

- Relative Error Distance (RED): The ratio of ED to the accurate output, given by:

$$RED = \frac{ED}{P} = \frac{|P - P'|}{P} \tag{2.5}$$

- Mean Relative Error Distance (MRED): The average value of all possible relative error distances (RED):

$$MRED = \frac{1}{2^{2n}} \sum_{i=1}^{2^{2n}} |RED_i| \tag{2.6}$$

- Mean Square Error (MSE): It is defined as the average of the squared ED values:

$$MSE = \frac{1}{2^{2n}} \sum_{i=1}^{2^{2n}} |P_i - P_i'|^2 = \frac{1}{2^{2n}} \sum_{i=1}^{2^{2n}} |ED_i|^2 \tag{2.7}$$

- Peak Signal-to-Noise Ratio (PSNR): The peak signal-to-noise ratio is a fidelity metric used to measure the quality of the output images, given by:

$$PSNR = 10 * log_{10}(\frac{MAX^2}{MSE}) \tag{2.8}$$

It indicates the ratio of the maximum pixel intensity to the distortion, where MAX stands for the maximum value of a pixel in the images, i.e., 255 for 8-bit image.

The presented metrics are not mutually exclusive, where one application may use several quality metrics. A detailed analysis of various error metrics, i.e., ER, MED, NED, MRED and PSNR, based on an exhaustive simulation for approximate designs, are explained in the next sections of this chapter as well as subsequent chapters of the thesis.

2.3 Approximate Building Blocks

Low power and energy-efficient approximate multipliers are generally constructed by replacing the accurate building blocks, i.e., full adders and compressors, with their approximate implementation. Next, we describe the basic building blocks we implemented to design our approximate multipliers.

2.3.1 Approximate Full Adders

We consider five approximate mirror adders, i.e., AMA1, AMA2, AMA3, AMA4 and AMA5 [8], three approximate XOR/XNOR based full adders, i.e., AXA1, AXA2 and AXA3 [19] and three inexact adder cells, i.e., InXA1, InXA2 and InXA3 [52]. All of them have been proposed and designed at the transistor-level with energy efficient

Table 2.1: Design Characteristics of Approximate Full Adders

FA Type	Size	Power(nw)	Delay(ps)	ER	PDP(fJ)
Exact FA	28	763.3	244	0	186.25
AMA1	20	612	195	2	119.34
AMA2	14	561.1	366	2	205.36
AMA3	11	558.1	360	3	200.92
AMA4	15	587.1	196	3	115.07
AMA5	8	412.1	150	4	61.82
AXA1	8	676.2	1155	4	781
AXA2	6	358.7	838	4	300.59
AXA3	8	396.5	1467	2	582
InXA1	6	410	740	2	303.4
InXA2	8	355.1	832	2	295.44
InXA3	6	648	767	2	753.5

implementation. Table 2.1 shows the design characteristics of the 11 considered approximate FAs including size (A), power (P), delay (D), number of erroneous outputs (E), which indicates the number of cases where at least one output (Cout or Sum) is wrong, and PDP. As shown in Table 2.1, all approximate FAs are Pareto-points, where they provide less area and power consumption compared to the exact design at the cost of compromising accuracy. Some of the FA designs have an enhanced performance (reduced delay), while other models have degraded performance due to the internal structure and node capacitance. In [55], AMA5 is considered as a *wire* with zero area and zero power consumption. However, this is unrealistic as the output of AMA5 has to drive other signals. Thus, we used two buffers instead of two wires to design it.

Assuming that the characteristics of approximate FAs are linearly applied to approximate arithmetic circuits (adders and multipliers), there is no single approximate FA, which is superior in all aspects. Therefore, we propose to use a *fitness function* to evaluate the designs based on its design metrics.

$$Fitness = C1 \times A + C2 \times P + C3 \times D + C4 \times E \tag{2.9}$$

where *C1*, *C2*, *C3* and *C4* are design coefficients within the range [0,1] to indicate the importance of a specific design metric, i.e., E equals zero for the exact design, and P is small for low power designs. The fitness of the approximate circuit depends on the application resiliency and input data distribution. The minimal fitness value is preferred since the goal is to minimize design metrics. We will utilize the 11 Pareto-design approximate FAs as primary building cells to construct approximate array multipliers.

2.3.2 Approximate Compressors

Full adders with 3 inputs and 2 outputs are called 3-to-2 compressors. Higher-order compressors, e.g., 5-to-3 and 8-to-4 [25], allow constructing high-speed tree multipliers. Therefore, we develop approximate compressors based on FAs, e.g., an 8-to-4 binary compressor is depicted in Figure 2.2, for illustration purposes. Tables 2.2 and 2.3 show the power consumption and the area of different approximate compressors implemented based on approximate FAs, respectively. The area for approximate compressors exhibits a linear relationship with the area of FAs. However, it does not look natural to obtain a closed-form analytical expression for the power consumption.

A hybrid approximate multiplier, i.e., based on approximate compressors with different approximation degrees and types, would have an ample design space. Therefore, to show the effectiveness of designing approximate compressors based on approximate FAs, we chose four FAs only. These FAs have superior design metrics. The best approximate FA in terms of delay and PDP is *AMA5*, and in terms of power and area is *AXA2*. Also, the best FA with a low error rate is *InXA1*. *AMA3* has moderate characteristics regarding the area, power, delay, and the number of errors. These selected FAs are used to design approximate high-order compressors, which in

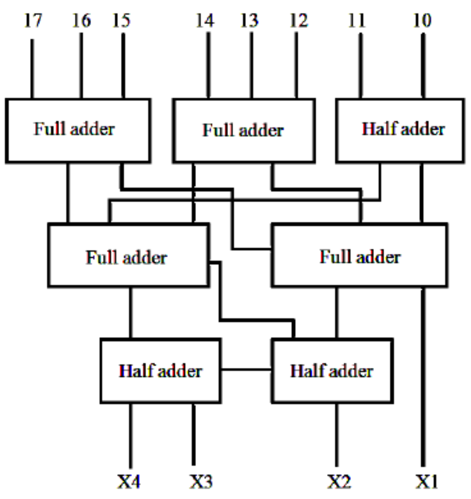


Figure 2.2: 8-to-4 Compressor Design

Table 2.2: Power Consumption (μw) for Different Compressors

	Compressor Type					
FA Type	**3-2**	**4-3**	**5-3**	**6-3**	**7-3**	**8-4**
Exact FA	0.562	1.469	1.659	1.466	1.355	2.198
AMA1	0.5474	0.9696	1.494	0.9258	1.138	1.65
AMA2	0.4525	1.224	1.189	1.536	1.321	1.609
AMA3	0.4489	0.6813	1.378	1.073	0.6157	0.9114
AMA4	0.5228	0.9988	1.176	1.183	1.037	1.449
AMA5	0.4802	0.9333	1.199	1.023	0.8753	1.791
AXA1	0.4511	1.586	1.128	1.562	1.521	2.142
AXA2	0.4584	1.296	1.563	0.8141	0.7489	2.77
AXA3	0.3544	1.349	1.429	1.231	0.4742	2.316
InXA1	0.1823	1.569	0.9423	0.4842	1.296	2.413
InXA2	0.5018	1.324	2.114	0.3441	0.8087	3.844
InXA3	0.5504	1.345	1.234	0.7866	1.636	5.27

turn can be used for creating approximate tree multipliers. However, these selected compressors are not guaranteed to be the optimal ones. But, they exhibit better features compared to the exact designs.

Table 2.3: Area (number of transistors) for Different Compressors

	Compressor Type					
FA Type	**3-2**	**4-3**	**5-3**	**6-3**	**7-3**	**8-4**
Exact FA	28	56	70	98	112	154
AMA1	20	48	54	74	80	122
AMA2	14	42	42	56	56	98
AMA3	11	39	36	47	44	86
AMA4	15	43	44	59	60	102
AMA5	8	36	30	38	32	74
AXA1	8	36	30	38	32	74
AXA2	6	34	26	32	24	66
AXA3	8	36	30	38	32	74
InXA1	6	34	26	32	24	66
InXA2	8	36	30	38	32	74
InXA3	6	34	26	32	24	66

2.4 Approximate Multipliers

In this section, we use the approximate FAs and compressors, described earlier, to design 8 and 16-bit approximate array and tree-based multipliers, respectively, which will act as our basic blocks for developing high-order multipliers, i.e., 32-bit multipliers.

2.4.1 8-bit Multiplier Basic Blocks

8-bit Approximate Array Multipliers: An n-bit array multiplier is composed of n^2 AND gates for partial product generation and *n-1* n-bit adders for partial product accumulation [56]. The design space of an $n \times n$ approximate array multiplier is quite huge since it depends on the type of FA used in the array, and the number of approximate FAs used in the n-bit adder. We use all 11 Pareto approximate FAs, described in Section 2.3.1, to construct 8-bit approximate array multipliers, based on only one FA type per design to avoid an exponential growth of the design space. Regarding the degree of approximation, we use two options: i) all FAs are

Table 2.4: Characteristics of 8-bit Approximate Array Multipliers

Multiplier Name	ER	NMED	Delay(ps)	POWER(μW)	SIZE
EE	0	0	527	31.41	1456
EM1	$9.70\text{x}10^{-1}$	$3.93\text{x}10^{-3}$	527	24.17	1288
M1M1	$9.96\text{x}10^{-1}$	$2.05\text{x}10^{-1}$	865	14.75	1072
EM2	$9.90\text{x}10^{-1}$	$3.52\text{x}10^{-3}$	557	22.97	1162
M2M2	1.00	$2.58\text{x}10^{-1}$	600	14.4	784
EM3	$9.99\ \text{x}10^{-1}$	$7.26\text{x}10^{-3}$	605	24.95	1099
M3M3	1.00	$2.64\text{x}10^{-1}$	598	15.31	640
EM4	$9.70\text{x}10^{-1}$	$1.71\text{x}10^{-3}$	573	21.85	1183
M4M4	$9.96\text{x}10^{-1}$	$9.86\text{x}10^{-2}$	313	11.17	832
EM5	$9.30\text{x}10^{-1}$	$1.56\text{x}10^{-3}$	573	22.15	1036
M5M5	$9.90\text{x}10^{-3}$	$1.27\text{x}10^{-1}$	250	10.69	496
EX1	$9.71\text{x}10^{-3}$	$3.21\text{x}10^{-3}$	546	31.86	1036
X1X1	$9.96\text{x}10^{-3}$	$1.61\text{x}10^{-1}$	558	21.33	496
EX2	1.00	$2.89\text{x}10^{-3}$	569	23.38	994
X2X2	1.00	$2.31\text{x}10^{-1}$	250	13.91	400
EX3	$6.15\text{x}10^{-1}$	$5.35\text{x}10^{-3}$	536	25.54	1036
X3X3	$9.96\text{x}10^{-1}$	$2.50\text{x}10^{-1}$	197	15.06	496
EIn1	$6.15\text{x}10^{-1}$	$4.91\text{x}10^{-3}$	517	26.07	994
In1In1	$8.54\text{x}10^{-1}$	$1.56\text{x}10^{-1}$	403	14.82	400
EIn2	$5.84\text{x}10^{-1}$	$2.76\text{x}10^{-3}$	528	28.79	1036
In2In2	$8.26\text{x}10^{-1}$	$1.27\text{x}10^{-1}$	340	12.56	496
EIn3	$9.90\text{x}10^{-1}$	$3.52\text{x}10^{-3}$	556	27.96	994
In3In3	1.00	$2.58\text{x}10^{-1}$	404	23.92	400

approximate; and ii) FAs that contribute to the least significant 50% of the resultant bits are approximated to maintain acceptable accuracy as recommended by [8] [20] [57]. Thus, we design, evaluate (based on sampled inputs) and compare 22 different options for 8-bit approximate array multipliers, as shown in Table 2.4, using the TSMC65nm technology.

The "Multiplier Name" in Table 2.4 consists of two parts, i.e., the name of the adder used for the most significant and least significant part. For example, in EM1, the most significant part is based on an exact adder (E), and the least significant part is based on mirror adder 1 (M1). Similarly, the least significant part for EX2 and EIn2 is based on XOR/XNOR 2 FA (X2) and Inexact 2 FA (In2), respectively. The

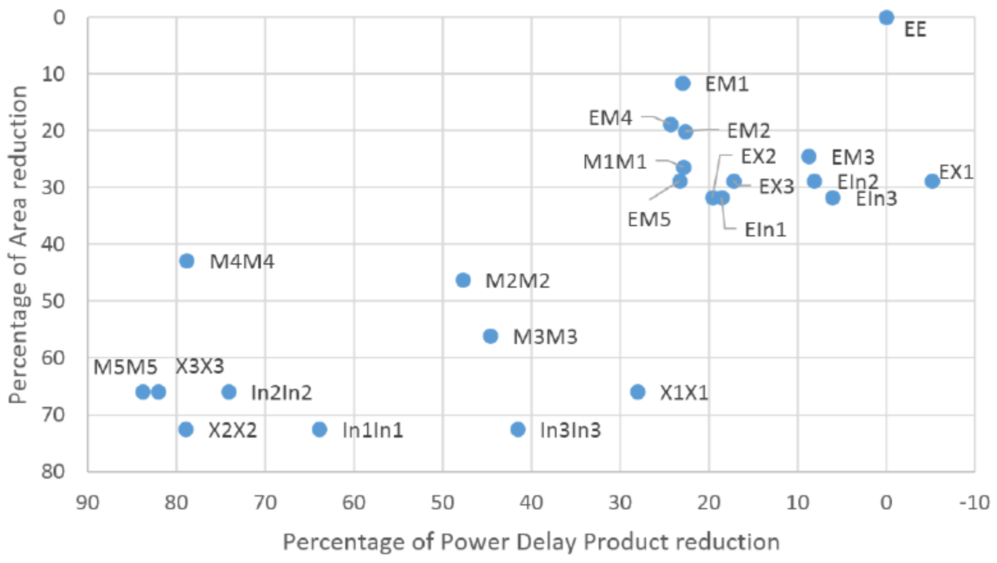


Figure 2.3: Area and PDP Reduction of 8-bit Approximate Array Multipliers

approximate multiplier size exhibits a linear relationship with the degree of approximation. As shown in Table 2.4, there is no single design that is superior in all design metrics. Therefore, a Pareto-analysis for various design characteristics is used for the different proposed designs. Figure 2.3 shows the area and PDP reduction of 8-bit approximate array multipliers. The best designs are located in the bottom left corner. *M5M5* is a Pareto-design with PDP reduction (84%) and area reduction (65%). The design *X3X3* is Non-Pareto because it has the same area reduction as the *M5M5* but with a smaller PDP reduction. However, we have to consider other *error metrics*. Some designs, such as *EX1*, have increased PDP due to excessive switching activity compared to the original design.

8-bit Approximate Tree Multipliers: The design space for 8-bit approximate tree multipliers is also quite large, depending on the compressor type and approximation degree [58]. To avoid the exponentially growing design space, we choose to use compressors of the same type in the multiplier design. Also, we use two options for approximation degree: i) all compressors are approximate; and ii) only those

Table 2.5: Characteristics of 8-bit Approximate Tree Multipliers

Multiplier Name	ER	NMED	Delay(ps)	POWER(μW)	SIZE
CEE	0.00	0.00	508	21.98	1218
CEM3	1.00	9.30x10^{-3}	537	19.65	912
CM3M3	1.000	2.16x10^{-1}	560	16.27	606
CEM5	9.79x10^{-1}	2.40x10^{-3}	356	18.63	858
CM5M5	9.99x10^{-1}	8.18x10^{-2}	282	13.99	498
CEX2	9.97x10^{-1}	5.70x10^{-3}	525	23.52	822
CX2X2	1.00	1.38x10^{-1}	513	22.6	426
CEIn1	8.73x10^{-1}	4.80x10^{-3}	505	25.12	822
CIn1In1	9.75x10^{-1}	7.81x10^{-2}	500	26.89	426

compressors that contribute to the lowest significant 50% of the resultant bits are approximated to maintain acceptable accuracy. Thus, based on the 4 shortlisted compressors, explained in Section 2.3.2, we compared 8 options for 8-bit approximate tree multipliers, and the results are given in Table 2.5. The name of the multiplier consists of three parts. For example, CEM1 represents a compressor-based multiplier (C), where the most significant part is based on an exact compressor (E), and the least significant part is composed of the mirror adder 1 (M1) based compressor.

As shown in Table 2.5, no design is superior in all metrics, but some designs are the best in a few parameters. As depicted in Figure 2.4, the best models are on the left bottom corner, i.e., *CM5M5* and *CX2X2* are Pareto-designs while *CEM5* is a non-Pareto-design.

2.4.2 16-bit Higher-Order Multiplier Configuration

The 8-bit multiplier basic modules can be used to construct higher-order target multiplier modules. Next, we use the example of designing a 16-bit multiplier to illustrate this process. The partial product tree of the 16-bit multiplication can be broken down into four products of 8-bit modules, which can be executed concurrently, as shown in Figure 2.5 of a recursive multiplier.

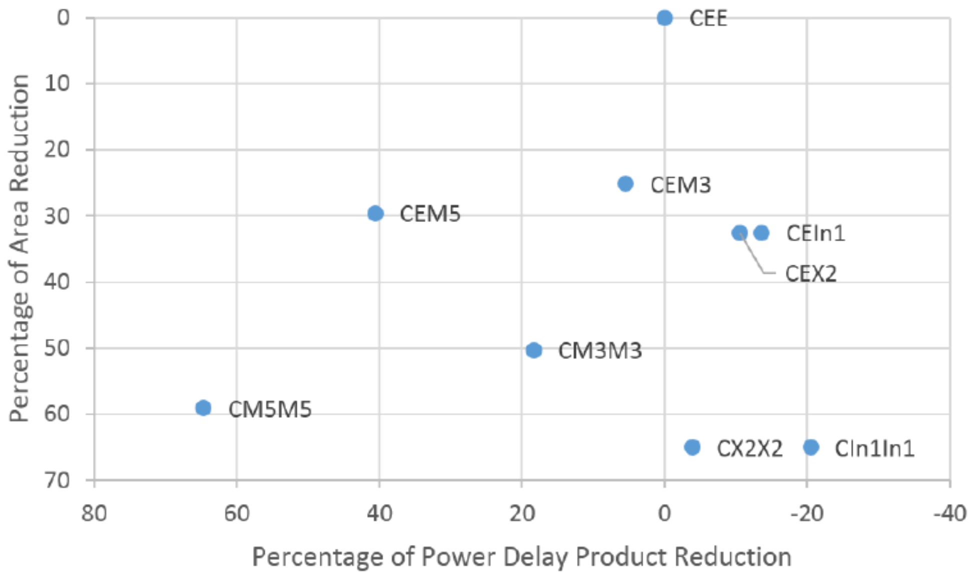


Figure 2.4: Area and PDP Reduction of 8-bit Approximate Tree Multipliers

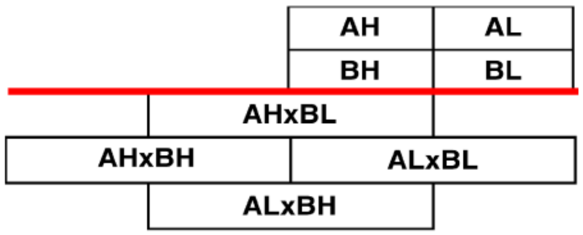


Figure 2.5: A High-Order Recursive Multiplier

In the case of high requirements of accuracy, we use an exact 8-bit multiplier for the three most significant products, i.e., $AH \times BH$, $AH \times BL$, and $AL \times BH$, and anyone of the approximate designs can be used for the least significant product, i.e., $AL \times BL$. For low accuracy requirements, only one 8-bit exact multiplier can be used for the most significant product, i.e., $AH \times BH$, and any of the other approximate designs can be used for the three least significant products, i.e., $AH \times BL$, $AL \times BH$, and $AL \times BL$. Modules that contribute to the lowest significant 50% of the resultant bits are approximated to maintain accuracy as recommended by [8] [20] [57][59].

We choose to design 16-bit multipliers with an exact $AH \times BH$ multiplier, and with exact MSBs and approximate LSBs for $AH \times BL$ and $AL \times BH$, and a fully imprecise or approximate in LSBs only $AL \times BL$. Any other approximation degree can be found based on the required quality function, i.e., maximum error, area, power or delay. Therefore, when we explain the 16-bit multipliers, we eliminate the types of $AH \times BH$, $AH \times BL$ and $AL \times BH$ from the name, and only the type of $AL \times BL$ is used.

16-bit Approximate Array Multipliers: Table 2.6 shows the simulation results for 16-bit approximate array multipliers, which show similarities with those in Table 2.4. The multiplier name is based on the type of $AL \times BL$ module. Fully approximate designs exhibit minimal delay due to reduced circuit complexity. Generally, models based on approximate mirror adders have the lowest power consumption, due to the elimination of static power dissipation. Since the design size grows linearly with the FA size, fully approximate designs based on six transistors cells have the smallest area. As depicted in Figure 2.6, the best designs are on the lower-left corner, i.e., *16In1In1* and *16In3In3* are Pareto-designs while *16M4M4* is a non-Pareto design.

16-bit Approximate Tree Multipliers: Table 2.7 displays the characterization for 16-bit approximate tree multipliers, and show similarities to Table 2.5. As depicted in Figure 2.7, i.e., *16CEM5*, *16CEIn1* and *16CM5M5* are all Pareto-designs while *16CEM3* is a non-Pareto design.

2.5 Discussion and Application

We implemented the considered approximate multipliers using Cadence's Spectre based on the TSMC65nm process, with $V_{dd} = 1.0V$ at T=27C°. Independent voltage sources provide the circuit inputs, and a load of 10fF is utilized. We then evaluated their design characteristics, i.e., area, power and delay. As shown in Tables 2.4 and

Table 2.6: Characteristics of 16-bit Approximate Array Multipliers

Multiplier Name	ER	NMED	Delay(ps)	POWER(μW)	SIZE
16EE	0	0.00	514	156.8	5824
16EM1	9.4x10^{-1}	7.69x10^{-10}	534	130.1	5320
16M1M1	1.76x10^{-2}	1.10x10^{-11}	526	118.4	5104
16EM2	1	1.82x10^{-5}	533	128.4	4942
16M2M2	1	2.28x10^{-5}	477	116.5	4562
16EM3	1	6.16x10^{-5}	519	131.6	4753
16M3M3	1	6.45x10^{-5}	490	120.4	4294
16EM4	9.285x10^{-1}	5.04x10^{-10}	522	118.8	5005
16M4M4	9.793x10^{-1}	5.11x10^{-10}	506	105.1	4654
16EM5	9.335x10^{-1}	5.30x10^{-10}	533	119	4564
16M5M5	9.335x10^{-1}	5.30x10^{-10}	535	105.1	4022
16EX1	9.509x10^{-1}	7.43x10^{-10}	513	154.9	4564
16X1X1	9.793x10^{-1}	8.34x10^{-10}	520	138.5	4024
16EX2	1	6.07x10^{-6}	521	138	4438
16X2X2	1	9.11x10^{-6}	514	127.4	3844
16EX3	9.646x10^{-1}	1.09x10^{-9}	515	134	4564
16X3X3	9.793x10^{-1}	1.27x10^{-9}	518	121.8	4024
16EIn1	5.239x10^{-1}	5.04x10^{-10}	519	134.2	4438
16In1In1	6.09x10^{-1}	5.43x10^{-10}	408	121.7	3844
16EIn2	2.137x10^{-1}	1.03x10^{-10}	537	146.9	4564
16In2In2	4.29x10^{-1}	1.42x10^{-10}	500	126.4	4024
16EIn3	1	1.82x10^{-5}	527	157.6	4438
16In3In13	1	2.28x10^{-5}	412	153.2	3844

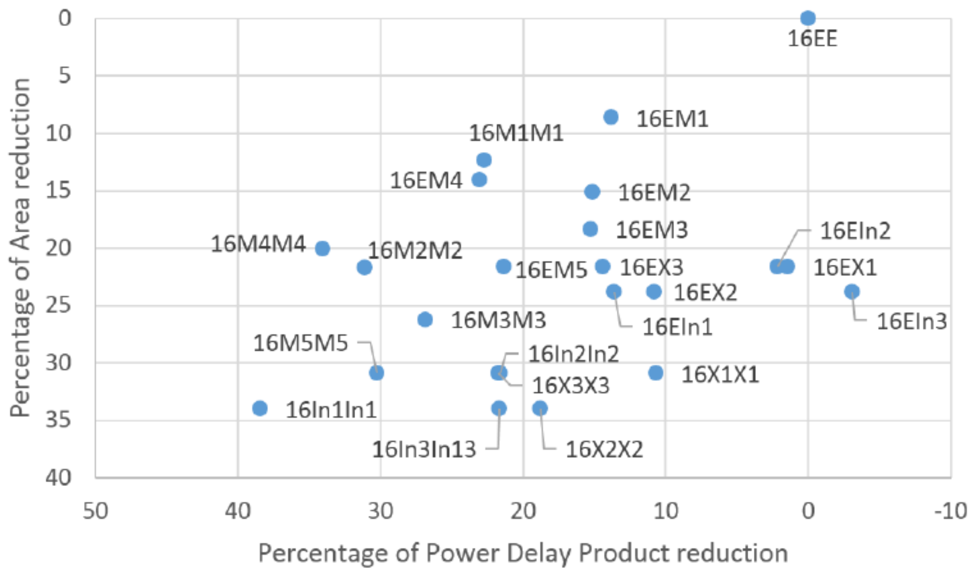


Figure 2.6: Area and PDP Reduction of 16-bit Approximate Array Multipliers

Table 2.7: Characteristics of 16-bit Approximate Tree Multipliers

Multiplier Name	ER	NMED	Delay(ps)	POWER(μW)	SIZE
16CEE	00	0.0	680	100.8	4872
16CEM3	1.00	$3.00\text{x}10^{-3}$	663	93.57	3954
16CM3M3	1.00	$3.10\text{x}10^{-3}$	693	90.6	3648
16CEM5	$9.41\text{x}10^{-1}$	$2.52\text{x}10^{-8}$	585	92.48	3792
16CM5M5	$9.79\text{x}10^{-1}$	$2.56\text{x}10^{-8}$	670	86.98	3432
16CEX2	1.00	$2.30\text{x}10^{-3}$	685	115	3684
16CX2X2	1.00	$2.40\text{x}10^{-3}$	671	114.3	3288
16CEIn1	$8.22\text{x}10^{-1}$	$4.83\text{x}10^{-8}$	516	112.5	3684
16CIn1In1	$9.04\text{x}10^{-1}$	$4.98\text{x}10^{-8}$	527	114.3	3288

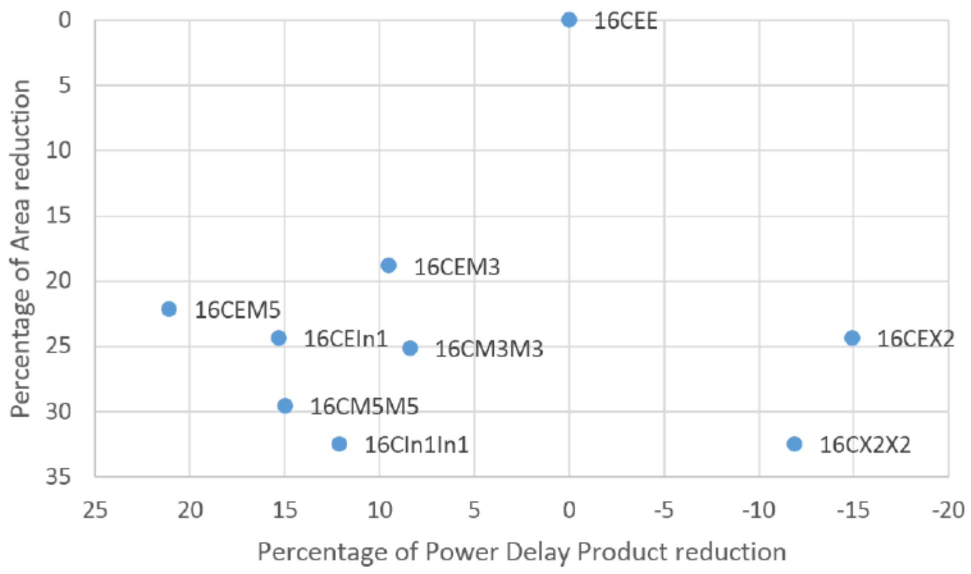


Figure 2.7: Area and PDP Reduction of 16-bit Approximate Tree Multipliers

2.5, the 8-bit exact tree multiplier exhibits lower delay, power and size compared to the 8-bit correct array multiplier.

