**Energy-Quality Scalable Hardware and Software Solutions for Energy-Efficient Approximate Computing**

by

Setareh Behroozi

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2022

Date of final oral examination: 08/15/2022

The dissertation is approved by the following members of the Final Oral Committee:

Younghyun Kim, Assistant Professor, Electrical and Computer Engineering
Mikko Lipasti, Professor, Electrical and Computer Engineering
Joshua San Miguel, Assistant Professor, Electrical and Computer Engineering
Matt Sinclair, Assistant Professor, Computer Sciences

## ACKNOWLEDGMENTS

This thesis work is dedicated to my husband, Ehsan Ahmadi, and my family members, who have supported and encouraged me during graduate school and life challenges.

First and foremost, I am incredibly grateful to my supervisor, Professor Younghyun Kim, for his invaluable advice, continuous support, and patience during my graduate studies. I would also like to thank every member of my defense committee, Professor Lipasti, Professor San Miguel, and Professor Sinclair, who generously provided knowledge and expertise with their valuable feedback.

I would also like to thank my friends, lab mates, and colleagues in the WISEST research group for a cherished time spent together in the lab. My appreciation also goes to my family and friends for their encouragement and support throughout my studies.

CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## ABSTRACT

Energy efficiency is one of the most crucial constraints in designing embedded systems that dictate performance, lifetime, form factor, and cost in modern computing system design. Approximate computing has recently emerged as a promising solution to address the challenge of energy efficiency in designing embedded systems by exploiting the error resiliency of applications where the quality of results can be traded for energy efficiency. Previous attempts to design approximate hardware components mostly support approximation levels determined at design-time only. However, this research attempts to design approximate hardware components that their energy-quality trade-off can be adjusted at run-time. Besides, previous efforts to leverage the trade-off of energy-quality have largely focused on designing individual approximate components and leaving the larger opportunity in energy saving through full-system approximations. This dissertation introduces an efficient design methodology to achieve full-system energy-quality capability using proposed quality-energy salable components. These objectives are achieved by two tasks as follows: i) designing energy-quality scalable components, ii) designing a control system for energy-quality scalable components. These contributions facilitate the widespread adoption of embedded systems in energy-constrained environments.

# 1 INTRODUCTION

Embedded devices bring computing capabilities to diverse disciplines such as health care, industry, and education. However, adopting embedded and smart devices in various sectors and disciplines is impossible or very difficult with the huge costs at the moment. The lack of adoption of computing technology in many fields results in the loss of millions of dollars annually to the economy. For example, the main cost in managing the United States infrastructures is the existing monitoring systems in inaccessible or hardly accessible areas such as roads, interstate highways, bridges, and dams. In such places, the lack of energy-efficient monitoring systems causes high costs for maintenance and battery replacement of existing embedded systems or labor effort and time to check those areas. This thesis provides solutions to design energy-efficient embedded devices to address existing problems in incorporating computing technology in different disciplines and create new opportunities in deploying computing technology in new areas.

## 1.1 Motivation

Energy efficiency is the major challenge in designing embedded systems that limits performance, lifetime, form factor, and cost of the modern computing system design. However, the energy efficiency improvement driven by semiconductor technology scaling is coming to an end with the prediction of the end of Moore's law in the near future.

*Approximate computing* is an emerging paradigm that facilitates low-power computing by relaxing accuracy requirements for applications that do not always demand exact computation results Han et al. (2013). Certain applications are resilient to small computation errors due to the absence of a unique, golden outcome, the intrinsic noise or redundancy of the input,

or the self-healing nature of the algorithm Venkataramani et al. (2015a). Examples of error-resilient applications include signal processing and neural Signal processing deals with physical quantities measured by a sensor, such as images and voices, which are intrinsically noisy, thus signal processing algorithms are designed to tolerate or recover from errors in the inputs and produce acceptable outputs. Neural networks are also highly error-resilient due to their self-healing nature. It is known that the output quality of neural networks is not significantly degraded even if the underlying computations and memory are greatly approximated. Approximate computing aggressively exploits error resiliency by producing "just good enough" results rather than exact results at a much lower effort in terms of power consumption, latency, logic area, etc. Adopting this broad approach, approximate hardware units have been proposed to produce approximate results that are not exact but good enough for the target application.

Over the past years, there has been proposed a large number of approximate computing methods to leverage the energy-quality trade-off for various components of embedded systems in different levels of abstraction, including i) device, ii) circuit, iii) architecture, and iv) algorithm or application. However, previous efforts mostly support approximation levels determined at design time.

*Dynamic quality configuration* is a requirement when using approximate computing because the notion of a good-enough result is application-specific and contextual. Accuracy requirement varies between applications and even within one application. One needs to configure the approximate designs with design time knobs to meet the maximum accuracy requirements. Once the design time configuration is fixed, there is no more opportunity for energy saving. Therefore, the underlying subsystems should support variant accuracy requirements through flexible energy-quality scaling and provide a control knob for run-time adjustment to

maximize energy saving.

This research proposal attempts to design approximate techniques that their energy-quality trade-off can be adjusted at runtime. Furthermore, existing approximate methods mainly focus on the energy-quality tradeoff of individual components, and a closed loop to control such an approximated component is mostly missing. To address the aforementioned demand, this research proposal attempts to develop a framework that generates a controller to estimate the system-level quality at a low overhead and automatically adjust the control knob to meet the energy-quality requirements. The other caveat of existing works is that the implication and design complexity of applying full-system approximation to different components of the system at the same time is not well studied. To close this gap, we will study how energy-quality scaling in one component of the system affects that of other components and the final system-level quality. Then we will develop a design methodology that can be used by system designers to achieve optimal approximate architecture and control systems for different system components.

## 1.2 Objectives and contributions

In this dissertation, I propose a series of solutions toward the goal of energy-efficient embedded devices. The research goals of this thesis will be carried out through the following research thrusts, as illustrated in Figure 1.1. The focus of this dissertation is shown in highlighted parts of the Figure 1.1.

- **Thrust 1: Designing energy-quality scalable components**
  We identify energy-quality scalability opportunities in computing and non-computing subsystems and propose energy-efficient designs with runtime reconfigurability capabilities. In other words, we

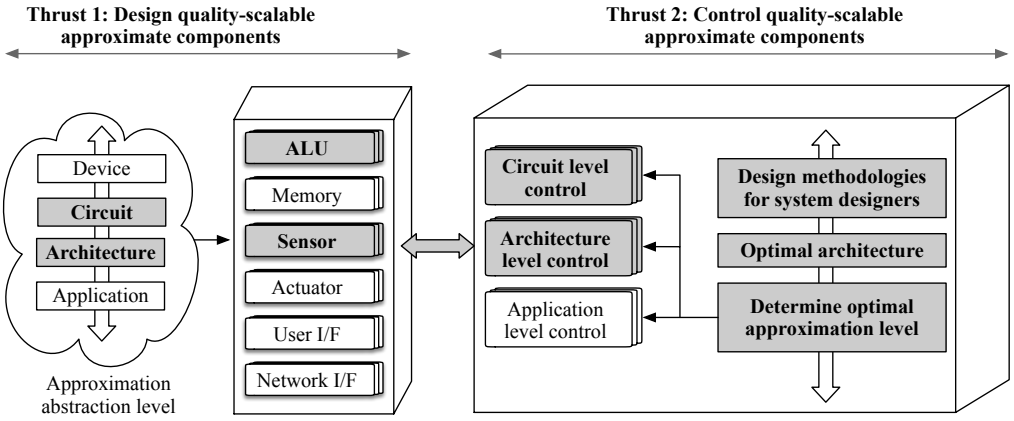



Figure 1.1: Research overview.

propose components with various accuracy levels through approximation knobs to set the accuracy level at runtime.

- **Thrust 2: Designing control systems for energy-quality scalable components**
  Based on the quality-scalable subsystems developed in thrust 1, we develop a framework that generates a controller to estimate the system-level quality at a low overhead and automatically adjust the control knob to meet the energy-quality requirements. Furthermore, we proposed methodologies to find optimal application-specific quality set points for approximation knobs of proposed quality-scalable components to exploit various accuracy requirements to maximize energy saving as well as to maintain the output quality in the presence of dynamic input variations.

**Projects team contributions:**

- SAADI: quality-configurable approximate divider Behroozi et al. (2019):

I am the main author of this project. I designed, simulated, and evaluated the approximate divider. Jingjie Li prepared the application level evaluation set-up and helped with experiments and paper writing. Jackson Melchert helped with the implementation of related works.

- SAADI-(EC): quality configurable approximate divider with error compensation Melchert et al. (2019)
  Jackson Melchert is the main author of this paper. I helped with some of the experiences as well as paper writing.

- AxSerBus: quality configurable approximate serial bus interface Kim et al. (2017):
  Professor Younghyun Kim is the main author of this paper. I helped with experimental set-up and evaluations.

- Application level quality control methodology for AxSerBus Behroozi et al. (2018):
  I am the main author of this project. I designed, simulated, and evaluated the proposed method.

- Scheduling of energy-quality scalable hardware for energy efficient approximate computing Behroozi et al. (2021):
  I am the main author of this project. I designed, simulated, and evaluated the proposed method. Yao Yao helped with the implementation of the approximate multiplier and some of the evaluation tasks. Professor Hoeseok Yang helped with designing the optimizer.

- Scheduling of quality-scalable neural network computation:
  This is an ongoing project, and I am the main author.

The rest of this dissertation is outlined as the following. Chapter 2 presents the background information and previous efforts of the research.

Next, Chapter 3 details the first thrust of this thesis on designing energy-quality scalable components. In this chapter, I describe a quality configurable approximate divider unit called SAADI Behroozi et al. (2019) and its extended version with error compensation methodologies called SAADI-(EC) Melchert et al. (2019). Next, an approximated serial bus encoding scheme called AxSerBus Kim et al. (2017) is presented. In Chapter 4, I target the second trust of this thesis by introducing techniques to control energy-quality scalable components. A control system to adjust the AxSerBus approximation knob Behroozi et al. (2018) is presented. Followed by a scheduling technique to control iterative computing units Behroozi et al. (2021). I discuss the future works and conclude the dissertation in Chapter 5 and Chapter 6, respectively.

# 2 BACKGROUND

This Chapter provides background information and a comprehensive discussion about previous works to improve the energy efficiency of embedded systems through approximate computing.

## 2.1 Approximate computing

With the end of Moore's law, integrated circuit designers are seeking new solutions to overcome the effect of the diminishing technology scaling opportunity and its subsequent benefits. Venkataramani et al. (2015b) At the same time, there has been a shift from applications with the requirement for precise numerical answers to good enough answers, particularly in embedded devices. These applications demand an acceptable user experience, which most of the time is achievable by good enough calculations Chippa et al. (2013b). This is mainly due to the characteristics of emerging applications. In signal processing, physical signal measurements are inherently noisy due to physical noise or analog-to-digital conversion. Therefore, they tolerate marginal noise. In data mining and search, no single golden answer exists, and multiple good answers are all acceptable. Machine learning-based solutions are usually trained with massive data rather than individual data points, and inner layers heal minor errors. Most of the small errors are suppressed and mostly do not propagate.

*Apparoximate computing* is a new computing paradigm where the correctness of the result is not guaranteed, and efforts are towards good enough answers. Therefore, significant benefits in energy efficiency can be achieved by relaxing the correctness requirement. For the past years, researchers have explored the potential of approximate computing for different levels of abstraction, such as device, circuit, architecture, and application. Previous efforts to achieve energy efficiency through approx-

imate computing targeted different subsystems such as sensors, actuators, arithmetic logic units (ALUs), user interfaces, network interfaces, etc. Examples of approximate ALUs are discussed in Gupta et al. (2011); Jayakumar et al. (2014); Park et al. (2000). Stanley-Marbell and Rinard (2015, 2020) present approximate sensor designs. Approximate memory designs such as Shoushtari et al. (2015); Liu et al. (2011); Sampson et al. (2014) attempt to save energy at the cost of data corruption in memory.

## 2.2 Approximate arithmetic units

Approximate arithmetic units perform near-accurate arithmetic calculations at a much lower energy cost using smaller core arithmetic units, with a superior energy-quality trade-off than that of simple truncation. The energy-accuracy trade-offs of static approximate arithmetic units are mainly determined by the design parameters of the core arithmetic units, such as bit width (BW) Gupta et al. (2011); Yang et al. (2013); Mrazek et al. (2017); Zendegani et al. (2016); Vahdat et al. (2017); Jiang et al. (2018a); Saadat et al. (2019); Lin and Lin (2013); Liu et al. (2014); Bhardwaj et al. (2014); Hashemi et al. (2015); Momeni et al. (2015); Nilsson et al. (2014); Mitchell (1962); Sullivan and Swartzlander (2013). On the other hand, some approximate arithmetic units support dynamic scaling by selectively applying error correction or performing progressive Taylor approximation Ye et al. (2013); Kahng and Kang (2012); Wu et al. (2019); Weirich et al. (2018).

## 2.3 Approximate divider units

Approximate dividers have been studied relatively more recently than other arithmetic units such as adders Gupta et al. (2011) and multipliers Hashemi et al. (2015). An approximate divider design presented

in Wu et al. (2015) calculates the quotient using a 2-D LUT indexed by the operands. The accuracy is determined by the granularity of the LUT and hence is fixed at design time. Another design presented in Hashemi et al. (2016) normalizes the operands to remove leading zeros and takes only a small number of most relevant bits. The truncated dividend is divided by the truncated divisor using a narrow-width precise divider. A recent design, called AAXD Jiang et al. (2018a) compares the magnitudes of the dividend and the divisor to address the overflow problems of Hashemi et al. (2016). The accuracy is preset by the divider width in both Hashemi et al. (2016) and Jiang et al. (2018a), which is not dynamically adjustable. SEERAD Zendegani et al. (2016) and TruncApp Vahdat et al. (2017) are multiplicative division, where the reciprocal of the divisor is obtained first and multiplied by the dividend to perform division. In SEERAD, the reciprocal is approximated using a table indexed by the upper bits of the divisor, whereas in TruncApp, the reciprocal is obtained by a simple bit manipulation such as inversion, truncation, and concatenation. In both designs, the accuracy is determined at design time and not configurable at runtime.

## 2.4 Approximate sensing

Low-power bus encoding mainly aims at reducing transition count (TC) on the bus and, in turn, reducing dynamic energy consumed for switching the highly capacitive off-chip interconnects. A typical approach to reducing TC is to utilize the temporal or spatial locality of data. Gray coding and T0 coding Benini et al. (1997) are applicable when most of the data are consecutive, such as in-memory address buses. However, these coding schemes are aimed at reducing inter-word (word-to-word) TC on parallel buses. Modern low-power sensors predominantly employ a serial bus, where intra-word (bit-to-bit) TC is more dominant than inter-word TC,

hence these parallel bus encoding schemes are not applicable.

Serial bus encoding has been studied relatively recently Bocca et al. (2004); Lee et al. (2004); Ghosh et al. (2014). The focus of these serial bus encoding schemes is on reducing intra-word TC by utilizing data locality. For example, differential bar encoding (DBE) converts a small differential into a 1-TC (single TC) codeword, where the number of consecutive bit 1's signifies the differential Bocca et al. (2004). However, the energy-saving of these encoding schemes is rather limited in sensing applications because they are designed for lossless data transfer and unnecessarily guarantee bit-by-bit accuracy between the source data and the received data.

Value-deviation-bounded serial (VDBS) encoding is an approximate encoding that maps each binary code to another binary code with a fewer TC, within a limited value deviation Stanley-Marbell and Rinard (2016). It is based on a fixed lookup table of codewords, hence runtime quality configuration is not supported. To address this, they introduced Rake, a runtime encoding algorithm with quality-configurability, but the algorithm is relatively complex and no hardware implementation has been proposed. Also, this encoding scheme does not take advantage of the locality of sensory data.

On the other hand, approximate differential encoding (ADE) Pagliari et al. (2016a) and Serial T0 Pagliari et al. (2016b) are differential encoding schemes that exploit locality. In Serial T0, a special code with no intra-word TC represents a negligibly small differential from the previous value, and exact values are sent out only if the differential is significant. However, the tradeoff between energy and quality is not fully exploited due to the single-threshold approach wherein the differentials are either zeroed out (at no energy cost) or accurately transferred (at a high energy cost).

## 2.5 Control quality-scalable approximate components

Although many approximate subsystems have been proposed in previous efforts to achieve energy efficiency, there are not many quality-scalable components, let alone their control systems. Some recent work Li et al. (2015); Hanif et al. (2020) proposed methods for optimizing the selection of hardware with different accuracy. A gradient descent-based method proposed Raha and Raghunathan (2017) to perform joint approximations across different subsystems. A more recent work Kemp et al. (2021) optimizes the latency of iterative hardware for a given accuracy constraint. Spantidi et al. (2021) attempts to set the approximation level of their proposed reconfigurable multiplier in a neural network application.

# 3 DESIGNING ENERGY-QUALITY SCALABLE COMPONENTS

The first step towards full-system approximation is designing energy-quality scalable components within a system and realizing their potential for quality-energy trade-off. The goals of this thrust of the project will be achieved through designing and developing quality-scalable components. While there are existing quality-scalable techniques for a variety of components and in different levels of abstractions, such as approximate DRAM with the refresh as approximation knob, on the other hand, approximate arithmetic unites, and ALU designs featuring energy-quality scalability are not studied well. For that, in this thrust, we present an energy-quality scalable divider unit. Another sub-system that has seen relatively less attention from researchers is energy-efficient sensing. In this thrust, we focus on addressing the energy challenge in sensing devices by proposing a run-time quality configurable serial bus architecture for off-chip sensory data transfer.

## 3.1 SAADI-(EC): quality-configurable approximate divider

Energy-efficient arithmetic units are key building blocks for realizing low-power signal processing. Researchers have sought to develop fast yet energy-efficient arithmetic algorithms and circuit implementations, and various designs have been proposed to improve the performance and reduce power consumption. However, traditional arithmetic units are designed for universal computing without the awareness of the application's accuracy requirement and produce exact results at all times, often consuming unnecessarily large energy.

Many approximate adders Gupta et al. (2011) and approximate mul-

tipliers Liu et al. (2014); Hashemi et al. (2015) have been proved useful for saving power or improving performance at the cost of minor accuracy loss.

