Using Approximate Computing Circuits to Optimize Power of an ASIC

SHI-TIEN HSING
PADMASHREE NUGGEHALLI SRINIVASA
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Using Approximate Computing Circuits to Optimize Power of an ASIC

Shi-Tien Hsing
ti1246hs-s@student.lu.se

Padmashree Nuggehalli Srinivasa
pa1565nu-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Academic Supervisor: Liang Liu
Senior Lecturer, EIT LTH liang.liu@eit.lth.se

Supervisor: Anders Lloyd, Axis Communication
Mikael Arnfelt, Axis Communication

Examiner: Erik Larsson

July 3, 2023

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

The growing demand for network cameras to support real-time image processing and machine-learning applications has created a need for low-power solutions. Although technology scaling makes complex computations feasible, voltage scaling is limited, leading to higher power density and dark silicon problems. One potential solution is the use of Approximate Arithmetic Circuits (AACs). This effectively reduces the number of logic gates required to perform arithmetic computations on ASICs. This technique is particularly suitable for image applications because the human eye's perceptual tolerance makes a degree of error admissible.

In this thesis, the Lanczos scaling is selected as the image application to evaluate the feasibility of applying AACs to trade off accuracy for power. To quantitatively evaluate the impact on image quality introduced by AAC errors, arithmetic accuracy metrics and image quality metrics are studied to find a correlation between them. Furthermore, power simulations are conducted on AACs using sub-10 nm technology to validate the power savings the chosen fabrication node achieves. Finally, the exact multipliers in the Lanczos scaling hardware are replaced with a selection of AACs, and then the system-level power consumption is assessed when scaling actual images.

The outcome of this study shows that the image scaling application produces a power saving exceeding 50% while maintaining a high Structural Similarity Index Metric (SSIM) of up to 0.9. This finding contributes to understanding the potential of an AAC in reducing power consumption in image processing circuits, paving the way for future advancements in approximate computing techniques.

Acknowledgments

We would like to express our deep gratitude to our industrial supervisors, Anders Lloyd and Mikael Arnfelt, for their exceptional guidance and expertise and for providing us with the opportunity to undertake this thesis. Their inspiring insights and assistance have played a vital role in our research.

We would also like to express our heartfelt gratitude to our academic supervisors, Liang Liu and Mohammad Attari, for their valuable suggestions and rigorous attitude in doing research. Their guidance and encouragement have been vital throughout the entire process.

We extend our sincere thanks to Axis Communications in Lund, for their support in the chosen thesis topic and for providing us with the necessary resources to carry out this project. Additionally, we would like to thank all the professors at the Faculty of Engineering (LTH), Lund University, for equipping us with the knowledge and skills needed to conduct this thesis.

Our deepest gratitude also goes to our families, partners, and friends for their unwavering support, encouragement, and friendship throughout our entire journey of studying and researching for this thesis.

Finally, we express our gratitude to each other for being excellent thesis partners. Together, we have provided inspiration, companionship, and laughter, making this journey less solitary and more enjoyable.

Popular Science Summary

In today's technology-driven world, images surround us in various forms, from social media posts to security camera feeds. Behind the scenes, powerful computational processes work tirelessly to enhance and analyze these images in real time. However, this quest for image perfection comes at a cost - power consumption. As our appetite for image processing grows, so does the need for power-efficient solutions to curb the skyrocketing power demands.

In the pursuit of power efficiency, engineers are faced with a conundrum. While technology advancements enable complex computations, scaling down voltage or frequency to reduce power is increasingly challenging. This has led to the emergence of Approximate Arithmetic Circuits (AACs), prioritizing power efficiency over absolute accuracy. By introducing controlled errors into arithmetic computations, AACs significantly reduce the logic gates needed for image processing while maintaining perceptible image quality. This breakthrough opens new possibilities for power-conscious image applications.

In this thesis, meticulous studies are conducted to assess the impact of approximation on image quality and power consumption by integrating AACs in image resizing techniques, such as the widely used Lanczos scaling. By exploring the correlation between arithmetic accuracy metrics and image quality metrics, optimal power savings can be achieved while preserving visual fidelity.

AACs offer a promising solution to address the power challenges in image processing. By leveraging the innate error tolerance of human perception, approximate circuits pave the way for greener and more sustainable image-processing technologies. With AACs, we can build power-efficient systems that meet the ever-growing demands of image applications while reducing power consumption. This research sets the stage for future advancements in approximate computing techniques, shaping a world where power efficiency and innovation go hand in hand.

Table of Contents

List of Figures

List of Tables

Abbrevations

AAC Approximate Arithmetic Circuit

ALU Arithmetic Logic Unit

ASM Application Specific Metric

CLA Carry-Lookahead Adder

CMOS Complementary Metal-Oxide-Semiconductor

CNN Convolutional Neural Network

CSA Carry-Select Adder

EAC Exact Arithmetic Circuit

ED Error Distance

EP Error Probability

ER Error Rate

ESA Equal Segmentation Adder

FA Full Adder

FSDB Fast Signal Database

HDL Hardware Description Language

HVS Human Visual System

LVT Low Voltage Threshold

MAC Multiply-Accumulate

MAE Mean Absolute Error

MED Mean Error Distance

MRE Mean Relative Error

MSE Mean Square Error

MSSIM Mean Structural Similarity Index Metric

NMED Normalization of Mean Error Distance

PE Processing Element

PSNR Peak Signal to Noise Ratio

RCA Ripple Carry Adder

RED Relative Error Distance

RTL Register-Transfer Level

SSIM Structural Similarity Index Metric

UDM Under-Designed Multiplier

WCE Worst Case Error

Chapter 1
Introduction

With the ever-increasing demand for network cameras to support real-time image processing and machine-learning applications, low-power solutions are needed. As computational demands increase, more sophisticated computations are performed on a single chip, leading to higher power density and heating issues in the integrated circuits. A percentage of transistors on the chip may be powered off to alleviate these thermal issues in network cameras. However, this approach can also lead to performance degradation, as powered-off transistors may be needed for certain computations. Thus, finding a balance between reducing power consumption and maintaining performance remains a significant challenge in developing low-power solutions for network cameras that require real-time image processing and machine learning.

During our pilot investigation into an image scaling application with a set-up similar to our thesis, we conducted circuit-level power simulations. The results revealed that the consumption of dynamic power constitutes about 90% of the total power consumption, exceeding the importance of static power. This significant proportion of dynamic power consumption is attributed to the frequent switching activity of transistors during computations, particularly in adders and multipliers, which are fundamental constituents of a Arithmetic Logic Unit (ALU).

To resolve the high dynamic power consumption of ALUs, many researchers have considered using Approximate Arithmetic Circuit (AAC) for image processing applications with the error resistance property [1] [2]. The inherent limitations in human perception allow certain errors that do not significantly degrade the output quality of images. Moreover, real-world applications often involve noisy inputs, which inherently pose challenges in accurately representing useful information with precision [3]. Using system-level error tolerance, controlled inaccuracies in computations can be introduced using AAC, reducing power consumption while maintaining acceptable output quality.