Different multiplier designs, based on *AMA5*, have the lowest delay and power consumption, due to the basic structure of the FA cell, which is composed of two buffers only. Also, they have the lowest NMED and small size. Regarding accuracy, the designs based on *InXA1* have low ER and NMED. Similarly, the designs based on the six transistors FA, have minimal size. Thus, we can observe that the characteristics of approximate FA are generally translated in the corresponding approximate multipliers as well. In terms of architecture, we found that the tree multiplier designs tend to have a lower power consumption than array multipliers, especially the designs based on low power consuming FAs, such as *AMA3* and *AMA5*. In terms of the 8-bit sub-module placement to form higher-order multipliers, with a fixed configuration for $AH \times BH$, $AH \times BL$ and $AL \times BH$ sub-module, we have noticed that ER and NMED increase, while the size, power consumption and delay decrease for designs with a high degree of approximation in $AL \times BL$.

Compared to the 24 different designs reported in [30], 92% of the models have ERs close to 100%, while only 80% of our proposed designs have high ERs. Regarding NMED, almost all our designs have a value less than 10^{-5}, which is the minimum value reported by the 24 approximate designs in [30]. Comparing the PDP reduction, most of the designs in [30] have a high PDP reduction because they are based on the truncation and a high degree of approximation. However, our models are superior in PDP reduction for designs with a high degree of approximation.

Image Blending Application

In order to validate and demonstrate the usefulness of the proposed approximate multipliers, we use them in a real-life image processing application (i.e., image multiplication). In previous sections, we used *Cadence Spectre* to build the circuits and evaluate their area, performance and power consumption. Now, for experimentation purposes, and in order to run an exhaustive simulation, we use MATLAB to evaluate error metrics for image processing. To this end, we have modeled the same approximate multiplier circuit architectures in MATLAB and run an exhaustive simulation. The quality of image blending is measured by signal to noise ratio (SNR), which is a measure for image fidelity and compares the level of a desired image to the level

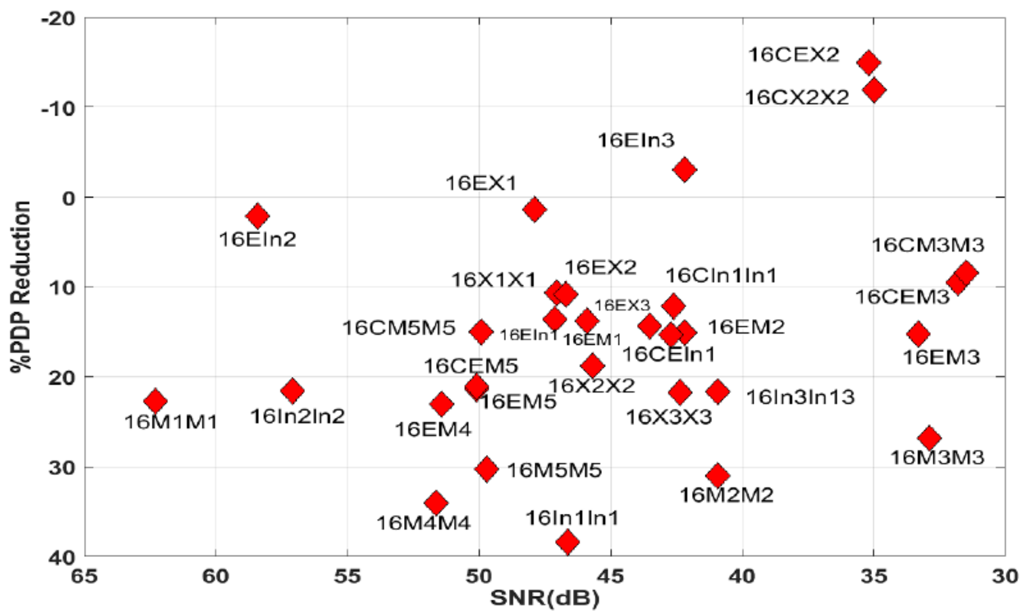


Figure 2.8: %PDP Reduction and SNR of Multipliers

of approximation error, as given in Equation 2.8. Figure 2.8 shows the values of the SNR obtained based on various approximate multipliers and the percentage of PDP reduction for each approximate multiplier. Designs on the bottom left corner, have the highest PDP reduction and the best quality (high SNR). Generally, all the designed multipliers have an acceptable average output quality with noticeable power savings.

2.6 Approximate Library (AxCLib)

In Section 2.4, we evaluated various approximate multipliers which have been designed based on three identified decisions. These 8 and 16-bit designs are implemented using Cadence's Spectre based on TSMC65nm process. Based on the obtained results, we have selected the most energy-efficient ones to be used in the "library of approximate designs" of our proposed methodology. These designs have following features: (1) they utilize five types of full adders which are called "approximate mirror adders",

Table 2.8: Library of 20 Static Approximate Designs based on *Degree* and *Type* Knobs

Approximate Design		**Degree**			
		D1	**D2**	**D3**	**D4**
Type	**AMA1**	*Design1*	*Design2*	*Design3*	*Design4*
	AMA2	*Design5*	*Design6*	*Design7*	*Design8*
	AMA3	*Design9*	*Design10*	*Design11*	*Design12*
	AMA4	*Design13*	*Design14*	*Design15*	*Design16*
	AMA5	*Design17*	*Design18*	*Design19*	*Design20*

i.e., Type = $\{AMA1, AMA2, AMA3, AMA4, AMA5\}$, chosen from the low power approximate FAs [8]; (2) their architecture is an array; and (3) their approximation degrees have four options, i.e., Degree = $\{D1, D2, D3, D4\}$, where D1 has 7 bits approximated out of the 16-bit result, while D2, D3, and D4 have 8, 9, and 16 approximate bits, respectively. Table 2.8 shows our library of approximate multipliers based on *Degree* and *Type* knobs, i.e., $ApprxMul = \{Design1, ..., Design20\}$. Also, the library includes the exact design to be used whenever the required TOQ cannot be satisfied.

To be consistent with the majority of the related work, we have implemented the library of approximate designs, shown in Table 2.8, at the register transfer level (RTL), in synthesizable VHDL. The general proposed methodology which is shown in Figure 1.1, is adaptable, i.e., applicable to approximate functional units other than multipliers, e.g., approximate multiply-accumulate units [60] and approximate meta-functions [61].

Next, we analyze various characteristics, i.e., accuracy, area, delay, energy and power consumption, of our approximate library. The considered 8-bit multipliers are commonly used in low power embedded systems and image/video processing applications [62] [63]. For example, as shown in Table 2.9, the CEVA-NP4000 DSP processor includes 4096 8-bit multiply-accumulate (MAC) units [1], which are used in various

Table 2.9: CEVA-NeuPro AI Processors Family [1]

Product	**Number of MAC Units**			**Target Market**
	8x8	**16x8**	**16x16**	
NP4000	4096	2048	1024	Automotive, Surveillance
NP2000	2048	1024	512	Surveillance, Drones
NP1000	1024	512	256	Smart-phones
NP500	512	256	128	IoT, Smart-phones

high-performance energy-constrained edge processing applications, i.e., IoT, smart-phones and enterprise surveillance. As shown in Section 2.4.2, the simulation results for 16-bit approximate multipliers exhibit significant similarities with the 8-bit version. Thus, we selected 8-bit approximate multipliers, as their analyses results can be extrapolated to 16 or 32-bit multipliers.

2.6.1 Accuracy of AxCLib

Table 2.10 summarizes various error metrics, i.e., error rate (ER), mean error distance (MED), normalized error distance (NED), mean relative error distance (MRED) and PSNR, that we obtained based on an exhaustive simulation for the approximate library. The shown results are averaged over the full range of the inputs and provide useful insights about design accuracy. There is a strong correlation between the ED and both the design *Type* and *Degree*. However, considering the ED metric as our TOQ, mandates changing the design for every applied input, which is impractical due to the associated overhead.

Analysis of Error Distance (ED)

Figure 2.9 shows the histogram distribution of the ED for the approximate multipliers given in Table 2.8. The ED for the designs based on *AMA1* with D1, D2, D3 and D4 have an average of 102, 246, 538 and 13162, respectively. While the ED for the designs

Table 2.10: Accuracy Metrics for the Library of Approximate Multipliers

Approximation Degree	FA Type	ER	MED	NED	MRED	MSE	PSNR
D1	**AMA1**	0.931	102	0.0165	0.0380	1.69×10^4	39.35
	AMA2	0.978	101	0.0213	1.0447	1.44×10^4	39.97
	AMA3	0.996	200	0.0416	2.8055	4.76×10^4	34.78
	AMA4	0.932	51	0.0083	0.0189	4.11×10^3	45.58
	AMA5	0.870	44	0.0069	0.0148	3.14×10^3	46.80
D2	**AMA1**	0.962	246	0.0366	0.0823	9.61×10^4	32.10
	AMA2	0.990	227	0.0515	2.1470	7.23×10^4	33.26
	AMA3	0.999	474	0.1077	7.0425	2.68×10^5	27.65
	AMA4	0.963	107	0.0164	0.0350	1.79×10^4	39.15
	AMA5	0.922	93	0.0138	0.0280	1.38×10^4	40.32
D3	**AMA1**	0.977	538	0.0758	0.1679	4.53×10^5	25.80
	AMA2	0.996	464	0.0976	4.2929	2.97×10^5	27.41
	AMA3	1.000	1010	0.2280	13.4428	1.17×10^6	21.75
	AMA4	0.977	185	0.0286	0.0582	5.27×10^4	34.30
	AMA5	0.952	185	0.0261	0.0488	5.34×10^4	34.56
D4	**AMA1**	0.9882	13162	1.1862	2.1	3.27×10^8	5.54
	AMA2	0.9998	16902	5.0033	272.6	4.00×10^8	8.18
	AMA3	0.9999	17211	3.9303	180.7	4.13×10^8	7.36
	AMA4	0.9882	6334	0.4705	0.6039	7.91×10^7	10.38
	AMA5	0.9825	8096	0.5309	0.6642	1.17×10^8	8.85

based on *AMA2* with D1, D2, D3 and D4 have an average of 101, 227, 464 and 16902, respectively. Models based on *AMA3* with D1, D2, D3 and D4 have average EDs of 200, 474, 1010 and 17211, respectively. However, the designs based on *AMA4* with D1, D2, D3 and D4 have average EDs of 51, 107, 185 and 6334, respectively, which is quite low. AMA5 based designs with D1, D2, D3 and D4 have the lowest average of ED, which is 44, 93, 185 and 8096, respectively.

We notice that the error distance varies for different inputs, for example, *Design1* (top-left corner of Figure 2.9) shows that the ED varies from 0 to 518 with an average of 102. If the value of ED were input-independent, then it would be constant for all input values. Such input-dependency is also present in the remaining designs. Moreover, we notice that the variation in ED based on approximation degree for a specific type

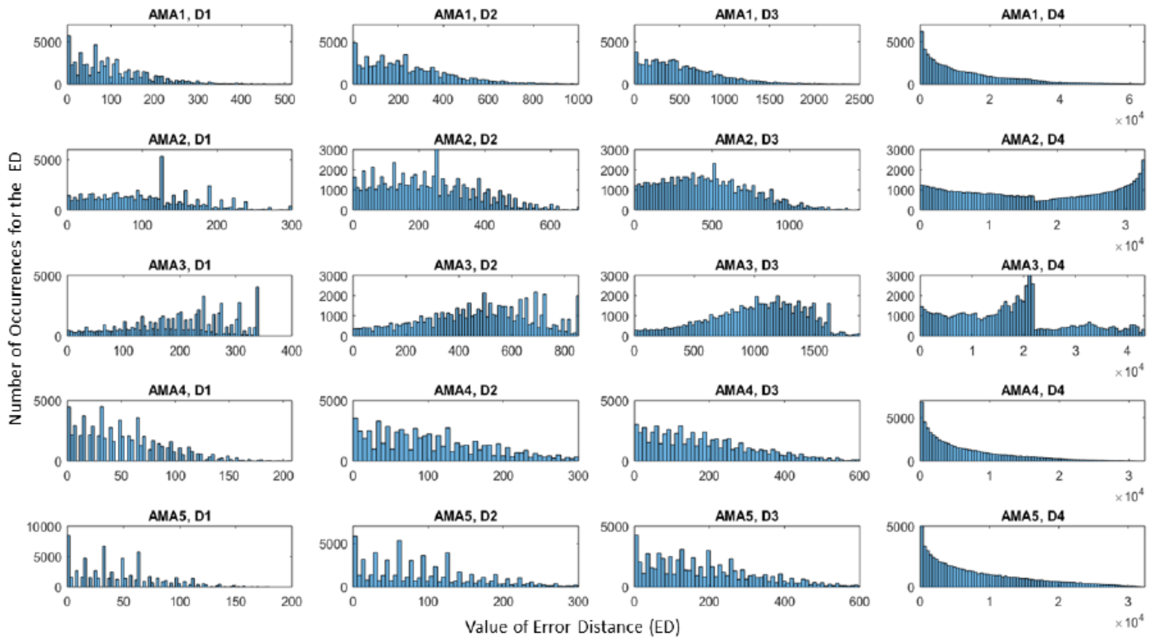


Figure 2.9: Histogram Distribution of ED for the Library of Approximate Multipliers

(represented horizontally in Figure 2.9) is more evident than the variation between different types for a particular degree (represented vertically in Figure 2.9), i.e., ED is correlated with the degree more than with the type. The ED increases almost doubles while increasing the approximation degree for any design type. Ideally, for every input, there exists a specific design, among the 20 models, with a minimal error distance (ED), which is the most suitable to be used in error-tolerant applications. However, changing the most appropriate design for every input is impractical due to the associated overhead.

Analysis of PSNR

The Peak Signal-to-Noise Ratio (PSNR), as given in Equation 2.8, depends on the Mean Square Error (MSE). For image processing applications, PSNR is an indication of image quality. Thus, it could be used as a TOQ metric while controlling the quality of approximate computing. A low value of PSNR indicates a low image quality

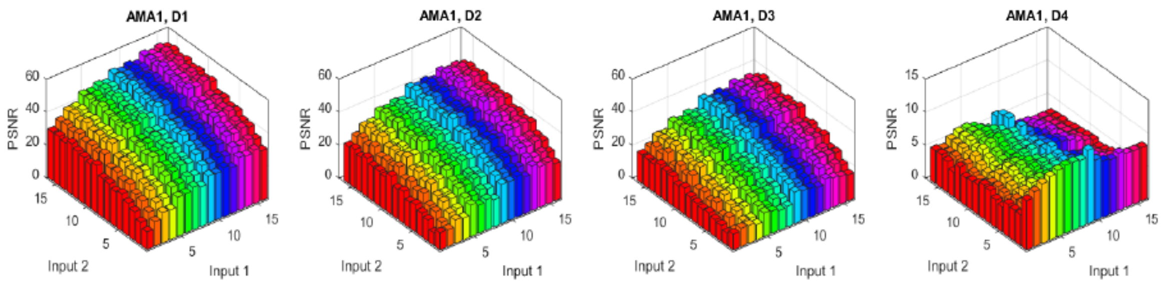


Figure 2.10: PSNR for Approximate Multiplier based on *AMA1* Type and D1, D2, D3 and D4 Approximation Degrees

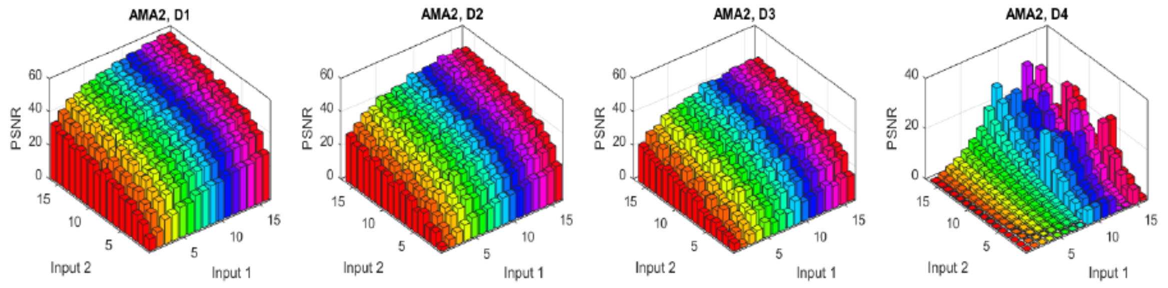


Figure 2.11: PSNR for Approximate Multiplier based on *AMA2* Type and D1, D2, D3 and D4 Approximation Degrees

associated with a large MSE. Similar to [64], we consider PSNR ≥ 25 as our threshold for "acceptable" quality.

Figures 2.10 - 2.14 show the PSNR for different approximate multipliers utilizing *AMA1* - *AMA5* types, respectively. Each column/bar shows the PSNR averaged over 256 different inputs, i.e., 16 consecutive inputs of *Input1* and 16 consecutive inputs of *Input2*, which we call a *cluster*. For instance, Figure 2.10 shows the PSNR for designs based on the *AMA1* type. The model based on *D1* has an average value of 39.4dB with a minimum amount of 10.7dB. It has 19 out of 256 input combinations with unacceptable quality, i.e., PSNR < 25dB. Similarly, the design based on *D2* has an average of 32.1dB with a minimum value of 7.7dB, and 54 input combinations with PSNR < 25dB. The *D3* based design has an average of 25.8dB for the PSNR, with 115 clusters out of the 256 groups having unacceptable quality. Regarding the

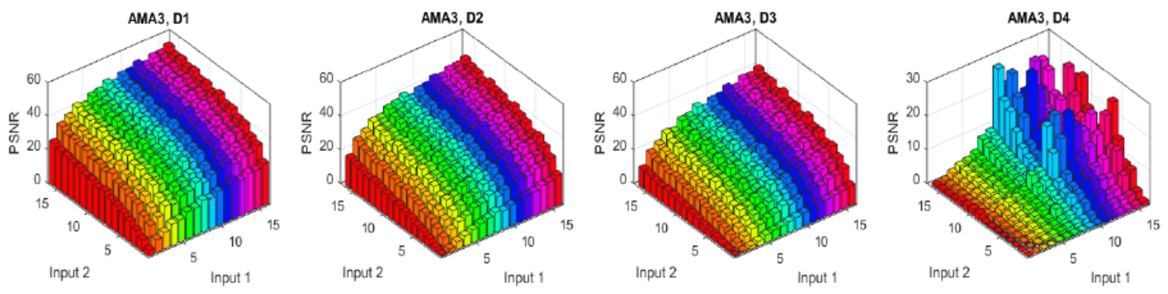


Figure 2.12: PSNR for Approximate Multiplier based on *AMA3* Type and D1, D2, D3 and D4 Approximation Degrees

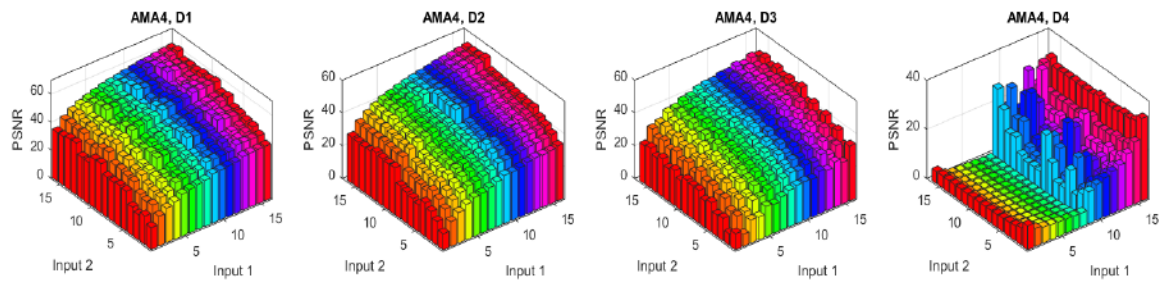


Figure 2.13: PSNR for Approximate Multiplier based on *AMA4* Type and D1, D2, D3 and D4 Approximation Degrees

D4 based design, all clusters have a low output quality, with PSNR < 25dB and a maximum value of 11dB.

The PSNR for designs based on *AMA2* is depicted in Figure 2.11. The *D1* based design has an average of 40dB with a minimum value of 7dB. It has 22 input combinations with PSNR < 25dB. Similarly, the design based on *D2* has an average of 33.2dB, with 55 input combinations having unacceptable quality. The *D3* based design has an average of 27.4dB with 99 input combinations having a PSNR < 25dB. Regarding the *D4* based design, only 7 clusters have an acceptable quality.

Figure 2.12 depicts a graphical representation for the PSNR of designs based on *AMA3*. The *D1* based design has an average of 34.8dB with 46 input combinations having PSNR < 25dB, while the *D2* based design has an average of 27.7dB with 92 input combinations having PSNR < 25dB. The *D3* based design has 151 input

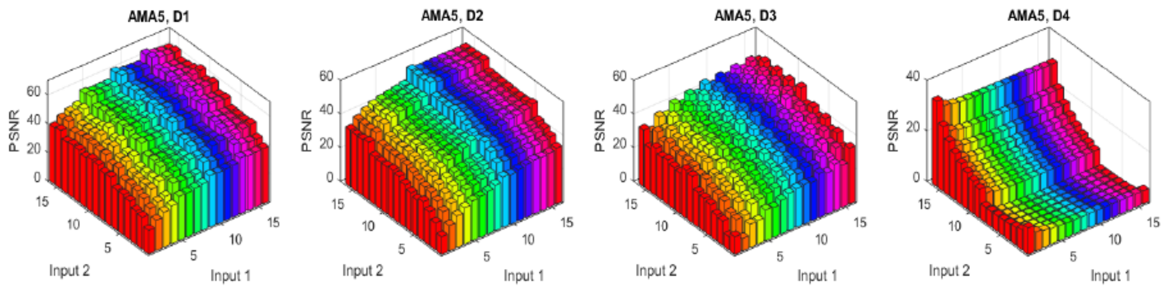


Figure 2.14: PSNR for Approximate Multiplier based on *AMA5* Type and D1, D2, D3 and D4 Approximation Degrees

combinations, which violate the PSNR, i.e., PSNR < 25, while the *D4* based design has only 6 clusters with a PSNR > 25dB.