Division is an arithmetic operation crucial in signal processing. A hardware divider is a costly module in terms of latency and energy consumption due to the high complexity of division algorithms. For example, the integer divider instruction (IDIV) of the AMD 12h family has a latency of 9–17 cycles for a 16-bit division and 9–25 cycles for a 32-bit division, while the integer multiplier instruction (IMUL) for the same widths takes only three cycles Advanced Micro Devices (AMD) (2011). As another example, a single-precision floating-point divider requires 1.35x to 3x more hardware resources and is 27% slower than the same precision multiplier on FPGAs Altera (2016). Despite the lack of energy-efficient and fast hardware dividers, the energy-efficient design of dividers has received less attention mainly due to relatively low utilization than other arithmetic operations. However, as distributed signal processing becomes more and more pervasive, the division is playing an increasingly important role in low-power systems, and the need for an energy-efficient divider is ever increasing. As a result, researchers have proposed several approximate divider designs based on rounding Zendegani et al. (2016), truncation Hashemi et al. (2016); Vahdat et al. (2017), or table look-up Wu et al. (2015).

Unfortunately, these existing approximate dividers support only a single level of accuracy and do not support *dynamic quality configuration*. In these designs, the accuracy level is determined by a hardware design parameter fixed at design time, such as the size of the look-up table (LUT) or the number of truncated bits. Once the design parameter has been fixed, there is no control knob for adjusting the accuracy at runtime, and there is no power/performance benefit from sacrificing accuracy. However, the accuracy requirement for arithmetic operations is not constant but

variable because the impact of approximation to the final quality at the application level is heavily input-dependent Raha and Raghunathan (2017). Moreover, the application-level quality requirement may change over time. Therefore, approximate computing hardware needs to provide a control knob to the application for the feedback control of the application-level quality Chippa et al. (2013a). Having no dynamic quality-control knob, the existing approximation dividers need to be designed to meet the most strict accuracy requirement predicted at design time, and, as a result, the potential of energy savings and performance improvement is not fully exploited when the actual accuracy requirement is less strict.

In this research project, we propose a novel approximate divider design called SAADI-EC (Scalable Accuracy Approximate Divider with Error Compensation), which is capable of dynamically trading accuracy for latency. SAADI-EC finds the approximate reciprocal of the divisor and multiplies it by the dividend to obtain the division result. The iterative reciprocal approximation process, where the accuracy gradually increases as more iterations are performed, enables the dynamic energy-quality trade-off. The number of iterations can be dynamically adjusted by the application for energy-quality scaling—the application can either obtain high-accuracy division results at the cost of a higher latency and energy consumption or reduce latency and improve energy efficiency by sacrificing accuracy. SAADI-EC has novel error compensation logic to greatly improve the accuracy of the approximate division. To compensate for the approximation error of the finite-order truncation of the Taylor series, the approximate quotient is multiplied by a constant in a LUT indexed by the number of iterations.

In summary, the contributions of this project are as follows:

- We introduce an approximate divider named SAADI-EC which features dynamic energy-quality scalability. SAADI-EC performs division by multiplying the dividend by the approximate reciprocal of

the divisor. The accuracy of division can be dynamically controlled by adjusting the number of iterations for reciprocal approximation.

- We introduce an efficient method that compensates for the negatively biased errors introduced in the reciprocal approximation by finite-order truncations of Taylor series.
- We present a low-power hardware architecture of SAADI-EC and analyze the trade-off between accuracy and energy efficiency. We evaluate the accuracy and energy consumption of SAADI-EC for different design-time parameters as well as runtime parameters.
- We demonstrate the application of SAADI-EC to i) k-means clustering, ii) JPEG image compression, and iii) an image division algorithm for video sequences with dynamic accuracy requirements. We illustrate the superiority of runtime accuracy scaling by comparing the area, energy, and delay of the resulting system with SAADI-EC compared to other approximate dividers.

**Proposed approximate division**

The proposed design, SAADI-EC, falls into a category of *multiplicative division* algorithms, where the reciprocal of the divisor is obtained first and then multiplied by the dividend. We first describe the basics of multiplicative division Oberman et al. (1997) before we describe our approach.

Let $A$ and $B$ be the dividend and the divisor, respectively. They can be represented in a normalized form as

$$A = 2^{e_a} \times a \qquad \text{and} \qquad B = 2^{e_b} \times b, \tag{3.1}$$

where $0.5 \leqslant a < 1$ and $0.5 \leqslant b < 1$, and $e_a$ and $e_b$ represent the position of the leading 1 bit if $A$ and $B$ were represented as binary numbers, respectively. During normalization, $a$ and $b$ are truncated to $n$ bits, where

$n$ is the width of multiplier, which is determined at design time. Then the quotient, Q, of division $A/B$ is

$$Q = \frac{A}{B} = 2^{e_a - e_b} \times \frac{a}{b} = 2^{e_a - e_b} \times a \times R(b), \tag{3.2}$$

where $R(b)$ is the reciprocal of $b$, that is, $R(b) = 1/b$. Instead of computing $a/b$ directly, the multiplicative division first finds $R(b)$, and then multiplies it by $a$, followed by a shift operation for denormalization. The key to fast division is finding the reciprocal as quickly as possible. However, finding an accurate reciprocal is almost as complex as the division itself. Therefore, it is typical that an approximate reciprocal is obtained initially, and then a functional iteration algorithm, such as the Newton-Raphson method, is used to converge to the accurate reciprocal.

When a moderate-accuracy quotient is good enough, using such a time-consuming functional iteration algorithm may be a waste of time and energy. Approximate multiplicative dividers discussed in Section **??** obtain an approximation of the reciprocal instead of the exact reciprocal to save energy. However, their fixed-accuracy approximation may not always produce results with "just good enough" accuracy at minimal power consumption. We address this challenge by an accuracy-adaptive reciprocal approximation based on incremental Taylor series expansion. Taylor series expansion is a widely used approach to find an approximate reciprocal. More specifically, the Taylor series expansion of $R(b) = 1/b$ is

$$R(b) = \frac{1}{1+x} = \sum_{i=0}^{\infty} (-x)^i = 1 - x + x^2 - x^3 + x^4 - \cdots, \tag{3.3}$$

where $x = b - 1$. This series always converges since $-0.5 \leqslant x < 0$ for normalized $b$ such that $0.5 \leqslant b < 1$. Note that the contribution of each term to accuracy improvement exponentially decreases as the order increases. Typically, a good reciprocal approximation can be obtained by adding up

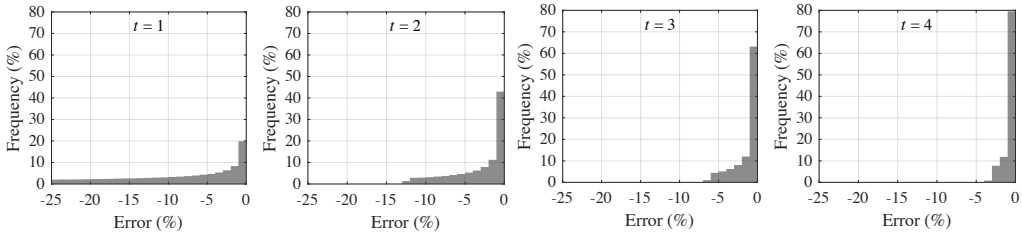



Figure 3.1: Error distribution of the approximated quotient with $t = [1, 2, 3, 4]$.

the first few low-order terms, and if higher accuracy is required, more high-order terms can be added to incrementally improve the accuracy. Let $\tilde{R}_t(b)$ denote the t-th order approximation of $R(b)$, such that

$$\tilde{R}_t(b) = \sum_{i=0}^{t} |x|^i = 1 + |x| + |x|^2 + |x|^3 + |x|^4 + \cdots + |x|^t, \tag{3.4}$$

which takes $t - 1$ multiplications. The result is a trade-off between higher accuracy and longer latency. This trade-off can be exploited by the application depending on its accuracy requirements and latency constraints. Finally, the t-th order approximate quotient $\tilde{Q}_t$ is then obtained as

$$\tilde{Q}_t = 2^{e_a - e_b} \times a \times \tilde{R}_t(b). \tag{3.5}$$

**Error compensation**

To improve the approximation accuracy of the base implementation we introduce novel logic for error compensation in SAADI-EC. As shown in (3.4), all terms in the reciprocal approximation are positive, meaning the resulting approximation error is always negative for a finite t. Therefore, this error can be mitigated by scaling the quotient of SAADI-EC by a

positive compensation factor, $c$. The new scaled quotient $\tilde{Q}_t^C$ is given by

$$\tilde{Q}_t^C = \tilde{Q}_t \times c. \tag{3.6}$$

Because $0 < |x| \leqslant 0.5$, and $t$ is always positive, the minimum possible error for $\tilde{R}_t(b)$ is 0% when $|x|$ is very small and $t$ is very large. Therefore, the minimum value of $c$ is 1, equivalent to no error compensation. The maximum possible error for $\tilde{R}_t(b)$ is 25%, when $|x| = 0.5$ and $t = 1$. Therefore, the maximum value of $c$ is 1.333.

Figure 3.1 shows how the approximation error of $\tilde{Q}_t$ decreases as $t$ increases. In addition, as $t$ increases the error distribution gets more closely centered around 0%. For each value of $t$, the error distribution is different, so a unique scaling factor should be determined at each $t$ to minimize the error. We discuss how this error compensation is implemented without using a costly floating-point multiplier in the next subsection.

**Hardware architecture**

Figure 3.2 shows the hardware architecture and a simplified diagram of a finite state machine (FSM) to control the hardware. Figure 3.3 shows the computation flow of SAADI-EC. First, the dividend $A$ and the divisor $B$ are normalized to $a$ and $b$ by the normalizer blocks, respectively, and truncated to $n$ bits. Normalization is done by finding the position of the leading 1 and left-shifting to make the most significant bit (MSB) 1. If $B$ is a power of two, the division is done by a simple shift operation. We do not show this in the diagram for simplicity. The 2's complement block converts $b$ to $|x|$. A single-cycle multiplier first computes $|x| \cdot |x| = |x|^2$, and the output is fed back to the multiplier again to subsequently obtain $|x|^3, |x|^4, \ldots |x|^t$ over $t-1$ cycles. Note that the output of the $n$-bit multiplier is truncated from $2n$ bits to $n$ bits because it is going to be fed back to itself. The results are added up by the accumulator block in the same cycle to

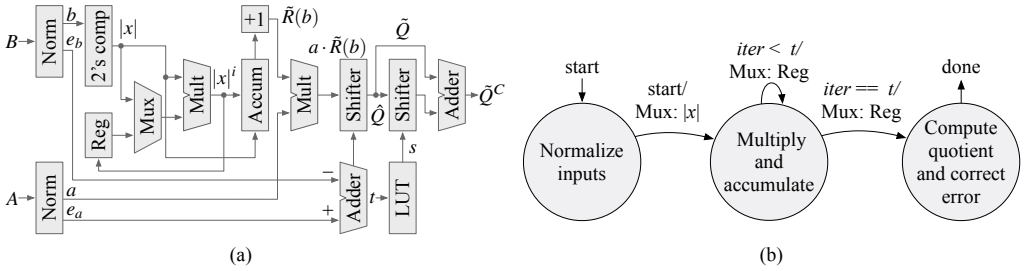



Figure 3.2: (a) Hardware architecture, and (b) control FSM of SAADI-EC.

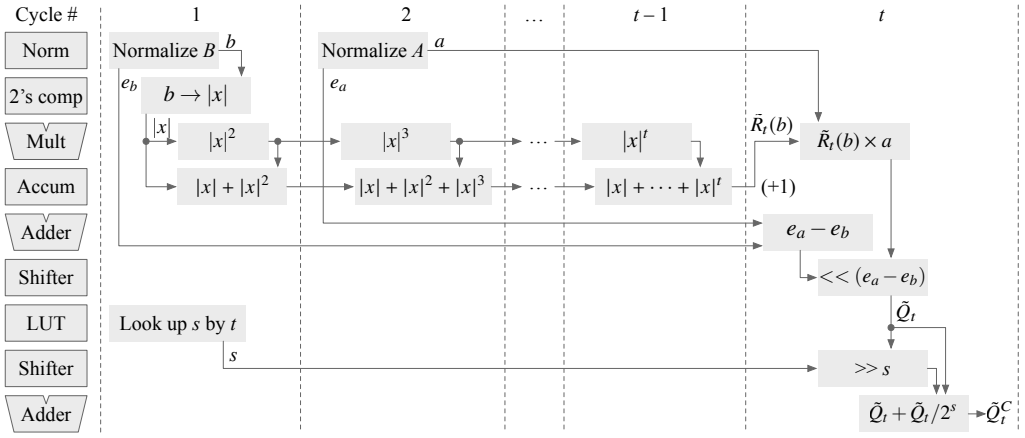



Figure 3.3: Hardware block usage and data flow. ($t \geqslant 2$)

calculate $|x| + |x|^2 + ... + |x|^t$. The output of the accumulator is $\tilde{R}_t(b) - 1$, thus it is added to 1 before being multiplied by $a$. In the t-th cycle, $\tilde{R}_t(b)$ is multiplied by $a$, and the output is denormalized using a barrel shifter to obtain the approximate quotient $\tilde{Q}$.

Finally, the error compensation discussed above is performed. We simplify the error compensation logic by replacing the costly multiplication in (3.6) by a simple table look-up and an addition. For a given $n$, which is a design parameter, the appropriate scaling factor $c$ is calculated for $1 \leqslant t \leqslant n - 1$ by finding the scaling factor that minimized the mean absolute error (MAE) of $\tilde{Q}_t$ over a large sample of inputs. Error compensation is

then done by adding a bit-shifted value of $\tilde{Q}_t$ to itself, that is

$$\tilde{Q}_t^C = \tilde{Q}_t \times c \approx \tilde{Q}_t + \tilde{Q}_t/2^s, \tag{3.7}$$

where the integer bit-shift amount $s$ is given by

$$s = -\text{Round}\left(\log_2(c-1)\right). \tag{3.8}$$

This simple error compensation hardware is composed of only a shifter, an adder, and a LUT holding the $n-1$ values of $s$ corresponding to all possible values of $t$ at a given $n$. The values of $c$ and $s$ for all values of $t$ at $n = [4, 8, 12, 16]$ are provided in our journal paper Melchert et al. (2019).

The width of the all the normalized numbers, such as $a$, $b$, $|x|$, $|x|^i$, etc., is $n$ bits. The accumulator needs to be wider than $|x|^i$ since the accumulation can generate a carryout. The maximum value in the accumulator is 1, since the maximum value of $\tilde{R}_t(b)$ is 2 when $|x| = 0.5$. Therefore, the width of the accumulator needs to be only one-bit wider than the multiplier, thus $n+1$ bits. After adding 1 to the accumulator output, $\tilde{R}_t(b)$ becomes $n+2$ bits, thus it is right-shifted by two bits to fit into the $n$-bit multiplier.

The reciprocal approximation requires up to $t-1$ multiplications, and $a \times \tilde{R}_t(b)$ requires another multiplication. Therefore, the maximum latency of one division operation is $t$ multiplications. Iterations for reciprocal approximation can be terminated earlier to reduce latency at the cost of accuracy loss, and we discuss this trade-off in the following.

To save area, the normalization of $A$ and $B$ can be done by a shared normalizer since the normalization of $A$ is required only after $\tilde{R}_t(b)$ is obtained. Similarly, the multiplications for reciprocal approximation and the multiplication of $a \times \tilde{R}_t(b)$ can be done by a single shared multiplier. Therefore, the major hardware components of SAADI-EC include one $n$-bit normalizer, one $n$-bit multiplier, one $(n+1)$-bit accumulator, one $\log_2 n$-bit adder, two $n$-bit barrel shifters, one LUT with $n-1$ entries, one

$2n$-bit adder, and a few multiplexers and registers. The usage of the shared hardware blocks is shown in Figure 3.3.

The proposed divider can be extended to compute signed division with the addition of minimal extra hardware. In the normalization blocks, the sign bit of the dividend and divisor need to be stripped off. If the sign bit was a 1 (i.e., negative), then the sign of the operand needs to be inverted using a two's complement block. At the end of the division, the sign bit is added back to the quotient and the quotient is inverted if necessary.

**Accuracy and latency trade-off**

Unlike prior approximate dividers whose division accuracy is fixed at design time, SAADI-EC provides a flexible energy-quality trade-off that can be selected by the application at runtime. When high-accuracy division is required, the application can increase $t$, and when low-accuracy division is sufficient, it can decrease $t$ to save energy. While SAADI-EC enables division operations to be performed at the optimal accuracy level, finding the optimal accuracy is highly application-dependent, which is out of the scope of this paper. In this section, we analyze how $t$ affects the error rate and energy consumption.

The loss of accuracy in SAADI-EC is caused by the following factors: (1) $\epsilon_1$: The input operands $A$ and $B$ are truncated to $n$ bits during normalization; (2) $\epsilon_2$: The approximate reciprocal $\tilde{R}_t(b)$ is the sum of a limited number of $|x|^i$ terms; (3) $\epsilon_3$: Each $|x|^i$ term is computed using a statically truncated multiplier that truncates $2n$-bit results to $n$ bits; (4) $\epsilon_4$: The approximate reciprocal $\tilde{R}_t(b)$ is truncated from $n+2$ bits to $n$ bits before multiplied by $a$, and the result is truncated to $n$ bits.