The feasibility and effectiveness of AACs depend on various factors, including specific system requirements, circuit-level design choices, and the characteristics of the fabrication technology used. Evaluating error metrics plays a crucial role in identifying the error characteristics of AACs and helps to select appropriate designs for specific applications [4]. Additionally, researchers should consider how AACs affect the resulting image quality, ensuring that acceptable levels of accuracy are

maintained for the specific application. Furthermore, gate-level logic optimization of the design is highly dependent on standard cell technology libraries, resulting in a substantial impact on power consumption.

Thorough analysis and understanding of these inter-dependencies are essential to ensure that AACs effectively reduce power consumption while maintaining acceptable levels of accuracy for the target application.

1.1 Aim of the Thesis

This thesis explores the design space of utilizing AACs in hardware to reduce power consumption for a specific image scaling algorithm and semiconductor technology node. The investigation encompasses multiple tasks, including 1. evaluating the image quality from an algorithmic perspective, 2. analyzing power reduction at the circuit level, and 3. evaluating the effectiveness of power reduction using a given fabrication technology.

From an algorithmic perspective, the thesis uses the Lanczos resampling as an image scaling algorithm to assess the error tolerance and maintain satisfactory image quality. Subsection 2.4.1 provides additional explanations on the rationale behind this choice of application. Different error metrics and colour spaces of the input images, as well as hardware adaptations, are investigated to determine the impact on the quality of the output image. Furthermore, an error visualization approach is presented to aid in selecting suitable approximate arithmetic circuits for specific algorithms.

At the circuit level, power simulations are conducted on arithmetic units to understand how AACs achieve power savings. Furthermore, a selection of state-of-the-art approximate arithmetic circuits are selected and subjected to power simulations to evaluate the claimed power-saving benefits.

To evaluate the effectiveness of power reduction using sub-10 nm Complementary Metal-Oxide-Semiconductor (CMOS) technology, synthesis is performed using a standard cell library for this process node. The post-synthesis netlist of the Lanczos resampling implemented with selected AACs is simulated to provide a more accurate power estimation from a system-level perspective.

In this thesis, 8-bit unsigned approximate adders and multipliers are chosen as arithmetic units, which are sufficient to handle computations of 8-bit colour image input. EvoApproxLib v1.2022 [5] is the primary source for the study of approximate circuits, offering a choice of up to 40 unique 8-bit unsigned approximate adders and 40 unique 8-bit unsigned multipliers in an open-source library. The error metrics in this thesis are chosen from the survey paper [3], which discusses and evaluates a selection of state-of-the-art approximate circuits based on these same metrics.

1.2 Thesis Outline

This thesis is organized in the following chapters.

- **Chapter 2: Background -** Provides the necessary background knowledge about power and power-saving techniques using existing AACs and their error metrics for evaluation in image applications.
- **Chapter 3: Methodology -** Explains the simulation method to evaluate the power of existing AACs and how to incorporate it into hardware-adapted image scaling applications to find the correlation between error metrics and image quality.
- **Chapter 4: Results and Discussion -** Explains the AACs selection process and their impact on power and image quality when used in image applications.
- **Chapter 5: Conclusion and Discussion -** Summarizes the conclusions derived from the thesis and contributes ideas for future work.

Chapter 2
Background

This chapter provides an overview of the background knowledge necessary to understand the method and discussion parts. The main emphasis is on the factors that impact dynamic power consumption in arithmetic circuits and various previously proposed methods for reducing power consumption using AACs and error metrics for assessing them. Additionally, the section outlines how these circuits can be applied in image processing using various image scaling algorithms and techniques for evaluating image quality. Finally, an explanation of how these applications can be implemented on actual hardware is given.

2.1 Dynamic Power

The total power consumption comprises two components: dynamic power and static power. Static power relies heavily on CMOS technology, whereas dynamic power is primarily influenced by specific applications. Based on our pilot experiment, we found that about 90% of the total power consumption in our image scaling application is consumed by dynamic power. Therefore, dynamic power is one of the main concerns that should be focused on for reducing power consumption in these applications.

The dynamic power consumption of a circuit, as in (2.1) is proportional to the total capacitance C_L of the switching nets, the operating frequency f of the circuit, the switching activity $A_{switching}$, and the square of the voltage swing V_{dd} [6].

$$P_{dynamic} = C_L \times V_{dd}^2 \times f \times A_{switching} \tag{2.1}$$

Although Dynamic Voltage and Frequency Scaling (DVFS) [7] and Adaptive Voltage Scaling (AVS) [8] are commonly used power management techniques to lower power consumption, in this thesis, we specifically focus on addressing the power consumption associated with the switching of signals between logic gates and nets. Since the supply voltage and operating frequency are limited by CMOS technology and the given requirements, reducing the switching activity can be a solution to reduce dynamic power consumption. In addition, the reduction in the number of charges and discharges of the load capacitance can be an additional benefit of this method.

The switching activity can be categorized into toggling and glitches. Signal toggling refers to a signal switching from one logical state to another within a single clock cycle. Conversely, a glitch is an undesired brief fluctuation in a signal that occurs when there is a delay in the output transition from one combinational logic to another. When a signal switches, the capacitive loads in the switching nets are charged and discharged, which consumes power. Thus, the more frequently the signal toggles, the higher the probability of glitches occurring and the higher the circuit's power consumption.

In [9], the behaviour of a 4-bit Ripple Carry Adder (RCA) circuit consisting of four cascaded Full Adders (FAs), as in Figure 2.1 is analyzed under different input conditions, and the worst and best-case scenarios for the signals are identified. The input combination that results in the least switchings is considered the best case, and the input with the maximum switching is the worst case. Here, a unit delay model for every FA is assumed for the worst-case and best-case input calculation to observe the behaviour of signal glitching. The input combination for which the worst-case switching is detected based on the experiments in [9] is shown in Figure 2.2.

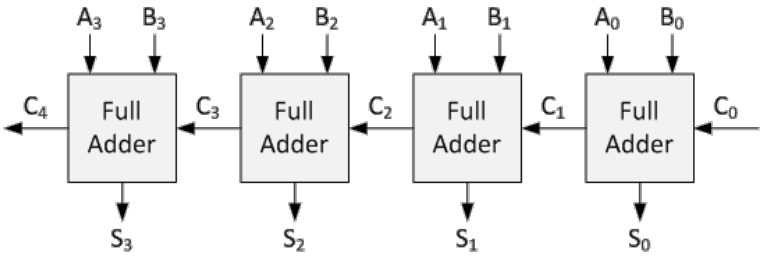


Figure 2.1: 4-bit Ripple Carry Adder

In real applications, the implementation of a FA circuit depends on the provided standard cell library. Therefore the delay on each gate depends on circuit loads and the logic optimizations done differently by the synthesis tool. To validate this assumption, we perform the same experiment on the 4-bit RCA with a clock frequency of 500 MHz, followed by netlist simulation to find the worst and best-case input combinations.

We can observe from Figure 2.3a that the number of switchings in the output sum propagates from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). Bit 0 of the sum has only one toggle and is considered a useful switching. However, in higher-order bits, multiple undesired switching events known as glitches occur within a single clock cycle. These glitches are considered useless, as they do not contribute to the final output while still consuming power. In contrast, Figure 2.3b represents the best-case input scenario, where such excessive glitch behaviour is not observed.