The PSNR for designs based on *AMA4* is shown in Figure 2.13. The model based on *D1* has an average of 45.6dB with a minimum value of 15.5dB. It has only six input combinations with PSNR < 25dB. Similarly, the *D2* based design has an average of 39.1dB with a minimum amount of 9.7dB, and 18 input combinations with PSNR < 25dB. The *D3* based design has an average of 34.3dB for the PSNR, and 43 input combinations with PSNR < 25dB. Regarding the *D4* based design, 230 clusters have a low output quality, with PSNR < 25dB and an average value of 10.4dB.

Figure 2.14 shows the PSNR for *AMA5* based designs. The *D1* based design has an average of 46.8dB with a minimum value of 16.3dB, where it has five input combinations with unacceptable quality. Similarly, the design based on *D2* has an average of 40.3dB with a minimum amount of 10.4dB, and 14 input combinations have PSNR < 25dB. The design with *D3* has an average of 34.6dB for the PSNR, and 33 input combinations have unacceptable quality. Regarding the *D4* based design, 239 clusters have a low output quality with an average of 8.8dB.

Based on the above analysis for the PSNR of 20 different approximate designs, we notice that every model has some input combinations that violate the specified quality constraint, e.g., the PSNR should be at least 25dB. Thus, we should avoid

Table 2.11: Power, Area, Delay, Frequency and Energy for the 20 Static Approximate Multipliers

Design		Dynamic	Slice	Occupied	Period	Frequency	Energy
Type	**Degree**	**Power (mW)**	**LUTs**	**Slices**	**(ns)**	**(MHz)**	**(pj)**
AMA1	**D1**	306	79	23	7.763	128.82	2375.5
	D2	253	76	29	7.814	127.98	1976.9
	D3	196	77	29	9.487	105.41	1859.5
	D4	38	75	27	7.339	136.26	278.9
AMA2	**D1**	271	69	23	9.306	107.46	2521.9
	D2	207	63	29	9.373	106.69	1940.2
	D3	165	57	23	9.775	102.30	1612.9
	D4	29	46	18	9.672	103.39	280.5
AMA3	**D1**	322	67	23	9.223	108.42	2969.8
	D2	262	58	20	7.488	133.55	1961.9
	D3	189	55	21	9.139	109.42	1727.3
	D4	36	32	14	3.093	323.31	111.3
AMA4	**D1**	263	56	29	6.121	163.37	1609.8
	D2	210	47	19	6.743	148.30	1416.0
	D3	124	40	16	5.889	169.81	730.2
	D4	31	13	7	1.383	723.07	42.9
AMA5	**D1**	242	53	26	7.017	142.51	1698.1
	D2	183	41	19	5.694	175.62	1042.0
	D3	113	31	11	4.625	216.22	522.6
	D4	30	6	6	0.706	1416.43	21.2
Exact		**442**	**85**	**33**	**8.747**	**114.32**	**3866.2**

such input-dependent cases with low output quality when requiring a high quality result.

2.6.2 Area, Power, Energy and Delay Analysis of AxCLib

For the analysis of design metrics of the approximate library, we utilized the XC6VLX75T FPGA, which belongs to the Virtex-6 family, and the FF484 package Xilinx. For functionality verification, we use VHDL simulation based on Mentor Graphics Modelsim [65]. We use Xilinx XPower Analyser for the power calculation based on exhaustive design simulation [66]. For logic synthesis, we use the Xilinx Integrated Synthesis Environment (ISE 14.7) tool suite [67].

Table 2.11 shows the characteristics of the approximate designs. As depicted in

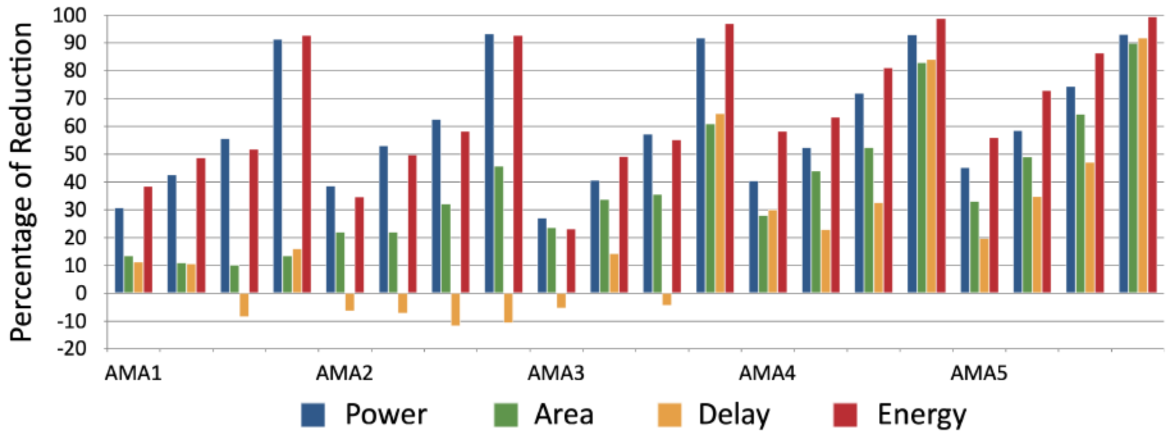


Figure 2.15: Power, Area, Delay and Energy Reduction for the 20 Static Approximate Multipliers

Figure 2.15, all designs have reduced power, area and energy compared to the exact design. However, a few designs have a delay higher than the delay of the exact design. Power reduction varies from 27.2% to 93.4%, with an average of 60.8%. Similarly, energy reduction ranges between 23.2% and 99.4%, with an average of 65.5%. The designs based on D1, D2, D3 and D4 have an average of 42.2%, 56.9%, 66.6% and 96.2% of energy reduction, respectively. These designs exhibit a 21.3% execution time reduction on average.

Each design of the 8-bit approximate array multipliers implemented on FPGA consists of 64 full adders (FAs). The designs D1, D2, D3 and D4, have 25, 33, 40 and 64 approximate FAs, respectively. Based on the experimental results, the power consumption is 1.3× lower for every bit of the results being approximated, e.g., going from D1 to D2. The dynamic power difference between D1 and D4 is about 10×, which equals ≈ 1.3^8. This reduction shows the benefits of design approximation regarding power consumption, which is obtained mainly due to the simplified design of approximate FA with reduced switching activity. The approximate FAs have a simplified structure compared with the exact FA. Thus, approximated designs have a noticeable reduction in the occupied LUTs.

The final energy reduction enabled by approximation is weighted by the portion of circuit design that is genuinely beneficial for approximation. Such modification is similar to the improvement of computer performance, which is obtained based on the enhanced performance of specific components, as quantified by Amdahl's law [68]. The final energy improvement can be expressed in the form of Amdahl's law as given in Equation 2.10, where E is the energy for the exact component, and E_{TOQ} is the energy of the approximate element, which satisfies the TOQ.

$$Improved\ Energy\ of\ Approximate\ Application = \frac{1}{X \times (\frac{E}{E_{TOQ}}) + (1 - X)} \tag{2.10}$$

The ratio $(\frac{E}{E_{TOQ}})$ is the energy saving in the approximate element in the design, and X is the fraction of the design, which is amenable to approximation. Thus, for the best energy efficiency, we aim to achieve $(\frac{E}{E_{TOQ}}) \gg 1$ and $X \approx 1$. Therefore, image processing applications, i.e., image blending and filtering, which use simple multiplication operations, will have a reduced energy consumption for such approximate application where $X \approx 1$.

2.7 Summary

In this chapter, we designed, evaluated and compared various 8 and 16-bit approximate array multipliers based on approximation in partial product summation. The design space of approximate multipliers is found to be primarily dependent on the type of the approximate FA used, the architecture, and the placement of smaller sub-modules in the higher-order n-bit multiplier. The proposed designs are compared based on their area, power, delay, accuracy and PDP. An image blending application is used to compare the proposed multiplier designs, which have competitive results compared to the state-of-the-art. Therefore, we selected a set of 20 designs as our "library of approximate designs". Then, we have implemented the library at the register transfer level (RTL), in synthesizable VHDL. The library is a basic building block in the proposed methodology for quality assured approximate design, as shown in Figure 1.1, where it will be integrated into an adaptive approximate accelerator designs, as it will be explained in the next chapters.

Chapter 3

Adaptive Approximate Accelerators: Software-based Design

In this chapter, we present a detailed description of the proposed methodology for designing adaptive approximate accelerators, while the proposed design can be implementable in both software and hardware, this chapter demonstrates the aspects of the software-based implementation. Chapter 4 is devoted for the FPGA-based hardware implementation.

3.1 Introduction

Approximation techniques require a *quality assurance* to adjust approximation settings/knobs and monitor the quality of fine-grained individual outputs. There are two strategies to adjust the settings of an approximate program to assure the quality of results i) *forward design* [69], which sets the design knobs and then observes the quality of results. However, the output quality of some inputs may reach unacceptable levels; and ii) *backward design* [70], which tries to find the optimal knobs setting for a given bound of output quality, this requires exploring a considerable space of knob settings for a given input, which is intractable.

To overcome the limitations mentioned above of design approaches, we propose an *adaptive approximate design* that allows altering the settings of approximation, at runtime to meet the desired output quality. The main idea is to develop a machine learning-based input-aware *design selector*, which can modify the approximate design based on the applied inputs, to meet the required quality constraints. Our approach is general in terms of quality metrics and supported approximate designs. It is primarily based on a library of 8 and 16-bit approximate multipliers with 20 different configurations and well-known power dissipation, performance and accuracy profiles as described in Section 2.6. Moreover, we utilize a backward design approach to dynamically adapt the design to meet the desired target output quality (TOQ) based on machine learning (ML) models. The TOQ is a user-defined quality constraint, which represents the maximum allowable error for a given application. The proposed design flow is adaptable, i.e., applicable to approximate functional units other than multipliers, e.g., approximate multiply-accumulate units [60] and approximate meta-functions [61]. The next section elaborates more on the main steps of the proposed methodology.

3.2 Adaptive Design Methodology

As shown in Figure 3.1, our proposed methodology encompasses two phases: (1) an offline phase, where we build a machine learning-based model; and (2) an online stage, where we continuously use the machine learning-based model based on the

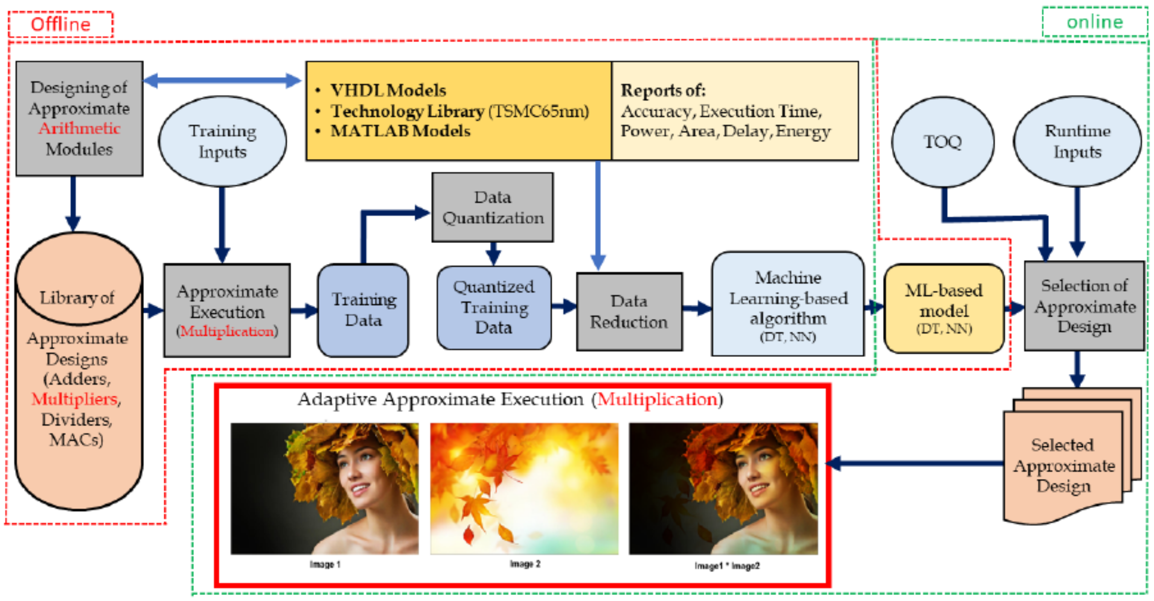

Offline
online
Designing of Approximate Arithmetic Modules
Training Inputs
VHDL Models
Technology Library (TSMC65nm)
MATLAB Models
Reports of: Accuracy, Execution Time, Power, Area, Delay, Energy
TOQ
Runtime Inputs
Data Quantization
Library of Approximate Designs (Adders, Multipliers, Dividers, MACs)
Approximate Execution (Multiplication)
Training Data
Quantized Training Data
Data Reduction
Machine Learning-based algorithm (DT, NN)
ML-based model (DT, NN)
Selection of Approximate Design
Adaptive Approximate Execution (Multiplication)
Image 1
Image 2
Image1 * Image2
Selected Approximate Design

Figure 3.1: Methodology of Software-based Adaptive Approximate Design

inputs to predict the settings of the adaptive design. The main step of the proposed methodology are the followings:

(1) Generating of Training Data: Inputs are applied exhaustively to the approximate library to create the training data for building the ML-based model (design selector). For 16 and 32-bit designs, the size of the input combinations is 2^{32} and 2^{64}, respectively. Thus, a sampling of the training data could be used because it is impossible to generate an exhaustive training dataset for such circuits.

(2) Clustering/Quantizing of Training Data: Evaluating the design accuracy for a single input can provide the error distance (ED) metric only. However, mean error metrics (e.g., mean square error (MSE), peak-signal-to-noise-ratio (PSNR) and normalized error distance (NED)) should be assessed over a set of consecutively applied data rather than a scalar input. Thus, inputs with a specific distance from each other are considered as a single cluster with the same estimated error metric. We propose to cluster every 16 consecutive input values. Based on that, each input for 8-bit

multiplier, encompasses 16 clusters rather than 256 inputs. Similarly, for the 16-bit multiplier design, the number of clustered inputs is reduced to 2^{24} rather than 2^{32}.

(3) Pre-processing/Reducing of Training Data: Inputs could be applied exhaustively for small circuits, e.g., 8-bit multipliers. However, the size of the input combinations for 16 and 32-bit designs is significant. Therefore, we have to reduce the size of the training data through sampling techniques in order to design a smaller and efficient ML-based model. Moreover, for 16-bit designs, we prioritize the training data based on their area, power and delay as well as accuracy, then reduce the training data accordingly.

(4) Building of Machine Learning-based Model: We built decision trees and neural networks-based models, which function as a design selector, to predict the most suitable settings of the design based on the applied inputs.

(5) Selection of Approximate Design: In the online phase, the user inputs, i.e., TOQ and inputs of the multiplier, are given to the ML-based models to predict the setting of the approximate design, i.e., *Type* and *Degree*, which is then utilized within an error-resilient application, e.g., image processing, in a software-based adaptive approximate execution.

The flow of the proposed methodology depicted in Figure 3.1 is given in Algorithm 3.1. The main steps are done once offline. During the online phase, the user specifies the TOQ, where we build our models based on normalized error distance (NED) and peak signal to noise ratio (PSNR) error metrics. An important design decision is to determine the configuration granularity, i.e., how much data to process before re-adapting the design, which is termed as the window size (N). For example, in image processing applications, we select N to be equal to the size of colored components of an image, i.e., 250x400=100,000 pixels for our example images. Then based

Algorithm 3.1. Proposed Adaptive Approximate Design

```
Input:
 1: (1) Input1: first input data; (2) Input2: second input data;
 2: (3) TOQ: specified quality;
Output: (1) Result:
 3:  Offline Phase::
 4: - Build a Library of Approximate Arithmetic Modules
 5: - Generate Training Data
 6: - Quantize/Preprocess Training Data
 7: - Build ML-based Design Selector, i.e., approximate computing quality manager
 8: Online Phase::
 9: N ← $Configure_Window_Size(TOQ)$ ;
10: L ← $Length_of_Input(Input1)$;
11: $T \leftarrow L/N$;                                   ▷ number of reconfiguration times
12: $i \leftarrow 1$;
13: while $i \leq T$ do
                                                        ▷ determine the $i^{th}$ cluster
14:     $C1_i$= $Detect_Cluster(Input1[N*(i-1)+1 : i*N])$
15:     $C2_i$= $Detect_Cluster(Input2[N*(i-1)+1 : i*N])$
                                          ▷ check if any of the $i^{th}$ inputs changed
16:     if ($C1_i \neq C1_{i-1}$) Or ($C2_i \neq C2_{i-1}$) then
17:         ($Degree_i$, $Type_i$) = Selector ($C1_i$, $C2_i$, $TOQ_i$)
18:     else                                            ▷ check if $TOQ_i$ changed
19:         if $TOQ_i \neq TOQ_{i-1}$ then
20:             ($Degree_i$, $Type_i$) = Selector ($C1_i$, $C2_i$, $TOQ_i$)
                                                        ▷ Use previous settings
21:         else
22:             $Degree_i \leftarrow Degree_{i-1}$
23:             $Type_i \leftarrow Type_{i-1}$
24:         end if
25:     end if
26:     UpdateBuffer($C1_i$, $C2_i$, $Degree_i$, $Type_i$, $TOQ_i$)
27: end while
28: function SELECTOR($C1_i$, $C2_i$, $TOQ_i$)
29:     Find $Degree_i$, $Type_i$
30:     Return $Degree_i$, $Type_i$
31: end function
32: function AXD($C1_i$, $C2_i$)
33:     Perform approximate Computation for $C1_i$ and $C2_i$
34:     Return Result
35: end function
36: function UPDATEBUFFER($C1_i$, $C2_i$, $Degree_i$, $Type_i$, $TOQ_i$)
37:     Save the inputs, used design and quality result
38: end function
```

on the length of inputs, i.e., L and N, we determine the number of times to reconfigure the design such that the final approximation benefits, i.e., reduced energy and execution time, are still significant. After N inputs, a design adaptation is done, if any of the inputs or TOQ changes. The first step in such adaptation is input quantization, i.e., specifying the corresponding cluster for each input based on its magnitude, since design adaptation for every scalar input is impractical. To evaluate the inputs of an approximate design, various metrics such as median, skewness and kurtosis, have been used [43]. However, our approximate library is designed irrespective of the applied inputs. Thus, the input magnitude is the most suitable characteristic of design selection.

3.3 Machine Learning Based Models

ML-based algorithms find solutions by learning through training data [71]. Supervised learning allows for a fast, flexible, and scalable way to generate accurate models that are specific to the set of application inputs and TOQ. The error for an approximate design with particular settings can be predicted based on the applied inputs. For this purpose, we developed a forward design-based model, as shown in Figure 3.2(a). The obtained accuracy for this model is 97.6% and 94.5% for PSNR and NED error metrics, respectively. Such high efficiency is attributed to the straightforward nature of the problem. However, we target the inverse design of finding the most suitable design settings for given inputs and error threshold, as shown in Figure 3.2(b).

Utilizing the *Rattle* package [72], in a preliminary evaluation conducted in [73],

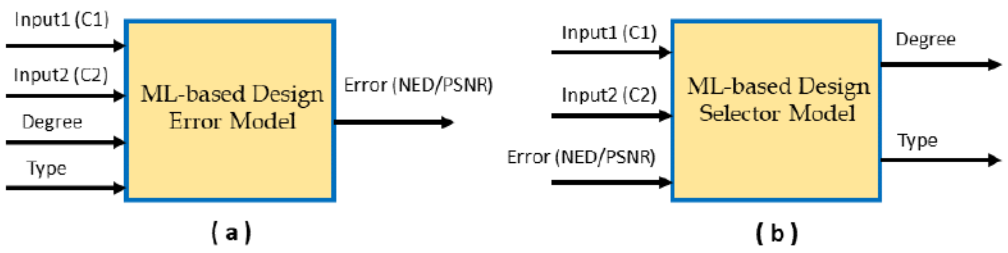


Figure 3.2: Models for AC Quality Manager, (a) Forward Design, (b) Inverse Design

we designed and evaluated various ML-based models, based on the analyzed data and several algorithms, developed in the statistical computing language R [74]. These models represent the design selector for the adaptive design. *Linear regression* models (LM) were found to be the simplest to develop; however, their accuracy is the lowest, i.e., around 7%. Thus, they are not suitable for our proposed methodology. On the other hand, *decision tree* (DT) models based on both *C5.0* [75] and *rpart* [76] algorithms achieve an accuracy of up to 64%, while *random forest* (RF) models, with an overhead of 25 decision trees, achieve an accuracy of up to 68%. The most accurate models are based on *neural networks* but they suffer from long development time, design complexity and high energy overhead [38]. In the sequel, we implement and evaluate two versions of the *design selector*, based on decision tree and neural network models. Accordingly, we identify and select the most suitable one to implement in our methodology.

3.3.1 Decision Tree-based Design Selector

The DT algorithm uses a flowchart-like tree structure to partition a set of data into various predefined classes, thereby providing the description, categorization, and generalization of the given data sets [77]. Unlike the linear model, it models non-linear relationships quite well. Thus, it is used in a wide range of applications, such as credit risk of loans and medical diagnosis [78]. Decision trees are usually employed for *classification* over data sets, through recursively partitioning the data, such that observations with the same label are grouped [78].

Initially, we implemented and evaluated three different DT-based models: (1) using the *C5.0* function [75], (2) using the *rpart* function [76], and (3) based on the *cubist* function [79]. The accuracy of the *rpart* based models is found to be

Table 3.1: Accuracy and Execution Time of DT and NN based Design Selectors

Model		**Accuracy**		**Execution Time (ms)**	
Inputs	**Output**	**DT**	**NN**	**DT**	**NN**
C1, C2, PSNR	Degree	77.8%	82.17%	8.87	18.9
C1, C2, PSNR, s2=D1	Type	75.5%	66.52%	25.03	18.0
C1, C2, PSNR, s2=D2	Type	76.1%	70.21%	19.3	9.0
C1, C2, PSNR, s2=D3	Type	71.3%	73.22%	11.94	18.7
C1, C2, PSNR, s2=D4	Type	74.1%	59.08%	6.61	7.4

lower than that of *C5.0* based models. *Cubist* is a rule-based model, which is an extension of Quinlan's M5 model tree [80], where a tree is developed such that the terminal leaves represent linear regression models. Therefore, we have to discretize the results, which degrade its accuracy. Moreover, the leaves with linear models need more processing than the scalar leaves in the *C5.0* model. Thus, due to tool limitations, we implemented a two-steps design selector utilizing the C5.0 function by predicting the design *Degree* and then its *Type*. This kind of *design selector* is supposed to be *lightweight* for continually monitoring the workloads, and continuously adapting the design every (N) input.

Based on the error analysis of the approximate designs, we noticed that the error magnitude is correlated to the approximation *Degree* in a more significant manner than the design *Type*. Such correlation is evident in the accuracy of the models, where these models have an average accuracy of 77.8% and 74.3% for predicting the design *Degree* and *Type*, respectively, as shown in Table 3.1. The time for executing the *software* implementation of these models is very short, i.e., 24.6ms in total with 8.87ms to predict the design *Degree* and 15.72ms to predict the design *Type*. This time is negligible compared to the time of running an application, such as image blending or filtering.

Generally speaking, a decision tree model could be replaced by a lookup table

(LUT) which contains all the training data that are used to build the DT-model [73]. When searching the LUTs, we could use the first matched value, i.e., design settings that satisfy the TOQ, which could be a better solution obtained with a little search effort. For DT-based models, we do not need to specify which value to retrieve. However, it is possible to obtain a result which is closer to the TOQ by changing the settings of the tree such as the maximum depth of any node of the tree, the minimum number of observations that must exist in a node in order for a split to be attempted and the minimum number of observations in any terminal node.

In general, for embedded and limited resources systems, a lookup table is not a viable solution if the number of entries becomes very large [81]. In fact, for a circuit with two 16-bit inputs, we need to generate 2^{32} input patterns to cover all possible scenarios of a circuit. However, a reduced precision LUT could be used as shown in Section 3.4.3. In order to build a DT-based model for 16-bit multiplier designs, we need to evenly distribute the sampled training data. Therefore, we tried different sample sizes, and similar to the work in [82], we found that the trial with $1M$ samples has the best accuracy. Thus, we randomly generate $50K$ input data for each 16-bit approximate design within the library. The obtained model accuracy was found to be 83.5%.