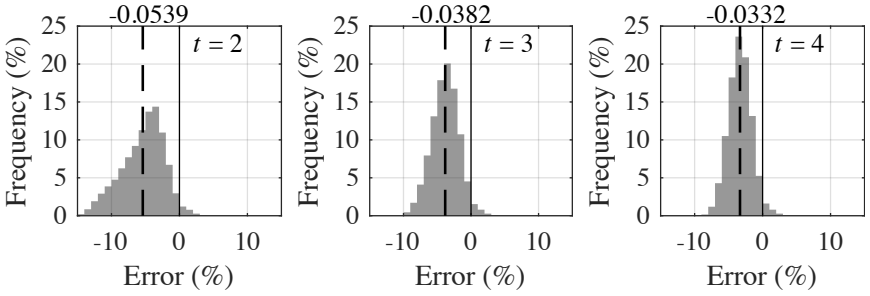
The sign of $\epsilon_1$ is input-dependent. Truncating $A$ contributes to negative factors of $\epsilon_1$ since the truncation results in a smaller dividend, while truncating $B$ has the reverse effect, as a smaller divisor produces a larger quotient. The error due to a finite-order approximation ($\epsilon_2$) is the runtime-

adjustable error factor that we exploit for accuracy-latency trade-off such that

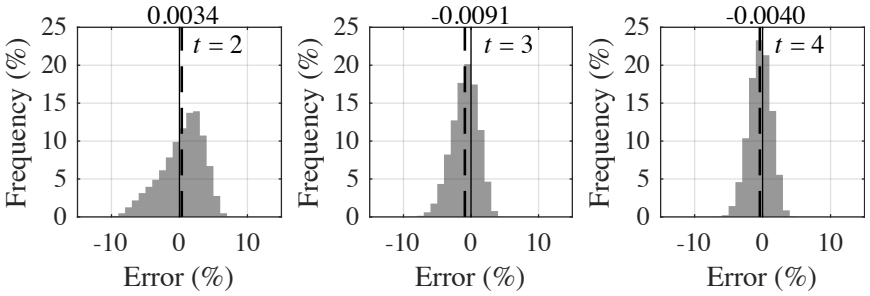
$$\epsilon_2 = \tilde{R}_t(b) - R(b) = -\sum_{i=t+1}^{\infty} |x|^i = -|x|^{t+1} - |x|^{t+2} - \cdots . \tag{3.9}$$

Note that $\epsilon_2$ is always negative, and the magnitude decreases as $t$ increases. Due to the accuracy loss in the multiplier ($\epsilon_3$), $|x|^i$ eventually becomes zero even if $x$ is non-zero. Note that multiplying $|x| \leqslant 0.5$ retires at least one bit per multiplication cycle, thus $|x|^i$ becomes zero within $n$ cycles. Therefore, only up to $|x|^{n-1}$, which is obtained at $(n-2)$-th cycle, contributes to $\tilde{R}_t(b)$, hence the effective range of $t$ for accuracy-latency trade-off is $1 \leqslant t \leqslant n-1$, considering one additional cycle at the end for $a \times \tilde{R}_t(b)$. Increasing $t$ beyond $n-1$ only increases latency but does not improve accuracy ($n = 4$ is an exception, where the accuracy saturates at $t = 2$ due to narrow multiplier width). On the other hand, decreasing $t$ below $n-1$ may terminate the reciprocal approximation for slow-converging $|x|^i$ and increases the average error rate. The error induced by the truncation in the multiplier and the accumulator ($\epsilon_3$ and $\epsilon_4$) is always negative since it results in an under-approximated reciprocal and thus an under-approximated quotient.

The proposed error compensation method mitigates the error introduced by $\epsilon_2$, $\epsilon_3$, and $\epsilon_4$. For a given $n$, a unique scaling factor $c$ is determined for each $t$ and is chosen to minimize the MAE over all inputs. Figure 3.4 shows an example of the effect of error compensation on the error distribution of SAADI-EC. In this example, $n = 6$ and $t = [2, 3, 4]$. Without error compensation, Figure 3.4 (a) shows that resulting error measurements are negatively biased. The mean error becomes closer to 0 as $t$ increases, but the majority of error measurements are negative. Figure 3.4 (b) shows the result of our error compensation according to (3.7). The error distributions have been shifted so that their mean is much closer to 0.




(a) Before error compensation




(b) After error compensation

Figure 3.4: Error distribution of approximate quotient for varying t with and without error compensation. The dotted line represents the mean error.

**Evaluation and discussion**

We evaluate the accuracy and latency of SAADI-EC implemented in Verilog HDL and present the accuracy-latency trade-off for various design-time and runtime parameters. SAADI-EC is compared with other approximate dividers: SEERAD Zendegani et al. (2016), TruncApp Vahdat et al. (2017), AAXD Jiang et al. (2018b), and PLApp Vaeztourshizi et al. (2018). We also implemented an equivalent Matlab model of SAADI-EC for application-level quality evaluation. (i) Color quantization using k-means clustering and (ii) JPEG compression are used as image processing applications, and the results are compared with those of an exact divider. (iii) Image

division for change detection is then used as an application that has a dynamic accuracy requirement, and the results of this experiment are compared against other proposed approximate dividers. In this report we only present the k-means clustering results, our journal paper Melchert et al. (2019) includes complete evaluations on this research project.

Table 3.1 compares the MAEs of SAADI-EC, SAADI-EC-P (pipelined version of SAADI-EC), SAADI (without error compensation), TruncApp, SEERAD, AAXD, and PLApp with different accuracy levels. The best MAE of TruncApp is 4.26% when the accuracy level is 4, and increasing it beyond 4 does not improve the accuracy due to its linear reciprocal approximation. The MAE of SEERAD(3) and SEERAD(4) is 4.41% and 2.12%, respectively. The MAE of AAXD ranges between 6.13% and 0.18% for accuracy level 6 to 16. While its accuracy is better than TruncApp and SEERAD, its delay is significantly longer than the other multiplicative dividers because it uses a divider as the base arithmetic unit. PLApp has two configuration parameters, the first is specifying the number of sub-intervals used for the reciprocal approximation, while the second is for the rounding width. The MAE of this divider ranges from 2.82% to 0.18%. This design achieves a comparable MAE to AAXD at a much lower delay.

The MAE, delay, and energy of SAADI-EC and SAADI are presented by upper bound and lower bound since they are a function of $t$. Although the previous dividers exhibit lower MAEs for some low accuracy levels, SAADI-EC outperforms others for higher accuracy levels, and, moreover, it is scalable.

Figure 3.5 better illustrates the comparison between the approximate dividers. It shows the trade-offs between the energy-delay product (EDP) and accuracy. Each curve represents the EDP-accuracy trade-offs of SAADI-EC for each $n$, and each dot on the curves represents the EDP-accuracy pair for different $t$. One can select $n$ based on the range of accuracy requirement

Table 3.1: Comparison with previously proposed approximate dividers and an exact divider.

| Divider | | Cyc. | MAE (%) | Area ($m^2$) | Total delay (ns) | Power (mW) | Energy (nJ) |
|---|---|---|---|---|---|---|---|
| SAADI-EC(4) | Max | 2 | 4.86 | 1,973 | 2.12 | 0.36 | 0.77 |
| | Min | 1 | 7.35 | | 1.06 | | 0.39 |
| SAADI-EC(8) | Max | 7 | 0.37 | 3,035 | 8.54 | 0.67 | 5.72 |
| | Min | 1 | 5.77 | | 1.22 | | 0.82 |
| SAADI-EC(12) | Max | 11 | 0.03 | 4,485 | 15.07 | 1.28 | 19.27 |
| | Min | 1 | 6.01 | | 1.37 | | 1.75 |
| SAADI-EC(16) | Max | 15 | 0.006 | 6,105 | 23.55 | 2.11 | 49.71 |
| | Min | 1 | 6.03 | | 1.57 | | 3.31 |
| SAADI(4) | Max | 2 | 11.32 | 1,199 | 2.14 | 0.31 | 0.66 |
| | Min | 1 | 14.63 | | 1.07 | | 0.33 |
| SAADI(8) | Max | 7 | 0.99 | 1,963 | 7.91 | 0.59 | 4.67 |
| | Min | 1 | 7.50 | | 1.13 | | 0.66 |
| SAADI(12) | Max | 11 | 0.07 | 3,068 | 15.73 | 1.09 | 17.22 |
| | Min | 1 | 6.98 | | 1.43 | | 1.56 |
| SAADI(16) | Max | 15 | 0.006 | 4,872 | 24.00 | 1.94 | 46.76 |
| | Min | 1 | 6.94 | | 1.60 | | 3.11 |
| SEERAD(3) | | 1 | 4.41 | 2,006 | 1.85 | 1.49 | 2.76 |
| SEERAD(4) | | 1 | 2.12 | 3,481 | 2.45 | 2.99 | 7.33 |
| TruncApp(3) | | 1 | 6.37 | 1,400 | 1.35 | 0.52 | 0.70 |
| TruncApp(4) | | 1 | 4.26 | 1,628 | 1.60 | 0.59 | 0.94 |
| AAXD(6) | | 1 | 6.13 | 985 | 2.30 | 0.50 | 1.15 |
| AAXD(8) | | 1 | 2.99 | 1,386 | 2.95 | 0.85 | 2.50 |
| AAXD(10) | | 1 | 1.48 | 1,526 | 4.21 | 1.02 | 4.29 |
| AAXD(12) | | 1 | 0.74 | 2,010 | 5.40 | 1.67 | 9.02 |
| AAXD(16) | | 1 | 0.18 | 2,432 | 9.10 | 2.31 | 21.02 |
| PLApp(4,4) | | 1 | 2.82 | 1,829 | 1.25 | 0.86 | 1.08 |
| PLApp(4,5) | | 1 | 1.40 | 2,073 | 1.64 | 1.15 | 1.89 |
| PLApp(4,6) | | 1 | 0.72 | 2,383 | 1.81 | 1.51 | 2.73 |
| PLApp(8,8) | | 1 | 0.18 | 3,822 | 2.52 | 2.86 | 7.21 |
| Exact divider (DesignWare) | | 1 | 0 | 3,206 | 50.53 | 1.89 | 95.50 |

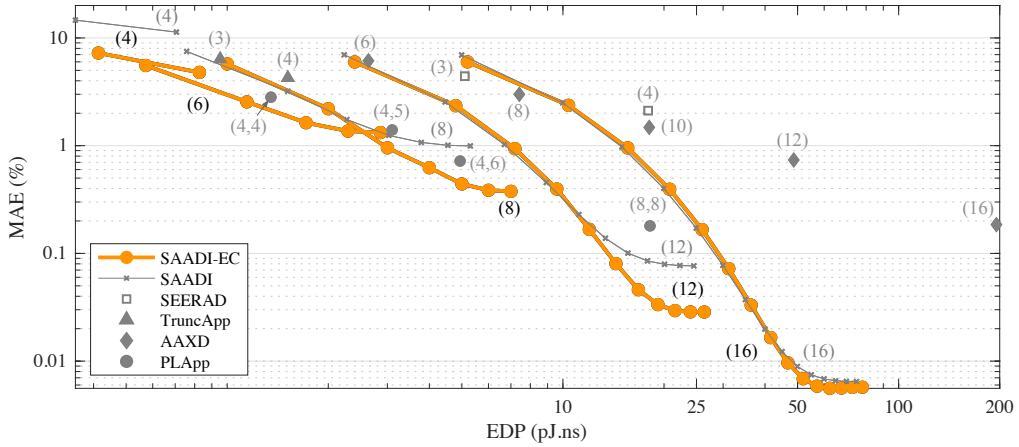



Figure 3.5: EDP-MAE trade-off comparison. TruncApp, SEERAD, AAXD, and PLApp have a single EDP-MAE point per accuracy level. SAADI-EC and SAADI have a flexible EDP-MAE curve per accuracy level.

and the energy budget at design time, and at runtime, $t$ can be adjusted to exploit the trade-offs. TruncApp, SEERAD, and PLApp are denoted by "▲", "□", and "" marks, respectively. TruncApp$(4)$ has the best trade-offs compared to other TruncApp configurations, but it is outperformed by SAADI-EC with $n = 6$ and $t = 2$. Similarly, SEERAD$(4)$ is outperformed by SAADI-EC with $n = 8$ and $t = 2$. PLApp is outperformed by SAADI-EC when $n = 6$ and $n = 8$. AAXD is denoted by "♦" marks, and it exhibits high EDP due to the precise divider used as the base arithmetic unit. SAADI-EC shows lower EDP since it uses a multiplier. SAADI-EC outperforms AAXD both in MAE and EDP after a few iterations, which is affordable due to its shorter cycle period.

For application-level evaluation, we perform color quantization, which reduces the number of unique colors in an image while preserving the color appearance as much as possible. Color quantization saves energy for the storage or transmission of the image, and a commonly used algorithm is k-means clustering. The mean update operation in k-means requires

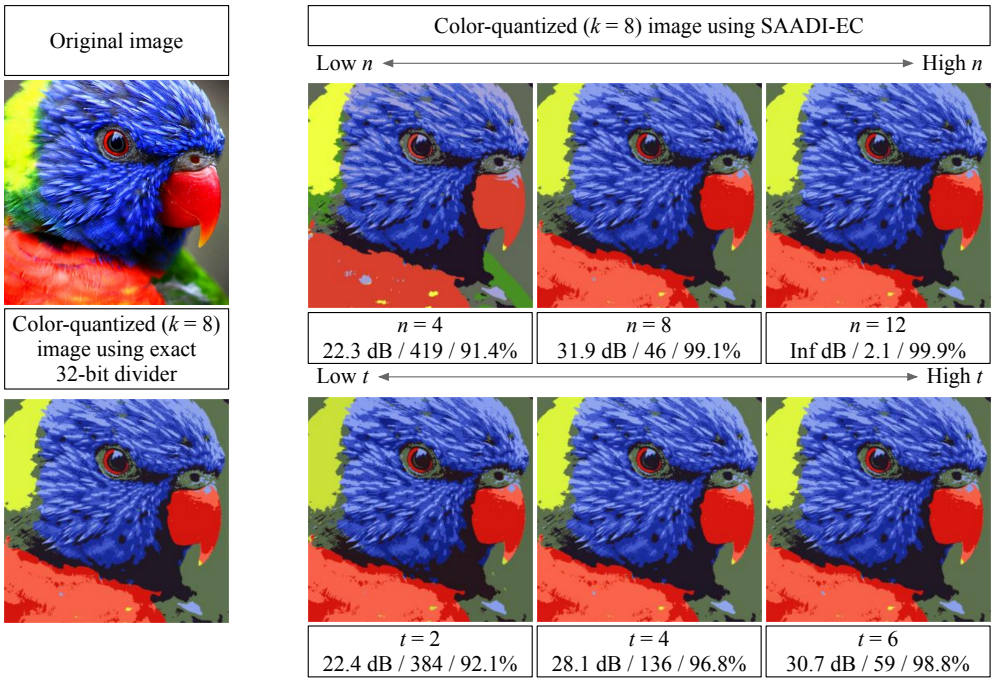



Figure 3.6: Color quantization using k-means clustering ($k = 8$) for varying $n$ and $t$. The output quality is shown in PSNR (dB)/MSE/SSIM (%) in comparison to the output of an 32-bit exact divider.

iterative division by non-constant values, thus an energy-efficient divider needs to be used. To examine the quality of SAADI compared with an exact divider in high level and evaluate the accuracy with varying $t$ or $n$, five images are respectively clustered into eight colors. We calculate three quality metrics: peak signal-to-noise ratio (PSNR), mean square error (MSE), and structural similarity (SSIM) with respect to the baseline results using an exact divider. To evaluate SAADI without the impact of the initial random centers, we feed the same initial centers into the baseline and SAADI each time and repeat ten times to take the average quality measures.

Figure 3.6 shows the original image and the color-reduced image using an exact divider as the reference. The first row shows the results by SAADI

with different $n$. We tie $t$ to $n-1$, providing sufficient time for SAADI to converge. As $n$ increases from 4 to 12, all three quality measures improve. Although SAADI is capable of producing an output of a reasonable quality for $n=4$, the quality dramatically increases for $n=8$, approaching 100% SSIM for $n=12$. The results with different $t$ are shown in the second row. In this case, $n$ is fixed to 8, and $t$ is set to 2, 4, or 6. We observe a clear trend of quality improvement with a larger $t$. Even with only two cycles, SAADI produces a high-quality result with SSIM higher than 90%, and further accuracy improvement is achieved by increasing $t$.

**Conclusion**

An energy-efficient divider is a crucial arithmetic unit for low-power signal processing. In this paper, we presented a quality-configurable approximate divider named SAADI-EC. It is based on an incremental approximation of the reciprocal of the divisor, where the accuracy gradually increases over multiple iterations. The application can set the number of iterations to exploit the trade-off between accuracy the latency to meet its requirement. We demonstrated that SAADI-EC produces the division results with adjustable accuracy. The accuracy-latency trade-off of the different implementations of SAADI-EC is evaluated, and the results for 32-bit di- vision with 8-bit approximation show the average accuracy between 94.2% and 99.6% with a latency between one and seven cycles. We also performed color quantization using k- means clustering and JPEG compression and demonstrated that SAADI-EC can produce high-quality results comparable to those generated by an exact divider. Finally, we showed that SAADI-EC outperforms other approximate dividers in applications that have dynamic accuracy requirements using image division. The challenging part of this project is to show how this quality-scalable hardware works in an end-to-end application with variant accuracy requirements and results in energy efficiency. In this work, we simplified the case and

simulated the showcase where the required accuracy of image division varies over time by dividing consecutive frames of a video sequence comprised of three change detection benchmark scenes. During each of these three scenes, we alter the accuracy requirement of the output image and dynamically adjust the divider to meet the requirement.

## 3.2 AxSerBus: quality-configurable approximate serial Bus Interface

Recent years have seen a rapid proliferation of wearable and implantable devices such as action cameras, health-tracking bands, smartwatches, and pacemakers. These devices spend significant amounts of time reading sensors such as microphones, image sensors, accelerometers, electrocardiogram (ECG) sensors, etc. and processing the sensed data. Frequent or continuous sensing poses a new challenge to energy efficiency because the data transfer from off-chip sensors consumes significant energy. Unlike *on-chip* buses that have greatly benefited from improvements in semiconductor process technology, *off-chip* buses, and in particular serial interfaces that are commonly used with sensors (e.g., Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), MIPI Camera Serial Interface (CSI)), have improved much more slowly. As a result, as much as 13% of energy dissipation of microcontrollers in such systems can be attributed to data transfer from off-chip sensors Stanley-Marbell and Rinard (2016).

High dynamic power consumption due to the high capacitance, high data rate, and high utilization of the sensor-to-processor off-chip bus motivates this research project. For example, consider an 8-megapixel image sensor capturing a video at a frame rate of 30 fps. Even if the distance between the sensor and the processor is only 3 cm, the dynamic power just for the data transfer (not including power consumed for processing the

data) can be as high as 49 mW[1]. This power consumption will be a bigger challenge with increasing resolution (e.g., 4K or 5K) and frame-rate (e.g., 60 fps or 240 fps for slow motion) in the video.

Since sensors convert physical quantities, such as light, temperature, or vibration, into a digital value, their outputs inherently include noise. Hence, the algorithms that process sensory data are designed to tolerate noisy inputs. For example, a small amount of noise in the readings from an accelerometer in a pedometer will not result in a significant error in the step count, which is what the user is ultimately interested in. Using traditional off-chip serial bus encodings for reading data from such sensors essentially amounts to *transferring noisy data precisely*, which is an overkill and wastes energy.