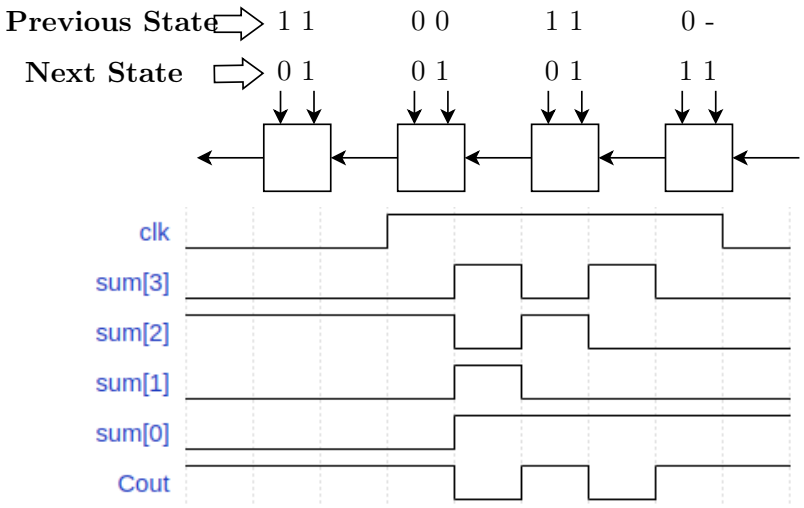
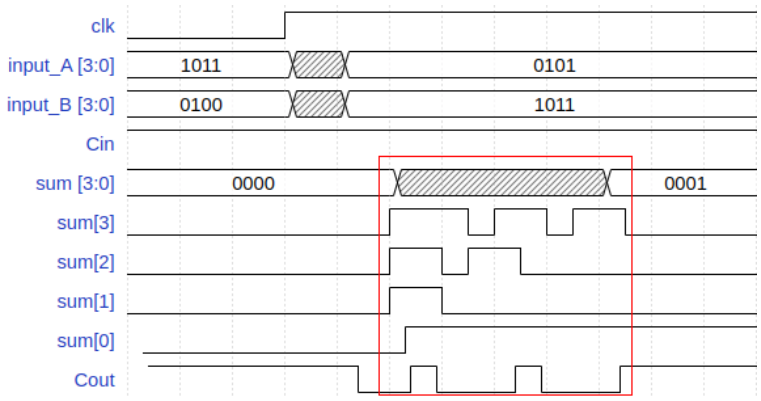
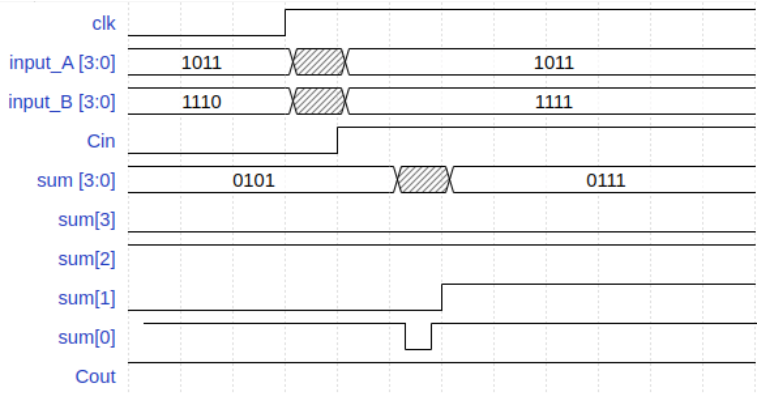


Figure 2.2: Worst case input and output switching as claimed in the reference paper



(a) Worst case input with maximum switching of 19 and power consumption of 690nW



(b) Best case input with minimum switching of 3 and power consumption of 340nW

Figure 2.3: Simulation result of 4-bit RCA to detect worst-case and best-case input combinations based on switching activity

Based on our experiment result, we found that the maximum switching of the output bits is detected in different worst-case input combinations compared to the result found in the paper. This result could be due to the technology-specific library and logical optimization techniques used in this analysis.

Power analysis in the design feeding all input combinations revealed that the input transition with maximum switching on output bits and a higher capacitance load would consume maximum power. Table 2.1 shows that the worst-case input with maximum switches causes almost twice the power consumption compared to the best-case input.

Table 2.1: Power consumption of a 4-bit Ripple Carry Adder

Input Type	Input Transition	Number of Toggles/Glitches	Total Power Consumption (nW)
Worst case	A - 1011 -> 0101 B - 0100 -> 1011 Cin - 0 -> 1	19	690
Best case	A - 1011 -> 1011 B - 1110 -> 1111 Cin - 0 -> 1	3	340

Thus, one solution to reduce the number of glitches will be to minimize the logic complexity of the circuit, resulting in lower switching and load capacitance, leading to lower circuit power consumption. AACs can be used in error-tolerant applications, such as image processing, resulting in a significant power reduction and still giving an acceptable outcome.

2.2 Arithmetic Circuits

Arithmetic circuits such as adders, multipliers, dividers, and shifters are the basic ALUs used in a data path to perform calculations. The design and implementation of these elements are often dominant in the power and speed of overall system performance. Therefore, careful design and selection of these arithmetic circuits are needed when designing hardware to trade off power, performance, and area

This thesis focuses on power optimization by replacing 8-bit adders and multipliers with their approximate alternatives. In the following, a brief introduction focusing on the power consumption of exact unsigned integer adders and multipliers will be discussed and extended to approximate unsigned integer adders and multipliers.

2.2.1 Exact Arithmetic Circuits

Exact Arithmetic Circuits (EACs) is designed to provide precise and accurate results for logic operations in arithmetic circuits. They are typically implemented using logic circuits, where the inputs and outputs are represented and processed with full precision. EACs are commonly used in critical parts of a circuit or applications where accuracy is crucial, such as the control logic of a processor.

EACs can be more power-hungry than AACs, requiring more complex circuitry to perform precise calculations. However, the actual power consumption of EACs depends on various factors such as the size of the circuit, the operating frequency, and the technology used for implementation.

The scope of this section focuses on the adders and multipliers among these EACs.

Basic Full Adder Circuit

A full adder circuit is one of the most common exact adders used to build other multi-bit adders and multipliers. In the subsequent chapters, we focus on the design of FAs and explore the potential for modifications into AACs. A FA is usually used by cascading N FAs to construct an N-bit adder. To analyze the implementation of an 8-bit unsigned adder, a 1-bit FA is discussed first in this section.

A FA adds three 1-bit inputs and produces two 1-bit outputs. The first two inputs are **A** and **B**, and the third input is an input carry **Cin**. The sum of the inputs is represented as **Sum**, and the output carry is designated as **Cout** as shown in Figure 2.4.

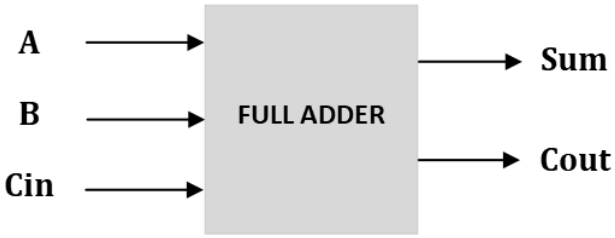


Figure 2.4: Full Adder

The FA design in Figure 2.5 using 2-input NAND gates is one way of implementation to analyze the switching activity and glitches at each stage of the NAND gate and find the correlation between peak power and switching. By verifying the observation in Section 2.1, power simulations of a FA prove that a circuit's power consumption is directly proportional to the number of switching of the output bits.