3.3.2 Neural Network-based Design Selector

We implemented a two-step NN-based design selector by predicting the design *Degree* first and then the *Type*. The model for *Degree* prediction has an accuracy of 82.17% while the four models for *Type* prediction have an average accuracy of 67.3%, as shown in Table 3.1. All of these models have a single hidden layer with a *sigmoid* activation function, given in Equation 3.1. The time for executing the *software* implementation of these models is very short, i.e., 32.18ms in total with 18.9ms to predict the design *Degree* and 13.28ms to predict the design *Type*. This time is negligible compared to the time of running an application, such as image processing.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

Compared to the DT-based model, the NN-based model has an execution time, which is 1.31X higher than the DT, while its average accuracy is almost 0.98X of the accuracy achieved by the DT-based model. The next section evaluates the *software* implementation of the proposed methodology, which utilizes the DT-based design selector that we described earlier in Section 3.3.1. We discard the NN-based design selector due to the absence of advantages over DT.

3.4 Experimental Results

In this section, we evaluate the effectiveness of the *software* implementation of the fully-automated proposed system, including the approximate library and the DT-based design selector. We run MATLAB on a machine with 8GB DRAM and i5 CPU with a speed of 1.8GHz. We evaluate the proposed methodology based on two applications of images processing: 1) Image blending, where we use two sets of images, i.e., *Set-1* with five examples and *Set-2* with 50 examples; and 2) Image filtering, where we use two images, i.e., *Lina* and *Cameraman*. Moreover, we evaluate an audio mixing application based on 16-bit models. The *execution time* is considered as a quality metric, where its overhead is found to be relatively small compared to the original applications, as shown in the sequel.

3.4.1 Image Blending

Image blending in multiplication mode allows us to blend multiple images to look like a single image. This process is widely used in developing animation and effect movies where video blending requires the multiplication of several consecutive photos. For example, blending two-colored videos, each with N_f frames of size N_r rows by N_c columns per image, involves a total of $3 \times N_f \times N_r \times N_c$ pixels. Each image has 3 colored components/channels, i.e., red, green and blue, where the values of their pixels are expected to differ. A static configuration uses a single design, from the library provided in Table 2.8, to perform all multiplications, even when their pixels are different. Therefore, for enhanced output quality, we propose to adapt the approximate design per channel as shown in Figure 3.3. However, for a video with a set of consecutive frames, e.g., 30 frames per second, the proposed methodology can be run for the first frame only since the other frames have very close pixel values. This way, the design selector continuously monitors the inputs and efficiently locates the most suitable design for each colored-component to meet the required TOQ.

Various metrics, e.g., median, skewness and kurtosis, have been used in the literature to characterize the inputs of approximate designs [43]. However, their proposed approximate circuits heavily depend on the training data used during the approximation process. Our approximate library is designed regardless of the applied inputs. Thus, the relationship between the consecutively applied inputs, such as skewness and kurtosis, is insignificant for our designs. Since the error magnitude depends on the user inputs, we rely on pixel values to select a suitable design. However, setting the configuration granularity at the pixel level is impractical. On the other hand, the design selection per coloued-component is more suitable.

We evaluate the average of the pixels of each colored-component to select the

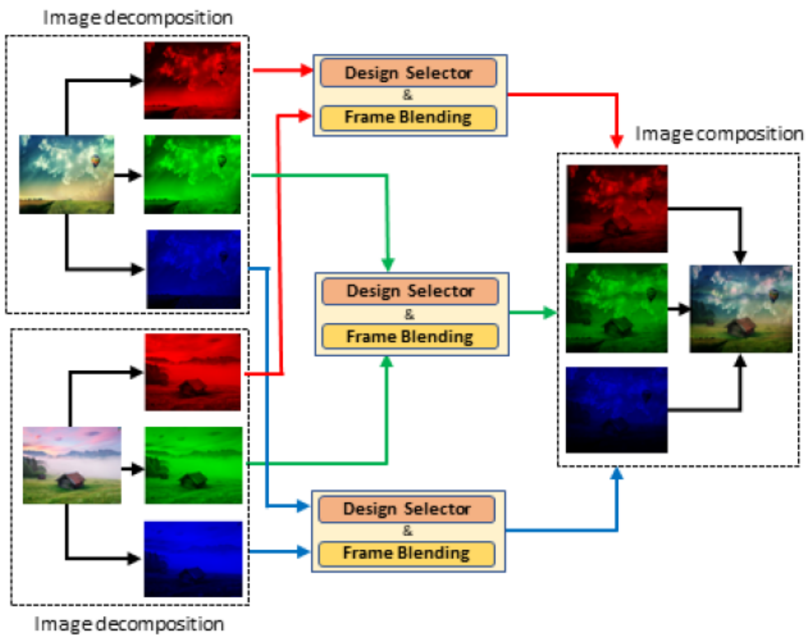


Figure 3.3: Adaptive Image/Video Blending at Component Level

most suitable design. Two completely different images may have the same average of their pixels. Unfortunately, this could result in the same selected approximate design because the training is performed at the multiplier level and not at the image level. To avoid this scenario, we reduce the configuration granularity by dividing the colored-component into multiple segments, e.g., four segments. After that, we propose to use various designs, rather than a single design, for each colored-component. This issue triggers training and building the model at the image level to control the quality of approximation in image processing applications. Next, we analyze the results of applying the proposed methodology on two sets of images; i) *Set*-1 with 10 images; and ii) *Set*-2 with 100 images based on a public database of images [2]. The photos of each set are then blended at the component level, as shown in Figure 3.3, to evaluate the efficiency of the proposed methodology.

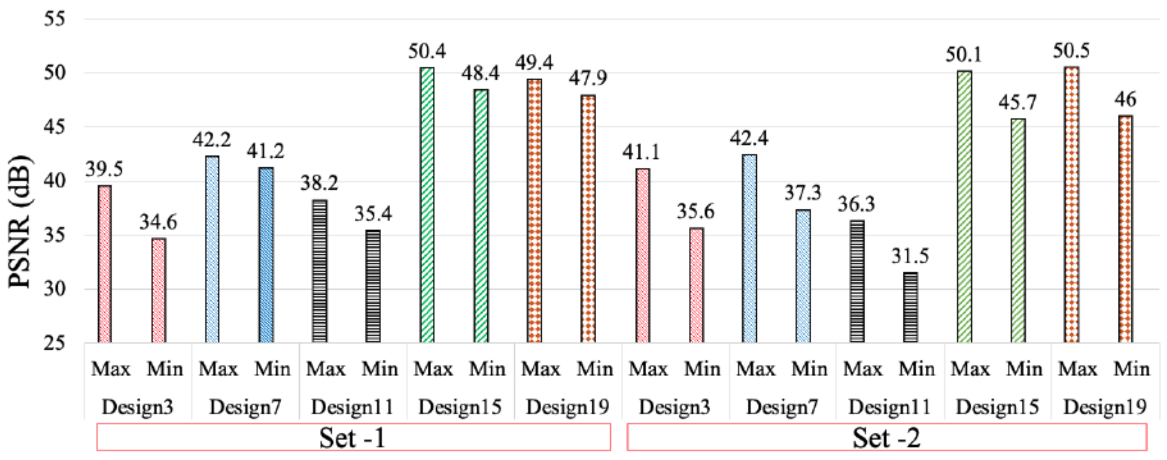


Figure 3.4: The Dependency of the Output Quality on the Applied Inputs

Accuracy Dependency on the Applied Inputs

Figure 3.4 shows the fluctuation in the value of the PSNR, that we obtained by applying different input images for a specific static design, where the fluctuation is the difference between the maximum and the minimum obtained PSNR. Based on the photos of *Set-1*, *Design3*, *Design7*, *Design11*, *Design15* and *Design19* have a fluctuation of 14.2%, 2.4%, 7.9%, 4.1% and 3.1%, of the obtained PSNR, respectively. Similarly, for the images of *Set-2*, the obtained PSNR fluctuates by 15.4%, 13.7%, 15.2%, 9.6% and 9.8% for *Design3*, *Design7*, *Design11*, *Design15* and *Design19*, respectively. The obtained PSNR for different images processed on a single design varies due to the dependency of the output quality on inputs, as observed in [12].

3.4.1.1 Blending of *Set-1* of Images

We use a set of 10 different images, each of size $N_r \times N_c = 250 \times 400 = 10^5$ pixels, and each image is segmented into three colored-components. Table 3.2 shows the average values of the pixels of each colored-component and the associated input cluster, which are denoted as *Average* and *Cluster*, respectively.

Table 3.2: Characteristics of *Set-1* Blended Images

Example	(Image1, Image2)	Frame Characteristic	Input 1 (Image1)			Input2 (Image2)		
			Red	Green	Blue	Red	Green	Blue
1	(Frame, City)	Average	131	163	175	172	153	130
		Cluster	9	11	11	11	10	9
2	(Sky, Landscape)	Average	121	149	117	160	156	147
		Cluster	8	10	8	11	10	10
3	(Text, Whale)	Average	241	241	241	48	156	212
		Cluster	16	16	16	4	10	14
4	(Girl, Beach)	Average	177	158	140	168	176	172
		Cluster	12	10	9	11	12	11
5	(Girl, Tree)	Average	102	73	40	239	193	118
		Cluster	7	5	3	16	13	8

We target 49 different values of TOQ, i.e., PSNR ranges from $17dB$ to $65dB$, for each blending example. Thus, we run the methodology 245 times, i.e., 5x49. For every invocation, based on the corresponding cluster for each input, i.e., $C1$ and $C2$, and the associated target PSNR, one of the 20 designs is selected and used for blending. For illustration purposes, in the sequel we explain *Example5* in detail. As shown in Table 3.2, the *Girl* image has a red-component with an average of 102, which belongs to *Cluster 7*, i.e., $C1_R = 7$. Similarly, the *Tree* image has a red-component with an average of 239, which belongs to *Cluster 16*, i.e., $C2_R = 16$. The green components belong to *Clusters* 5 and 13 ($C1_G = 5$, $C2_G = 13$) while the blue components belong to *Clusters* 3 and 8 ($C1_B = 3$, $C2_B = 8$). Then, we adapt the design by calling the design selector thrice, i.e., once for every colored-component, assuming TOQ=$17dB$. The selected designs (based on Line 29 of Algorithm 3.1) are given by:

$$Selector(C1_{\mathrm{R}}, C2_{\mathrm{R}}, TOQ) \rightarrow Design_{\mathrm{R}} \rightarrow Design_8 \tag{3.2}$$

$$Selector(C1_{\mathrm{G}}, C2_{\mathrm{G}}, TOQ) \rightarrow Design_{\mathrm{G}} \rightarrow Design_{16} \tag{3.3}$$

$$Selector(C1_{\mathrm{B}}, C2_{\mathrm{B}}, TOQ) \rightarrow Design_{\mathrm{B}} \rightarrow Design_{11} \tag{3.4}$$

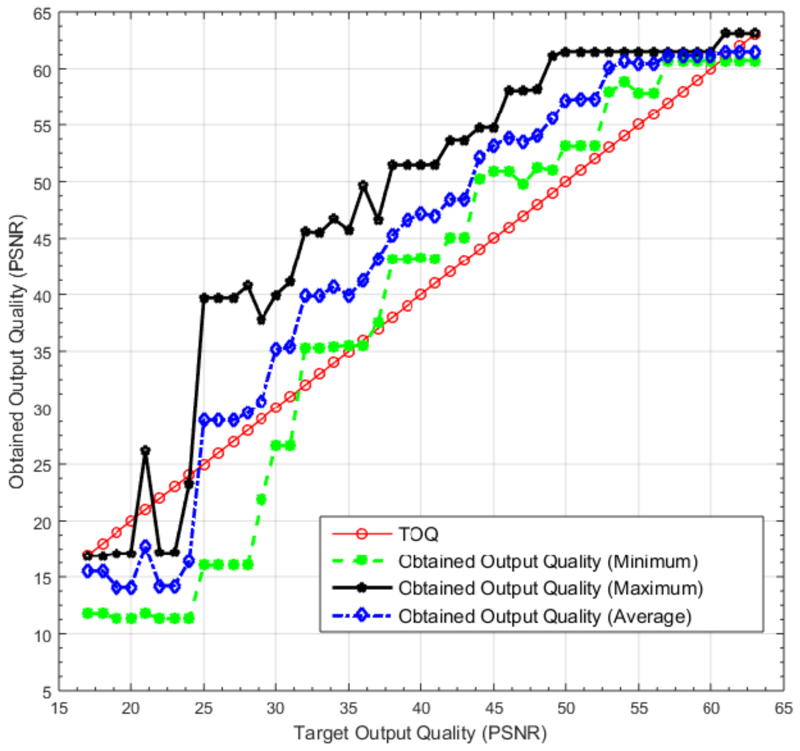


Figure 3.5: Obtained Output Quality for Image Blending of *Set-1*

The selected $Design_R$, $Design_G$ and $Design_B$ are $Design_8$, $Design_{16}$ and $Design_{11}$, respectively. Based on that, the obtained quality is $16.9dB$, which is insignificantly less than the TOQ.

Accuracy Analysis of Adaptive Design

Figure 3.5 shows the minimum, maximum and average curves of the obtained output quality, each evaluated over five examples of image blending. Out of the 245 selected designs, 49 predicted designs are violating the TOQ, even insignificantly, i.e., the obtained output quality is below the red line. The unsatisfied output quality is attributed mainly to model imperfection. The best achievable prediction accuracy is based on the accuracy of the two models executed consecutively, i.e., *Degree* model with 77.8%, and *Type* model with 76.1%. The accuracy of our model prediction is

80%, which is in agreement with the average accuracy of the DT-based models, as shown in Tabl 3.1.

Execution Time Analysis of Adaptive Design

Figure 3.6 shows the average execution time of the five examples of image blending evaluated over 20 static designs. The shown time is normalized for the execution time of the exact design. All designs have a time reduction ranging from 1.8% to 13.6% with an average of 3.96%.

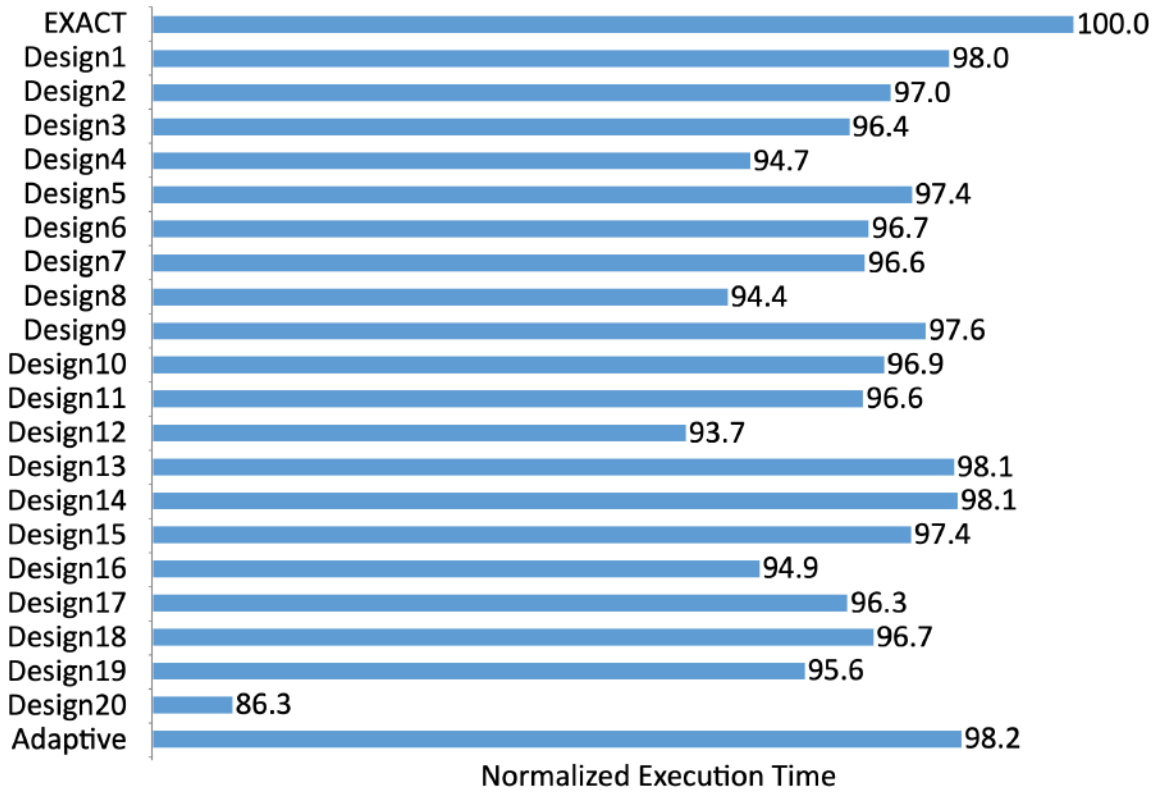


Figure 3.6: Normalized Execution Time for Blending of *Set-1* Images using 20 Static Designs

For the five examples of image blending, we evaluated the execution time of the adaptive design, where the target PSNR ranges between $17dB$ and $65dB$ for each case. Figure 3.7 shows the execution time for the five examples using the exact design,

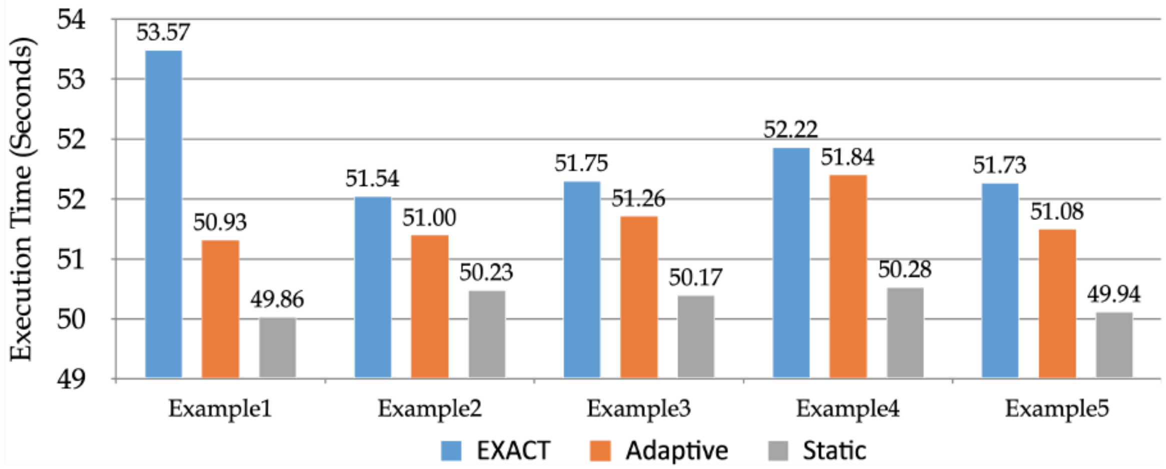


Figure 3.7: Execution Time of the Exact, Static and Adaptive Design

the adaptive design averaged over 49 different TOQ, and the static design averaged over 20 approximate designs. Design adaptation overhead, which represents the time for running the ML-based design selector is 30.5ms, 93.9ms, 164.6ms, 148.6ms and 42.1ms, for the five examples, respectively. Moreover, the five examples have a data processing time based on three selected designs per example of 50.90s, 50.91s, 51.10s, 51.69s and 51.04s, respectively. Thus, for these five examples, the design adaptation time represents 0.06%, 0.18%, 0.32%, 0.28% and 0.08% of the total execution time, respectively, which is a negligible overhead.

3.4.1.2 Blending of *Set-2* of Images

We used a set of 100 images from the database of "*8 Scene Categories Dataset*" [2], which is downloadable from [83]. It contains eight outdoor scene categories; coast, mountain, forest, open country, street, inside the city, tall buildings and highways. A sample of the used images is shown in Figure 3.8.

Similar to *Set-1*, we target 49 different value of TOQ, for 50 examples of blending and execute the proposed methodology 2450 times. Figure 3.9 shows the minimum,

Figure 3.8: A Sample of *Set-2* Images [2]

maximum and average curves of the obtained output quality, each evaluated over 50 examples of image blending. Out of the 2450 selected designs, 430 predicted designs are violating the TOQ, where the obtained output quality is below the red line. Thus, the accuracy of our model prediction is 82.45%, which is slightly higher than the accuracy obtained for *Set-1* images. We consider PSNR $\geq 25dB$ as a threshold for an *acceptable* quality of pictures, as proposed by [64]. As shown in Figures 3.5 and 3.8, the proposed methodology has a high prediction accuracy for an *acceptable* PSNR, while its prediction accuracy is low when the TOQ $< 25dB$.

Energy Analysis of Adaptive Design

One of the foremost goals of designing a library of approximate arithmetic modules, i.e., multipliers and adders, is to enhance energy efficiency. To calculate the energy consumed by the approximate multiplier to process an image, we use the following equation:

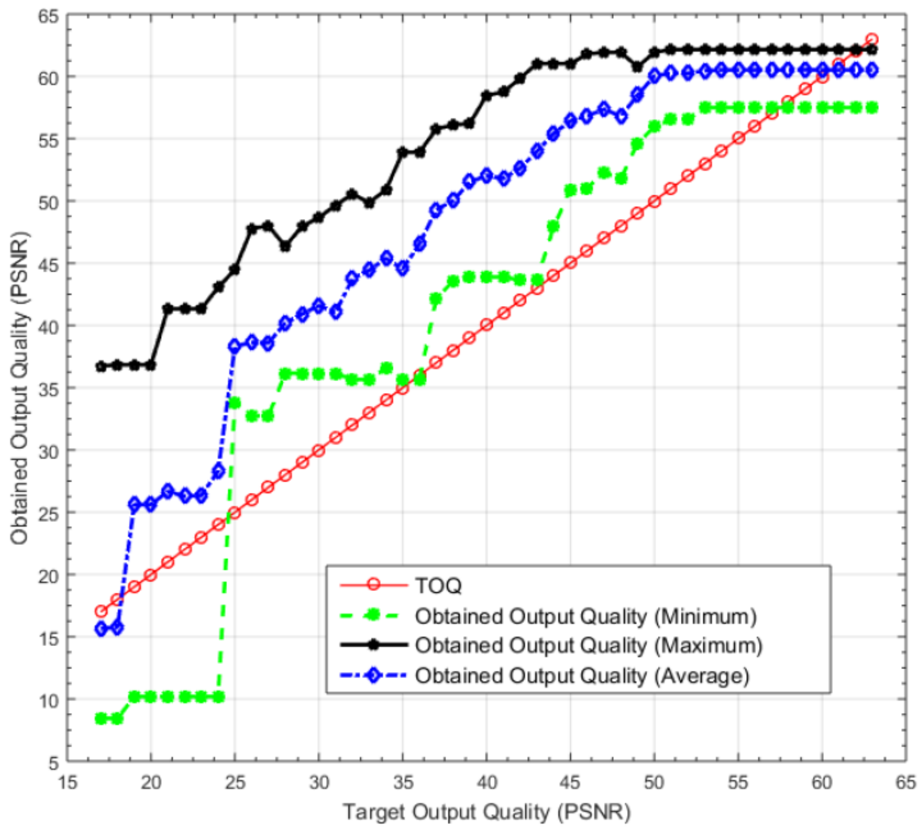


Figure 3.9: Obtained Output Quality for Image Blending of *Set-2*

$$Energy = Power \times Delay \times N \tag{3.5}$$

where *Power* and *Delay* are obtained from the synthesis tool, as shown in Table 2.11, and N is the number of multiplications required to process an image, which equals 250 $\times$ 400=10^5 pixels. As shown in Table 2.11, *Design9* has the highest energy consumption with 2970 pj and a saving of 896 pj compared to the exact design. Thus, the design adaption overhead of 733.7 pj (based on Table 4.1) is almost negligible compared to the total minimal energy savings of 89.6 μj (896 pj $\times$ 10^5) obtained by processing a single image. These results validate our lightweight *design selector*.

3.4.2 Gaussian Smoothing

We evaluate the accuracy of the adaptive design on a Gaussian smoothing low-pass filter, which reduces image details through attenuating high-frequency signals. Applying a Gaussian smoothing is the same as convoluting the image with a circularly symmetric 2-D Gaussian function, as given in Equation 3.6 [84]. The Gaussian smoothed output is a weighted average of each pixel's neighborhood. As in Equation 3.7, we use a (3×3) kernel, based on $\sigma = 1.5$, where the average kernel weight depends significantly on the value of the central pixels.