*Approximate computing* exploits the error resiliency of such applications to save energy by relaxing the exactness requirement and producing results that are *just good enough* Venkataramani et al. (2015a). Adopting this broad approach, *approximate serial bus* techniques have been proposed recently to encode data in a *lossy* manner to reduce bit transition counts (TC), leading to reduced switching power in the serial bus Stanley-Marbell and Rinard (2016); Pagliari et al. (2016b,a). Approximate serial bus encoding techniques need to address three main challenges. First, the encoding scheme should encode as many data values as possible using energy-efficient bit patterns, minimizing the transmission of high-energy cost bit patterns. Second, the encoding scheme should provide the application with a control knob for quality configuration because the notion of a good-enough result is application-specific and contextual. Third, the encoding and decoding need to be highly energy-efficient to prevent the energy overheads from nullifying the energy savings obtained from the bus.

In this task, we propose a novel *quality-configurable approximate serial bus* called AxSerBus (Approximate Serial Bus) to address the aforementioned

[1]Assuming a parasitic capacitance of 1 pF/cm, a driver voltage level of 3.3 V, and a color depth of 8 bits.

challenges. AxSerBus encodes data in three modes: (i) If the difference between the current and previous data values is very small, the difference is zeroed out, and a 0-TC (no intra-word TC) pattern is transmitted, and the receiver reuses the previous data value (this is similar to Pagliari et al. (2016b)). (ii) If the difference is in an intermediate range (defined by adjustable thresholds), an approximate value of the difference is encoded as a 1-TC (single intra-word TC) pattern, which incurs minimal energy dissipation since it contains only a single bit transition. The use of approximate differences allows a wider range of differences to be encoded in this energy-efficient encoding mode. An interesting property of this encoding scheme is that the error due to approximation scales with the magnitude of the differences, which minimizes the error. (iii) Finally, if the difference is large, we transmit the absolute value and not the difference. The energy cost for transmission of the absolute value is relatively high because it may contain several bit transitions, but only a small fraction of data is encoded in this mode. AxSerBus does not require extra bits to indicate the encoding mode because the mode information is implicit in the TC of each codeword. In summary, the contributions of this project are as follows:

- We propose a quality-configurable approximate serial bus called AxSerBus that can significantly reduce the energy consumed by off-chip serial buses with minimal quality degradation. AxSerBus achieves more than 80% reduction in TC compared to a traditional (exact) serial bus and up to 26.5% more reduction in TC compared to a state-of-the-art approximate serial bus given the same accuracy requirement.

- Later in Section 4.1 we propose a dynamic quality configuration technique with no runtime overhead of quality comparison. We demonstrate that it facilitates a near-optimal energy-quality trade-off for variable input data at runtime.

- We present an energy-efficient hardware implementation of the proposed encoder and decoder. Encoding and decoding of a multi-bit word are done in a single cycle in hardware, consuming minimal power.
- We demonstrate the trade-off between application-level quality and energy saving using an optical character recognition (OCR) application. AxSerBus achieves 79.4% reduction in dynamic power dissipation while maintaining OCR accuracy above 95%.

**AsXerBus encoding and decoding**

Before describing the details of the encoding and decoding schemes, we first define some symbols. Let $s_i$ be the *original source word*, $t_i$ the *encoded word* (*codeword*), and $s'_i$ the *decoded word* of length $l$ in the $i$-th transmission. We define difference $\delta_i$ as the difference between the current source word $s_i$ and the previous source word $s_{i-1}$. The difference between $s_i$ and $s'_i$ is defined as error $\epsilon_i$. In order to prevent error accumulation, the sender should transmit the sum of the current difference $\delta_i$ and the error in the previous transmission $\epsilon_{i-1}$. We define the sum as $\Delta_i$. These can be formulated as:

$$\delta_i = s_i - s_{i-1} \qquad \epsilon_i = s_i - s'_i \qquad \Delta_i = \delta_i + \epsilon_{i-1}$$

Also, $tc(w)$ denotes the intra-word TC of a single word $w$, where $w$ is either $s$ or $t$. The cumulative sum of TC of multiple consecutive words, including both intra- and inter-word TCs, is defined as serial TC, or STC. The dynamic power consumption is directly proportional to STC, and, throughout the paper, STC is the objective function to be minimized.

The encoding mode for the $i$-th transmission is selected among the *low difference* (*LD*) *encoding mode*, *intermediate difference* (*MD*) *encoding mode*, and *high difference* (*HD*) *encoding mode* based on $\Delta_i$. The following

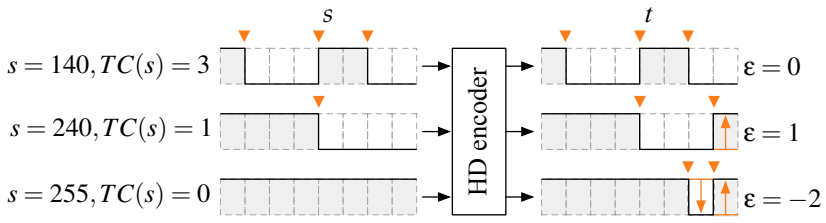



Figure 3.7: Encoding examples in the HD encoding mode for $l = 8$.

subsections describe the encoding and decoding mechanisms in each mode.

### *A. LD encoding mode*

If $|\Delta|$ is very small, i.e., $\Delta < D_0$, where $D_0$ is the *zero-out* threshold, we approximate $\Delta$ as zero. We call the error introduced by this approximation the *zero-out error*. In this case, $\Delta = 0$ is encoded as a 0-TC codeword, which is the most low-energy codeword. In $l$-bit codewords, there exist two 0-TC codewords (all 1's and all 0's), and we use all 1's to represent $\Delta = 0$. This is preferable in some serial bus physical-layer standards that have asymmetric static power consumption where a bit 0 consumes more power than a bit 1 because the line is pulled low by an open-drain driver, such as in I2C.

### *B. MD encoding mode*

If $D_0 \leqslant |\Delta| < D_m$, where $D_m$ is the *mode threshold*, $|\Delta|$ is encoded as a 1-TC codeword. There exist $2(l-1)$ 1-TC codewords, and they are used to represent small (non-zero) differences in an approximate manner, with non-uniform approximation bounds. Let us assume $\Delta \geqslant 0$. If $\Delta < 2^{l-1}$, $\Delta$ is approximated as

$$\Delta' = 2^{\lfloor \log_2 \Delta \rfloor} + 2^{\lfloor \log_2 \lfloor \Delta/2 \rfloor \rfloor}. \tag{3.10}$$

Table 3.2: Approximate encoding in the LD and MD encoding modes for $l = 8$.

| $\lvert\Delta\rvert$ range | $\lvert\Delta'\rvert$ | t if $\Delta > 0$ | t if $\Delta < 0$ | Max. $\epsilon$ |
|---|---|---|---|---|
| 0 | 0 (00000000) | 11111111 | 11111111 | 0 |
| 1 | 1 (00000001) | 1111111**0** | **0**1111111 | 0 |
| [2, 3] | 3 (0000001**1**) | 111111**00** | **00**111111 | $\pm$1 |
| [4, 7] | 6 (000001**10**) | 11111**000** | **000**11111 | $\pm$2 |
| [8, 15] | 12 (00001**100**) | 1111**0000** | **0000**1111 | $\pm$4 |
| [16, 31] | 24 (0001**1000**) | 111**00000** | **00000**111 | $\pm$8 |
| [32, 63] | 48 (0011**0000**) | **11**000000 | **00000**11 | $\pm$16 |
| [64, 127] | 96 (0**1100000**) | **1**0000000 | **00000001** | $\pm$32 |

In other words, all $\Delta$ between $2^r$ and $2^{r+1}-1$ are approximated as $2^r + 2^{r-1}$, which is the median of the range. Then, $\Delta$ is encoded as a run of $l - r$ bits of 1's, followed by $r$ bits of 0's. In case that $\Delta < 0$, $|\Delta|$ is encoded in the same manner, but the order of bit 1's and bit 0's is reversed. The error due to this approximation is called the *approximation error*. This error is added to the next $\delta$ for error compensation.

At the receiver side, $t$ is decoded as $\Delta'$ and added to the previous $s'$. Table 3.2 shows the LD and MD mode codewords for $l = 8$. Codewords with more number of bit 1's are assigned to lower $\Delta$ because bit 1 is preferred for low power, as mentioned above.

### *C. HD encoding mode*

If $|\Delta|$ is greater than the mode threshold (i.e., $|\Delta| \geqslant D_m$), the absolute value of $s$, instead of the difference, is encoded as a multi-TC (two or more intra-word TCs) codeword. If $tc(s)$ is already two or more, the source data is not modified and transmitted as its binary representation, i.e., $t = s$. However, if $tc(s)$ is 0 or 1, it cannot be transmitted without modification because 0-TC and 1-TC codewords are reserved for the LD and MD mode,

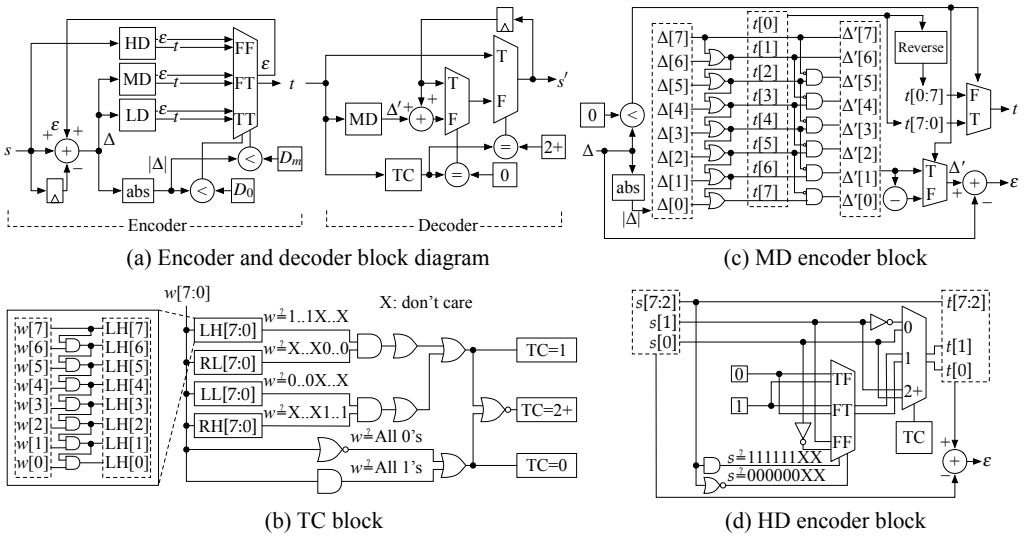



Figure 3.8: (a) Encoder and decoder block diagrams, (b) TC block in the encoder and the decoder, (c) MD encoder block in the encoder, and (d) HD encoder block in the encoder.

respectively. This collision of codewords could be avoided by adding extra bits to indicate the encoding mode, but this would incur a heavy overhead. Instead, knowing that an additional minor error will not affect the overall quality and that the HD encoding mode will not be used frequently, we intentionally increase $\mathtt{tc}(\mathtt{t})$ to 2 by flipping a minimum number of bits so that the decoder can identify the encoding mode from $\mathtt{tc}(\mathtt{t})$. To minimize the value deviation, the last one or two least significant bits (LSBs) are flipped, as shown in Figure 3.7. The maximum error due to the bit flips is only $\pm 2$. No decoding is required at the receiver side because $\mathtt{t}$ is already the binary representation of the exact value or an approximate value of $s$.

### Implementation

AxSerBus is validated by implementing hardware for the encoding and decoding logic. Figure 3.8(a) is the high-level hardware block diagram

of the encoder and the decoder to show how the encoding and decoding modes are selected. We describe the detailed implementation of the major blocks.

**TC counter block:** The TC counter block is common in the encoder and the decoder, and it determines whether the input word $w$ ($s$ or $t$) is a 0-TC, 1-TC, or multi-TC word. Determining whether $w$ has 0 TC or 1 TC is relatively simple since the 0-TC and 1-TC words are limited. Two 0-TC words, i.e., all 1's and all 0's, can be detected by AND and NOR gates, as shown at the bottom of Figure 3.8(b). The logic for detecting 1-TC words is shown at the top of Figure 3.8(b). LH, RL, LL, and RH stand for "left high", "right low", "left low", and "right high", respectively. They indicate whether an input word $w$ has a specific bit pattern. For example, LH[3] is true if $w$ starts with three 1's, i.e., if $w$ is 111XXXXX, where X is a 'don't care'. Similarly, RL[5] is true if $w$ ends with five 0's, i.e., XXX00000. Therefore, if both RH[3] and LL[5] are true, $w$ is 11100000, which has a TC of 1. Other 1-TC words can be detected by applying the same method. More specifically, $w$ is a 1-TC word if either

$$(\text{LH}[1]\&\text{RL}[7])|(\text{LH}[2]\&\text{RL}[6])|(\text{LH}[3]\&\text{RL}[5])|(\text{LH}[4]\&\text{RL}[4])$$
$$|(\text{LH}[5]\&\text{RL}[3])|(\text{LH}[6]\&\text{RL}[2])|(\text{LH}[7]\&\text{RL}[1])$$

or

$$(\text{LL}[1]\&\text{RH}[7])|(\text{LL}[2]\&\text{RH}[6])|(\text{LL}[3]\&\text{RH}[5])|(\text{LL}[4]\&\text{RH}[4])$$
$$|(\text{LL}[5]\&\text{RH}[3])|(\text{LL}[6]\&\text{RH}[2])|(\text{LL}[7]\&\text{RH}[1])$$

is true. LH, RL, LL, and RH can be implemented using only a few number of gates, as shown in the box in Figure 3.8(b). If $w$ is neither a 0-TC nor 1-TC word, we do not count the TC and simply determine that it has two or more TCs since all multi-TC codewords take the same datapath in the encoder and the decoder, regardless of their actual TC.

**LD encoder block:** If $|\Delta| < D_0$, the LD encoder block generates the 0-TC code as described in Section 3.2. The codeword is always $l$ bits of 1's, and the error $\epsilon$ is equal to $-\Delta$. On the decoder side, if the TC of the received codeword $t$ is 0, it simply recirculates the previous $s'$.

**MD encoder/decoder blocks:** If $D_0 \leqslant |\Delta| < D_m$, the MD encoder block generates a 1-TC code as described in Section 3.2. Figure 3.8(c) shows the logic to generate a 1-TC code $t$ and the error $\epsilon$ for intermediate differences. First, a 1-TC codeword $t$ is generated for $|\Delta|$ using the cascaded OR gates that detect the most significant 1 and make all less significant bits to 1's. If $\Delta < 0$, the bit order of $t$ is reversed by flipping the vector (Reverse block in the Figure 3.8(c)). To compute the error, the encoder decodes $t$ to $\Delta'$ using the cascaded AND gates that finds the most significant 1 and makes the next less significant bit 1, followed by 0's, as presented in Table 3.2. The difference between $\Delta$ and $\Delta'$ is $\epsilon$. The MD decoder block is identical to the part of the MD encoder that converts $t$ to $\Delta'$. Note that $\epsilon$ is not maintained on the decoder side.

**HD encoder block:** If $|\Delta| \geqslant D_m$, the HD encoder block, shown in Figure 3.8(d), generates a multi-TC code as described in Section 3.2. The HD encoder block modifies only lower two bits to ensure that $t$ has two or more TCs. The TC counter block is used to determine how the lower two bits should be modified. Since 2-TC words are interpreted by the decoder as an absolute value, the error $\epsilon$ is simply the difference between $s$ and $t$. On the decoder side, there is no HD decoder for multi-TC codewords since they are absolute values to be used as $s'$ as they are.

**Evaluation and discussion**

In this section, we first evaluate the reduction in STC achieved by AxSerBus and the data value deviations. Next, we evaluate the quality degradation at the application level.

We use 200 natural images from INRIA Holidays image dataset Jégou

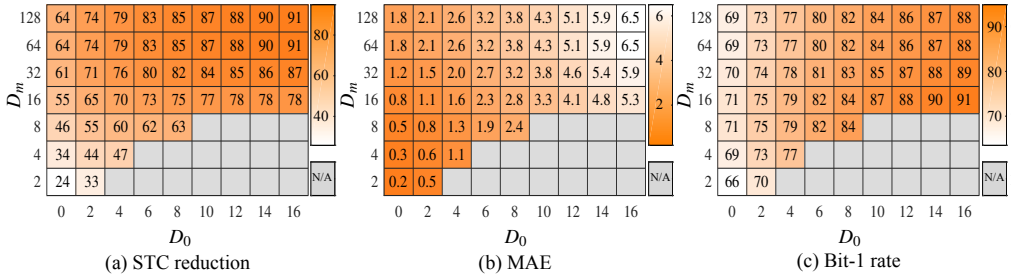



Figure 3.9: (a) STC reduction, (b) MAE, and (c) bit-1 rate for all possible threshold pairs used for 200 test images.

et al. (2008) as input. We first evaluate the impact of the threshold values on STC and data values. Figure 3.9(a) shows the average STC reduction for 200 natural images for different threshold pairs. We vary $D_0$ between 0 and 16 and $D_m$ between 2 and 128 independently. Some infeasible threshold pairs that $D_0 \geqslant D_m$ are denoted as N/A. As $D_0$ increases, more small differences are encoded in the LD mode, resulting in more data values encoded as a 0-TC code, hence a lower STC. Similarly, increasing $D_m$ results in that more data values are encoded in the MD mode, in which the approximate differences are transferred, resulting in a more STC reduction.

Figure 3.9(b) shows the mean absolute error (MAE) for different threshold pairs. Larger $D_0$ and $D_m$ introduces greater zero-out errors and approximation errors, respectively, and, therefore, the MAE increases as well. Large STC reduction of 44% is observed even with low approximation ($D_0 = 2$ and $D_m = 2$) yet with MAE as low as 0.6. Figure 3.9(c) shows the average rate of bit 1's per codeword. The bit-1 rate increases as $D_0$ increases because the 0-TC code composed of only bit 1's is used more frequently. On the other hand, it decreases as $D_m$ increases because MD mode codewords for higher differences have more number of bit 0's than codewords for lower differences. However, the rate is consistently higher than the expected bit-1 rate of unencoded words (50%).