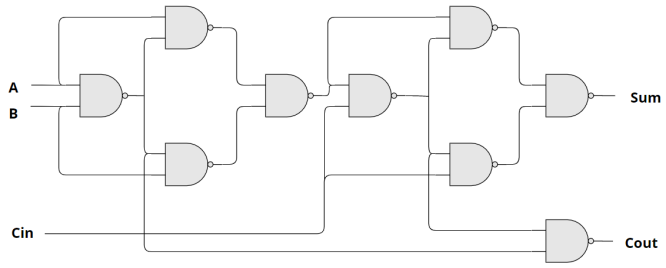


Figure 2.5: 1-bit Full Adder using NAND Gates

In practical implementations, the FA circuit may not exclusively comprise NAND gates. The synthesis tool can select optimized logic from a standard cell library when optimizing for power, area, or performance. These FAs are then cascaded differently to design different exact arithmetic adders.

A binary FA is also the basic component for building multipliers. Multipliers are essentially adder arrays that typically comprise three components: (a) partial product generation, (b) accumulation, and (c) final additions. The operation of partial product generation primarily involves the use of AND gates. On the other hand, the (b) and (c) stages can be implemented using adders of different bit lengths depending on the type of multipliers.

The authors in [10] have done some experiments and compared the power consumption of different exact and approximate circuits. The adders include Ripple Carry Adder (RCA), Carry-Select Adder (CSA), and Carry-Lookahead Adder (CLA). The multipliers include Ripple Carry Array, Multiple Carry-Save Array, and Wallace Tree circuits. When performing an accurate calculation, RCA is concluded to be the optimal circuit for power saving while CLA demonstrates a better performance with less delay.

In the subsequent section, we will dive into strategies for reducing power consumption by deliberately introducing errors within the arithmetic circuits using approximate circuits. This thesis investigates various approximate circuits researchers propose in [10][11].

2.2.2 Approximate Arithmetic Circuits

Approximate arithmetic circuits are becoming increasingly popular due to their ability to reduce delay, area, and power consumption compared to accurate arithmetic circuits. This is achieved at the expense of some loss in accuracy. However, if the errors resulting from approximate computation are within acceptable limits, approximate arithmetic circuits can be utilized in various practical applications such as digital signal processing, image processing, and the implementation of hardware for neural networks in artificial intelligence and machine learning. [12]

Approximate Adders

Some common approaches to designing approximate adders can be (a) by approximating the full adders, (b) by reducing the critical path of the circuit, and (c) by truncating the carry propagation.

One of the previous works [13] on the design of approximate adders using the approach (a) proposes a new design for a multi-bit FA circuit that can be used in low-power approximate computing systems. The proposed design uses a carry-based approach by altering the sum bit as shown in Figure 2.6 to avoid critical XOR operation in conventional ones, which reduces the circuit's power consumption. The research claims that approximate circuits with wider datapaths, such as 64 bits and 128 bits, significantly reduce delay, area, and power.

As this thesis focuses on only 8-bit computations, the carry-based approach for designing approximate adders does not give an acceptable accuracy. The authors aim to balance accuracy and power consumption. However, there is still a limit to how much accuracy can be sacrificed for power savings in smaller approximate adders.

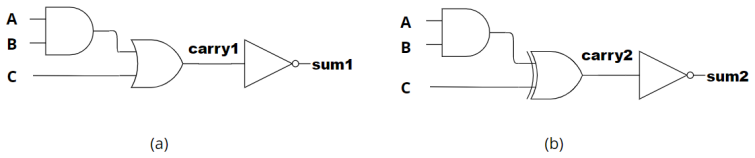


Figure 2.6: Carry Based Approximate Adders

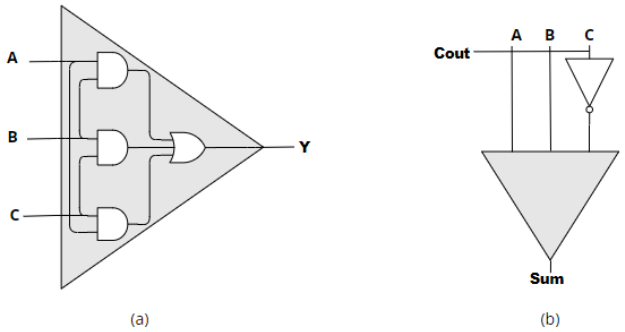


Figure 2.7: (a)Majority Gate Circuit, (b) Their proposed 1-bit Full Adder schematic

The multi-bit approximation adders in [11] are one type of approximate adder using the approach (b). Multiple 1-bit full adders are cascaded with a majority logic circuit as in Figure 2.7. With this implementation, there is a 50% reduction in delay and a 71% decrease in the area achieved in an 8-bit approximate adder but at the cost of a decrease in accuracy.

In category (c), one of the approaches is the segmented adder [14] [15]. It mitigates carry propagation delays by partitioning the input into multiple shorter segments. Each segment is processed independently by parallel sub-adder blocks with individual carry-in signals. This design effectively shortens the length of the carry propagation chain, thereby enabling faster addition operations. Figure 2.8 shows the fundamental structure of the segmented adders, illustrating their key components.

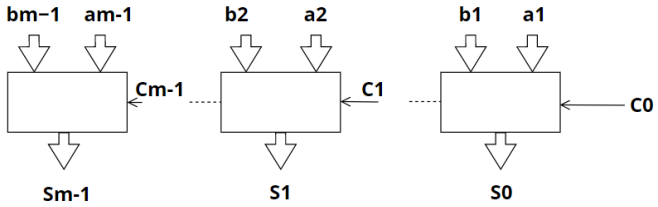


Figure 2.8: Segmented Adder

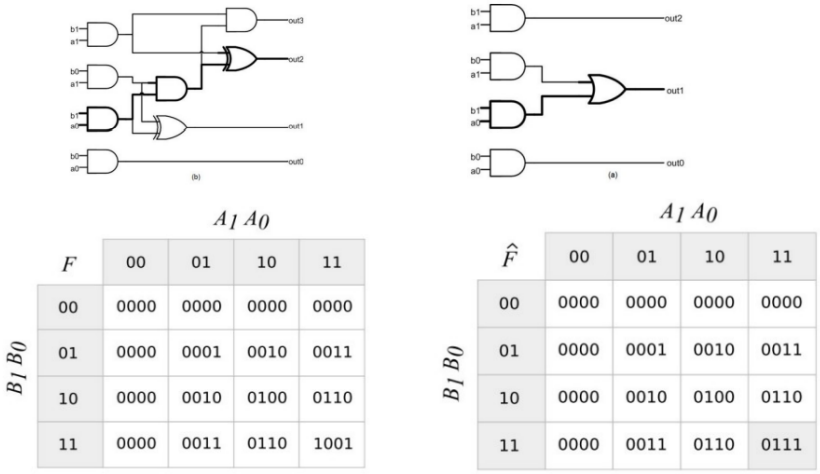
The most common segmented adder design is the N-bit Equal Segmentation Adder (ESA) [15], which employs multi-bit sub-adders without any carry input. Unlike other adders, the input bits used for carry computation in ESA do not overlap. This property significantly reduces the hardware cost of ESA compared to other adders. For convenience, sub-adders of equal size are considered in this thesis.