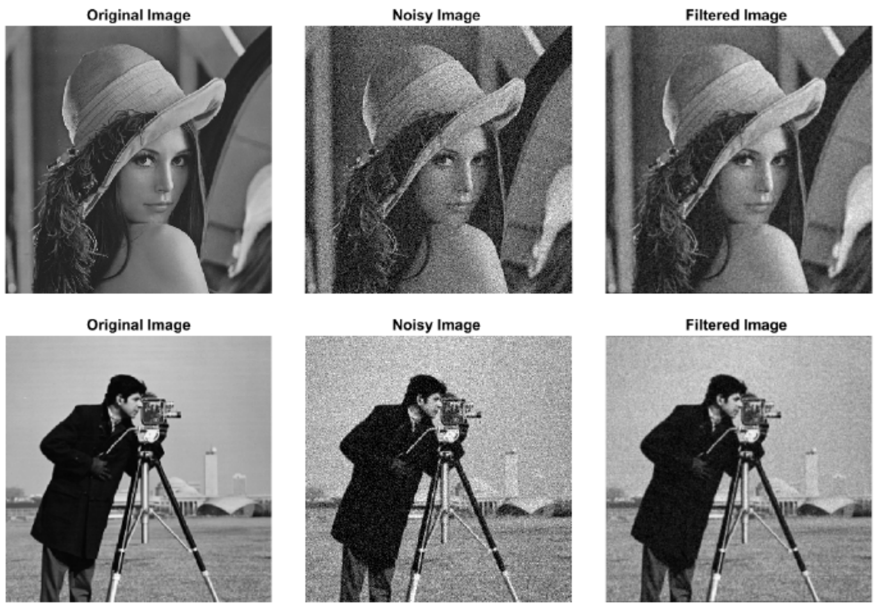


Figure 3.10: Exact, Noisy and Filtered Images

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.6}$$

$$Kernel = \frac{1}{254} \begin{bmatrix} 24 & 30 & 24 \\ 30 & 38 & 30 \\ 24 & 30 & 24 \end{bmatrix} \tag{3.7}$$

We use the benchmarks, *Lena* and *Cameraman*, as input images, with the addition of zero-mean Gaussian white noise, with a variance of 0.01 to the original gray-scale image. Figure 3.10 shows the benchmark images with Gaussian-noise added and the noisy images filtered with the exact design. The Gaussian kernel, given in Equation 3.6, is applied to the 8-bit gray-scale input images of size (512×512) pixels. Figure 3.11 shows the PSNR obtained by using the 20 static designs. The resulting values are computed for the image obtained by applying the Gaussian filter with the exact multiplier design on the noisy image. The approximate designs were able to get a maximum PSNR of 53.2dB.

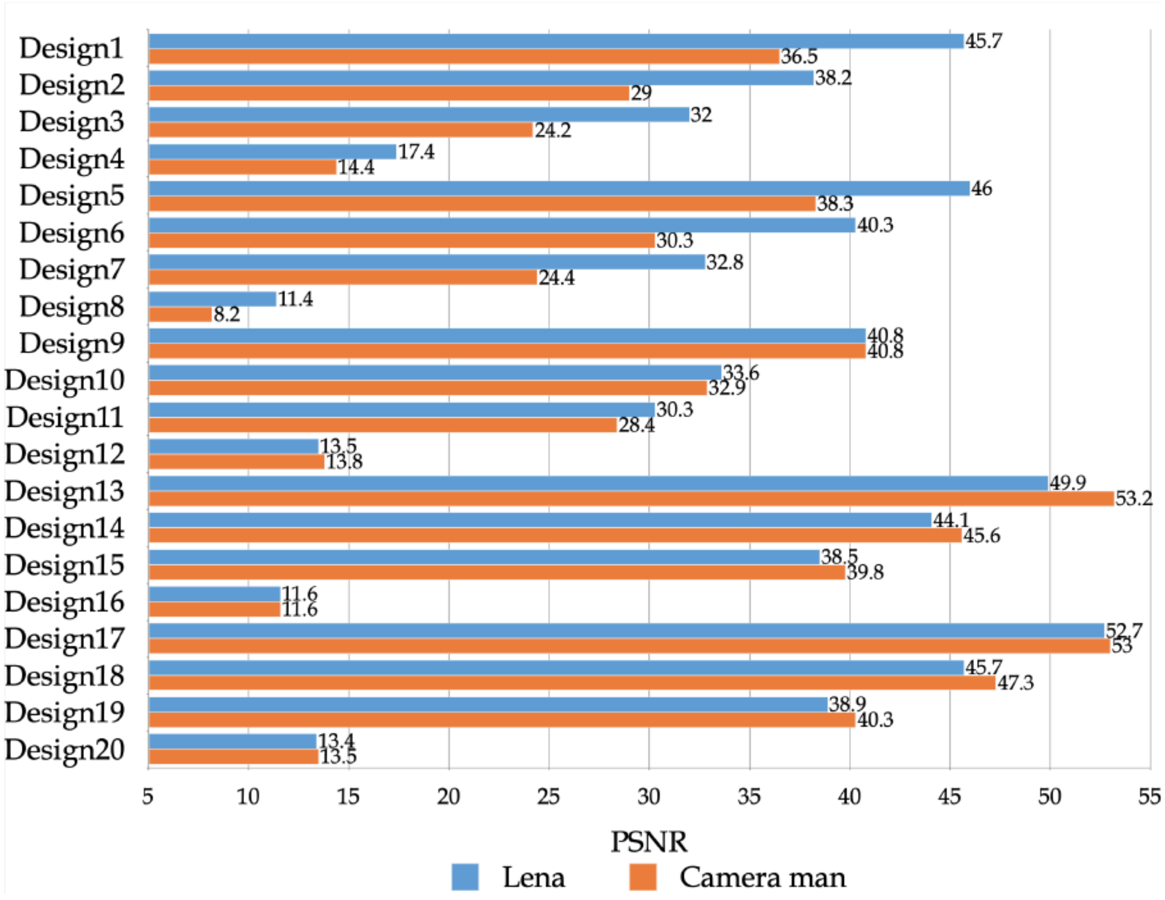


Figure 3.11: Output Quality (PNSR) for 2 Examples of Image Filtering Using 20 Static Designs

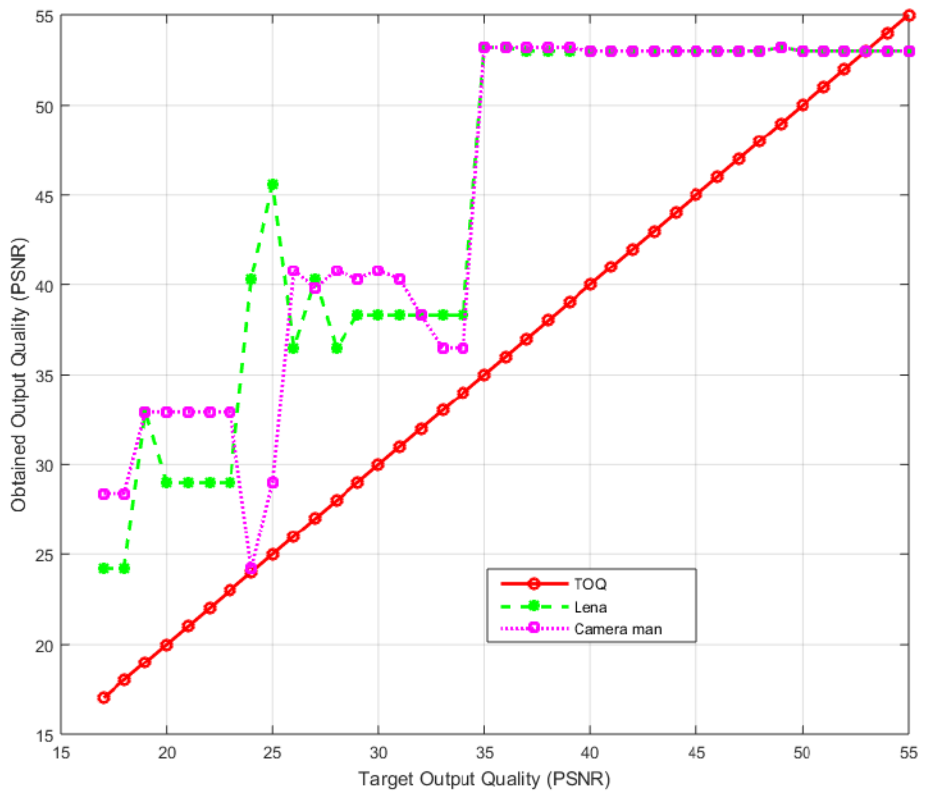


Figure 3.12: Obtained Output Quality for 2 Examples of Adaptive Image Filtering

Figure 3.12 shows the obtained PSNR for two examples of image filtering based on the adaptive design, where the TOQ ranges between 17dB and 53dB. The proposed methodology was able to satisfy the user given quality constraints. When the TOQ ranges between 17dB and 34dB, the obtained output quality for the two examples are different. However, when the TOQ is 35dB or more, both cases have almost the same obtained output quality.

3.4.3 Audio Mixing/Blending

Sounds are propagating waves represented in a binary format of 16-bit depth, which is able to cover a wide range of amplitudes with an enhanced quality. We perform a set of audio blending applications, to evaluate the DT-based model and reduced precision

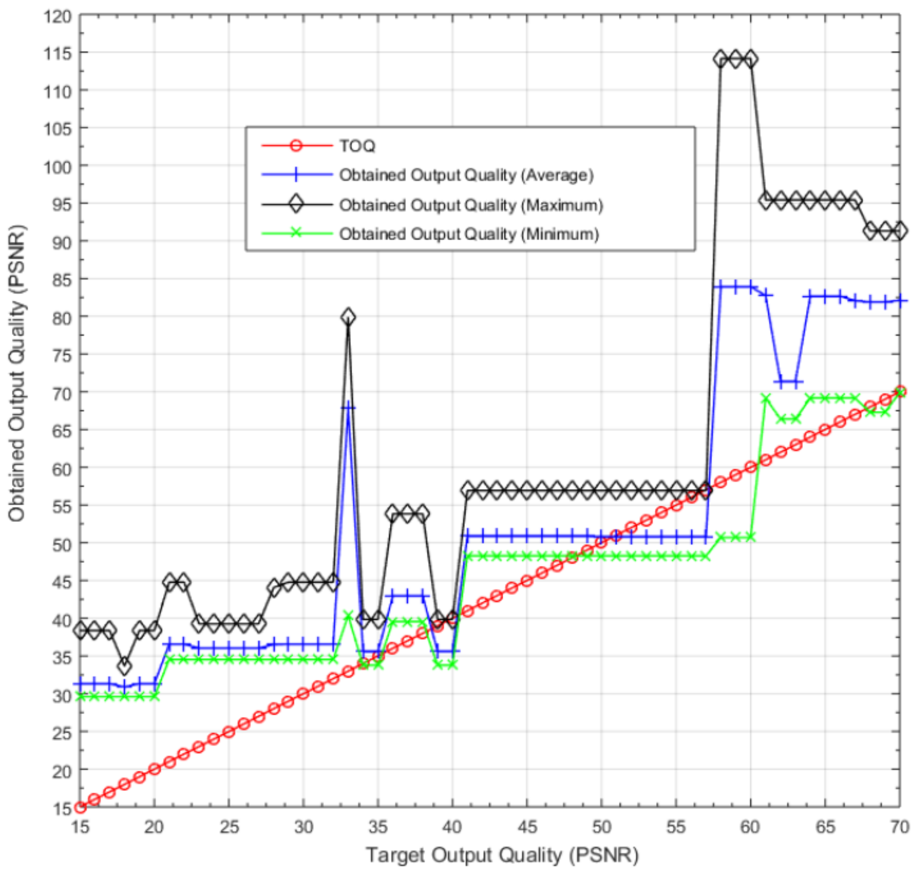


Figure 3.13: Obtained Output Quality for Audio Blending

LUT for 16-bit designs. We evaluate the proposed methodology over 45 examples, where the target PSNR ranges between 15dB and 70dB. The used WAV sound files were obtained from [85].

3.4.3.1 DT-based Model

Figure 3.13 presents the achieved PSNR, i.e., minimum, maximum and average, based on the decision tree model, which is designed to learn from the limited input data, to predict a result for the unseen data. The accuracy of the "average obtained TOQ" (blue curve) is 85.7%. Figures 3.5 and 3.9 for the image blending application were showing a smoother curve of predicted results since the DT-model was built using a full set of training data. However, the stairs/steps in the predicted PSNR, shown in Figure 3.13, are due to the expectations of the model, which was built based on sampled training data. The comparable accuracy of both models shows the effectiveness of machine learning even with sampled training data.

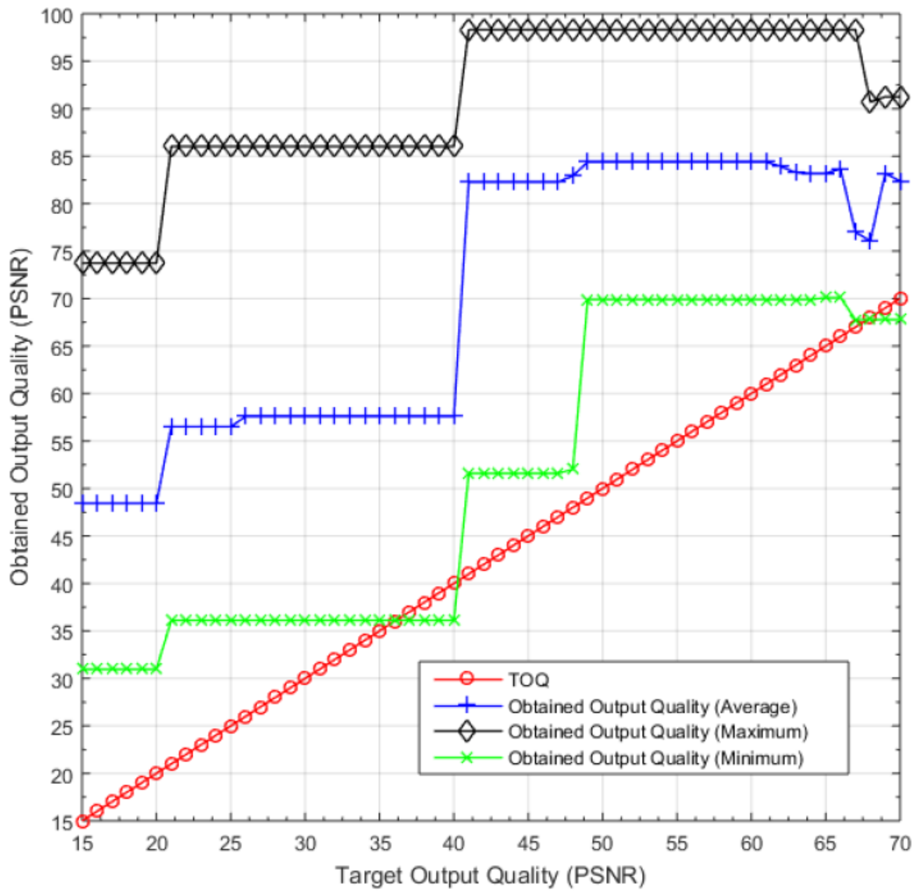


Figure 3.14: Obtained Output Quality for Audio Blending using LUTs

3.4.3.2 Reduced Precision LUT

A LUT with 16M instances provides a full accuracy with significant area overhead [86]. Therefore, as explained in Section 3.3.1, we used a sample of 1M instances to build a reduced-precision LUT. Figure 3.14 shows the obtained output quality for audio blending utilizing LUT. Since the LUT does not include all input values, we use "interpolation" to cover all possible inputs and obtain estimated output values for the inputs which are not found in the LUT. The accuracy of the "average obtained TOQ" (blue curve) is 100%, which could be due to interpolation by using the first solution that satisfy the TOQ. We notice that the obtained PSNR using LUTs is higher than the obtained PSNR by DTs, where the LUT uses the first match while the DT uses the closest match. We realize that both DTs and LUTs are applicable with different design characteristics. For example, DTs with acceptable accuracy are more suitable for embedded systems with limited resources, while LUTs are a better choice for PCs with large memory as this would lead to a higher accuracy.

Table 3.3: Obtained Accuracy (PSNR) for Various Approximate Designs

Application		**KUL** [28]	**ETM** [29]	**ATCM** [88]	**Adaptive Design (Proposed)**
Blending	***Set-1***, Ex. 1	24.8	27.9	41.5	61
	Set-1, Ex. 2	29.2	29.1	43.7	61.1
	Set-1, Ex. 3	20.3	24.8	33.1	63
	Set-1, Ex. 4	23	28	38.2	60.7
	Set-1, Ex. 5	27.6	29.4	40.3	61.5
Filtering	Lena	36.5	36.3	38.4	52.7
	Cameraman	35.9	36.9	29.2	53.2

3.5 Comparison with Related Work

We now compare the output accuracy achieved by our adaptive design with the precision of two static approximate models based on approximate multipliers proposed by Kulkarni et al. [28] and Kyaw et al. [29] that have *similar structures* as our approximate array multipliers. Moreover, we compare the accuracy of our work with a third approximate design based on the approximate tree compressor multiplier (ATCM), proposed by Yang et al. [88], which is a Wallace tree multiplier. Table 3.3 shows a summary of the obtained PSNR for image blending and filtering based on KUL

[28], ETM [29], ATCM [88] and the proposed adaptive design. The proposed model achieves better output quality than static designs due to the ability to select the most suitable design from the approximate library.

3.6 Summary

For dynamic inputs, a static approximate design may lead to substantial output errors for changing data. Previous work has ignored the consideration of the changing inputs to assure the quality of individual outputs. In this chapter, we proposed a novel fine-grained input-dependent adaptive approximate design, based on machine learning models. Then, we implemented a fully-automated toolchain utilizing a DT-based design selector. The proposed solution considers the inputs in generating the training data, building ML-based models, then adapting the design to satisfy the TOQ. The "software" implementation of the proposed methodology, developed in this chapter, provide a negligible delay overhead and was able to satisfy an output accuracy of 80% to 85.7% for various error-resilient applications. Such quality assured results come at the one-time cost of generating the training data, deploying and evaluating the design selector, i.e., machine learning-based model. With runtime design adaptation, the model always identifies and selects the most suitable design for controlling the quality loss. In the next chapter, we describe a fully FPGA-based *hardware* implementation of our proposed methodology utilizing dynamic partial reconfiguration.

Chapter 4

Adaptive Approximate Accelerators: FPGA-based Design

As demonstrated in Chapter 3, a *software* implementation of the proposed adaptive approximate accelerate was able to satisfy the required TOQ with a minimum accuracy of 80%. In this chapter, we present an *FPGA-based* implementation of the adaptive approximate accelerator utilizing the feature of dynamic partial reconfiguration, with a database of 21 reconfigurable modules.

4.1 Introduction

An essential advantage of FPGAs is their flexibility, where these devices can be configured and re-configured on-site and at runtime by the user. In 1995, Xilinx introduced the concept of *partial reconfiguration* (PR) in its XC6200 series to increase the flexibility of FPGAs by enabling re-programming parts of design at runtime while the remaining parts continue operating without interruption [89]. The basic premise of PR is that the device hardware resources can be time-multiplexed, similar to the ability of a microprocessor to switch tasks. PR eliminates the need to re-configure and re-establish links fully and dramatically enhances the flexibility that FPGAs offer. PR enables adaptive and self-repairing systems with reduced area and dynamic power consumption.

In Chapter 3, we demonstrated an adaptive design based on the applied inputs to improve the individual output quality. Similarly, in this chapter, we propose to dynamically adapt the functionality of the FPGA-based approximate accelerators using machine learning and dynamic partial reconfiguration. We utilize the previously proposed DT and NN-based *design selectors* that continually monitors the input data and quickly determines the most suitable approximate design. Then, accordingly, partially reconfigures the FPGA with the selected approximate design while keeping the whole error-tolerant application intact. The proposed methodology applies to any error-tolerant application where we demonstrate its effectiveness using an image processing application. As FPGA vendors announced the technical support for the runtime partial reconfiguration, such systems are becoming feasible. The works, reported in [45] - [47], show the benefits of DPR. However, to our best knowledge, the design framework for adaptively changeable approximate functional modules with input-awareness does not exist.

4.2 Dynamic Partial Reconfiguration

In this section, we describe the FPGAs architecture and main characteristics that can utilize partial reconfiguration. After that, we explain their components and features that enable them to build an adaptive design in more detail.

Field Programmable Gate Arrays (FPGA) devices conceptually consist of [90]: i) hardware logic (functional) layer which includes flip-flops, lookup tables (LUTs), block random-access memory (BRAM), digital signal processing (DSP) blocks, routing resources and switch boxes to connect the hardware components; and ii) configuration memory which stores the FPGA configuration information through a binary file called *configuration file* or *bitstream* (BIT). Changing the content of the *bitstream* file allows us to improve the functionality of the hardware logic layer. Xilinx and Intel (formerly Altera) are the leading manufacturing companies for FPGA devices. Due to the hardware available for implementation, in this thesis, we use the VC707 evaluation board from Xilinx, which provides a hardware environment for developing and evaluating designs targeting the Virtex-7 XC7VX485T-2FFG1761C FPGA [91].

Partial Reconfiguration (PR) is the ability to modify portions of the modern FPGA logic by downloading partial *bitstream* files while the remaining parts are not altered [92]. PR is a hierarchical and bottom-up approach and is an essential enabler for implementing *adaptive* systems. It can be static or dynamic, where the reconfiguration can occur while the FPGA logic is in the reset state or running state, respectively [90]. The dynamic partial reconfiguration (DPR) process consists of two phases: i) fetching and storing the required bitstream files in the flash memory, which is not time-critical; and ii) loading bitstreams into the reconfigurable region through a controller, i.e., internal configuration access port (ICAP). Implementing a partially reconfigurable FPGA design is similar to implementing multiple non-partial reconfiguration designs that share a common logic. Since the device is switching tasks in hardware, it has the benefit of both flexibility of software implementation and the performance of hardware implementation. However, it is not very commonly employed in commercial applications [92].

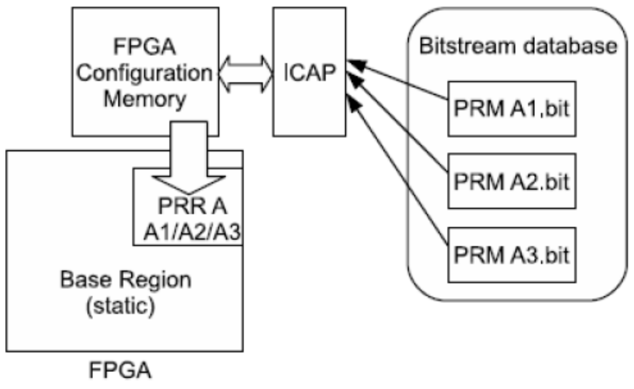


Figure 4.1: Principle of Dynamic Partial Reconfiguration on Xilinx FPGAs

Logically, the part that will host the reconfigurable modules (dynamic designs) is the dynamic *partial reconfigurable region* (PRR), which is shared among various modules at runtime through multiplexing. Figure 4.1 illustrates a reconfigurable design example on Xilinx FPGAs, with a partially reconfigurable region (PRR) A, which is reserved in the overall design layout mapped on the FPGA, with three possible partially reconfigurable modules (PRM). For FPGA full reconfiguration, all functional blocks should coexist while partial reconfiguration loads a single functional block at a time into the reconfigurable module. During PR, a portion of the FPGA needs to keep executing the required tasks, including the reconfiguration process. This part of the FPGA is known as the *static region*, which is configured only once at the boot-time with a full *bitstream*. This region will also host static parts of the system, such as I/O ports as they can never be physically moved.

Xilinx Partial Reconfiguration Controller (PRC) provides management functions for self-controlling partially re-configurable designs [93]. It is designed for enclosed systems where all reconfigurable modules are known to the controller. The Xilinx PRC consists of multiple (up to 32) *virtual socket* (VS) managers, which connect to a single fetch path. The *virtual socket* refers to a re-configurable partition (RP) with any supporting logic that exists in the static design. Each VS can contain up to 128 re-configurable modules (RMs) and up to 512 software and hardware triggers. Our proposed design includes a single VS with 21 RMs, each with a specific trigger. The mapping from a particular trigger, i.e., hardware or software, to a given RM is configurable during core configuration and at runtime. The PRC fetches the *bitstream* data from an Advanced eXtensible Interface (AXI) bus [93], which allows the controller to access *bitstreams* no matter where they are stored. Initially, the partial *bitstreams* are stored in a configuration library, i.e., a *bitstream* database, as shown in Figure 4.1. When a hardware (signal) or a software (register write) trigger event occurs, the PRC fetches/pulls partial *bitstreams* from the memory/database through the AXI bus and delivers them to a configuration port, e.g., ICAP.

The Xilinx Internal Reconfiguration Access Port (ICAP) is the core component of any dynamic partial reconfigurable system implemented in Xilinx *SRAM-based* FPGAs [94]. The ICAP controller, which is *flexible* with high reconfiguration *throughput*, is responsible for executing all commands to access and modify the configuration memory [90]. The speed of configuration is directly related to the size of the partial *bitstream* file and the bandwidth of the configuration port. For ICAP, the data width is 32 bit, and the bandwidth is 3.2Gb/s [95]. The ICAP enables DPR from within an FPGA chip, leading to the possibility of fully autonomous FPGA-based systems.

The Linear Byte Peripheral Interface (BPI) Flash memory provides 128 MB of nonvolatile storage. The .BIT binary configuration data (*bitstream*) file contains the header information that does not need to be downloaded to the FPGA. Therefore, we convert it into .BIN binary configuration data file, which is without the header information. With each trigger, a partial bit file is pulled from the BPI flash by the PRC and delivered to the ICAP, changing the functionality in that Reconfigurable Partition.

4.3 Machine Learning Based Models

In this section, we describe the FPGA-based implementation of the design selector based on DT and NN models.