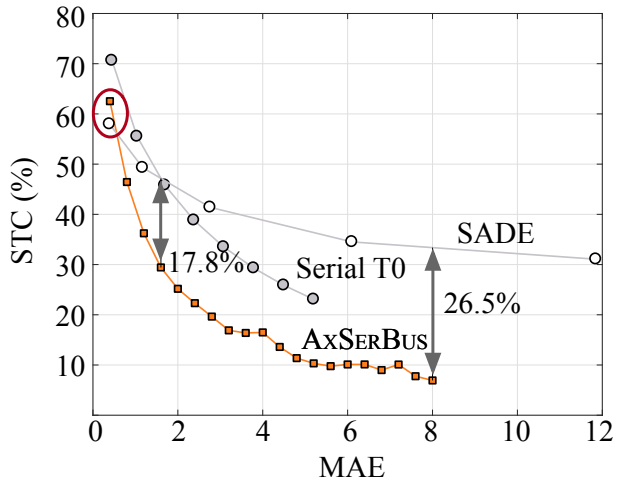



Figure 3.10: Comparison of Pareto-optimal energy-quality trade-offs to Serial T0 Pagliari et al. (2016b) and SADE Pagliari et al. (2017) for 200 test images.

In Figure 3.10, the energy-quality trade-off of AxSerBus is shown and compared to that of Serial T0 Pagliari et al. (2016b) and SADE Pagliari et al. (2017). It shows that AxSerBus performs better than the other two encoding schemes in most cases. AxSerBus achieves a lower STC by up to 17.8% and 26.5% than Serial T0 and SADE for the same MAE. SADE demonstrates a better energy-quality trade-off than AxSerBus when the error is very low, as marked by a red circle in Figure 3.10.

We evaluate the quality of data at the application level using image processing applications. Figure 3.11 shows the image quality with different threshold pairs. Inside each figure, the STC reduction from the original image and the PSNR is shown. The figure only shows a partial image of the entire test image to clearly show the detail, but the STC and PSNR are measured for the entire image. Even for the most aggressive threshold pair $(D_0, D_m) = (8, 64)$, the image degradation is hardly noticeable (PSNR of 35 dB), but the STC is dramatically decreased by 90%. Table 3.3 and Table 3.4 show the mean and standard deviation of STC reduction and

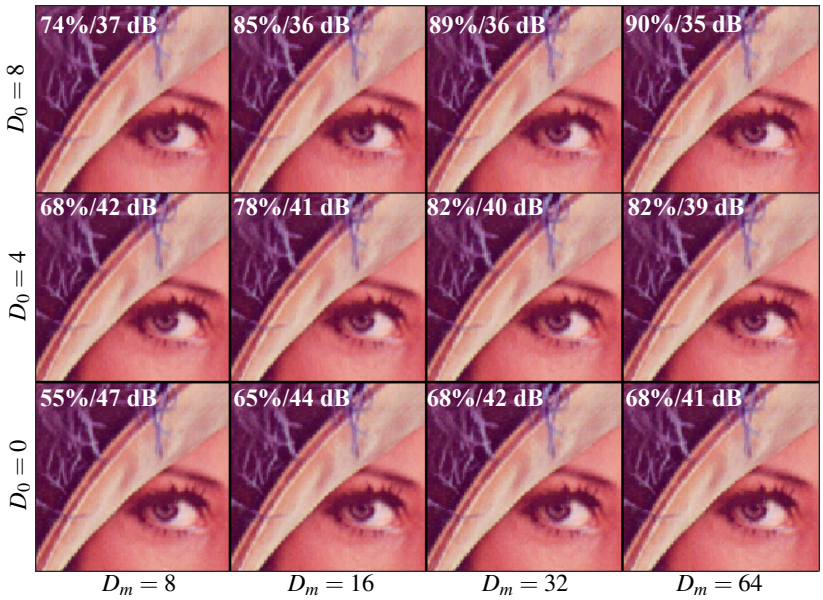



Figure 3.11: Magnified test image after encoding and decoding with different threshold configurations. STC reduction and PSNR are denoted inside each figure.

PSNR statistics for 200 images under test, respectively. Testing on 200 images, AxSerBus achieves 84% STC reduction on average with 6% standard deviation with threshold configuration $(\mathrm{D_0}, \mathrm{D_m}) = (8, 64)$ and keeping the average PSNR over 34 dB with only 1 dB of standard deviation.

Finally, We synthesized the Verilog description of the encoder and decoder separately using Synopsis Design Compiler, targeting a 45 nm CMOS standard cell library from IBM. The estimated power consumption is only 0.184 mW for the encoder and 0.153 mW for the decoder. The total power consumption of the encoder and the decoder is 0.337 mW, which is less than 1% of the potential power savings using this sensor communication encoding scheme.

Table 3.3: STC reduction statistics for 200 test images.

| | | | $D_m$ | | | |
|---|---|---|---|---|---|---|
| | | | 8 | 16 | 32 | 64 |
| $D_0$ | 8 | Mean | 62% | 74% | 81% | 84% |
| | | Std | 16% | 12% | 8% | 6% |
| | 4 | Mean | 58% | 69% | 75% | 78% |
| | | Std | 17% | 13% | 9% | 8% |
| | 0 | Mean | 43% | 53% | 60% | 63% |
| | | Std | 18% | 14% | 11% | 3% |

Table 3.4: PSNR statistics for 200 test images.

| | | | $D_m$ | | | |
|---|---|---|---|---|---|---|
| | | | 8 | 16 | 32 | 64 |
| $D_0$ | 8 | Mean | 37 dB | 37 dB | 36 dB | 34 dB |
| | | Std | 0.5 dB | 0.4 dB | 0.3 dB | 1.0 dB |
| | 4 | Mean | 43 dB | 41 dB | 39 dB | 37 dB |
| | | Std | 0.6 dB | 0.3 dB | 0.8 dB | 2.0 dB |
| | 0 | Mean | 48 dB | 45 dB | 42 dB | 38 dB |
| | | Std | 0.8 dB | 1.3 dB | 1.9 dB | 2.9 dB |

**Conclusion**

In this project, we tackled the problem of the high dynamic power consumption of sensory data transfer from off-chip sensors to a microcontroller. We introduced a quality-configurable approximate serial bus encoding technique called AxSerBus. AxSerBus encodes the differences of sensor outputs in an approximate manner, allowing smaller approximation errors for more frequently occurring smaller differences. The quality of data can be dynamically controlled by adjusting the approximation thresholds, and we presented a low-overhead dynamic quality configuration scheme that achieves near-Pareto optimal energy-quality trade-offs. AxSerBus achieves a significant reduction in STC while maintaining a

high application-level quality both for natural images and text images. We evaluated the reduction in STC and the impact on the quality using an OCR application and showed that AxSerBus achieves 79.4% reduction in dynamic power dissipation while maintaining OCR accuracy above 95%. Showcasing this energy-efficient serial bus encoding scheme the main challenge was prototyping and building a working demo of the AxSerBus use case in a real application. To do so one needs to implement a complete system performing a task completely. We implemented the AxSerBus decoder and encoder in FPGA and performed an image processing application with the off-chip camera. Driving the sensor peripherals to capture the frames was the most challenging part of this demo implementation.

# 4 DESIGNING CONTROL SYSTEM FOR ENERGY-QUALITY SCALABLE COMPONENTS

In this thrust of the project, we introduce an efficient control framework that automatically adjusts the level of approximation without the user having to explicitly interfere. In doing so, a translation from component-level quality metric to application-level quality metric is needed. For example, for the approximate serial bus encoding proposed in 3.2, the component quality metric is pixel values, and the application level quality metric could be PSNR or success rate in an image processing application or an optical character recognition (OCR) respectively. In this thrust, we attempt to develop application and component quality metric mapping functions to be used in the control framework. This translation accounts for the component-level error propagation to high-level application quality. Quality mapping functions enable high-level quality estimation based on low-level error metrics. The goal of this thrust of the project is to design a feedback control system to maintain the application-level quality above the desired accuracy requirement.

In addition, we develop a design methodology that provides system designers with optimal approximate hardware components as well as their control system given system constraints such as energy or performance. For example, the iterative divider introduced in Section 3.1 computes the division result in a progressive manner. In other words, given more number of iterations, the SAADI divider calculates a more accurate result. In this approximate architecture, the number of iterations is an approximate knob that needs to be determined.

In the following, at first, we present a dynamic quality configuration based on offline training for an approximate serial bus encoder. Then, we introduce an integer linear programming (ILP)-based scheduling method to determine the optimal latency of quality-scalable computing hardware

in an offline manner in order to maximize the energy saving given application output quality and performance constraints.

## 4.1 Application level quality control methodology for a quality-configurable approximate serial Bus interface

The quality of data is application-dependent and does not scale with the approximation error. Hence the degree of approximation should be dynamically adjusted depending on the need of the application. To address the demand for quality control, in this Section, we focus on the runtime control of the approximate serial bus proposed before in Section 3.2 for dynamic quality management.

*Dynamic quality configuration* refers to dynamically changing the accuracy of approximate data in order to meet the data quality requirement of applications. The quality requirement of sensory data varies by application; for example, an image recognition application and a text recognition application may have different image quality requirements, even though they both use data from an image sensor. Moreover, even for a single application, the quality requirement may vary over time. Therefore, the approximate serial bus needs to provide the application with a control knob that can be used to modulate the data quality.

In AxSerBus, the quality of decoded data is varied by adjusting $D_0$ and $D_m$ at runtime. By simply writing new values in the registers that hold $D_0$ and $D_m$ (denoted as boxes with $D_0$ and $D_m$ in Figure 3.8(a)), the data quality is adjusted at runtime. We define a *threshold pair* $D$ as a pair of $D_0$ and $D_m$, i.e., $D = (D_0, D_m)$. For a given threshold pair, lowering $D_0$ or $D_m$ will improve the data quality, while raising $D_0$ or $D_m$ will deteriorate it. Ideally, we can provide the application with a full degree of freedom to

adjust $D_0$ and $D_m$ independently and let it decide which to adjust after comparing the results. However, this will introduce significant energy and performance overheads for the evaluation of energy-quality trade-offs if the configuration occurs frequently. Moreover, since the characteristics of the source data vary over time, the selected threshold pair will soon become sub-optimal for the new source data. Therefore, adjusting $D_0$ and $D_m$ independently for dynamic quality configuration has no practical advantage.

To facilitate practical dynamic quality configuration, we introduce a simple heuristic scheme that does not pose any runtime overheads for energy-quality evaluation yet achieves near-optimal trade-offs. We base on the observation that, even if the energy-quality trade-offs vary by source data, the *energy-quality Pareto optimal threshold pairs* are similar to each other. We define energy-quality Pareto optimal threshold pairs for individual source data as $\mathbf{D}_{Pareto}$, which can be obtained by evaluating the energy-quality trade-offs for all possible threshold pairs. Instead of using $\mathbf{D}_{Pareto}$ that is different for each source data, our quality configuration scheme uses a set of *general* energy-quality Pareto optimal threshold pairs obtained from a sensor-specific and application-specific *training source dataset*. To obtain this, we average STC and MAE for each threshold pair for each source data in the training source dataset, and find the Pareto optimal configurations among them. From the Pareto optimal configurations, we select a monotonic sequence from the Pareto optimal configurations from the minimum $(D_0, D_m)$ to the maximum $(D_0, D_m)$ such that both $D_0$ and $D_m$ change monotonically.

This sequence may not include all the Pareto optimal configurations if $D_0$ or $D_m$ is not monotonic; for example, if $(8, 16)$ and $(10, 16)$ are Pareto optimal, $(8, 32)$ is not included in the sequence even if it is Pareto optimal. This is because these non-monotonic Pareto optimal configurations are more sensitive to datasets used for training, and including them increases

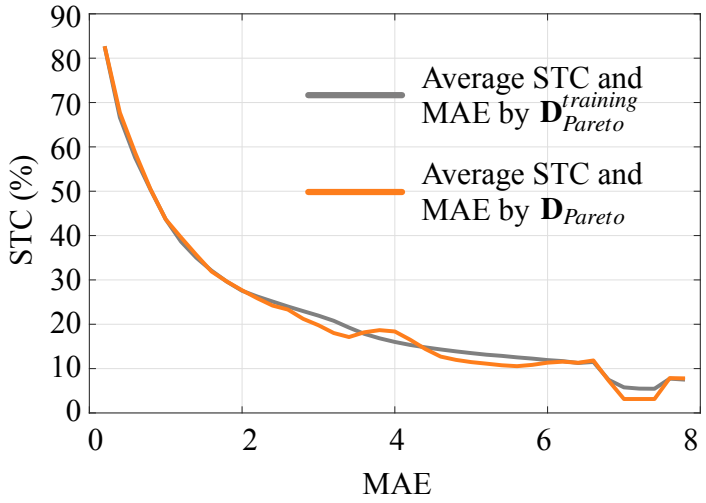



Figure 4.1: Trade-offs of average STC and average MAE for $\mathbf{D}_{\mathrm{Pareto}}$ and $\mathbf{D}_{\mathrm{Pareto}}^{\mathrm{training}}$ for 200 test images.

the complexity of dynamic configuration with little energy-quality benefit for general input data. We define this sequence as $\mathbf{D}_{\mathrm{Pareto}}^{\mathrm{trained}}$ and use as the *general energy-quality Pareto optimal threshold pairs* for runtime quality configuration. At runtime, the application simply switches to the next or previous threshold pair to adjust the quality without any runtime energy-quality evaluation. Since the data observed at runtime are different from the data used for training, this quality configuration may result in a sub-optimal energy-quality trade-off. However, in practice, most threshold pairs in $\mathbf{D}_{\mathrm{Pareto}}^{\mathrm{training}}$ are also Pareto optimal for other source data of the same type, and therefore even the thresholds chosen statically exhibit a near-optimal energy-quality trade-off.

### Evaluation and discussion

We demonstrate that the dynamic quality configuration based on the static selection of $\mathbf{D}_{\mathrm{pareto}}^{\mathrm{training}}$, described above achieves very near-optimal energy-quality trade-offs. The threshold pairs in $\mathbf{D}_{\mathrm{pareto}}^{\mathrm{training}}$ are obtained

from offline training of AxSerBus using a static training source dataset. Out of 200 images in INRIA Holidays dataset, 100 training images are used for selecting $\mathbf{D}_{pareto}^{training}$ and the rest of 100 are used for test purpose. Figure 4.1 shows the trade-offs between the average STC and the average MAE obtained by $\mathbf{D}_{pareto}$ and image-specific $\mathbf{D}_{pareto}^{training}$, respectively for 200 test images. When averaged, the discrepancy between $\mathbf{D}_{pareto}$ and $\mathbf{D}_{pareto}^{training}$ become even more insignificant.

## 4.2 Scheduling of energy-quality scalable hardware for energy-efficient approximate computing

The past decade has witnessed the ever-increasing demand for energy-efficient computing driven by the prevalence of power-demanding applications (e.g., machine learning) on power-constrained computing devices at the edge. *Approximate computing* has emerged as a promising solution for energy-efficient computing that produces "just good enough" results for applications where imprecise results are sufficient for its purpose at a much lower effort Han et al. (2013); Venkataramani et al. (2015a); Alioto (2017). Various error-resilient applications where quality can be gracefully traded off for energy efficiency, such as sensing, signal processing and, more recently, machine learning applications Han et al. (2016, 2015); Shin and Gupta (2010), have greatly benefited from the new computing paradigm and hardware techniques that have been proposed in recent years Mittal (2016); Alioto et al. (2018); Kim et al. (2020).

Energy-quality scaling opportunities in the system vary for different applications and even within one application it differs over time and with different inputs. Integrating approximate units without run-time configurability requires setting the approximation level at the maximum

accuracy requirement for the application and neglects the run-time energy-quality scaling opportunities which genuinely wastes energy which should be avoided in energy-constrained systems.

Most of the prior work in the area of approximate computing has mainly focused on designing approximate hardware components such as approximate adders and multipliers Hashemi et al. (2016); Babić et al. (2011); Ahmed and Srinivas (2019). However, most, if not all, applications involve more than one approximate computations that require system-level orchestration. Specifically, despite the rich set of approximate hardware components that have been actively proposed, a systematic method to determine which computations in the application should be approximated to maximize energy efficiency with minimal quality loss has been largely lacking. Moreover, for those approximate hardware components that support dynamic reconfiguration of energy-quality trade-off to meet energy efficiency and quality requirements adaptive to varying input or user's need at runtime Behroozi et al. (2019); Babić et al. (2011); Ahmed and Srinivas (2019), making the orchestration even more difficult.

*Iterative computing* is one such approach that progressively improves the output quality over multiple iterations. Using iterative computing hardware, one can either achieve high accuracy by allowing for more latency or high power efficiency by finishing computation sooner. To take advantage of this approach, various iterative computing hardware designs have been proposed, mainly for arithmetic operations Babić et al. (2011); Ahmed and Srinivas (2019); Behroozi et al. (2019); Saadat et al. (2020); Melchert et al. (2019); Wu et al. (2019). However, in spite of the benefit, the adoption of iterative computing hardware has been impeded by the lack of an efficient method to determine the optimal latency to meet the energy and quality requirements at the same time.

There are several major challenges in determining the optimal latency of iterative computing hardware. First, the optimal latency is heavily input-

dependent because the accuracy of approximate computing hardware varies significantly dependent on their input. Moreover, most applications consist of multiple inter-dependent computations where the output of one computation is input to another computation, making the accuracy estimation even more challenging. Finally, the design space complexity is exponential to the application complexity, rendering naive design space exploration methods (e.g., exhaustive search) impractical. Unfortunately, there exist no prior approximate computing hardware optimization methods are not suitable for the problem of scheduling of iterative computing hardware. Some recent work Li et al. (2015); Hanif et al. (2020) proposed methods for optimizing the selection of hardware with different accuracy, but they do not consider the case where the same hardware has varying accuracy depending on latency. A gradient descent-based method proposed Raha and Raghunathan (2017) is not directly applicable since it does not consider the case where the same hardware is used multiple times throughout the processing pipeline. A more recent work Kemp et al. (2021) optimizes the latency of iterative hardware for a given accuracy constraint, but it does not consider multiple units.

In this project, we propose an integer linear programming (ILP)-based scheduling method to address the aforementioned challenges. We consider the input dependence of output accuracy by using a data-driven error modeling of iterative computing hardware depending on its location within the application. The proposed error modeling enables accurate estimation of the final output quality for a given input distribution without running time-consuming simulations. We demonstrate that the proposed method successfully finds optimal or near-optimal scheduling solutions orders of magnitude faster than exhaustive search or decision tree-based optimization.