Approximate Multipliers

An approximate multiplier can be designed by modifying the three primary components of the exact multiplier. These three stages of approximation include (a) the approximation of the generation of partial products, (b) the approximation of the accumulation of partial products from different stages, and (c) the approximation of the addition of the final product. These approximations allow for a trade-off between accuracy and computational complexity in the multiplier design.

The Under-Designed Multiplier (UDM) employs a specific method for (a), as shown in Figure 2.9 [16]. The multiplier introduces an error by modifying a single output entry in this approach. At the same time, the adder circuit used in the final stage of the partial product addition remains unaffected and accurate. Despite the error introduced, it is within acceptable limits, and UDM allows for constructing larger multipliers while achieving significant power savings.

The mentioned approaches for approximate arithmetic circuits serve as examples of reducing logic gates and computations to minimize power consumption. However, this thesis primarily aims to evaluate the power-saving benefits of utilizing AACs. Consequently, not all of the aforementioned approaches are thoroughly examined. Instead, our primary focus is evaluating a specific set of AACs available within an open-source library. These AACs are designed using a novel automated process for creating approximate arithmetic circuits, which will be presented in the upcoming section.



$B_1 B_0$ \ $A_1 A_0$	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	1001

$B_1 B_0$ \ $A_1 A_0$	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	0111

Figure 2.9: 2x2 Under-Designed Multiplier

EvoApprox Library Arithmetic Circuits

EvoApprox8b is a library of approximate adders and multipliers that can be used for circuit design and bench-marking of approximation methods [5]. The library contains various types of approximate adders and multipliers and their implementation of exact arithmetic circuits.

The researchers who developed this library used a multi-objective Cartesian Genetic Algorithm to design the circuits, which allowed them to use an automatic way to generate approximate circuits [17]. The idea is to achieve the lowest possible power consumption of a given error metric by evaluating the switching activity of all input combinations. This method is used to optimize approximate circuits based on the power-saving trade-off for different error metrics such as Mean Square Error (MSE), Mean Absolute Error (MAE), Mean Relative Error (MRE), Worst Case Error (WCE), and Error Probability (EP). Each error metric has a collection of circuits with different power-saving ranges that allow users to select their basic building blocks for target applications.

In this thesis, we mainly focus our evaluation on examining 8-bit unsigned approximate adders and 8-bit unsigned approximate multipliers circuits from the library (v1.2022) and check if the claimed power saving ratio in the library still holds valid with the latest technology and if it can be used in image applications. In addition to the AACs present in the EvoApprox Library, we have also implemented the Segmented Adder and UDM for conducting power simulations. This allows us to compare the tradeoff between power consumption and error metrics between the AACs in the EvoApprox Library and the AACs designed with a conventional method.

2.3 Error Metrics

Error metrics play a crucial role in evaluating AACs. These metrics provide different quantitative measures to assess the error characteristics of the approximate results compared to the exact results, allowing researchers and designers to make decisions in the selection of AACs.

In this thesis, we have two types of error metrics to evaluate for approximate circuits; one is Arithmetic Accuracy, and the other is Application Specific Metrics (ASMs). Arithmetic accuracy assumes evenly distributed input values and calculates the accuracy of circuits based on those input values. In contrast, ASMs evaluate approximate circuits from a specific application's perspective. Based on different applications, we can expect that the distribution of input values varies and, therefore, has other impacts on various applications. This thesis focuses on applying approximate circuits to image applications so that we will introduce some commonly used ASMs for image quality evaluation.

2.3.1 Arithmetic Accuracy

To evaluate the accuracy of an approximate circuit, different aspects such as error distribution, worse-case error, and average error distance should be considered. Therefore, there are a few error metrics that have been adopted. Here are some commonly used error metrics to evaluate arithmetic differences. For a total N-output circuit, i is the output i^{th}. The metrics in this section are referenced from [3].

Error Distance (ED)

The error distance (ED_i) shows the arithmetic difference between the i^{th} output of an approximate circuit and the i^{th} output of an accurate circuit. It is calculated as (2.2)

$$ED_i = |M_i' - M_i| \tag{2.2}$$

where M_i is the decimal format output of an accurate circuit and M_i' is the decimal format output of an approximate circuit. The absolute value is the difference between an approximate and an accurate circuit.

Relative Error Distance (RED)

To consider the relative difference compared to the accurate output value, the RED is a commonly used metric. It is calculated as (2.3) where ED_i is divided by the i^{th} output M_i of the accurate circuit. When using this metric, a higher error tolerance is expected if M_i is also larger.

$$RED_i = \frac{ED_i}{M_i} \tag{2.3}$$

Above, individual differences are discussed. In the following, these errors are accumulated to evaluate the overall accuracy of the circuits.

Mean Absolute Error (MAE) or Mean Error Distance (MED)

MED (also known as MAE) is calculated in (2.4) as accumulating the error distance of each output multiplied by the probability that the i^{th} input occurs. In this thesis, the discussion revolves around 8-bit adders and 8-bit multipliers. Consequently, when two inputs each have a probability of $\frac{1}{256}$ and are combined, the resulting probability would be $\frac{1}{256\times256}$. This metric provides a mean error distance based on how many input combinations it has.

$$MAE = \sum_{i=1}^{N} ED_i \times P(ED_i) \tag{2.4}$$

Mean Relative Error (MRE)

MRE is calculated the same as MAE but replaces ED with RED as (2.5). Therefore, for any two types of 8-bit unsigned approximate multipliers, when ED is the same between two circuits, but one circuit produces errors in the lower bits of the output, it will exhibit a higher MRE than the other.

$$MRED = \sum_{i=1}^{N} RED_i P(RED_i) \tag{2.5}$$

Normalization of Mean Error Distance (NMED)

NMED is the normalization of MED by dividing the maximum value of the accurate circuit by (2.6). This metric is useful when comparing the different sizes of approximate circuits. Only 8-bit AACs are explored in this thesis, so this metric is not discussed in the later sections.

$$NMED = \frac{MED}{max(M_1, ...M_N)} \tag{2.6}$$

Mean Square Error (MSE)

MSE is also a common method used to evaluate approximate circuits. The MSE is calculated by accumulating the square of the error distance times the probability that this error occurs as (2.7). It is sensitive to large EDs, as MSE amplifies the impact of large errors by accumulating the square of EDs.

$$MSE = \sum_{i=1}^{N} ED_i^2 P(ED_i) \tag{2.7}$$

Error Rate (ER) or Error Probability (EP)

ER or EP are sometimes used interchangeably in some studies. It accumulates 1 whenever an approximate output differs from the accurate output (2.8). Then divide the total count of the output number to calculate the probability of an

erroneous result. This error metric does not consider the value of ED. As long as there is any difference, 1 is accumulated.

$$EP = \sum_{i=1}^{N} \frac{1}{N}, M' \neq M \tag{2.8}$$

Worst Case Error (WCE)

WCE is used to find the maximum ED in an approximate circuit as (2.9). Here, normalization is done by dividing the maximum error by the output size N so that different sizes of AACs are comparable. Although the comparison of different AACs is not within the scope of this thesis, we use the same formula to maintain consistency.

$$WCE = \frac{max(ED_1, ...ED_N)}{N} \tag{2.9}$$

2.3.2 Application Specific Metrics

The next section will focus on application-specific metrics, specifically concerning the evaluation and discussion of image quality assessment. This analysis aims to delve into the various aspects and criteria used to assess the quality of images in the context of the specific application under consideration.