4.3.1 Decision Tree-based Design Selector

As described in Section 3.3.1, the DT-based models have an average accuracy of 77.8% and 74.3% for predicting the design *Degree* and *Type*, respectively, as shown in Table 3.1. The overhead time for executing the *software* implementation of these models is around 24.6ms in total, with 8.87ms to predict the design *Degree* and 15.72ms to predict the design *Type*.

This section evaluates the power, area, delay and energy of the *FPGA-based* implementation of the DT-based *design selector*. Similar to evaluating the approximate library, we utilize the XC6VLX75T FPGA, which belongs to the Virtex-6 family. The Configurable Logic Block (CLB) comprises two slices, each containing four 6-input LUTs and eight flip-flops, for a total of eight 6-input LUTs and 16 flip-flops per CLB. Moreover, we use Mentor Graphics Modelsim [65] for functionality verification. We use Xilinx XPower Analyser for the power calculation based on exhaustive design simulation [66], while for logic synthesis, we use the Xilinx Integrated Synthesis Environment (ISE 14.7) tool suite [67].

Table 4.1 shows the obtained characteristics of the DT-based model. The power consumption of the model ranges between 35mW and 44mW. This value is insignificant when compared to the power consumption of approximate multipliers, as shown in Table 2.11, where these multipliers being selected are used for N inputs. Similarly, the introduced area, delay and energy overhead are amortized by running the approximate design for N inputs. The area of the model, represented in terms of the number

Table 4.1: Power, Area, Delay, Frequency and Energy of DT and NN-based Design Selectors

Model		Dynamic Power (mW)		Slice LUTs		Occupied Slices		Period (ns)		Frequency (MHz)		Energy (pj)	
Inputs	**Output**	**DT**	**NN**	**DT**	**NN**	**DT**	**NN**	**DT**	**NN**	**DT**	**NN**	**DT**	**NN**
C1, C2, PSNR	Degree	16	155	602	7835	231	2683	22.910	31.504	43.65	31.74	366.6	4883.1
C1, C2, PSNR, s2=D1	Type	19	164	497	8427	189	2791	18.596	31.746	53.78	31.50	353.3	5206.3
C1, C2, PSNR, s2=D2	Type	23	153	449	6625	221	2309	15.962	31.718	62.65	31.53	367.1	4852.8
C1, C2, PSNR, s2=D3	Type	23	159	390	5360	149	1731	15.494	29.164	64.54	34.29	356.4	4637.0
C1, C2, PSNR, s2=D4	Type	28	170	298	4549	134	1420	11.838	28.678	84.47	34.87	331.5	4875.3

of slice LUTs, is 1099, at maximum. Also, the number of occupied slices could reach 452 slices. The worst-case (slowest) frequency that the model could run is 43.65MHz, with a period of 22.91ns. The designed model could consume the maximum energy of 733.7pj. The set of approximate multipliers exhibits acceptable savings in their characteristics, i.e., area, power, delay and energy, compared to the exact design, as shown in Table 2.11.

It is important to note that the *design selector*, which is synthesized only once, is specific for the considered set of approximate designs. However, the proposed methodology is readily applicable to other approximate designs as well. The implementation overhead, i.e., power, area, delay and energy, for the DT-based model is negligible compared to the approximate accelerator since it is a simple nesting of if-else statements with a maximum depth of 12 to reach a node of a final result.

4.3.2 Neural Network-based Design Selector

Neural networks (NNs) have generally been implemented in software. However, recently with the exploding number of embedded devices, the hardware implementation of NNs is gaining significant attention. FPGA-based implementation of NN is complicated due to the large number of neurons and the calculation of complex equations such as activation function [96]. We use the sigmoid function $f(x)$, which was given by Equation 3.1, as an activation function. A piecewise second-order approximation scheme for the implementation of the sigmoid function is proposed in [97] as provided by Equation 4.1. It has inexpensive hardware, i.e., one multiplication, no look-up table and no addition.

$$f\left(x\right)=\begin{cases}1 & x>4.0\\ 1-\frac{1}{2}(1-\frac{|x|}{4})^2, & 0<x\leq 4.0\\ \frac{1}{2}(1-\frac{|x|}{4})^2, & -4.0<x\leq 0\\ 0, & x\leq -4.0\end{cases} \tag{4.1}$$

As described in Section 3.3.2, we implemented a two-step design selector by predicting the design *Degree* first and then the *Type*, with an accuracy of 82.17% and 67.3%, respectively, as shown in Table 3.1. The execution time of the NN-based model ranges between 37.6ms and 26.3ms, with an average of 32.7ms.

We implemented the NN-based model on FPGA, and its characteristics, including dynamic power consumption, slice LUTs, occupied slices, operating frequency and consumed energy, are shown in Table 4.1. These values are found to be insignificant when compared to the characteristics of approximate multipliers, as shown in Table 2.11, where these multipliers are used for N inputs. However, compared to the DT-based model, the NN-based model, as shown in Section 3.3.2, has an execution time, which is 1.31× higher than the DT, while its average accuracy is almost 0.98× of the accuracy achieved by the DT-based model. Moreover, regarding other design metrics, including power, slice LUTs, occupied slices, period and energy, the NN-based model has a value of 8.06×, 13.93×, 11.74×, 1.61× and 6.8×, consecutively, compared with the DT-based model. Unexpectedly, the DT-based model is better than the NN-based model in all design characteristics, including accuracy and execution time. The next section evaluates the *FPGA-based* implementation of the proposed methodology, which utilizes the DT-based design selector that we described earlier. We discarded the NN-based design selector due to the absence of advantages over DT.

4.4 System Architecture

Figure 4.2 shows the FPGA-based methodology for quality assurance of approximate computing through design adaptation, inspired by the general methodology shown in Figure 1.1. In order to utilize the available resources of the FPGA and show the benefits of design approximation, we integrate 16 multipliers into an accelerator to be used all together. Figure 4.3 shows the internal structure for the approximate accelerator with 16 multipliers. Each input, i.e., A_i and B_i where $16 \geq i \geq 1$, is 8-bit wide.

The implemented machine learning-based models (design selectors) are decision trees-based only, where model training is done once offline, i.e., off-FPGA. Then, the VHDL implementation of the obtained DT-based model, which is the output of the offline phase, is integrated as a functional module within the online phase of the

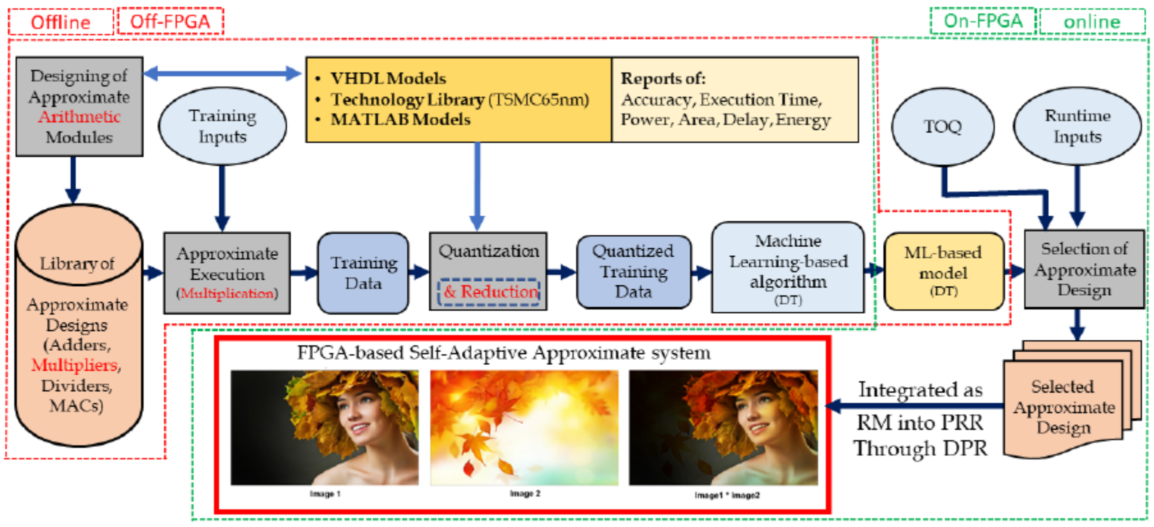

Offline
Off-FPGA
On-FPGA
online
Designing of Approximate Arithmetic Modules
Training Inputs
• VHDL Models
• Technology Library (TSMC65nm)
• MATLAB Models
Reports of: Accuracy, Execution Time, Power, Area, Delay, Energy
TOQ
Runtime Inputs
Library of Approximate Designs (Adders, Multipliers, Dividers, MACs)
Approximate Execution (Multiplication)
Training Data
Quantization & Reduction
Quantized Training Data
Machine Learning-based algorithm (DT)
ML-based model (DT)
Selection of Approximate Design
Selected Approximate Design
Integrated as RM into PRR Through DPR
FPGA-based Self-Adaptive Approximate system
Image 1
Image 2
Image1 * Image2

Figure 4.2: Methodology of FPGA-based Adaptive Approximate Design

FPGA-based adaptive system, as shown in Figure 4.4. The proposed FPGA architecture contains a set of IP cores, connected through a standard bus interface. The developed approximate accelerator core is with the capability of adjusting processing features as commanded by the user to meet the given TOQ. For the parallel execution, we utilize the existing block RAM in the Xilinx 7 series FPGAs, which have 1030 blocks of 36Kbits. Thus, we store the input data (images) in a distributed memory, e.g., save each image of size 16 KByte into 16 memory slots each of 1 KByte. Other configurations of the memory are also possible and can be selected to match the performance of the processing elements within the accelerator.

The *online* phase of the proposed adaptive approximate design, based on the decision tree is presented in Figure 4.4, where the annotated numbers, i.e., ① to ⑧, show the flow of its execution for image blending application. The target device is xc7vx485tffg1761-2, and the evaluation kit is Xilinx Virtex-7 VC707 Platform [91]. The main components are the reconfiguration engine, i.e., DT-based design selector, and the reconfigurable core (RC), i.e., approximate accelerator. The RC is placed in a well-known partially reconfigurable region (PRR) within the programmable logic. The AXI-HWICAP controller establishes communication with the ICAP.

We evaluate the effectiveness of the proposed methodology for an FPGA-based adaptive approximate design utilizing DPR. For that, we select an *image blending*

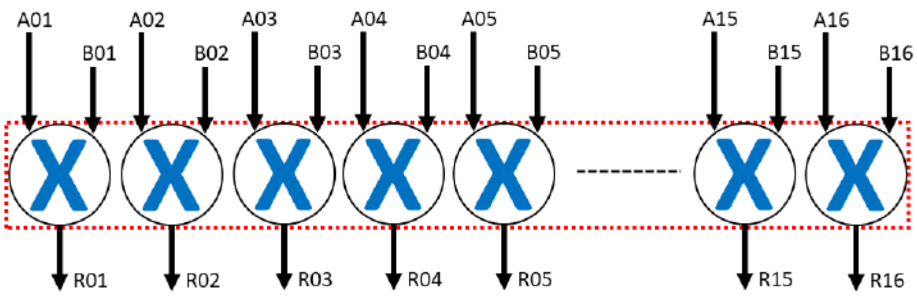


Figure 4.3: An Accelerator with 16 Identical Approximate Multipliers

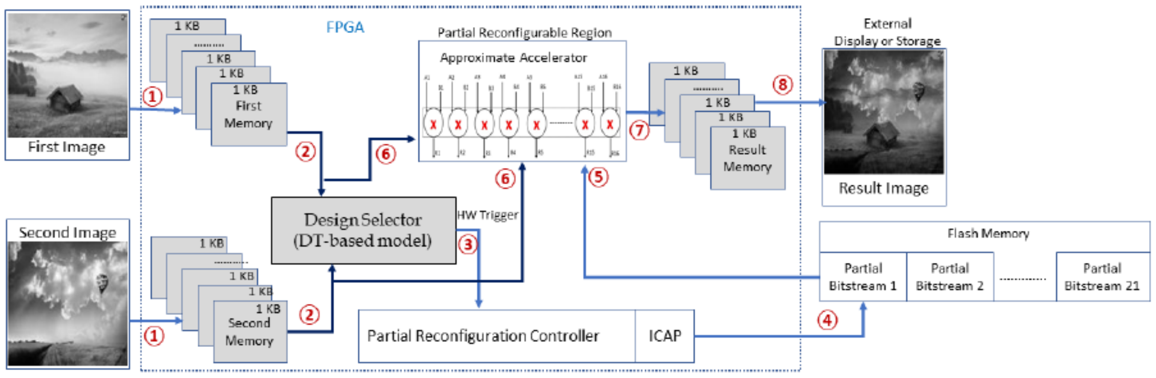


Figure 4.4: Methodology of FPGA-based Adaptive Approximate Design - Online Phase

application due to its computationally intensive nature and its amenability to approximation. As a first step, to prove the validity of the proposed design adaptation methodology, we evaluate a design without the DPR feature, utilizing the exact accelerator as well as 20 approximate accelerators that exist simultaneously, based on the proposed methodology. Thus, 21 different accelerators evaluate the outputs. Next, based on the inputs and the given TOQ, the design selector chooses the output of a specific design, which has been selected based on the DT-model. Finally, the selected result will be forwarded as the final result of the accelerator. The evaluated area and power consumption of such design is 15X and 24X more significant than the exact implementation, respectively.

We use MATLAB to read the images, re-size them to 128 $\times$ 128 pixels, convert them to grayscale and then write into coefficient (.COE) files. Such files contain the image pixels in a format that the Xilinx Core Generator can read and load. We store the images in an FPGA block RAM (BRAM). The design evaluates the average of the pixels of each image retrieved from the memory; then, the *hardware selector* decides which reconfigurable module, i.e., *bitstream* file, to load into the reconfigurable region.

The full bitstream is stored in flash memory to be booted up into the FPGA at power-up. Moreover, the partial bitstreams are stored in well-known addresses of the flash memory.

4.5 Experimental Results

In the following, we discuss the results of our proposed methodology when evaluated on image processing applications. In particular, we present the obtained accuracy results along with reports of the area resources utilized by the implemented system.

Accuracy Analysis of the Adaptive Design

We evaluate the accuracy of the proposed design over 55 examples of image blending. For each example, our TOQ (PSNR) ranges from $15dB$ to $63dB$. The images we use are from the database of "8 Scene Categories Dataset" [2], which is downloadable from [83]. It contains eight outdoor scene categories; coast, mountain, forest, open country, street, inside the city, tall buildings and highways. Figure 4.5 shows the minimum, maximum and average curves of the obtained output quality, each evaluated over 55 examples. Generally, for image processing applications, the quality is typically considered acceptable if $PSNR \geq 30dB$, and otherwise unacceptable [98]. Based on that, the design adaptation methodology has been executed 1870 times while the TOQ has been satisfied for 1530 times. Thus, the accuracy of our obtained results in Figure 4.5 is 81.82%.

Area Analysis of the Adaptive Design

Table 4.2 shows the primary resources of the XC7VX485T-2FFG1761 FPGA [99]. Moreover, it shows the resources required for the image blending application utilizing

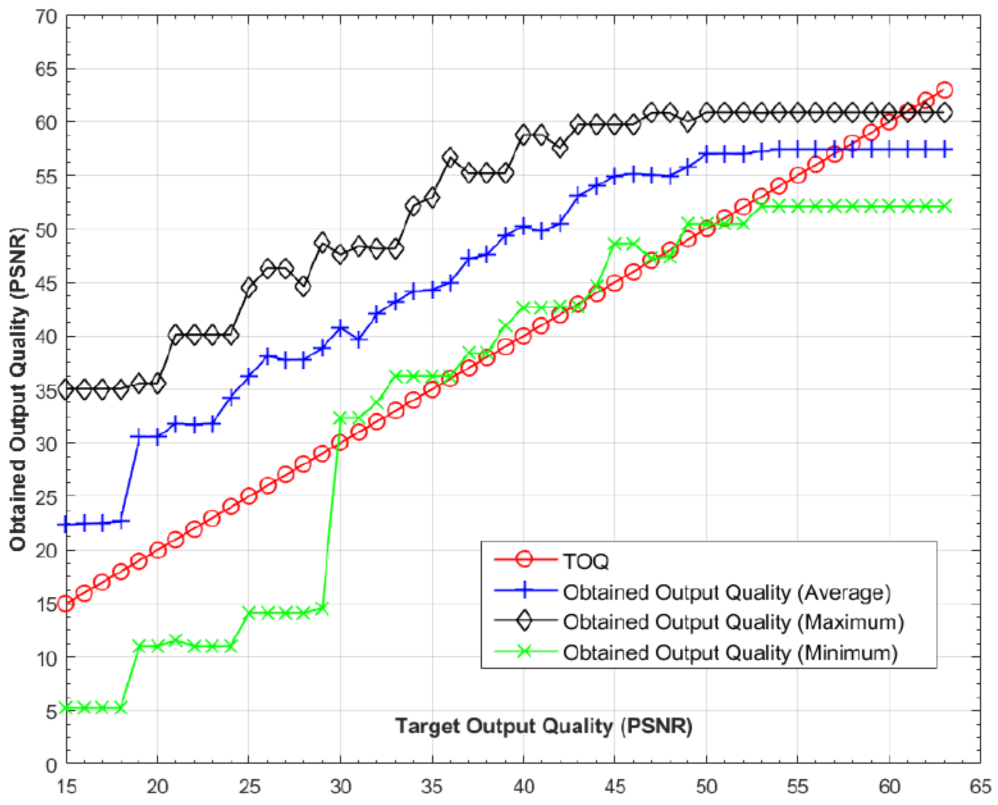


Figure 4.5: Obtained Output Quality for FPGA-based Adaptive Image Blending

an approximate accelerator, both static and adaptive implementation. Design checkpoint files (.DCP) are a snapshot of a design at a specific point in the flow, which includes the current netlist, any optimizations made during implementation, design constraints and implementation results. For the static implementation the .DCP file is 430 KByte only, while for the dynamic implementation, it is 17411 KByte. This increase in the file size is due to the logic which has been added to enable dynamic partial reconfiguration, as well as the 20 different implementations for the reconfigurable module (RM). Moreover, the overhead of such logic is shown in the increased number of the occupied Slice LUTs and Slice Registers. However, both static and dynamic implementations have the same size of the *bitstream* file (692 KByte), which is to be downloaded into the FPGA. DPR enables downloading the partial *bitstream*

Table 4.2: Area/Size of Static and Adaptive Approximate Accelerator

Design	.DCP File KByte	Slice LUTs	Slice Registers	RAMB36	RAMB18	Bonded IOB	DSPs	Bitsream size (KByte)
XC7VX485T-2FFG1761 FPGA	—	303600	607200	1030	2060	700	2800	—
Static Design	430	1472	357	235	51	65	0	19799
Adaptive - Top	17411	12876	15549	235	51	65	0	19799
Adaptive - Exact RM	770	1287	0	0	0	0	0	692
Adaptive - Max Approx RM	647	800	0	0	0	0	0	692
Adaptive - Min Approx RM	458	176	0	0	0	0	0	692

into the FPGA rather than the full *bitstream*. Thus, downloading 692 KByte rather than 19799 KByte would be 28.6 $\times$ faster. Since different variable-size reconfigurable modules will be assigned to the same reconfigurable region, it must be large enough to fit the biggest one, i.e., the exact accelerator in our methodology.

Table 4.2 shows the main features for the Xilinx XC7VX485T-2FFG1761 device, including the number of slice LUTs, slice registers, and the number of block RAM. The total capacity of block RAM is 37080 Kbit, which could be arranged as 1030 blocks of size 36Kbit each or 2060 blocks of size 18Kbit each. The reconfigurable module (RM) with exact implementation occupies 1287 Slice LUTs. However, the number of Slice LUTs occupied by the RM with approximate implementation varies from 800 to 176 LUTs. Thus, the area of the approximate RM varies from 62.16% to 13.68% of the area of the exact RM. Despite all of that, all 21 RMs have the same *bitstream* size of 692 KB.

4.6 Summary

In order to assure the quality of approximation by design adaptation, in this chapter, we described the proposed methodology to adapt the architecture of the FPGA-based approximate design using dynamic partial reconfiguration. The proposed design with low power, reduced area, small delay and high throughput is based on runtime adaptation for changing inputs. For this purpose, we utilized a lightweight and energy-efficient *design selector* built based on decision tree models. Such input-aware *design selector* determines the most suitable approximate architecture which satisfies user given quality constraints for specific inputs. Then, the partial *bitstream* file of the selected design is downloaded into the FPGA. Dynamic partial reconfiguration allows quickly reconfiguring the FPGA devices without having to reset the complete device. The obtained analysis results of image blending application showed that it is possible to satisfy the TOQ with an accuracy of 81.82%, utilizing a partial *bitstream* file that is 28.6× smaller than the full *bitstream*. The next chapter explains the quality assurance of approximate design by error compensation rather than design adaptation.

Chapter 5

Self-Compensating Approximate Accelerators

In the previous chapters, we proposed and evaluated a methodology to assure the quality of the results of approximate accelerators through building a runtime adaptive approximate design. For that, we utilized machine learning techniques, i.e., DT and NN, to select the most suitable module from a set of 20 designs. Similarly, in this chapter, we propose a runtime self-healing approximate accelerator utilizing decision trees to compensate for the error based on the inputs. For that, a corrective term, which is predicted based on the input data using a decision tree model, is added to the erroneous result.

5.1 Introduction

Approximation errors persist permanently during the operational lifetime of the approximate accelerators. Thus, it is necessary to develop techniques that can alleviate approximation error and enhance the accuracy with minimal overhead, when a high inexactness cannot be afforded. Therefore, it is crucial to tackle this issue at the early design stage and change the architecture of approximate accelerators by building a lightweight internal error compensation/recovery module with minimal overhead. However, approximate computing is still an immature computing paradigm, where to the best of our knowledge, a formal model of the impact of approximation on accuracy metric is still missing [100]. Accuracy performance of approximate designs is highly input-dependent, where we know relatively little about enhancing the accuracy of approximation in a disciplined manner. In this chapter, we propose a novel machine learning-based self-compensating approximate accelerator, aiming to improve the accuracy of the approximated results. There is no clear relationship between the inputs of approximate accelerators and their errors. Therefore, such accelerators are designed based on the inputs by employing an ML-based compensation module to capture input dependency of error. This technique leads to a noteworthy reduction in error magnitude, with negligible overhead.

A severe limitation of the state-of-the-art self-compensating methodology is that it mainly employed in parallel architectures by integrating approximate components with their complementary designs, i.e., having the same error magnitude with opposite polarity. However, obtaining such integral parts is not always guaranteed. Therefore, an approximation methodology is required that can provide a complementary effect within a single computing element. In Chapters 3 and 4, we have designed and implemented a machine learning-based technique that seek to control the quality of approximate computing by selecting the most suitable approximate design based on the inputs. Nevertheless, this technique is efficient when having a set of approximate designs to choose the most adequate among them, which is not always applicable.

As a proof of concept, we consider approximate accelerators with 8-bit approximate array multipliers. Such accelerators have 9 bits of the results being approximated. Also, they utilize full adder (FA) cells, known as approximate mirror adder 5 (*AMA5*) [8], which provides a simplified design with the reduced area, power and delay (*Design19* in Table 2.8). The challenge is to build an efficient error compensation module, which considers the value of the inputs. Thus, machine learning techniques are used to capture such dependency. Finally, we employ an image blending application, where two images are multiplied pixel-by-pixel to demonstrate a practical application of self-compensating approximate accelerators.

5.2 Error Compensation Approach

In a self-compensating approximate accelerator, we propose to integrate an input-dependent compensation module in such a way that the accumulative error is reduced. Figure 5.1(a) shows the design of a simplified accelerator with two approximate multipliers. The magnitude of error $e1$ depends on inputs A and B, while the magnitude of error $e2$ depends on inputs C and D. Whereas, $e1$ does not equal $e2$, i.e., $e1 \neq e2$, unless $\{A, B\} = \{C, D\}$. The final accelerator error is e, where $e = e1 + e2$. The maximum error is $|e1| + |e2|$.

Here, without loss of generality, we consider accelerators constructed utilizing 8-bit approximate array multipliers based on *AMA5* FAs with 9-bits of the results being approximated. However, the proposed methodology applies to any approximate accelerator design, e.g., approximate adders [19], dividers [21] and multiply-accumulate units [60].