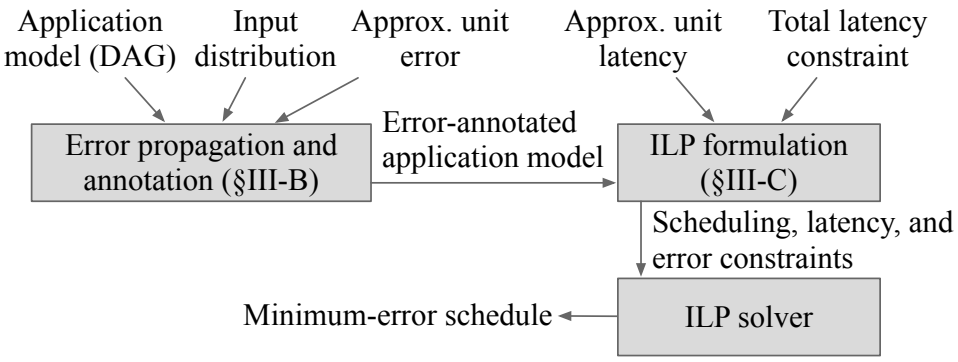



Figure 4.2: Solving the scheduling problem for approximate hardware units using ILP.

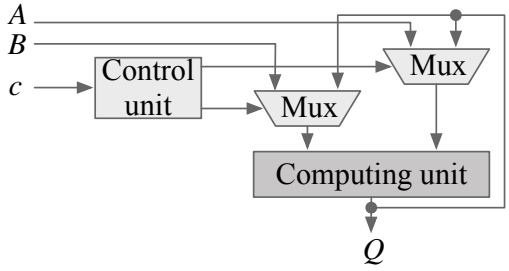



Figure 4.3: Architecture of iterative computing-based approximate hardware.

**ILP-based Scheduling of Iterative Computing Hardware**

In this section, we define the scheduling problem of iterative computing hardware along with our system model. Then we describe the error propagation model and annotation, followed by the formulation of the scheduling problem in ILP. Figure 4.2 illustrates the overall flow of scheduling described in this section.

We consider the hardware architecture composed of precise hardware units and iterative computing-based approximate hardware units. Precise hardware units have a fixed latency and always produce exact results. On the other hand, approximate hardware units have a finite number of discrete approximation levels to dynamically trade-off accuracy for

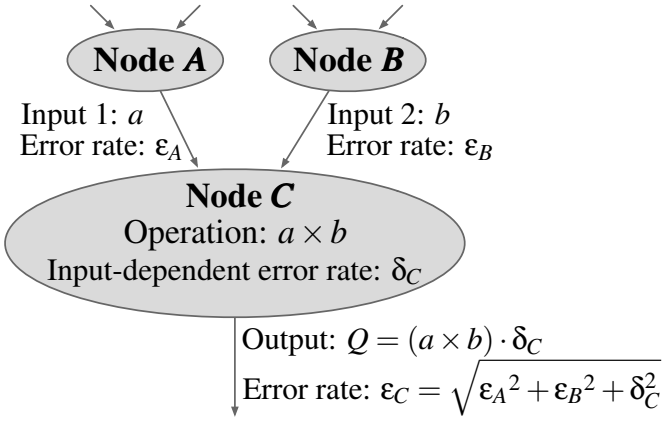



Figure 4.4: Propagation of error through approximate multiplication in Node C.

latency. Figure 4.3 shows a general architecture of iterative computing-based approximate hardware, where the accuracy of output Q improves over $c$ iterations. The implementation of iterative computing hardware can be based on various iterative approximation algorithms such as logarithmic operations Ahmed and Srinivas (2019) or Taylor approximation Behroozi et al. (2019); Melchert et al. (2019); Wu et al. (2019).

An application is modeled as a directed acyclic graph (DAG), $G = \langle V, E \rangle$, where operations are represented as nodes V, and dependencies between operations are presented as edges E between corresponding nodes. The distribution of the input is known at design time, which is common in application-specific system design. If not known, one can assume any statistical distribution. The scheduling problem is, given a total latency constraint, to find the optimal starting time and the latency of each operation that minimizes the error rate of the final output.

### Error Propagation and Annotation

The output error of an approximate operation is a result of the input error rates and the error of the approximate operation, which can be modeled

as the propagation of error Hughes and Hase (2010). The error rate of an approximate operation, $\delta$, can be considered as a scaling factor that is multiplied to the precise result of the operation. That is, for an operation $f(a, b)$, the approximate output is

$$Q = f(a, b) \cdot (1 + \delta), \tag{4.1}$$

and the error rate of Q is

$$\epsilon = \sqrt{\epsilon_{f(a,b)}{}^2 + \delta^2}, \tag{4.2}$$

where $\epsilon_{f(a,b)}$ is the error rate when the operation $f(a, b)$ is precisely performed on erroneous inputs $a$ and $b$. For example, consider the approximate multiplication in Node C in Figure 4.4. When the inputs $a$ and $b$ have error rates of $\epsilon_A$ and $\epsilon_B$, respectively, and the error rate of the approximate multiplication is $\delta_C$, the error rate of the output of Node C is

$$\epsilon_C = \sqrt{\epsilon_A{}^2 + \epsilon_B{}^2 + \delta_C{}^2}, \tag{4.3}$$

since the error propagation of multiplication $a \times b$ is

$$\epsilon_{a\times b} = \sqrt{\epsilon_A{}^2 + \epsilon_B{}^2}. \tag{4.4}$$

We estimate error rates for all approximate operations and all approximate levels ($\delta_i[k]$, described later in Table 4.1). For that, we measure the error rate of a target operation (Node $i$) in all approximate levels ($k$) when all the other operations in the application are precise. We repeat this process for all operations within the application. Error rates for all approximate operations are normalized to reflect the amount of accuracy improvement for varying approximate levels in the error propagation model. Once all operations in the DAG are annotated with error rates, the error rate of the final node is the objective function to be minimized.

To ensure the high accuracy of the error model, we take the varying distribution of input of each node into consideration. Even for the same approximate operation and the same approximate level, the average error rate can be different depending on where in the DAG the operation is performed because input distribution is not the same. Input distribution is obtained by running the application with a given dataset without approximation and statistically characterizing the input distributions at each node.

**ILP Problem Formulation**

From the error-annotated DAG, we generate ILP constraints. Table 4.1 lists the notations we use to describe the ILP formulation in this section. *Inputs to ILP-based scheduling* are (i) error-annotated application DAG generated in Section 4.2, (ii) latencies of approximate hardware units at each approximation level, and (iii) total latency constraint $L$. The scheduling problem is to find the starting time ($t_i$) and the latency ($c_i$) of each node that *minimizes the error rate of the final node*, with respect to the constraints formulated in what follows.

**Scheduling constraints:** Since a single hardware unit is available per operation type, an arbitrary pair of nodes of the same operator type should not be overlapped in execution time. That is, $\forall i, j$ such that $type_i = type_j$,

$$\begin{cases} t_i + c_i \leqslant t_j & \text{if } pr_{i,j} = 1 \\ t_j + c_j \leqslant t_i & \text{otherwise} \end{cases}, \tag{4.5}$$

where $pr_{i,j}$ is a binary variable that is 1 if and only if the execution of $v_i$ precedes that of $v_j$. In addition, the execution dependencies specified in $E$ should be satisfied in the scheduling decision. Thus, $\forall (v_i, v_j) \in E$, the

Table 4.1: Notation summary.

| Notation | Definition |
|---|---|
| $G$ | Application DAG $G = \langle V, E \rangle$ |
| $M_i$ | Set of approximation levels of Node $i$ |
| $P_i$ | Set of predecessors of Node $i$ |
| $L$ | Total latency constraint |
| $type_i$ | Operation type of Node $i$ |
| $mode_{i,k}$ | If 1, approximation level of Node $i$ is $k$; otherwise 0 |
| $pr_{i,j}$ | Nodes $i$ and $j$ share a hardware unit, and Node $i$ precedes $j$ |
| $d_i[k]$ | Latency of Node $i$ at approximation level $k$ |
| $\delta_i[k]$ | Error rate of Node $i$ at approximation level $k$ |
| $t_i$ | Starting time of Node $i$ |
| $c_i$ | Latency of Node $i$ |
| $\epsilon_i$ | Error rate of the output of Node $i$ |
| $i, j$ | Index of node |
| $k$ | Index of approximation level |

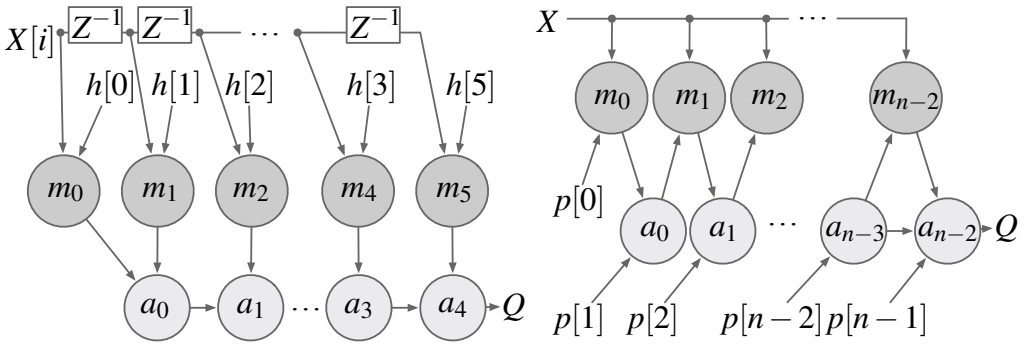
following inequality should hold:

$$t_i + c_i \leqslant t_j. \tag{4.6}$$

The latency of each node, $c_i$, is determined by the following equation, i.e., for each $v_i \in V$,

$$c_i = \sum_{k=1}^{|M_i|} mode_{i,k} \cdot d_i[k], \tag{4.7}$$

where $mode_{i,k}$ is a binary variable that is 1 if $v_i$ is chosen to operate at the $k$-th approximation level selected from $M_i$, a set of approximation levels of $v_i$, and $d_i[k]$ denotes the latency of $v_i$ at the $k$-th approximation




(a) $n$-tap FIR (b) $n$-th order PoE

Figure 4.5: Data dependency as a DAG diagram for (a) n-tap FIR and (b) n-th order PoE.

Table 4.2: Benchmarks used for evaluation.

| Benchmark | FIR-5 | FIR-16 | FIR-32 | PoE-6 | PoE-16 | PoE-32 |
|---|---|---|---|---|---|---|
| Adders | 5 | 16 | 33 | 5 | 15 | 31 |
| Multipliers | 6 | 17 | 32 | 5 | 15 | 31 |
| Min. latency | 7 | 18 | 34 | 10 | 30 | 62 |
| Max. latency | 13 | 34 | 65 | 15 | 45 | 93 |
| No. of solutions | 64 | 131,072 | >8M | 32 | 32,768 | >2M |

level. Finally, each node has a single approximation level chosen, i.e., for all $v_i \in V$.

$$\sum_{k=1}^{|M_i|} mode_{i,k} = 1. \tag{4.8}$$

**Total latency constraint:** A valid scheduling solution should always be completed by the given total latency constraint L, i.e.,

$$\max_{v_i \in V}(t_i + c_i) \leqslant L. \tag{4.9}$$

**Error constraints:** The error rate of an approximate operator node is determined based on the errors transferred from input operands and the error generated from the node itself as shown in (4.3). In order to keep the linearity of the formulation, both left and right sides of (4.3) are squared:

$$\epsilon_i{}^2 = \sum_{\forall v_j \in P_i} \epsilon_j{}^2 + \sum_{k=1}^{|M_i|} mode_{i,k} \cdot \delta_i[k]^2, \tag{4.10}$$

where $P_i$ denotes a set of predecessor nodes of $v_i$, i.e., $\forall v_j \in P_i$, $\exists (v_j, v_i) \in E$. Note that this linearization (squared error rate representation) does not jeopardize the optimality of the solution as $\epsilon$ is non-negative, and minimizing $\epsilon$ is equivalent to minimizing $\epsilon^2$.

**ILP objective function:** In this project, we considered the total latency of the application as an optimization constraint and minimized the final output error as the objective function. This formulation is easily transferable to the case where the maximum error at the final output is a constraint, and we solve the problem to minimize the overall latency or energy consumption of the benchmark.

### Experimental Setup

We consider two types of benchmark applications: (i) finite impulse response (FIR) filters applied to electrocardiogram (ECG) signals and (ii) polynomial evaluation (PoE) applied to normally distributed random inputs. Figure 4.5 illustrates the dependency in a $n$-tap FIR and a $n$-th order PoE benchmark in a DAG. Table 4.2 shows the number of addition and multiplication operations in the benchmarks and the number of all possible scheduling solutions. The 16-bit TIM multiplier Ahmed and Srinivas (2019) is used for approximate multiplication. Its average accuracy for uniformly distributed inputs is 94.77% in the first cycle and 98.45%

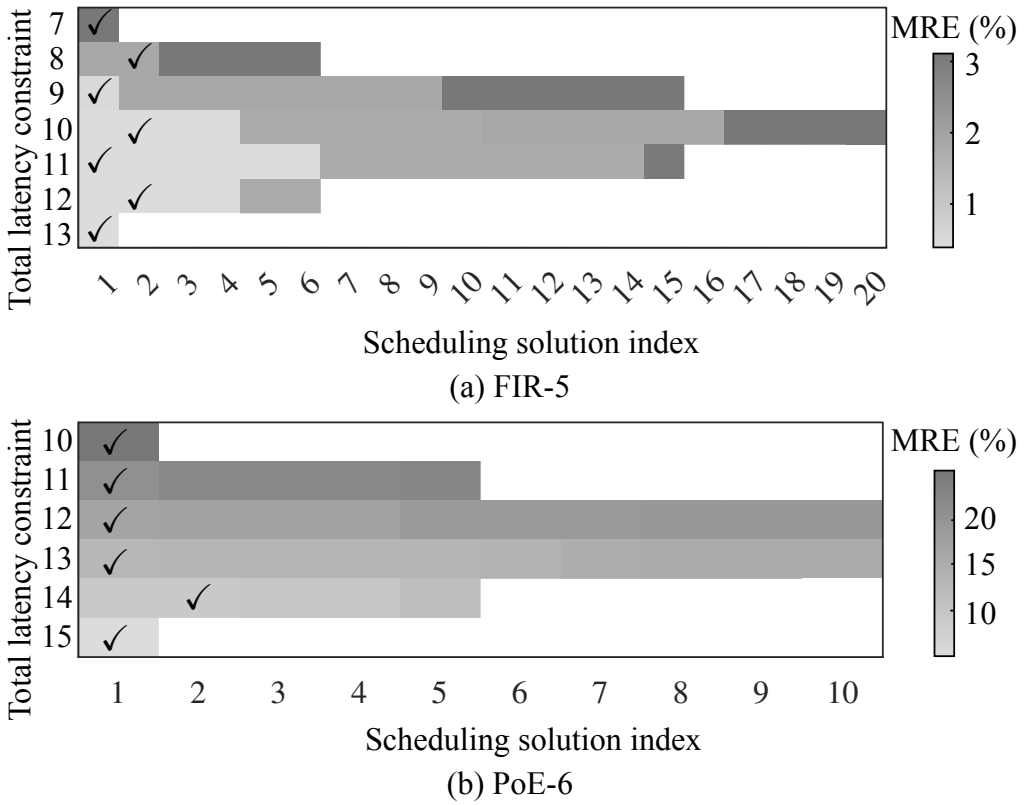



Figure 4.6: MRE of all possible scheduling solutions of (a) FIR-5 and (b) PoE-6. The proposed scheduler's solutions are highlighted with checkmarks (✓).

in the second cycle. For the accuracy evaluation, we compare the mean relative error (MRE) of the schedules obtained by the proposed scheduler and the true optimal solution obtained by exhaustive search, or decision tree-based optimization if the exhaustive search is not tractable due to the large number of possible solutions. As mentioned in Section **??**, we use these generic optimization methods as the baselines since there exists no prior optimization method for this problem. The execution time is measured on a PC with the 3.4 GHz quad-core Intel Core i5 CPU with 16 GB of RAM. We use Gurobi **?** as the ILP solver.

**Optimality of Scheduling Solutions**

We first evaluate the proposed ILP-based scheduler by comparing the accuracy of the scheduling solutions obtained by ILP to that of all possible scheduling solutions. Because of the sheer number of the all possible solutions, exhaustive search is not tractable even for medium-size benchmarks, thus we use the smallest benchmarks for demonstration. Figures 4.6(a) and 4.6(b) show the accuracy of all scheduling solutions of FIR-5 and PoE-6, respectively. Each cell represents one scheduling solution. Each row is a set of solutions with the same latency, sorted by accuracy in ascending order, ranging from the minimum (all multipliers run for one cycle) to the maximum (all multipliers run for two cycles). The left-most solution is the optimal solution for each latency constraint. It exhibits a wide variation in accuracy between scheduling solutions, not only by latency but also within the same latency. For each latency constraint, cells highlighted by checkmarks (✓) represent the scheduling solutions obtained by the proposed scheduler. The results show that the proposed scheduler is able to find the optimal solutions in most latency constraints in both benchmarks, and even the solutions that are not optimal are still as almost accurate as the optimal.

Next, we evaluate the accuracy for larger benchmarks in Figure 4.7. We use an exhaustive search as the baseline for FIR-5 and PoE-6 that are small, and decision tree for others. We compare the proposed scheduler's solution to the best-case and worst-case schedules obtained by the baseline schedulers. For FIR-5 and PoE-6, the proposed scheduler successfully finds the optimal or near-optimal solutions in all cases, which is also seen in Figure 4.6. The proposed scheduler solutions also achieve good accuracy for larger benchmarks as well at a much lower execution time as will be shown in Section 4.2.