Mean Square Error (MSE)

In image application, MSE is calculated by

$$MSE = \frac{1}{XY} \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} |I(x,y) - I'(x,y)|^2 \tag{2.10}$$

it accumulates the square of error distance between each pixel from the reference image I and each pixel from the processed image I', divided by the total number of pixels. Symbols x and y are expressions of pixel coordinates from the x and y directions. The width and height of an image are described as X and Y, respectively. The mathematical expression is the same as MSE in the arithmetic precision section, while i is replaced by (x,y) and N is replaced by the total number of pixels $X \times Y$ of an image.

Peak Signal to Noise Ratio (PSNR)

PSNR is an engineering term used to find the ratio between the signal's peak power versus the power of distortion. When applied in image applications, the peak power is the maximum pixel value of an image. In our application, the images are represented and operated in 8-bit format, so the maximum pixel MAX_i here is 255.

$$PSNR = 20log_{10}\frac{MAX_i}{\sqrt{MSE}} \tag{2.11}$$

Based on (2.11), we can interpret the expression of PSNR as the higher, the better. For an 8-bit lossy image, having PSNR between 30dB and 50dB [18] is common. When the calculated MSE is 0, there is no distortion; therefore, PSNR is infinite.

Structural Similarity Index Metric (SSIM)

MSE and PSNR are relatively simple to calculate and are widely used in different research fields. However, those metrics assume that the image quality perceived by the Human Visual System (HVS) can be directly quantified by the error measurements. In practice, different distortions on images can have the same MSE but have a different assessment by human eyes. The authors in [19] have found that HVS is more sensitive to extracting structural information when assessing image quality. Based on this finding, one of the common and relatively accurate indices called SSIM is proposed in that paper. This index separates image signals into three relatively independent components, luminance(l(x,y)), contrast(c(x,y)), and structure(s(s,y)). The overall image quality is considered by evaluating these three factors. It can be represented as

$$SSIM(x, y) = f(l(x, y), c(x, y), s(x, y)) \tag{2.12}$$

Luminance is considered as the average over the entire image, where μ_x is the mean intensity of image x and N is the total number of pixels of image x, and x_i is the pixel value:

$$\mu_x = \frac{1}{N} \sum_{i=0}^{N-1} x_i \tag{2.13}$$

Standard deviation is used as an estimation of signal contrast where σ_x is the standard deviation (the square root of variance) as an estimate of the signal contrast in image x, it is given by

$$\sigma_x = \sqrt{\frac{1}{N-1} \sum_{i=0}^{N-1} (x_i - \mu_x)^2} \tag{2.14}$$

The structure is derived after removing the mean intensity and normalized by its standard deviation, and the structure expression of image x is calculated as:

$$\frac{x - \mu_x}{\sigma_x} \tag{2.15}$$

The calculation for μ_y, σ_y, and the structure factor of image y is the same as image x.

To compare those factors between image x and image y They are calculated respectively as follows:

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.16}$$

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.17}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3} \tag{2.18}$$

where C_1,C_2, and C_3 are small values to avoid instability when denominators are zeros. It can be calculated by

$$C_1 = (K_1 L)^2 \tag{2.19}$$

$$C_2 = (K_2 L)^2 \tag{2.20}$$

$$C_3 = \frac{C_2}{2} \tag{2.21}$$

where K_1, K_2 <<1 are small constants, and L is the dynamic range of the pixel values. Here, we use 255 for standard 8-bit images.

σ_{xy} is given as

$$\sigma_{xy} = \frac{1}{N-1} \sum_{i=0}^{N-1} (x_i - \mu_x)(y_i - \mu_y) \tag{2.22}$$

Finally, the SSIM is calculated by combining $l(x,y)$,$c(x,y)$, and $s(x,y)$ to below:

$$SSIM(x,y) = [l(x,y)]^\alpha \cdot [c(x,y)]^\beta \cdot [s(x,y)]^\gamma \tag{2.23}$$

where $\alpha > 0$, $\beta > 0$, and $\gamma > 0$ are parameters used to decide the relative importance of those three components. In the proposed paper, all are set to 1. Based on the above formula, the SSIM index results in:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2) + C_2} \tag{2.24}$$

Mean Structural Similarity Index Metric (MSSIM)

The authors in [19] have proven that SSIM achieves better results when applied locally rather than globally. An 11 by 11 circular-symmetric Gaussian weighting function $w = w_i | i = 1, 2, ..., N$ with a standard deviation of 1.5 samples is normalized to a unit sum. The estimated values of local statistics, namely μ_x, σ_x, and σ_{xy}, are defined as:

$$\mu_x = \sum_{i=1}^{N} \omega_i x_i \tag{2.25}$$

$$\sigma_x = (\sum_{i=1}^{N} \omega_i (x_i - \mu_x)^2)^{\frac{1}{2}} \tag{2.26}$$

$$\sigma_{xy} = \sum_{i=1}^{N} \omega_i (x_i - \mu_x)(y_i - \mu_y) \tag{2.27}$$

Once the above calculation has been done for all image pixels, a mean SSIM (MSSIM) index is calculated to evaluate the overall image quality:

$$MSSIM(X,Y) = \frac{1}{M} \sum_{j=1}^{M} SSIM(x_j, y_y) \tag{2.28}$$

where M is the sample number of the quality map.

2.3.3 Image quality metrics on colour image

Metrics such as PSNR and SSIM are commonly used to assess the quality of compressed grayscale images. These metrics are based on comparing the pixel values between the original and compressed images and are, therefore, well suited for grayscale images with only one information channel. Since this thesis focuses on discussing the image quality of colour images, adaptation is required to apply them to colour images. In this thesis, the scaling application has experimented on RGB and YCbCr colour spaces. Based on HVS, human eyes are more sensitive to luminance than chroma; thus, the colour space YCbCr is designed to separate luminance from chroma, while the RGB colour space is a mixture of both luminance and chroma in all colour channels. When a colour image is analyzed in YCbCr colour format, they are weighted according to [20]:

$$\mathrm{imq}_{\mathrm{YCbCr}} = 0.8 \times Y + 0.1 \times Cb + 0.1 \times Cr \tag{2.29}$$

while RGB channels are treated as equally weighted.

$$\mathrm{imq}_{\mathrm{RGB}} = \frac{R + G + B}{3} \tag{2.30}$$

In the upcoming sections, scaled images will be evaluated using these equations to adapt them to colour images. Each image will have three separate image metric values, and then, depending on the colour space employed for the scaling process, a final value of the image metric is calculated.