The main challenge in the design of self-compensating accelerators is the development of the input-dependent compensation module that has a minimal area, delay

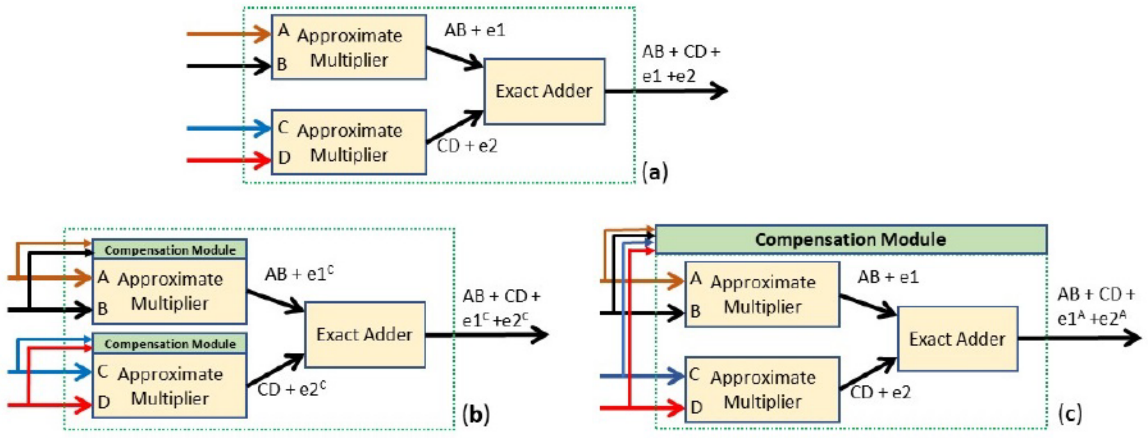
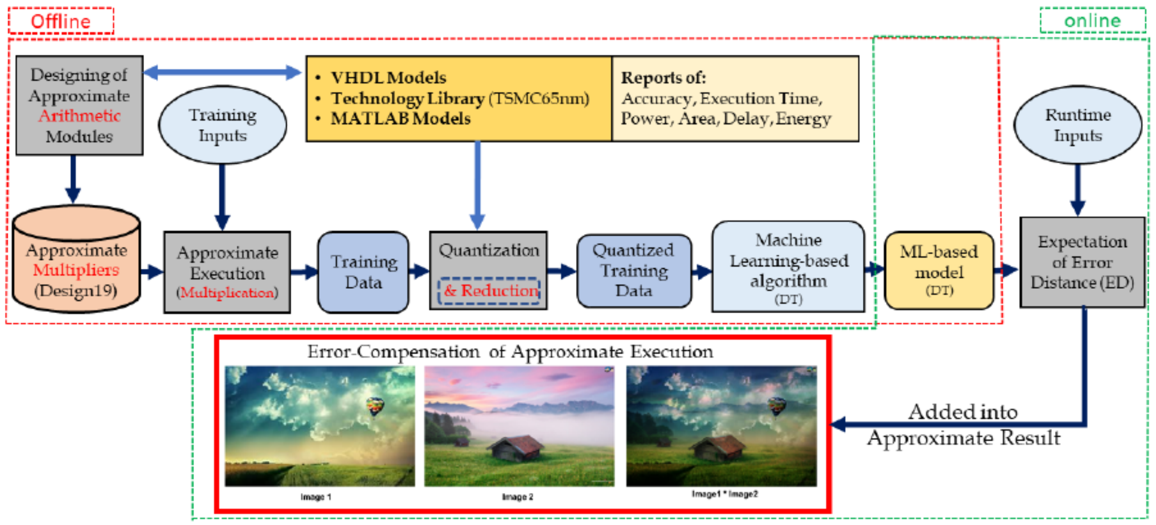


Figure 5.1: Simplified Architecture for Accelerator of Two Approximate Multipliers, (a) Without Error Compensation, (b) With Error Compensation Module per Approximate Component, (c) With Error Compensation Module per Approximate Accelerator.

and power overhead. An overview of the proposed design methodology is depicted in Figure 5.2. It is a customized version from the general proposed methodology shown in Figure 1.1, where we use 1) a single approximate design rather than 20 designs; 2) DT-based model only, which is simpler and faster than the NN-based model; and 3) the error distance (ED) metric in building the DT-based model rather than the PSNR, which is used in the adaptive design.



Offline
online
Designing of Approximate Arithmetic Modules
Training Inputs
VHDL Models
Technology Library (TSMC65nm)
MATLAB Models
Reports of: Accuracy, Execution Time, Power, Area, Delay, Energy
Runtime Inputs
Approximate Multipliers (Design19)
Approximate Execution (Multiplication)
Training Data
Quantization & Reduction
Quantized Training Data
Machine Learning-based algorithm (DT)
ML-based model (DT)
Expectation of Error Distance (ED)
Error-Compensation of Approximate Execution
Image 1
Image 2
Image1 * Image2
Added into Approximate Result

Figure 5.2: Design flow for Approximate Accelerator Compensation Module

Table 5.1: Characteristics of Approximate Accelerator Components, i.e., Approximate Multiplier and Compensation Module

Design	Dynamic Power (mW)	Slice LUTs	Occupied Slices	Period (ns)	Frequency (MHz)	Energy (pj)
Exact Multiplier	442	85	33	8.747	114.32	3866.2
Approximate Multiplier	113	31	11	4.625	216.22	522.6
Compensation Module	2.79	23	8	2.213	451.88	6.6

The fundamental step in the proposed flow is designing an approximate multiplier, which is the essential building component of the accelerator. Table 5.1 shows the design characteristics of the 8-bit approximate array multiplier, including its area, delay, power and energy consumption. The attributes of the exact array multiplier are also shown in the table. We evaluate the power, area, delay and energy utilizing the XC6VLX75T FPGA, which belongs to the Virtex-6 family, and the FF484 package. We use Mentor Graphics Modelsim [65], Xilinx XPower Analyser [66] and Xilinx Integrated Synthesis Environment (ISE 14.7) tool suite [67].

Since the magnitude of approximation error is input dependent, we apply an exhaustive simulation by having $2^8 = 256$ different values for each input. Thus, we have $256 \times 256 = 65{,}536$ different input combinations with their associated error distance (ED), which constitute our training data. Figure 5.3 shows the histogram distribution for the ED of the approximate multiplier. Approximate computing relies on the principle of failing small or rarely fail [14]. However, out of the 65536 possible input combinations, 62420 have inexact results. Thus the error rate (ER) is 95.25%. Therefore, a high error rate (ER), requires having a small value of ED to get an acceptable final result. As shown in Figure 5.3, minor errors occur more frequently than large errors. For example, we have only 1575 input combinations with ED>500, which is almost 2.48% of the erroneous inputs. Considering such extreme values in ED may simplify building the compensation module. Error distance has 176 distinctive values, where the minimum ED is 4, the maximum ED is 756, and the average is 185.

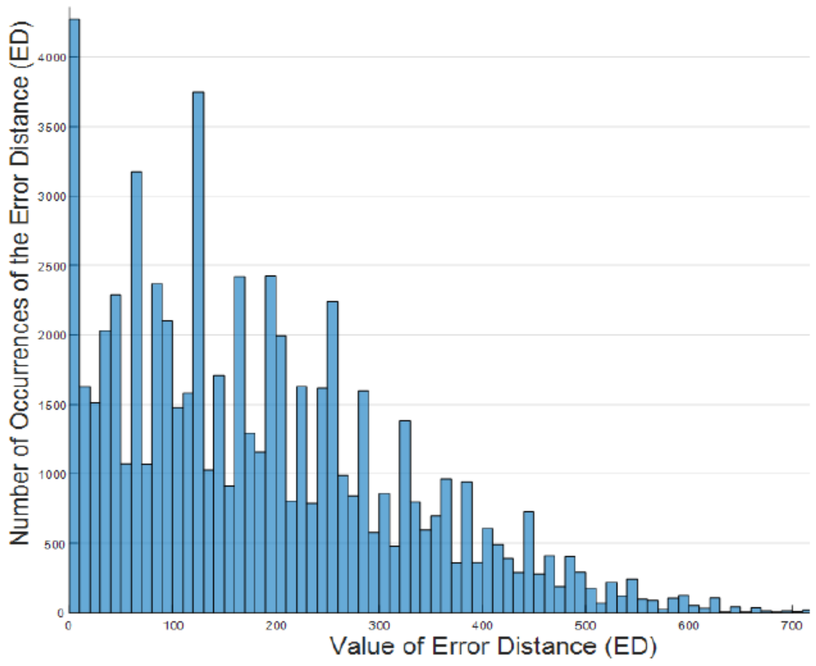


Figure 5.3: Histogram Distribution of the Error Distance (ED) of the Approximate Multiplier

Generally, whenever the error occurs for a small fraction of input combinations, i.e., error rate (ER) is low, an approximate design with simple error correction, such as adding a constant corrective magnitude, exhibits better performance compared to the exact model. However, our approximate accelerator has a high ER of 95.25%, which makes simple error correction inapplicable.

To predict the ED based on the value of the inputs, we use a lightweight machine learning-based algorithm, i.e., classification decision tree (DT), based on the *C5.0* algorithm [75], given in R [74]. Decision trees that are fast, memory-efficient and have a simple structure of nesting if-else statements are quite well able to model the non-linear relationship between the inputs and error distance. We notice that the

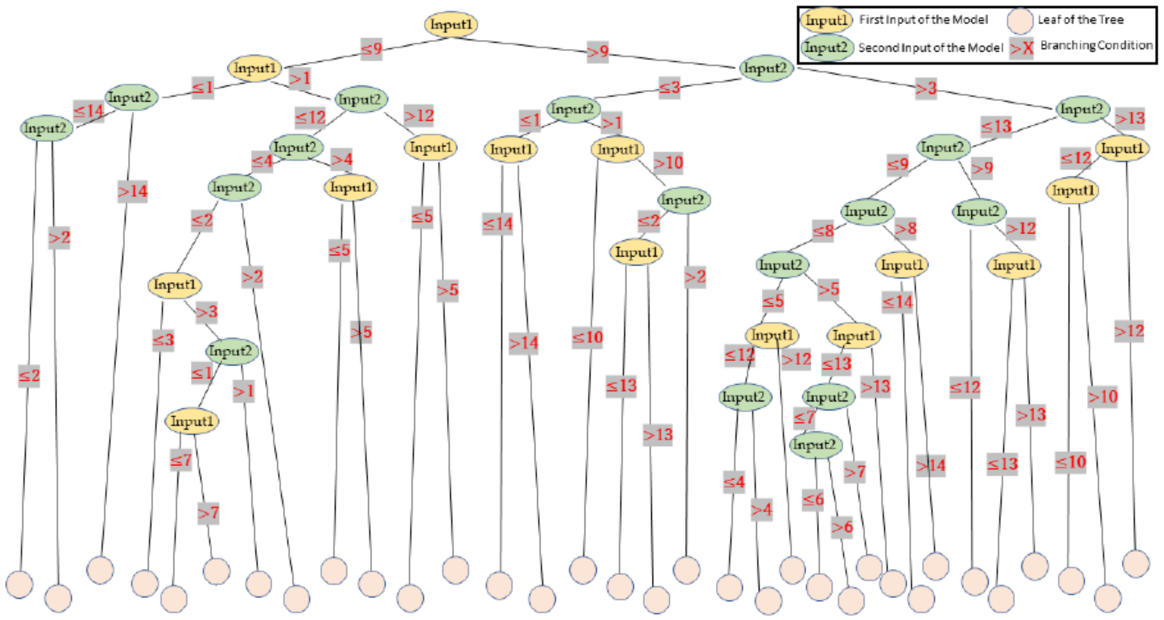

Input1 First Input of the Model
Leaf of the Tree
Input2 Second Input of the Model
>X Branching Condition
Input1
≤9
>9
Input2
Input1

Figure 5.4: The Structure of the Decision Tree-based Model

inputs of the approximate design with close magnitudes are associated with a very close ED. Consequently, we quantize the inputs based on their magnitudes into 16 different clusters. Thus, the model has $16 \times 16 = 256$ different combinations of input clusters, rather than 65,536 inputs, which significantly simplifies its internal structure.

Figure 5.4 shows the structure of the decision tree that we obtained. The leaves of the tree represent the expected values of the error metric, i.e., ED, that should be added to the approximate result in order to correct the final output, while the internal nodes represent the conditional decision points which are the inputs of the model, i.e., the first input (*Input1*) and the second input (*Input2*) of the approximate design. The values associated with the connections between the *conditional decision points* represent the cluster of the inputs, i.e., from 1 to 16. For example, the first branch in Figure 5.4 examines the class of *Input1*, and then it traces to the left-side if it is ≤ 9 or traces to the right side if the class is >9.

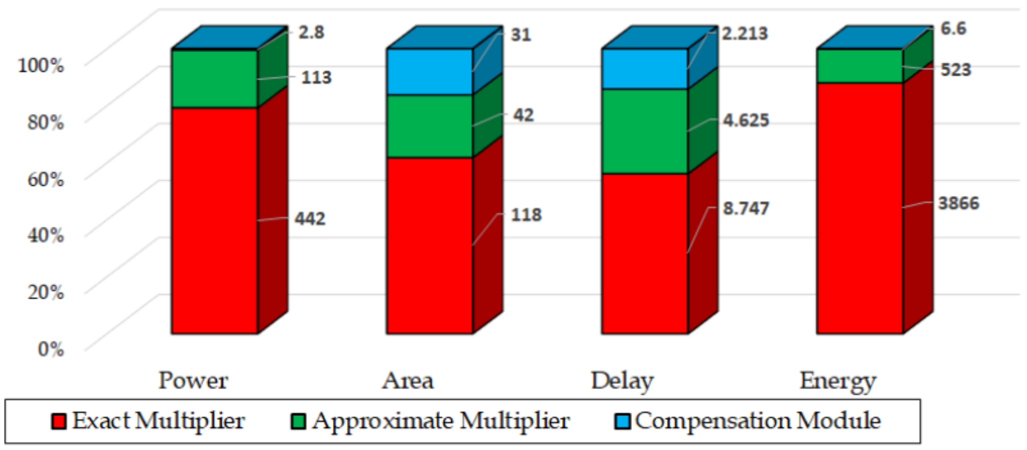


Figure 5.5: Power, Area, Delay and Energy of Approximate Accelerator Components

Compensation Module Overhead

To show the effectiveness of the proposed compensation module, we perform an accuracy evaluation utilizing its implementation in MATLAB. Moreover, we evaluate its power, area, delay and energy. Table 5.1 shows the obtained results, where the power consumption of the module is about $2.8mW$, which forms about 2.4% added power to the approximate multiplier. Similarly, the introduced area, delay and energy overhead of the error compensation module with respect to the approximate multiplier is about 42.5%, 32.4% and 1.2%, respectively. Such cost is negligible when compared to the approximate multiplier where we integrate multiple instances of it within the approximate accelerator.

Figure 5.5 shows a relative representation of the power, area, delay and energy of the approximate multiplier, compensation module as well as the exact multiplier. Despite the module added overhead, the approximate multiplier with the accompanying module (as shown in Figure 5.1(b)) still has a reduction of 73.8%, 38.1%, 21.8% and 86.3% in power, area, delay and energy, respectively, compared to the exact multiplier.

The error of the approximate multiplier, i.e., $e1$, will be reduced to $e1^C$, which represents $e1$ after being alleviated by the compensation module at the component level, where $e1^C \ll e1$. Moreover, to amortize the overhead of the proposed module, we suggest another architectural configuration with a single compensation module for the approximate accelerator, as shown in Figure 5.1.(c), rather than having a dedicated module for each approximate component, as shown in Figure 5.1.(b). Such a proposed design is applicable when different data processed at various parts have similar values, e.g., adjacent image pixels. Thus, the introduced error is roughly identical.

In image processing applications, the accelerator processes adjacent image pixels, which usually have close values. Therefore, for image blending in a multiplicative mode where the pixels of the two images are multiplied pixel-by-pixel, we propose to divide the image into three segments (coloured-components), i.e., red, green and blue, as in Chapter 3. Each coloured component is processed on a separate accelerator. For that, the compensation module of the approximate accelerator evaluates the average value of the pixels for each frame coloured-component. Based on that, a compensation value is calculated, i.e., predicted by decision tree-based model, and then added to all the pixels of the frame coloured-component. Thus, the error of the approximate accelerator, i.e., $e1 + e2$, will be reduced to $e1^A + e2^A$, based on the error compensation module at the accelerator level. The next section evaluates the accuracy of the implemented compensating module that we developed.

5.3 Experimental Results

This section presents the experimental results obtained by introducing the compensation module both at the component and the accelerator levels.

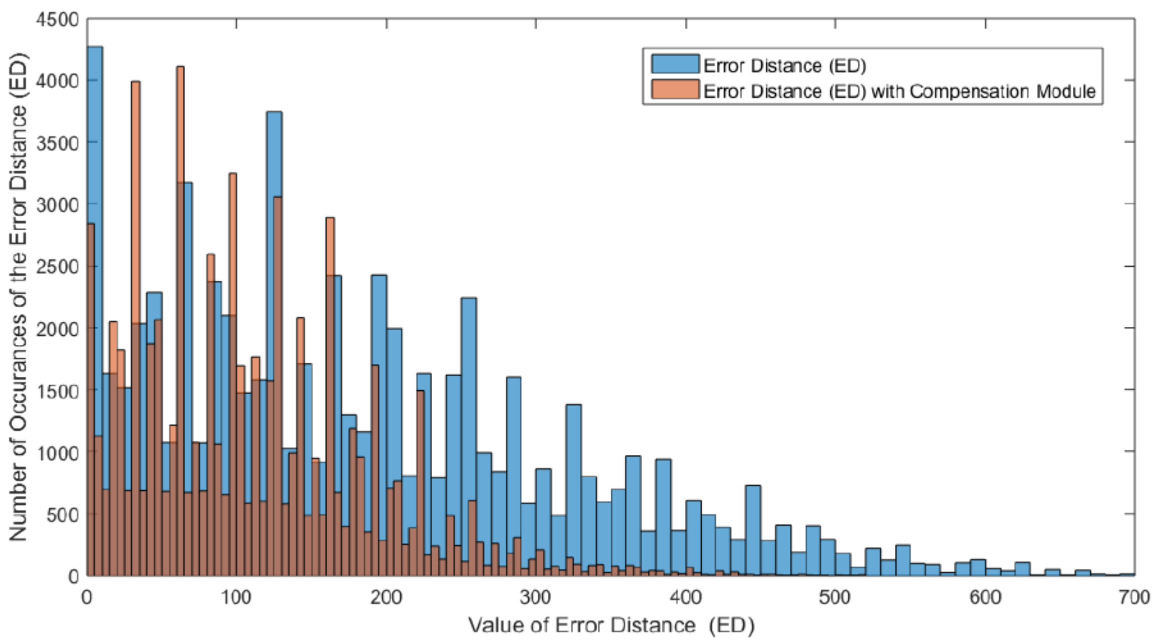


Figure 5.6: Distribution of Error Distance (ED) of Approximate Multiplier with/without the Error Compensation Module

Compensation Module at Component Level

To evaluate the performance of the compensation module, which is shown in Figure 5.1.(b), we perform an exhaustive simulation of the approximate multiplier. Figure 5.6 shows the histogram of the error distance of the approximate multiplier without compensation as well as and the compensated value by integrating the compensation module into the approximate component. Such a module will enhance the accuracy of the result by adding a compensation value based on the decision tree-model to reduce the final error distance (ED). There is a significant reduction in error characteristics, i.e., in both error magnitude and error frequency.

As shown in Table 5.2, the error proposed compensation module, reduces the maximum ED from 756 to 520, while the mean ED is decreased from 185 to 110. The number of input combinations with the erroneous result, where ED>500, is reduced from 1575 input combinations to 16, which is a significant quality improvement. The

Table 5.2: Error Distance (ED) of Approximate Multiplier without/with the Error Compensation Module

	ED=0	ED!=0	Maximum ED	Mean ED	ED>300	ED>400	ED>500
Original Error Distance	3116	62420	756	185	12922	5454	1575
Compensated Error Distance	2177	63359	520	110	1458	218	16

number of input combinations with an ED>400 and ED>300 is reduced from 5454 to 218, and from 12922 to 1458, respectively. This noteworthy improvement in the quality of results validates the importance of the added compensation module. Moreover, the number of distinct values of the ED is lowered from 176 to 129. Without the proposed compensation module, the approximate multiplier has 3116 error-free input combinations, i.e., the error rate is 95.25%. However, adding an ML-based compensation module reduces the error-free input combinations into 2177, i.e., the error rate is 96.68%, by erroneously adding a compensation value into error-free results. The increase in the ER is due to model imperfection, even though the final accuracy has significant improvement. Similarly, in some cases, the compensation module increases the ED rather than reduce it. Overall, there is a substantial reduction in error magnitude and error frequency, where this will enhance the final accuracy of the utilized error-resilient application.

Image Blending Application with Compensation Module

To evaluate the proposed self-compensating approximate accelerators in an error-tolerant application, we deployed them in an image blending application, where two images are multiplied pixel-by-pixel. The photos used in blending and their corresponding accurate results are shown in Figure 5.7, where the size of each image is 250 × 400 pixels. Two configurations of compensation modules are used: 1) a compensation module for each approximate component; and 2) a single compensation module

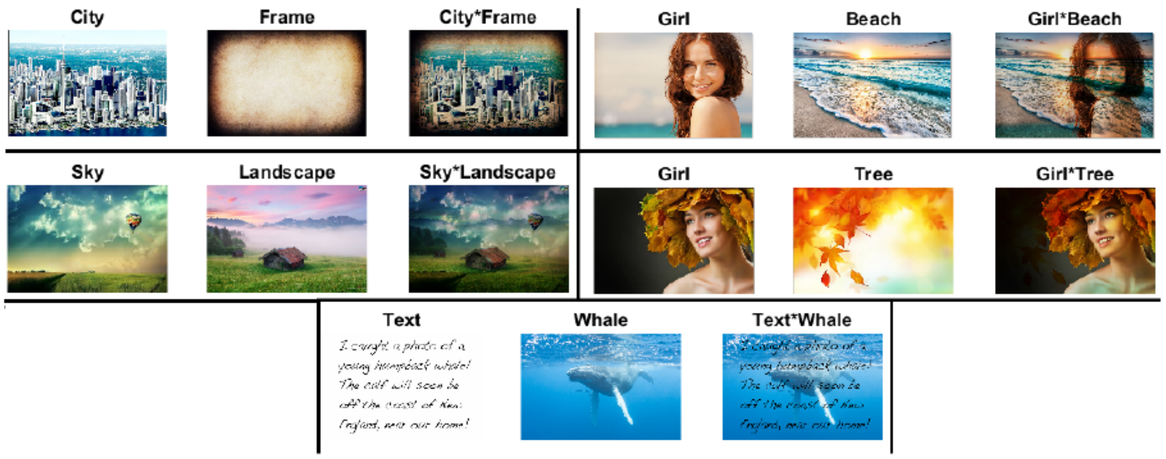


Figure 5.7: Images used to Evaluate the Error Compensation Module

for all approximate components.

The PSNR of the obtained results is displayed in Figure 5.8, where all blending examples have improved quality, i.e., PSNR, whenever we use the compensation module. The improvement in the output quality when the compensation module is incorporated at the component level is more significant than the case when the module is used at the accelerator level. The shown results of image blending with error compensation have an enhanced quality, where the increase in the PSNR ranges from $2.6dB$ to $4.7dB$ with an average of $4.2dB$ for the considered examples. Thus, we can obtain an average of 9% improvement in the final quality of image blending application with negligible overhead. However, using the compensation module at the accelerator level achieves a lower accuracy enhancement, where the compensation value is evaluated for 10^5 multiplication. The accuracy of approximate accelerators can be enhanced by integrating the compensation module at a finer granularity level.

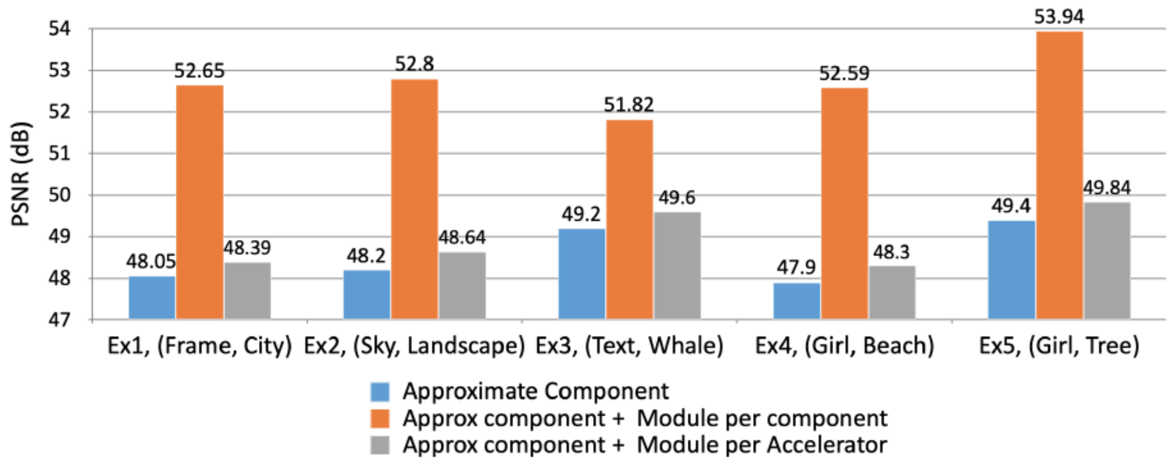


Figure 5.8: Output Quality (PSNR) of Image Blending, (a) Without Error Compensation, (b) With Error Compensation Module per Approximate Component, (c) With Error Compensation Module per Approximate Accelerator

5.4 Summary

In this thesis, we proposed a general methodology for quality assurance of approximate computing. We explained how to assure the quality of approximation results through design adaptation. Then, we implemented the proposed design in both software and hardware, as explained in Chapters 3 and 4, respectively. In this chapter, based on our general methodology for quality assurance of approximation, we developed a third novel machine learning-based self-compensating approximate computing for quality enhancement. It has shown an opportunity for error reduction without requiring similar approximate computing elements, i.e., a mirror pair. The proposed decision tree-based error compensation module can achieve a noteworthy enhancement in accuracy performance without compromising the power consumption, area and speed. Thus, we were able to enhance the quality of approximation through three different approaches successfully.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Approximate computing has re-emerged as an energy-efficient computing paradigm for error-tolerant applications. Thus, it is promising to be integrated within the architecture and algorithms of brain-inspired computing, which has massive device parallelism and the ability to tolerate unreliable operations. However, there are essential questions to be answered before approximate computing can be made a viable solution for energy-efficient computing, such as [101]: (1) how much to approximate at the component level to be able to observe the gains at the system level; (2) how to measure the final quality of approximation; and (3) how to maintain the desired output quality of an approximate application.