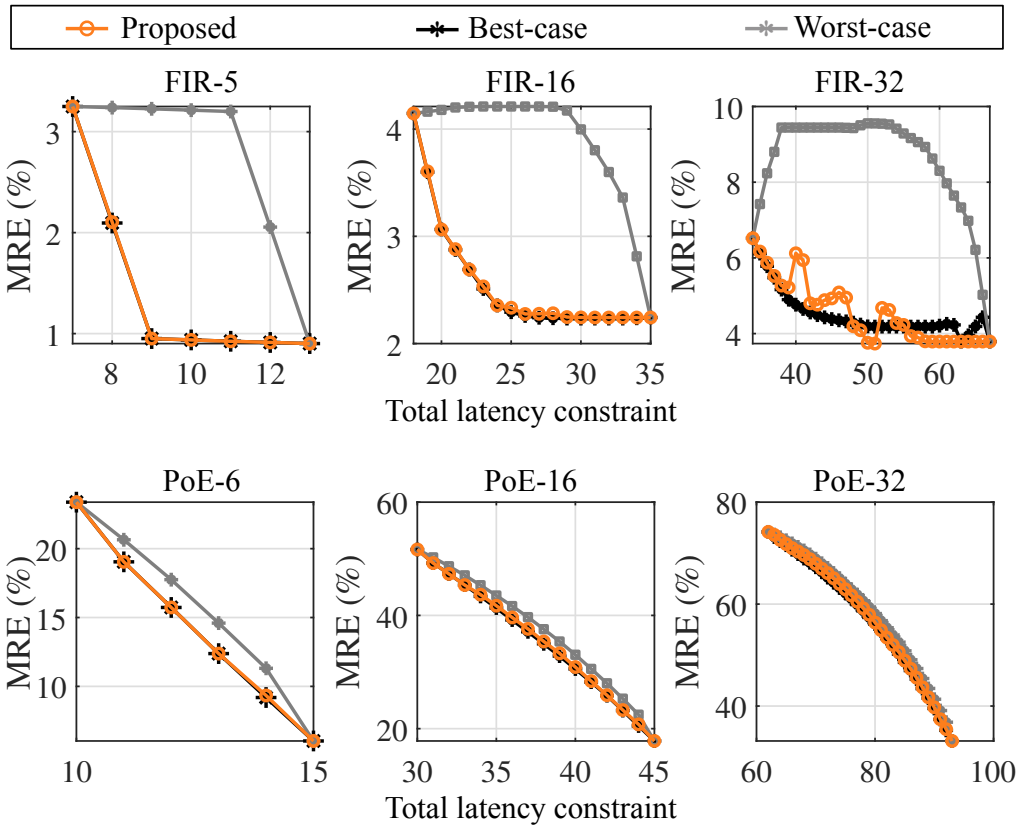



Figure 4.7: Comparison of proposed scheduling with baselines. Exhaustive search is used as baselines for FIR-5 and PoE-6; decision tree for FIR-16, FIR-32, PoE-16, and PoE-32.

### Optimality of Scheduling Solutions under Input Variation

Error modeling in Section 4.2 relies on the distribution of input known at design time, but the actual input at run time may be different and may have a different optimal schedule. To investigate the impact, we evaluate the optimality of scheduling solutions that are optimal for one dataset when applied to another dataset. We use two sets of data: (i) a train dataset, based on which a scheduling solution is optimized, and (ii) a test dataset, to which the solution is applied to. For FIR benchmarks, we use

Table 4.3: Number of same latencies in the optimal schedules for train and test datasets, and application accuracy degradation of scheduling under input variation.

| Benchmark | FIR-5 | FIR-16 | FIR-32 | PoE-6 | PoE-16 | PoE-32 |
|---|---|---|---|---|---|---|
| Same latencies | 2/6 | 18/18 | 31/34 | 4/6 | 2/16 | 2/32 |
| MRE increase | 0.13% | 0.00% | 1.29% | 1.03% | 0.57% | 0.31% |

two different sets of ECG signals as the train and test datasets. For PoE benchmarks, we use two sets of normally distributed random numbers with different mean values ($\mathcal{N}(0, 2^8)$ and $\mathcal{N}(2^4, 2^8)$). We find the optimal scheduling solution for the test dataset, and if it is different from that of the train dataset, we compare the accuracy by both scheduling solutions. All benchmarks result in the same or only slightly different optimal solutions as shown in Table 4.3 in spite of the different input distributions and error models. The results confirm the need for input-aware error modeling and also demonstrate the robustness of scheduling solutions under input variation.

**Scheduling Optimization Execution Time**

Finally, we evaluate the execution time of the proposed scheduling method against the measured or estimated execution time of other scheduling methods. As shown in Table 4.4, using the exhaustive search or decision tree-based optimization is not tractable due to the large number of possible solutions. The proposed scheduling method dramatically reduces the execution time by more than a billion times for the complex FIR-32 and PoE-32 benchmarks, making the scheduling of approximate hardware units feasible for complex applications.

Table 4.4: Execution time in seconds. (†: estimated from time for evaluating one solution and the number of solutions.)

| Benchmark | FIR-5 | FIR-16 | FIR-32 | PoE-6 | PoE-16 | PoE-32 |
|---|---|---|---|---|---|---|
| Exhaustive search | 2,760 | 4.0E6† | 4.9E11† | 47 | 1.5E5† | 1.5E10† |
| Decision tree | 861 | 14,455 | 37,169 | 23 | 730 | 15,672 |
| Proposed ILP | 0.1 | 1.3 | 19.4 | 0.1 | 0.8 | 5.1 |

**Conclusion**

An ILP-based scheduling is proposed to determine the optimal latency of iterative approximate hardware to fully exploit the energy-accuracy trade-off. The proposed scheduling method benefits from the novel data-driven error modeling of iterative computing hardware to ensure the optimality of the solutions. The presented error model reflects the effect of input variation on approximation accuracy to better assist with the scheduling. Based on the error modeling, we presented an ILP formulation for the scheduling problem. The evaluation results show that the proposed scheduler achieves optimal or near-optimal scheduling solutions in significantly less execution time compared to enumerating approaches such as exhaustive search or decision tree.

In this project, we showcased the efficiency of work on simple benchmarks such as FIR filet and PoE. We did not consider complicated applications, mainly due to the lack of low-level data dependencies. In other words, all the existing data dependencies for the benchmark are for the application level and there exists no data dependency at the level of translated code for a target physical architecture. One future work to help with approximate computing research is to implement a cycle-accurate simulator to generate a detailed log of application data dependencies for a target hardware architecture. The second challenge in this project was proposing an efficient error estimation. The efficiency of optimization methodology depends heavily on the efficiency of error estimation to calculate the

score of each solution. In this project, we considered uncertainty propagation rules. A future work aligned with this project would be an efficient methodology to estimate the error at a given node in the application.

# 5 FUTURE WORKS

## 5.1 Scheduling of quality-scalable neural network computation

Machine learning and neural networks (NNs) are the driving forces in the advancement of various applications. Demand for NN computation on embedded devices to perform recognition tasks such as voice recognition or predictions and more is ever increasing. NNs with larger model sizes and heavier computation are preferred from the functionality point of view. At the same time, embedded devices deal with limited resources such as computation, storage, and energy Kim et al. (2020); Amrouch et al. (2020). Approximate computation is a promising solution to run NNs on embedded devices Sarwar et al. (2018); Mrazek et al. (2019); Zervakis et al. (2021); Tasoulas et al. (2020); Venkataramani et al. (2020). Many approximate hardware architecture designs such as approximate adders and approximate multipliers have been proposed to increase the computation capability of embedded devices under various constraints such as energy and storage Kim et al. (2020).

Not all the computations within one application has the same accuracy requirement. This insight opens room to perform more approximations to increase the energy saving when the accuracy requirement allows. Based on this insight, many quality reconfigurable approximate hardware designs have been proposed to maximize energy saving. Despite the benefits of quality-scalable computing units, controlling such schemes has not been studied well. Scheduling the approximation knobs in quality scalable computing units poses a scalability challenge. First, due to the nature of the neural network application, the number of computations is huge. For example, Resenet-50 NN architecture has $120m$ multiplication operations. The number of computations goes higher with larger and more complex

NN architecture, which is the seen trend in NN advancements. Therefore proposing an online optimization to find the best approximation level for each arithmetic operation is challenging.

Prior works to determine the approximation level of reconfigurable hardware components resolved the scalability challenge by proposing coarse grain optimizations with the cost of less energy saving. Mrazek et al. (2019) proposed a layer-based optimization methodology where all computation of each layer has the same approximation level. Tasoulas et al. (2020); Zervakis et al. (2021) reduced the granularity to the weight ranges in their optimization methodology. Previous works have two main shortcomings, which limit their energy-saving capabilities. First, coarse grain approximation knob scheduling does not fully take advantage of reconfigurable hardware units for maximum energy saving. Second, their methodology relies on offline optimization rather than online approximation knob setting.

In this work, we propose a run-time approximate knob scheduling scheme with fine granularity of each multiplication in neural computation. At each multiplication, an efficient module estimates the produced error using the input combination. First, approximation error is compensated to some degree using the error estimation result. Then the scheduler can determine future operations' approximation level using the accumulated estimated error. Figure 5.1 shows the modified multiply and accumulate (MAC) hardware architecture enhanced with the proposed approximation level scheduler. In this figure, A and B are input to multiplier hardware, and Q is the output of the MAC unit. Modifications are first incorporating an efficient error predictor, which estimates the amount of error produced in the multiplier ($\epsilon$) based on the inputs. Second, the error accumulator accumulates the E, the total error produced in neuron computation ($\sum \epsilon$). And third, an accuracy level control scheme to set the accuracy level ($r$) of reconfigurable approximate multiplier. In the following sections, we

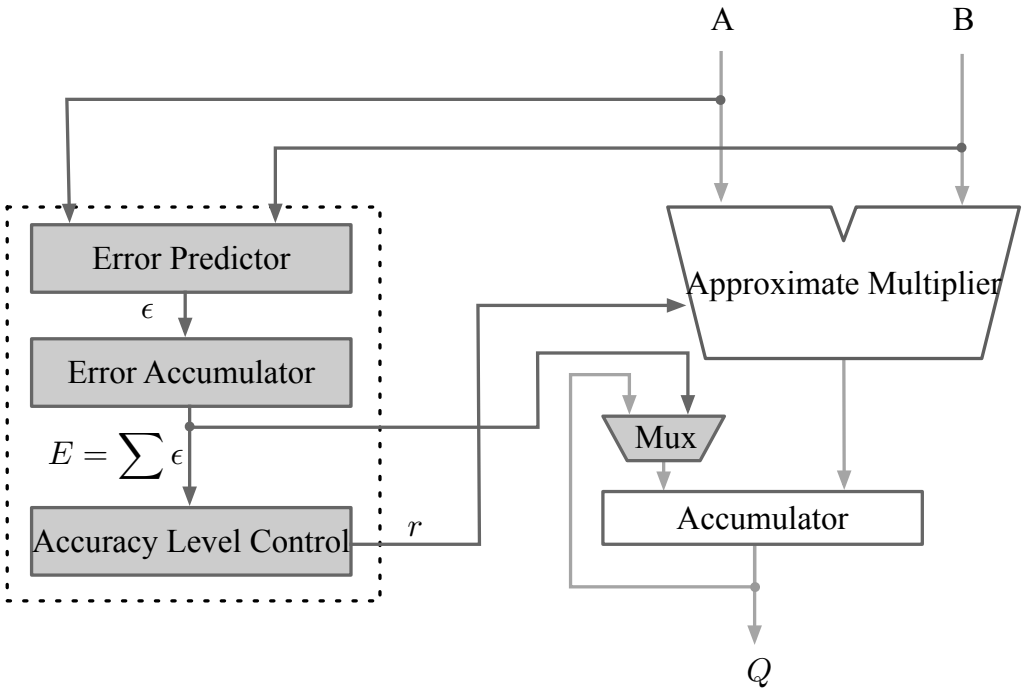



Figure 5.1: Hardware schematic of enhanced MAC unit with proposed approximation level scheduler.

introduce an efficient error predictor methodology based on a decision tree algorithm. Then we elaborate the run-time scheduler to set the approximation knob of reconfigurable multiplier. In the end, we present preliminary results evaluating the proposed scheme.

**Proposed Error Predictor**

The approximation error is input-dependent. This means that the magnitude of error varies by different input pairs. To be able to set the approximation level, we need to determine the amount of error the hardware is producing. The actual error, which is the golden reference, is not available at run time. Error estimation design needs to be power, area, and latency

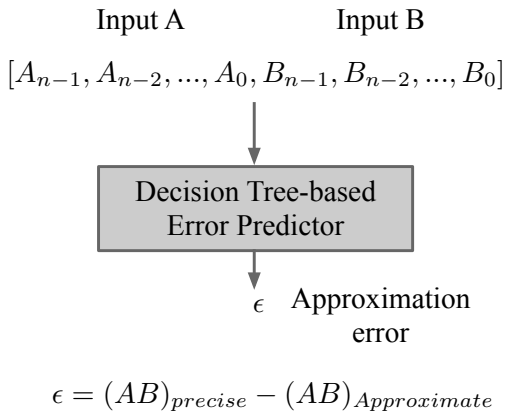



Figure 5.2: Decision-tree based error predictor overview.

efficient. Otherwise, it detracts from the energy saving achieved by approximation. In this project, we propose an efficient decision tree-based (DT-based) scheme to estimate the error shown in Figure 5.2.

We profiled the NN inputs to multiplication and count every input combination ($2^16$ for 8-bit quantized NN) for the inference tasks on a certain dataset. Then we generate a training dataset with the distribution of profiled data to train the DT. Input pair in binary format forms input features of proposed DT. Using the approximate multiplier model, we can train DT to estimate the multiplication error.

Preliminary evaluation shows that DTs as shallow as four levels could estimate the multiplication error with a negligible difference from the true error. To account for approximation techniques in the target approximate multiplier, we can omit the least significant bits of the input pair from the feature list. We use the predicted error in two folds. First, we compensate the approximation error in multiplication by adding the predicted error to the result. Then, we use the accumulated predicted error to determine the future approximation levels.

**Proposed Run-time Scheduler of Approximate Levels**

The major challenge in scheduling approximation levels for reconfigurable computing units is the scalability issue. Because the number of multiplication operations in a NN is the order of a hundred million. Prior works in this area resolved the approximation level scheduling problem by coarse grain optimizations Mrazek et al. (2019); Tasoulas et al. (2020); Zervakis et al. (2021) which results in loss of opportunity in energy saving.

Input dependence of approximation error complicates this optimization as well due to the lack of true reference for the error at optimization time. Previous scheduling techniques can estimate the NN accuracy by representative datasets in their optimization phase. However, due to reducing the design space size by coarse grain solutions, such works waste existing energy-saving opportunities. On the other hand, performing errors for fine-grain design space is not tractable due to a large number of possible solutions.

We can tackle the task of approximation knob in an offline way or at run-time. Offline scheduling techniques perform optimization on the representative datasets and deploy the achieved configuration at run-time. The benefit of this method is that the scheduling decision-making does not hurt the performance of NN execution with the cost of less energy saving in total.

To overcome the scalability challenge in optimization and input dependence of approximation error, we propose a run-time solution that performs error estimation in an efficient way at run-time and can adjust the approximation level on the fly based on the current dataset.

Figure 5.3 shows the accuracy level control scheme that performs the scheduling task. Considering that approximate multiplier has $n$ level of approximation, scheduler can select an accuracy level knob $r$ from set of $\{r_0, r_1, r_(n-1)\}$. With $r_0$ as most accurate configuration and $r_{n-1}$ as least accurate configuration. To perform the scheduling task on run-time, we

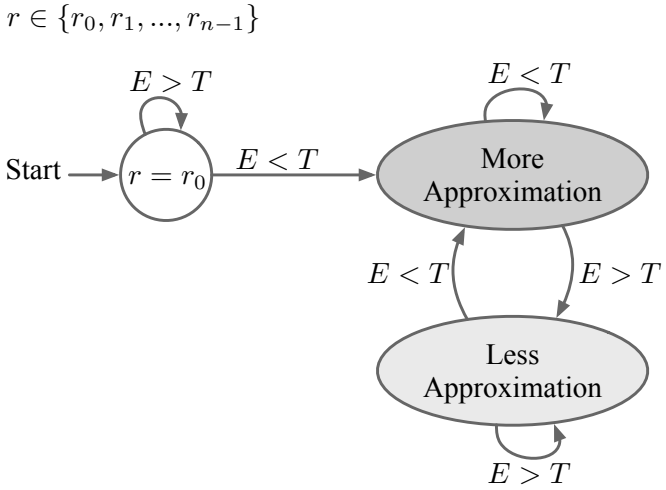



Figure 5.3: State machine of accuracy level controller.

consider T as an error threshold that will be tuned in the offline stage using a representative dataset. If the accumulated error is less than the error threshold, the scheduler keeps reducing the accuracy by a more aggressive approximation configuration for future operations until it exceeds the error threshold, where it stops and selects a more accurate configuration.

**Preliminary Experimental Results**

We demonstrate that the proposed error prediction method mostly compensates for the produced error due to multiplication approximation with negligible energy and area overhead.

**Experimental setup:** We used ADAPT Dimitrios et al. (2022), a DNN emulation framework that extends PyTorch Paszke et al. (2019) to support approximate inference. For preliminary evaluation, we used the MNIST dataset and a three-layer 8-bit quantized DNN architecture consisting of two convolutional layers and one fully connected layer to perform the inference task.

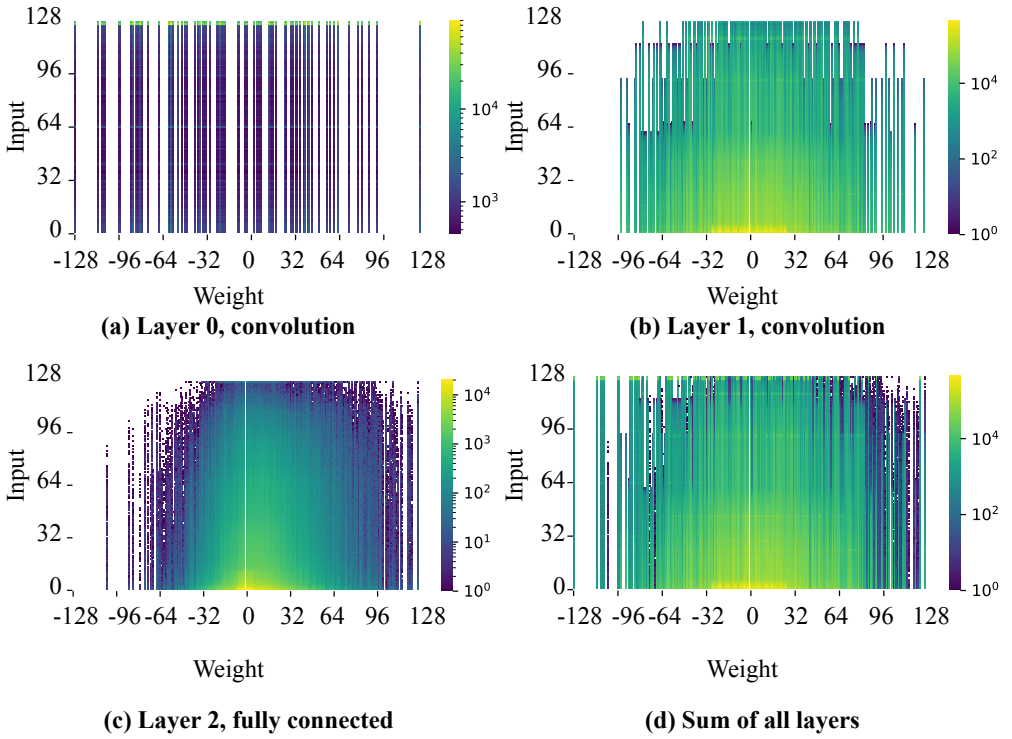



Figure 5.4: Distribution of multiplication input pair (NN input and weight) count for the MNIST dataset in (a) layer 0 (concolutional layer), (b) layer 1 (convolutional layer), and (c) layer 2 (fully connected layer).