2.4 Image Applications

2.4.1 Image scaling

Scaling is the process of resizing images to fit specific resolutions or aspect ratios, often necessary for various purposes such as image segmentation, Convolutional Neural Network (CNN) classifications, and preprocessing for other image applications. Several scaling algorithms exist for image resizing, including nearest-neighbour interpolation, bilinear interpolation, bicubic interpolation, and Lanczos resampling. Nearest-neighbour interpolation is the simplest method but may not yield optimal results compared to other algorithms. Bilinear interpolation and bicubic interpolation operate by interpolating new pixels based on the weighted values of surrounding pixels, with bilinear considering a 2×2 pixel neighbourhood and bicubic considering a 4×4 pixel neighbourhood. In contrast, Lanczos resampling is known to provide better image sharpness preservation, but its computational complexity poses limitations when implemented on hardware.

This thesis selects Lanczos resampling as the target algorithm due to its ability to preserve better image quality after scaling. Its complexity makes it meaningful to explore the application of AACs to improve performance and power efficiency. As referenced from [21], the algorithm possesses distinct characteristics that make it suitable for different image types and applications.

Lanczos resampling

Lanczos resampling is a technique used for image scaling that involves using a kernel with a size determined by a positive integer parameter, denoted as a. Typically, a is set to 2 or 3 to achieve optimal results. In this case, we choose a to be 2, resulting in a kernel size of 4 pixels in both the x and y directions. Therefore, a total kernel size of 4×4 is required to interpolate one pixel in the scaled image. This algorithm uses a sinc function as (2.31) to interpolate the pixel values for the output image. The Lanczos kernel is calculated as follows:

$$L(x) = \begin{cases} 1, & \text{if } x = 1 \\ \frac{asin(\pi x)sin(\pi x/a)}{\pi^2 x^2}, & \text{if } -a \leq x < a \text{ and } x \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.31}$$

To perform the interpolation, we utilize an interpolation formula represented by Equation (2.32). This formula involves two main variables: x and i. The variable x represents the new position of the pixel that needs to be interpolated, either in the x or y direction. On the other hand, i represents the location of the neighbouring pixel in the original image that corresponds to x.

In Equation (2.32), the pixel value $S(x)$ at the new position x is calculated by summing the contributions of neighbouring pixels S_i, weighted by the Lanczos function $L(x - i)$. The Lanczos function determines the influence of each neighbouring pixel based on its distance from the new position.

$$S(x) = \sum_{i=\lfloor x \rfloor - a + 1}^{\lfloor x \rfloor + a} S_i L(x - i) \tag{2.32}$$

The Lanczos resampling is illustrated in Figure 2.10 where in the horizontal direction, four input pixels P_{11}, P_{12}, P_{13}, P_{14} are weighted according to the corresponding Lanczos coefficient to interpolate the value of 1 intermediate pixel I_1. The same procedure is repeated in the y direction, where the input pixels P are replaced with intermediate pixel values I.

To provide an example, given the situation where we want to interpolate a new pixel located at (3.5, 3.5) using Lanczos resampling. In the x direction, we take into account the pixels at positions (2, 3.5), (3, 3.5), (4, 3.5), and (5, 3.5) of the original image, along with their respective pixel values. These neighbouring pixels play a crucial role in interpolation calculation, using the Lanczos function to determine their contributions.

However, since the y coordinate 3.5 is unavailable in the original image, we must apply the same interpolation technique in the y direction. To accomplish this, we repeat the interpolation process four times in the x direction, resulting in interpolations involving 16 pixels. Later, we perform the interpolation in the y direction again to achieve the final interpolated pixel value.

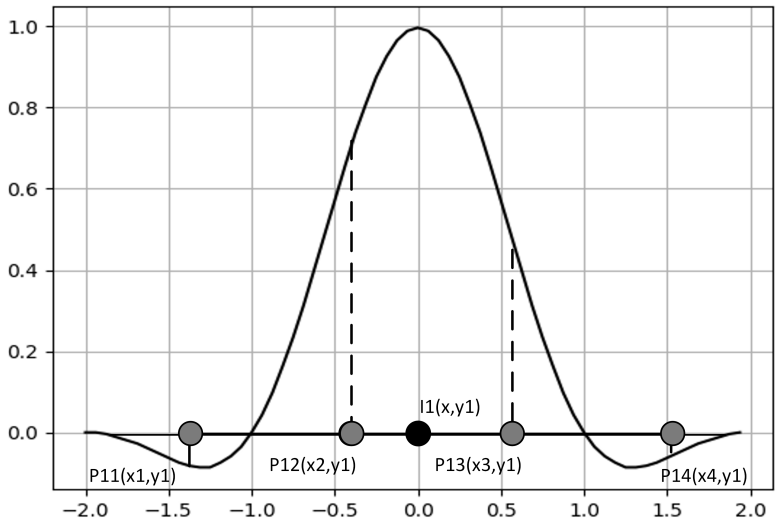


Figure 2.10: Lanczos resampling

Therefore, we would need to consider the pixel locations for the x direction as below, where the location of each pixel is expressed as (x,y):

$$\begin{pmatrix} (2,2) & (3,2) & (4,2) & (5,2) \\ (2,3) & (3,3) & (4,3) & (5,3) \\ (2,4) & (3,4) & (4,4) & (5,4) \\ (2,5) & (3,5) & (4,5) & (5,5) \end{pmatrix} \tag{2.33}$$

By performing (2.32) where i corresponds to the x location of the pixels in (2.33), we can interpolate four intermediate pixels as shown in (2.34). By performing the y-direction interpolation once, the pixel value at (3.5,3.5) is calculated.

$$\begin{pmatrix} (3.5,2) \\ (3.5,3) \\ (3.5,4) \\ (3.5,5) \end{pmatrix} \tag{2.34}$$

The above example can be extended to perform on all new pixels to obtain a final scaled image. For a kernel with a set to 2, 16 input pixels are needed to calculate one output pixel. Therefore, the Lanczos resampling is slower than the bilinear and bicubic interpolation. On the other hand, it preserves the sharpness of the image with these Lanczos kernels.

The choice of algorithm depends on the specific application and the trade-off between speed and image quality. Lanczos resampling produces better image quality while having more complex computations than Bilinear interpolation. Therefore, bilinear interpolation is still the most commonly used algorithm for scaling. In this thesis, we discuss the possibility of reducing the computation complexity of Lanczos resampling with approximate multipliers.

Chapter 3
Methodology

This thesis employs a methodology that includes various stages, starting with software-level evaluation involving algorithmic evaluation. This evaluation then extends to circuit-level experimentation and finally leads to system integration. Section 3.1 provides an overview chart that summarizes the methodology adopted in this thesis. Further, Section 3.2 and Section 3.3 elaborate on the software-level algorithmic assessment and the circuit-level experimentation, respectively. Finally, Section 3.4 delves into the integration aspect of the system.

3.1 Methodology Overview

An overview of the design methodology is shown in Figure 3.1. The methodology steps represented by dashed lines indicate the tasks implemented in a high-level programming language to demonstrate the proof of concept for the thesis. On the other hand, the steps represented by solid lines involve implementation in a hardware description language, followed by synthesis and power simulation. It involves three key steps:

- Scaling Applications: The software implementation of image scaling applications is an efficient means of evaluating and analyzing the image quality of scaled images using approximate multipliers. These implementations include hardware-friendly adjustments, as described in Subsection 3.2.2, which are crucial to ensure the compatibility and seamless integration of approximate multipliers in software and hardware systems. Subsection 3.2.3 will mention which part of the design is being replaced with approximate multipliers. The expected output would allow us to evaluate image quality and its correlation with error metrics.