Towards addressing these challenges, in this thesis, we proposed a methodology that assures the quality of approximate computing through design adaptation based on fine-grained inputs and user preferences. For that, we designed a lightweight machine learning-based model, that functions as a design selector, to select the most suitable approximate designs to ensure the final quality of the approximation. Moreover, we proposed another technique for quality assurance through error compensation. For that, we designed a decision tree-based model to predict the error magnitude based on the applied inputs. Thus, the final result is compensated accordingly. In the sequel, we summarize each of the main contributions of this thesis.

The first contribution of this thesis is the design and evaluation of a library of 8 and 16-bit approximate multipliers with various settings based on approximation in partial product summation. The proposed designs were evaluated at circuit and application level based on their accuracy, area, delay and power consumption, where various optimal designs have been identified in terms of the considered design metrics.

Second, we proposed a novel methodology to generate an adaptive approximate design that satisfies a user given quality constraints, based on the applied inputs. For that, we have built a machine learning-based model (that functions as a design selector) to determine the most suitable approximate design for the applied inputs based on the associated error metrics. To solve the *design selector* model, we used decision tree and neural network techniques to select the approximate design that matches the closest accuracy for the applied inputs.

The next contributions of this thesis are the *software* and *hardware* implementations of the proposed methodology, with negligible overhead. The obtained analysis results of the image processing application showed that it is possible to satisfy the TOQ with accuracy ranging from 80% to 85.7% for various error-resilient applications. The *FPGA-based* adaptive approximate accelerator with constraints on size, cost and power consumption relies on dynamic partial reconfiguration to assist in satisfying these requirements.

Moreover, we proposed a novel machine learning-based *self-compensating* approximate accelerators for enhancing the quality of approximate computing applications. The proposed methodology has shown an opportunity for error reduction without requiring similar approximate computing elements, i.e., a mirror pair. The proposed decision tree-based compensation module achieves a noteworthy enhancement in accuracy performance without compromising the power consumption, area and speed of the original design.

In summary, the general proposed design adaptation methodology, with three different implementation techniques, can be seen as a basis for automatic quality assurance. It offers a promising solution to reduce the approximation error while maintaining approximation benefits. Some of the potential future enhancements and directions for further research are detailed in the next section.

6.2 Future Work

This thesis lays the ground for a promising framework for monitoring and controlling approximation quality, through input-dependent design adaptation. Building on the proposed methodology and experimental results presented in this thesis, several enhancements and directions for further research can be explored and pursued. Besides approximation quality, approximate computing is getting closer to be one of the mainstream computing approaches in future systems. Towards this goal, there are tremendous research directions that connect approximate computing with design verification, testing, reliability and safety, including:

- In its present form, the methodology considers only approximate integer multipliers, i.e., 8 and 16-bit array multipliers, with 20 different settings. However, real numbers could be represented by fixed-point or floating-point. An exciting extension of this work can be the integration of floating-point and fixed-point multipliers in the approximate library, as well as other arithmetic modules such as adders and dividers.

- Testing of approximate circuits is quite challenging, where both approximate and manufacturing faults should be distinguishable. In [102], we proposed an approximation-conscious multi-level IC test flow, which classifies the output of the test process to be good, bad, or good-enough IC. As future work, we plan to target fault models other than the Stuck-At-Fault model with various accuracy metrics.

- We implemented an entirely software and hardware adaptive designs. However, a hardware/software co-design of AC systems would combine the benefits of both techniques, which we may consider in the future. Moreover, an optimal approximate solution will not be achievable in a single layer of the computing stack. Therefore, a cross-layer design methodology, including hardware, software, architecture, application and algorithms, needs more investigation.

- Finally, in hardware-based fault-tolerant systems, components redundancy techniques, such as double modular redundancy (DMR) and triple modular redundancy (TMR), are used for fault detection and correction, respectively. Thus, the effectiveness of approximate computing in fault-tolerant systems is under consideration.

Bibliography

[1] CEVA NeuPro: a family of AI processors for deep learning at the edge, 2019. `https://www.ceva-dsp.com/product/ceva-neupro/`, Last accessed on 2020-05-10.

[2] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[3] B. Moons and M. Verhelst. Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 4(4):475–486, 2014.

[4] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *European Test Symposium*, pages 1–6, 2013.

[5] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan. Approximate computing and the quest for computing efficiency. In *Design Automation Conference*, pages 1–6, 2015.

[6] R. Ragavan, B. Barrois, C. Killian, and O. Sentieys. Pushing the limits of voltage over-scaling for error-resilient applications. In *Design, Automation Test in Europe*, pages 476–481, 2017.

[7] P. Roy, R. Ray, C. Wang, and W. F. Wong. ASAC: Automatic Sensitivity Analysis for Approximate Computing. *SIGPLAN Not.*, 49(5):95–104, 2014.

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

[9] R. Nair. Big data needs approximate computing: Technical perspective. *Communications of the ACM*, 58(1):104–104, 2014.

[10] A. Mishra, R. Barik, and S. Paul. iACT: A software-hardware framework for understanding the scope of approximate computing. In *Workshop on Approximate Computing Across the System Stack*, pages 1–6, 2014.

[11] J. Bornholt, T. Mytkowicz, and K. McKinley. UnCertain: A first-order type for uncertain data. *SIGPLAN Notices*, 49(4):51–66, 2014.

[12] M. Laurenzano, P. Hill, M. Samadi, S. Mahlke, J. Mars, and L. Tang. Input responsiveness: Using canary inputs to dynamically steer approximation. In *Programming Language Design and Implementation*, pages 161–176. ACM, 2016.

[13] D. Mahajan, A. Yazdanbakhsh, J. Park, B. Thwaites, and H. Esmaeilzadeh. Towards statistical guarantees in controlling quality tradeoffs for approximate acceleration. In *International Symposium on Computer Architecture*, pages 66–77, 2016.

[14] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Design Automation Conference*, pages 1–9, 2013.

[15] E. Nogues, D. Menard, and M. Pelcat. Algorithmic-level approximate computing applied to energy efficient hevc decoding. *IEEE Transactions on Emerging Topics in Computing*, 7(1):5–17, 2019.

[16] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural acceleration for general-purpose approximate programs. In *International Symposium on Microarchitecture*, pages 449–460, 2012.

[17] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel. Invited: Cross-layer approximate computing: From logic to architectures. In *Design Automation Conference*, pages 1–6, 2016.

[18] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique, and A. Kumar. Area-Optimized Low-Latency Approximate Multipliers for FPGA-Based Hardware Accelerators. In *Design Automation Conference*, pages 1–6, 2018.

[19] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi. Approximate xor/xnor-based adders for inexact computing. In *Nanotechnology*, pages 690–693, 2013.

[20] K. M. Reddy, Y. B. N. Kumar, D. Sharma, and M. H. Vasantha. Low power, high speed error tolerant multiplier using approximate adders. In *VLSI Design and Test*, pages 1–6, 2015.

[21] M. Imani, R. Garcia, A. Huang, and T. Rosing. Cade: Configurable approximate divider for energy efficiency. In *Design, Automation Test in Europe Conference*, pages 586–589, 2019.

[22] J. Melchert, S. Behroozi, J. Li, and Y. Kim. Saadi-ec: A quality-configurable

approximate divider for energy efficiency. *IEEE Transactions on Very Large Scale Integration Systems*, 27(11):2680–2692, 2019.

[23] S. Hashemi, R. I. Bahar, and S. Reda. A low-power dynamic divider for approximate applications. In *Design Automation Conference*, pages 1–6, 2016.

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

[25] A. Momeni, J. Han, P. Montuschi, and F. Lombardi. Design and analysis of approximate compressors for multiplication. *IEEE Transactions on Computers*, 64(4):984–994, 2015.

[26] G. Zervakis, K. Tsoumanis, S. Xydis, N. Axelos, and K. Pekmestzi. Approximate multiplier architectures through partial product perforation: Power-area tradeoffs analysis. In *Symposium on VLSI*, pages 229–232, 2015.

[27] W. Yeh and Ch. Jen. High-speed and low-power split-radix FFT. *IEEE Transactions on Signal Processing*, 51(3):864–874, 2003.

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

[29] Kh. Y. Kyaw, W. L. Goh, and K. S. Yeo. Low-power high-speed multiplier for error-tolerant application. In *International Conference of Electron Devices and Solid-State Circuits*, pages 1–4, 2010.

[30] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han. A comparative evaluation of approximate multipliers. In *Nanoscale Architectures*, pages 191–196, 2016.

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

[32] M. Imani, D. Peroni, and T. Rosing. CFPU: Configurable floating point multiplier for energy-efficient computing. In *Design Automation Conference*, pages 1–6, 2017.

[33] M. Imani, R. Garcia, S. Gupta, and T. Rosing. RMAC: Runtime Configurable Floating Point Multiplier for Approximate Computing. In *International Symposium on Low Power Electronics and Design*, pages 12:1–12:6, 2018.

[34] W. Baek and T. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. *SIGPLAN Notices*, 45(6):198–209, June 2010.

[35] M. Samadi, J. Lee, D. Jamshidi, A. Hormati, and S. Mahlke. SAGE: Self-tuning approximation for graphics engines. In *International Symposium on Microarchitecture*, pages 13–24, 2013.

[36] B. Grigorian, N. Farahpour, and G. Reinman. BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing. In *International Symposium on High Performance Computer Architecture*, pages 615–626, 2015.

[37] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke. Rumba: An online quality

management system for approximate computing. In *International Symposium on Computer Architecture*, pages 554–566, 2015.

[38] T. Wang, Q. Zhang, N. Kim, and Q. Xu. On effective and efficient quality management for approximate computing. In *International Symposium on Low Power Electronics and Design*, pages 156–161, 2016.

[39] X. Chengwen, W. Xiangyu, Y. Wenqi, X. Qiang, J. Naifeng, L. Xiaoyao, and J. Li. On quality trade-off control for approximate computing using iterative training. In *Design Automation Conference*, pages 1–6, 2017.

[40] A. Raha, H. Jayakumar, and V. Raghunathan. Input-based dynamic reconfiguration of approximate arithmetic units for video encoding. *IEEE Transactions on Very Large Scale Integration Systems*, 24(3):846–857, 2016.

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

[42] J. Pirkl, A. Becher, J. Echavarria, J. Teich, and S. Wildermann. Self-adaptive FPGA-based image processing filters using approximate arithmetics. In *International Workshop on Software and Compilers for Embedded Systems*, SCOPES, pages 89–92. ACM, 2017.

[43] S. Xu and B. C. Schafer. Approximate reconfigurable hardware accelerator: Adapting the micro-architecture to dynamic workloads. In *International Conference on Computer Design*, pages 113–120. IEEE, 2017.

[44] S. Xu and B. C. Schafer. Toward self-tunable approximate computing. *IEEE Transactions on Very Large Scale Integration Systems*, pages 1–12, 2018.

[45] M. Orlandić and K. Svarstad. An adaptive high-throughput edge detection filtering system using dynamic partial reconfiguration. *Journal of Real-Time Image Processing*, 16(1):2409–2424, 2018.

[46] B. Krill, A. Ahmad, A. Amira, and H. Rabah. An efficient FPGA-based dynamic partial reconfiguration design flow and environment for image and signal processing IP cores. *Signal Processing: Image Communication*, 25(5):377 – 387, 2010.

[47] M. Nguyen, R. Tamburo, S. Narasimhan, and J. C. Hoe. Quantifying the benefits of dynamic partial reconfiguration for embedded vision applications. In *International Conference on Field Programmable Logic and Applications*, pages 129–135, 2019.

[48] F. N. Taher, J. Callenes-Sloan, and B. C. Schafer. A machine learning based hard fault recuperation model for approximate hardware accelerators. In *Design Automation Conference*, pages 80:1–80:6. ACM, 2018.

[49] S. Mazahir, O. Hasan, and M. Shafique. Self-compensating accelerators for efficient approximate computing. *Microelectronics Journal*, 88:9 – 17, 2019.

[50] K. Chen, W. Liu, J. Han, and F. Lombardi. Profile-based output error compensation for approximate arithmetic circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers (Early Access)*, pages 1–12, 2020.

[51] H. Jiang, J. Han, and F. Lombardi. A comparative review and evaluation of approximate adders. In *Symp. VLSI*, pages 343–348, 2015.

[52] H. A. F. Almurib, T. N. Kumar, and F. Lombardi. Inexact designs for approximate low power addition by cell replacement. In *Design, Automation Test in Europe*, pages 660–665, 2016.

[53] R. Hrbacek, V. Mrazek, and Z. Vasicek. Automatic design of approximate circuits by means of multi-objective evolutionary algorithms. In *International Conference on Design and Technology of Integrated Systems in Nanoscale Era*, pages 1–6, 2016.

[54] J. Liang, J. Han, and F. Lombardi. New metrics for the reliability of approximate and probabilistic adders. *IEEE Transactions on Computers*, 62(9):1760–1771, 2013.

[55] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel. Architectural-space exploration of approximate multipliers. In *International Conference on Computer-Aided Design*, pages 1–8. ACM, 2016.

[56] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic. *Digital integrated circuits.* Prentice Hall, 2002.

[57] B. Shao and P. Li. Array-based approximate arithmetic computing: A general model and applications to multiplier and squarer design. *IEEE Transactions on Circuits and Systems*, 62(4):1081–1090, 2015.

[58] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford University Press, 2010.

[59] D. Sengupta and S. S. Sapatnekar. Femto: Fast error analysis in multipliers through topological traversal. In *International Conference on Computer-Aided Design*, pages 294–299, 2015.

[60] M. Masadeh, O. Hasan, and S. Tahar. Input-conscious approximate multiply-accumulate (MAC) unit for energy-efficiency. *IEEE Access*, 7:147129–147142, 2019.

[61] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy. Design of voltage-scalable meta-functions for approximate computing. In *Design, Automation Test in Europe*, pages 1–6, 2011.

[62] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina. Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, Automation & Test in Europe*, pages 258–261, 2017.

[63] R. Fatemeh. High performance 8-bit approximate multiplier using novel 4:2 approximate compressors for fast image processing. *International Journal of Integrated Engineering*, 10, 2018.

[64] Y. Xu, J. D. Deng, and M. Nowostawski. Quality of service for video streaming over multi-hop wireless networks: Admission control approach based on analytical capacity estimation. In *International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 345–350, 2013.

[65] Mentor Graphics Modelsim, 2019. `https://www.mentor.com/company/higher_ed/modelsim-student-edition`, Last accessed on 2020-05-06.

[66] Xilinx XPower Analyser, 2019. `https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ug733.pdf`, Last accessed on 2020-05-10.

[67] Xilinx Integrated Synthesis Environment, 2019. `https://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html`, Last accessed on 2020-05-10.

[68] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, 2018.

[69] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. A low latency generic accuracy configurable adder. In *Design Automation Conference*, pages 86:1–86:6. ACM, 2015.

[70] X. Sui, A. Lenharth, D. Fussell, and K. Pingali. Proactive control of approximate programs. In *International Conference on ASPLOS*, pages 607–621. ACM, 2016.

[71] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[72] G. J. Williams. *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Springer, 2011.

[73] M. Masadeh, O. Hasan, and S. Tahar. Controlling approximate computing quality with machine learning techniques. In *Design, Automation and Test in Europe*, pages 1575–1578, 2019.

[74] The R project for statistical computing, 2019. `https://www.r-project.org/`, Last accessed on 2020-05-19.

[75] Package C50, 2019. `https://cran.r-project.org/web/packages/C50/C50.pdf`, Last accessed on 2020-05-19.

[76] Package rpart, 2019. `https://cran.r-project.org/web/packages/rpart/rpart.pdf`, Last accessed on 2020-05-19.

[77] L. Breiman, J. Friedman, R. Olshen, and Ch. Stone. *Classification and Regression Trees*. Chapman and Hall, Wadsworth, 1984.

[78] R. C. Barros, A. C.P.L.F de Carvalho, and A. A. Freitas. *Automatic Design of Decision-Tree Induction Algorithms*. Springer, 2015.

[79] Package Cubist, 2019. `https://cran.r-project.org/web/packages/Cubist/Cubist.pdf`, Last accessed on 2020-05-19.

[80] J. R. Quinlan. Learning with continuous classes. In *Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[81] A. Raha and V. Raghunathan. qLUT: Input-Aware Quantized Table Lookup for Energy-Efficient Approximate Accelerators. *ACM Transactions on Embedded Computing Systems*, 16(5s):130:1–130:23, 2017.

[82] X. Jiao, D. Ma, W. Chang, and Y. Jiang. LEVAX: An Input-Aware Learning-Based Error Model of Voltage-Scaled Functional Units. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2020.

[83] Modeling the shape of the scene: a holistic representation of the spatial envelope, 2020. `http://people.csail.mit.edu/torralba/code/spatialenvelope/`, Last accessed on 2020-05-08.

[84] Ch. Solomon. *Fundamentals of digital image processing a practical approach with examples in Matlab*. Wiley-Blackwell, 2011.

[85] BBC Sound Effects, 2020. `http://bbcsfx.acropolis.org.uk/`,.

[86] P. Ashok, M. Jackermeier, P. Jagtap, J. Křetínský, M. Weininger, and M. Zamani. Dtcontrol: Decision tree learning algorithms for controller representation. In *International Conference on Hybrid Systems: Computation and Control*, 2020.

[87] P. Ashok, J. Křetínský, K. G. Larsen, A. Le Coënt, J. H. Taankvist, and M. Weininger. Sos: Safe, optimal and small strategies for hybrid markov decision processes. In *Quantitative Evaluation of Systems*, pages 147–164. Springer International Publishing, 2019.

[88] T. Yang, T. Ukezono, and T. Sato. Low-power and high-speed approximate multiplier design with a tree compressor. In *International Conference on Computer Design*, pages 89–96, 2017.

[89] Partial Reconfiguration User Guide, 2013. `https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ug702.pdf`, Last accessed on 2020-05-13.

[90] K. Vipin and S. A. Fahmy. FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications. *ACM Computing Surveys*, 51(4):72:1–72:39, 2018.

[91] VC707 Evaluation Board for the Virtex-7 FPGA: User Guide, 2019. `https://www.xilinx.com/support/documentation/boards_and_kits/vc707/ug885_VC707_Eval_Bd.pdf`, Last accessed on 2020-05-13.

[92] D. Koch. *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications*. Springer, 2012.

[93] Xilinx Partial Reconfiguration Controller v1.3, LogiCORE IP Product Guide, 2019. `https://www.xilinx.com/support/documentation/ip_documentation/prc/v1_3/pg193-partial-reconfiguration-controller.pdf`, Last accessed on 2020-05-13.

[94] AXI HWICAP v3.0: LogiCORE IP Product Guide, 2020. `https://www.xilinx.com/support/documentation/ip_documentation/axi_hwicap/v3_0/pg134-axi-hwicap.pdf`, Last accessed on 2020-05-13.

[95] Vivado Design Suite User Guide: Dynamic Function eXchange, 2020. `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug909-vivado-partial-reconfiguration.pdf`, Last accessed on 2020-05-13.

[96] S. Ngah, R. Abu Bakar, A. Embong, and S. Razali. Two-steps implementation of sigmoid function for artificial neural network in field programmable gate array. *ARPN Journal of Engineering and Applied Sciences*, 11(7):4882–4888, 2016.

[97] M. Zhang, S. Vassiliadis, and J. G. Delgado-Frias. Sigmoid generators for neural computing using piecewise approximations. *IEEE Transactions on Computers*, 45(9):1045–1049, 1996.

[98] M. Barni. *Document and Image compression.* CRC Press, 2006.

[99] 7 Series FPGAs Data Sheet: Overview, 2020. `https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf`, Last accessed on 2020-05-13.

[100] F. Regazzoni, C. Alippi, and I. Polian. Security: The dark side of approximate

computing. In *International Conference on Computer-Aided Design*, pages 44:1–44:6. ACM, 2018.

[101] J. Han. Introduction to approximate computing. In *VLSI Test Symposium*, pages 1–1, 2016.

[102] M. Masadeh, O. Hasan, and S. Tahar. Approximation-Conscious IC Testing. In *International Conference on Microelectronics*, pages 56 – 59, 2018.

Biography

Education

- **Concordia University**: Montreal, Quebec, Canada.

  Ph.D., Electrical & Computer Engineering (September 2016 - August 2020)

- **Delft University of Technology**: Delft, The Netherlands.

  M.Sc, Computer Engineering (September 2011 - August 2013)

- **The Arabic Academy for Banking & Financial Sciences**: Amman, Jordan.

  M.Sc, Management Information Systems (September 2006 - March 2009)

- **Yarmouk University**: Irbid, Jordan.

  B.Sc, Computer Engineering (September 1998 - June 2003)

Awards

- Concordia University International Tuition Award of Excellence, Canada (2017, 2018).

- Concordia University Conference and Exposition Award, Canada (2019, 2020).

- Yarmouk University Doctoral Scholarship, Jordan (2016 - 2020).
- Yarmouk University Master's Scholarship, Jordan (2011 - 2013).

Work History

- **Research Assistant**, Hardware Verification Group, Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada (2016 - 2020)
- **Teaching Assistant**, Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada (2018 - 2020)
- **Full Time Teaching Assistant**, Computer Engineering Department, Yarmouk University, Irbid, Jordan (2004 - 2011)
- **Full Time Faculty Member (Instructor)**, Computer Engineering Department, Yarmouk University, Irbid, Jordan (2013 - 2016)

Publications

- **Journal Papers**
  - **Bio-Jr1** M. Masadeh, O. Hasan, S. Tahar: "A Quality-Assured Approximate Hardware Accelerators Based on Dynamic Partial Reconfiguration for Efficient Machine Learning". ACM Journal on Emerging Technologies in Computing. *Submitted.*

- **Bio-Jr2** M. Masadeh, O. Hasan, S. Tahar: "Machine Learning-Based Self-Adaptive Design of Approximate Computing". IEEE Transaction on CAD of Integrated Circuits and Systems. *Submitted.*
- **Bio-Jr3** M. Masadeh, O. Hasan, S. Tahar: "Input-Conscious Approximate Multiply-Accumulate (MAC) Unit for Energy-Efficiency". IEEE Access 7: 147129-147142 (2019)
- **Bio-Jr4** M. Taouil, M.Masadeh, S. Hamdioui, E. J. Marinissen: "Post-Bond Interconnect Test and Diagnosis for 3-D Memory Stacked on Logic". IEEE Transaction on CAD of Integrated Circuits and Systems 34(11): 1860-1872 (2015)

- **Refereed Conference Papers**
  - **Bio-Cf1** M. Masadeh, A. Aoun, O. Hasan, S. Tahar: "Decision Tree-based Adaptive Approximate Accelerators for Enhanced Quality". International Systems Conference (SysCon) 2020: 1-5 (To Appear).
  - **Bio-Cf2** M. Masadeh, O. Hasan, S. Tahar: "Machine Learning-Based Self-Compensating Approximate Computing". International Systems Conference (SysCon) 2020: 1-6 (To Appear).
  - **Bio-Cf3** M. Masadeh, O. Hasan, S. Tahar: "Using Machine Learning for Quality Configurable Approximate Computing". Design, Automation Test in Europe (DATE) 2019: 1575-1578.
  - **Bio-Cf4** M. Masadeh, O. Hasan, S. Tahar: "Approximation-Conscious IC Testing". International Conference on Microelectronics (ICM) 2018: 56-59.

- **Bio-Cf5** M. Masadeh, O. Hasan, S. Tahar: "Comparative Study of Approximate Multipliers". Great Lakes Symposium on VLSI (GLSVLSI) 2018: 415-418.
- **Bio-Cf6** M. Taouil, M. Masadeh, S. Hamdioui, E. J. Marinissen: "Interconnect test for 3D stacked memory-on-logic". Design, Automation Test in Europe (DATE) 2014: 1-6.

- **Technical Reports**
  - **Bio-Tr1** M. Masadeh, O. Hasan, S. Tahar: Error Analysis of Approximate Array Multipliers. http://arxiv.org/abs/1803.06587 (2019)
  - **Bio-Tr2** M. Masadeh, O. Hasan, S. Tahar: Comparative Study of Approximate Multipliers. http://arxiv.org/abs/1908.01343 (2018)