**Distribution of input pairs:** We profiled the inference task on MNIST dataset on the target NN architecture. Figure 5.4 shows the distribution of multiplication input pair count for the MNIST dataset. This data excludes pairs containing value zero due to the fact that we need this profiled data to train the error predictor and pairs with zero values are not of interest. In the preliminary phase, we used the sum of all layer pair combination counts to train the DT because all layers had similar data distribution.

**Inference accuracy:** Using the profiled data distribution of multiplication input, we generated one million random pairs with the same distribution

Table 5.1: Evaluation of DNN inference accuracy using DT-based error predictor to compensate error of approximate multipliers. Inference accuracy using a precise multiplier is 98.28%.

| z | Mode | Energy saving | MRE (%) | Inference accuracy (%) | |
|---|---|---|---|---|---|
| | | | | w/o compensation | w compensation |
| 1 | PE | 8.30 | 31.19 | 98.37 | 98.30 |
| 2 | PE | 20.23 | 218.97 | 98.01 | 98.27 |
| 3 | PE | 36.66 | 969.00 | 81.36 | 98.00 |
| 1 | NE | 5.50 | 94.80 | 98.21 | 98.28 |
| 2 | NE | 16.17 | 285.05 | 97.76 | 98.28 |
| 3 | NE | 31.8 | 1047.10 | 68.33 | 98.26 |

to train the DT-based error predictor. DT with only four layers (16 leaves) is used for error compensation. Table 5.1 demonstrates the efficiency of DT-based error predictor in compensating the approximation error. In this experiment, we used a reconfigurable approximate multiplier proposed in Spantidi et al. (2021). This approximate multiplier is an approximated booth multiplier with two configuration knobs. $Mode$ knob selects the positive or negative error, and $z$ determines the approximation level. In all cases, the proposed error predictor could compensate for the approximation error successfully and close the gap between the precise multiplier and the approximate one with negligible reduction in energy saving.

**Overhead:** We implemented the approximate $8 \times 8$ multiplier enhanced with DT-based error predictor in Verilog and synthesized targeting the NanGate 45-nm CMOS Technology. The proposed DT-based error predictor with the depth of four (target preditor used in previous experiments) increases the power consumption and area of approximate multiplier $\sim 5\%$ and $\sim 11\%$, respectively.

# 6 CONCLUSION

Approximate computing is a new design paradigm for energy-efficient computing, which exploits intrinsic error resiliency for saving energy. Significant benefits in energy efficiency can be achieved by relaxing exactness requirement as long as the output meets a certain level of quality requirement. This notion of quality is highly application- and input-dependent and can vary over time, thus the underlying approximate components in a system should support flexible energy-quality scaling and provide a control knob for run-time configuration.

In this proposal, we attempt to answer the demand for energy-quality scalable components by proposing several run-time quality configurable components. The proposed schemes in addition to exiting quality-scalable techniques for the rest of the system components offer a great potential for energy savings for various emerging error-resilient applications, such as machine learning, yet their adoption is largely limited to isolated applications, rather than the integrated application of multiple techniques to achieve the system-level optimality. To take full advantage of energy-quality scalability across the system, we proposed a full-system design framework to efficiently handle the increased design complexity due to the new added dimension in design, i.e. quality.

Advanced Micro Devices (AMD). 2011. *Software optimization guide for amd family 10h and 12h processors*.

Ahmed, Syed Ershad, and MB Srinivas. 2019. An improved logarithmic multiplier for media processing. *Journal of Signal Processing Systems* 91(6): 561–574.

Alioto, M. 2017. Energy-quality scalable adaptive VLSI circuits and systems beyond approximate computing. In *Design, automation test in europe conference exhibition (date)*, 127–132.

Alioto, Massimo, Vivek De, and Andrea Marongiu. 2018. Energy-quality scalable integrated circuits and systems: Continuing energy scaling in the twilight of moore's law. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8(4):653–678.

Altera. 2016. *Floating-point ip cores user guide*.

Amrouch, Hussam, Georgios Zervakis, Sami Salamin, Hammam Kattan, Iraklis Anagnostopoulos, and Jörg Henkel. 2020. Npu thermal management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39(11):3842–3855.

Babić, Zdenka, Aleksej Avramović, and Patricio Bulić. 2011. An iterative logarithmic multiplier. *Microprocessors and Microsystems* 35(1):23–33.

Behroozi, Setareh, Jingjie Li, Jackson Melchert, and Younghyun Kim. 2019. SAADI: A scalable accuracy approximate divider for dynamic energy-quality scaling. In *Asia and south pacific design automation conference (asp-dac)*, 481–486.

Behroozi, Setareh, Vijay Raghunathan, Anand Raghunathan, and Younghyun Kim. 2018. A quality-configurable approximate serial bus

for energy-efficient sensory data transfer. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8(3):379–390.

Behroozi, Setareh, Yao Yao, Hoeseok Yang, and Younghyun Kim. 2021. Scheduling of iterative computing hardware units for accuracy and energy efficiency. In *2021 ieee international symposium on circuits and systems (iscas)*, 1–5. IEEE.

Benini, Luca, Giovanni de Micheli, Enrico Macii, Donatella Sciuto, and Cristina Silvano. 1997. Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems. In *Proceedings of the acm great lakes symposium on vlsi (glsvlsi)*, 77–82.

Bhardwaj, K., P. S. Mane, and J. Henkel. 2014. Power- and area-efficient approximate Wallace tree multiplier for error-resilient systems. In *International symposium on quality electronic design (isqed)*, 263–269.

Bocca, Alberto, Sabino Salerno, Enrico Macii, and Massimo Poncino. 2004. Energy-efficient bus encoding for LCD displays. In *Proceedings of the acm great lakes symposium on vlsi (glsvlsi)*, 240–243.

Chippa, V. K., et al. 2013a. Approximate computing: An integrated hardware approach. In *Acssc*.

Chippa, Vinay K, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013b. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th annual design automation conference*, 1–9.

Dimitrios, Danopoulos, Zervakis Georgios, Siozios Kostas, Soudris Dimitrios, and Henkel Jörg. 2022. Adapt: Fast emulation of approximate dnn accelerators in pytorch. *arXiv preprint: 2203.04071*.

Ghosh, Somrita, Prasun Ghosal, Nabanita Das, Saraju P. Mohanty, and Oghenekarho Okobiah. 2014. Data correlation aware serial encoding for

low switching power on-chip communication. In *Proceedings of the ieee computer society annual symposium on vlsi (isvlsi)*, 124–129.

Gupta, V., D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy. 2011. IMPACT: Imprecise adders for low-power approximate computing. In *Ieee/acm international symposium on low power electronics and design (islped)*, 409–414.

Han, J., et al. 2013. Approximate computing: An emerging paradigm for energy-efficient design. In *Ets*.

Han, Song, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. Eie: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44(3):243–254.

Han, Song, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Hanif, Muhammad Abdullah, Rehan Hafiz, Osman Hasan, and Muhammad Shafique. 2020. PEMACx: A probabilistic error analysis methodology for adders with cascaded approximate units. In *Proceedings of the design automation conference (dac)*, 1–6.

Hashemi, S., et al. 2016. A low-power dynamic divider for approximate applications. In *Dac*.

Hashemi, Soheil, R. Iris Bahar, and Sherief Reda. 2015. DRUM: a dynamic range unbiased multiplier for approximate applications. In *Ieee/acm international conference on computer-aided design (iccad)*, 418–425.

Hughes, Ifan, and Thomas Hase. 2010. *Measurements and their uncertainties: a practical guide to modern error analysis*. Oxford University Press.

Jayakumar, Hrishikesh, Kangwoo Lee, Woo Suk Lee, Arnab Raha, Younghyun Kim, and Vijay Raghunathan. 2014. Powering the Internet of Things. In *Ieee/acm international symposium on low power electronics and design (islped)*, 375–380.

Jégou, Hervé, Matthijs Douze, and Cordelia Schmid. 2008. Hamming embedding and weak geometry consistency for large scale image search-extended version. Tech. Rep.

Jiang, H., et al. 2018a. Adaptive approximation in arithmetic circuits: A low-power unsigned divider design. In *Date*.

Jiang, Honglan, Leibo Liu, Fabrizio Lombardi, and Jie Han. 2018b. Adaptive approximation in arithmetic circuits: A low-power unsigned divider design. In *2018 design, automation & test in europe conference & exhibition (date)*, 1411–1416. IEEE.

Kahng, Andrew B., and Seokhyeong Kang. 2012. Accuracy-configurable adder for approximate arithmetic designs. In *Design automation conference (dac)*, 820–825.

Kemp, Taylor, Yao Yao, and Younghyun Kim. 2021. MIPAC: Dynamic input-aware accuracy control for dynamic auto-tuning of iterative approximate computing. In *Proceedings of the asia and south pacific design automation conference (asp-dac)*, 248–253.

Kim, Younghyun, Setareh Behroozi, Vijay Raghunathan, and Anand Raghunathan. 2017. AxSerBus: A quality-configurable approximate serial bus for energy-efficient sensing. In *Ieee/acm international symposium on low power electronics and design (islped)*, 1–6.

Kim, Younghyun, Joshua San Miguel, Setareh Behroozi, Tianen Chen, Kyuin Lee, Yongwoo Lee, Jingjie Li, and Di Wu. 2020. Approximate hardware techniques for energy-quality scaling across the system. In

*2020 international conference on electronics, information, and communication (iceic)*, 1–5. IEEE.

Lee, Kangmin, Se-Joong Lee, and Hoi-Jun Yoo. 2004. SILENT: Serialized low energy transmission coding for on-chip interconnection networks. In *Proceedings of the ieee/acm international conference on computer-aided design (iccad)*, 448–451.

Li, Chaofan, Wei Luo, Sachin S Sapatnekar, and Jiang Hu. 2015. Joint precision optimization and high level synthesis for approximate computing. In *Proceedings of the 52nd annual design automation conference*, 1–6.

Lin, C., and I. Lin. 2013. High accuracy approximate multiplier with error correction. In *Ieee international conference on computer design (iccd)*, 33–38.

Liu, Cong, Jie Han, and Fabrizio Lombardi. 2014. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *Design, automation & test in europe conference & exhibition (date)*, 1–4.

Liu, Song, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G Zorn. 2011. Flikker: Saving dram refresh-power through critical data partitioning. In *Proceedings of the sixteenth international conference on architectural support for programming languages and operating systems*, 213–224.

Melchert, Jackson, Setareh Behroozi, Jingjie Li, and Younghyun Kim. 2019. SAADI-EC: A quality-configurable approximate divider for energy efficiency. *IEEE Transactions on VLSI Systems* 27(11):2680–2692.

Mitchell, John N. 1962. Computer multiplication and division using binary logarithms. *IRE Transactions on Electronic Computers* (4):512–517.

Mittal, Sparsh. 2016. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)* 48(4):1–33.

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

Mrazek, V., R. Hrbacek, Z. Vasicek, and L. Sekanina. 2017. EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, automation & test in europe conference & exhibition (date)*, 258–261.

Mrazek, Vojtech, Zdenek Vasícek, Lukás Sekanina, Muhammad Abdullah Hanif, and Muhammad Shafique. 2019. Alwann: automatic layer-wise approximation of deep neural network accelerators without retraining. In *2019 ieee/acm international conference on computer-aided design (iccad)*, 1–8. IEEE.

Nilsson, P., A. U. R. Shaik, R. Gangarajaiah, and E. Hertz. 2014. Hardware implementation of the exponential function using Taylor series. In *Ieee nordic circuits and systems conference (norchip)*, 1–4.

Oberman, S. F., et al. 1997. Division algorithms and implementations. *IEEE Transactions on computers* 46(8).

Pagliari, Daniele Jahier, Enrico Macii, and Massimo Poncino. 2016a. Approximate differential encoding for energy-efficient serial communication. In *Proceedings of the acm great lakes symposium on vlsi (glsvlsi)*, 421–426.

———. 2016b. Serial T0: Approximate bus encoding for energy-efficient transmission of sensor signals. In *Design automation conference (dac)*, 1–6.

———. 2017. Approximate energy-efficient encoding for serial interfaces. *ACM ACM Transactions on Design Automation of Electronic Systems* 22(4): 64:1–64:25.

Park, Jongsun, Hunsoo Choo, Khurram Muhammad, SeungHoon Choi, Yonghee Im, and Kaushik Roy. 2000. Non-adaptive and adaptive filter

implementation based on sharing multiplication. In *2000 ieee international conference on acoustics, speech, and signal processing. proceedings (cat. no. 00ch37100)*, vol. 1, 460–463. IEEE.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32*, ed. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, 8024–8035. Curran Associates, Inc.

Raha, A., and V. Raghunathan. 2017. Towards full-system energy-accuracy tradeoffs: A case study of an approximate smart camera system. In *Dac*.

Saadat, Hassaan, Haris Javaid, Aleksandar Ignjatovic, and Sri Parameswaran. 2020. WEID: Worst-case error improvement in approximate dividers. In *Proceedings of the asia and south pacific design automation conference (asp-dac)*, 593–598.

Saadat, Hassaan, Haris Javaid, and Sri Parameswaran. 2019. Approximate integer and floating-point dividers with near-zero error bias. In *Design automation conference (dac)*, 161:1–161:6.

Sampson, Adrian, Jacob Nelson, Karin Strauss, and Luis Ceze. 2014. Approximate storage in solid-state memories. *ACM Transactions on Computer Systems (TOCS)* 32(3):1–23.

Sarwar, Syed Shakib, Swagath Venkataramani, Aayush Ankit, Anand Raghunathan, and Kaushik Roy. 2018. Energy-efficient neural computing

with approximate multipliers. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 14(2):1–23.

Shin, Doochul, and Sandeep K Gupta. 2010. Approximate logic synthesis for error tolerant applications. In *2010 design, automation & test in europe conference & exhibition (date 2010)*, 957–960. IEEE.

Shoushtari, Majid, Abbas BanaiyanMofrad, and Nikil Dutt. 2015. Exploiting partially-forgetful memories for approximate computing. *IEEE Embedded Systems Letters* 7(1):19–22.

Spantidi, Ourania, Georgios Zervakis, Iraklis Anagnostopoulos, Hussam Amrouch, and Jörg Henkel. 2021. Positive/negative approximate multipliers for dnn accelerators. In *2021 ieee/acm international conference on computer aided design (iccad)*, 1–9. IEEE.

Stanley-Marbell, Phillip, and Martin Rinard. 2015. Lax: Driver interfaces for approximate sensor device access. In *15th workshop on hot topics in operating systems (hotos xv)*.

———. 2016. Reducing serial I/O power in error-tolerant applications by efficient lossy encoding. In *Proceedings of the design automation conference (dac)*, 62:1–62:6.

———. 2020. Warp: A hardware platform for efficient multimodal sensing with adaptive approximation. *IEEE Micro* 40(1):57–66.

Sullivan, M. B., and E. E. Swartzlander. 2013. Truncated logarithmic approximation. In *Ieee symposium on computer arithmetic (arith)*, 191–198.

Tasoulas, Zois-Gerasimos, Georgios Zervakis, Iraklis Anagnostopoulos, Hussam Amrouch, and Jörg Henkel. 2020. Weight-oriented approximation for energy-efficient neural network inference accelerators. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67(12):4670–4683.

Vaeztourshizi, Marzieh, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2018. An energy-efficient, yet highly-accurate, approximate non-iterative divider. In *Proceedings of acm/ieee international symposium on low power electronics and design (islped)*, 14:1–14:6.

Vahdat, S., et al. 2017. TruncApp: A truncation-based approximate divider for energy efficient dsp applications. In *Date*.

Venkataramani, S., et al. 2015a. Approximate computing and the quest for computing efficiency. In *Dac*.

Venkataramani, Swagath, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. 2015b. Approximate computing and the quest for computing efficiency. In *2015 52nd acm/edac/ieee design automation conference (dac)*, 1–6. IEEE.

Venkataramani, Swagath, Xiao Sun, Naigang Wang, Chia-Yu Chen, Jungwook Choi, Mingu Kang, Ankur Agarwal, Jinwook Oh, Shubham Jain, Tina Babinsky, et al. 2020. Efficient ai system design with cross-layer approximate computing. *Proceedings of the IEEE* 108(12):2232–2250.

Weirich, M. R., G. Paim, E. A. C. da Costa, and S. Bampi. 2018. A fixed-point natural logarithm approximation hardware design using Taylor series. In *New generation of circuits & systems conference (ngcas)*, 53–56.

Wu, D., T. Chen, C. Chen, O. Ahia, J. San Miguel, M. Lipasti, and Y. Kim. 2019. SECO: A scalable accuracy approximate exponential function via cross-layer optimization. In *Ieee/acm international symposium on low power electronics and design (islped)*, 1–6.

Wu, L., et al. 2015. A curve fitting approach for non-iterative divider design with accuracy and performance trade-off. In *Newcas*.

Yang, Z., A. Jain, J. Liang, J. Han, and F. Lombardi. 2013. Approximate XOR/XNOR-based adders for inexact computing. In *Ieee international conference on nanotechnology (nano)*, 690–693.

Ye, R., T. Wang, F. Yuan, R. Kumar, and Q. Xu. 2013. On reconfiguration-oriented approximate adder design and its application. In *Ieee/acm international conference on computer-aided design (iccad)*, 48–54.

Zendegani, R., et al. 2016. SEERAD: A high speed yet energy-efficient rounding-based approximate divider. In *Date*.

Zervakis, Georgios, Ourania Spantidi, Iraklis Anagnostopoulos, Hussam Amrouch, and Jörg Henkel. 2021. Control variate approximation for dnn accelerators. In *2021 58th acm/ieee design automation conference (dac)*, 481–486. IEEE.