- Arithmetic Circuits: In Sections 2.1 and 2.2, we comprehensively analyse how switching activities affect the overall power consumption of the adder circuits. This study provides valuable information on the mechanisms through which power reduction can be achieved in AACs. Based on this understanding, in Subsections 3.3.1 and 3.3.2, we present the synthesis and power simulation methodologies used in this thesis to evaluate the power gains achieved using AAC.

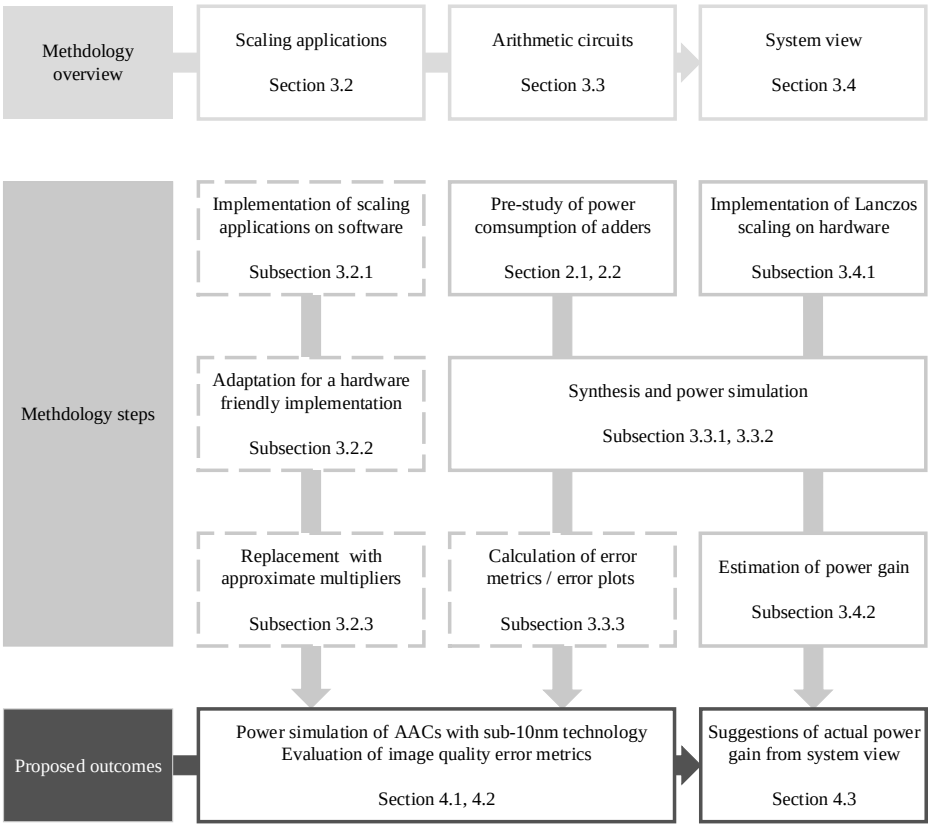


Figure 3.1: Methodology overview

Furthermore, in Subsection 3.3.3, we describe the software implementation utilized to calculate error metrics and generate error plots. These methodologies enable us to assess the effectiveness of AACs in reducing the power consumption of the selected fabrication technology while maintaining acceptable levels of precision.

- System view: In Subsection 3.4.1, we present a hardware implementation of the designed circuit based on the results obtained from the previous steps. By implementing and evaluating the circuit in hardware, we can verify the power reduction achieved through the use of AAC.

  In Subsection 3.4.1, we will evaluate and estimate the power consumption of the entire system based on the hardware implementation described above. This estimation considers the power consumption of the AACs and other components and modules within the system. Comparisons were made between the estimated system power consumption and the actual power gain achieved by using AAC, and the results are shown in Section 4.3. This

comparison allows us to assess the effectiveness of AACs from a system perspective. Based on this evaluation, we can provide recommendations and suggest specific AACs suitable for integration into the system, considering both power reduction and system-level performance requirements.

3.2 Scaling Applications

This section follows the steps required in designing and evaluating the image scaling application using AACs in the software domain.

3.2.1 Implementation of scaling applications on software

By implementing the image scaling algorithms in software, we can simulate and assess the performance of the approximate multipliers without the need for physical hardware. This software-based approach allows for a more efficient evaluation of multiple approximate multipliers and enables us to analyze the impact of these multipliers on the resulting image quality. Figure 3.2 shows the image scaling application flow in software. A combination of Python and C code is used to achieve better performance in implementing image scaling software. The Python code handles initial processing tasks, such as loading the input image and setting the scaling ratio. It prepares the necessary data and parameters for the scaling operation.

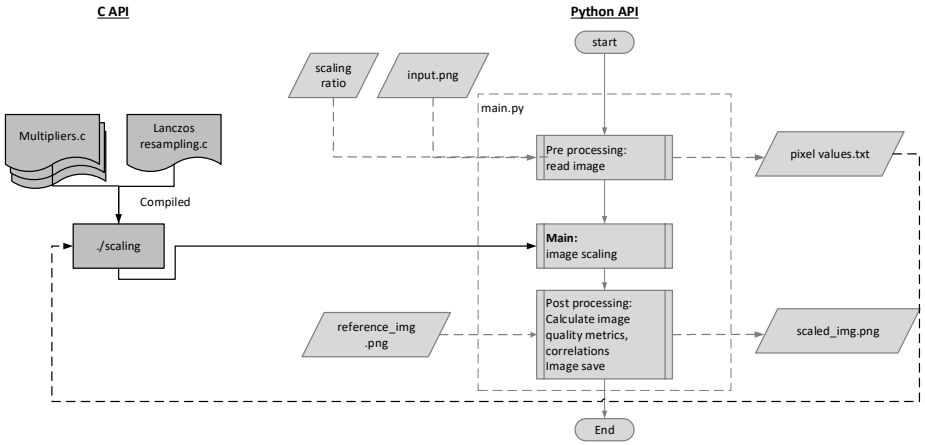


Figure 3.2: Flow of image scaling in software

To ensure efficient execution, the image scaling process is implemented in the C code. The Python code passes the image data and scaling parameters to the C code, which performs the scaling operation. This implementation in C takes advantage of the lower-level capabilities in the programming language, resulting in improved performance and faster processing times.

Once the scaling operation is complete, the resulting scaled image is returned to the Python code for further analysis. The Python code then calculates the image quality metrics MSE, PSNR, and SSIM of the resulting image and a reference image that is scaled with the same scaling method using an exact multiplier. These metrics are compared to assess the quality of the scaled image. The Python code can also save the final scaled image for later analysis and comparison. This allows us to visualize and subjectively understand the scaling results and assess the image quality.

3.2.2 Adapting for a hardware-friendly implementation

The arithmetic logic units of a low-power, real-time image processing ASIC are constrained to 8-bit unsigned integer numbers in this thesis. Therefore, an adaptation to image scaling is required.

The specification of image scaling is as follows:

- Input images are all square-shaped, with the horizontal and vertical directions having the same dimensions. In this thesis, we only consider square images to simplify our analysis, but all conclusions are valid for any image size.
- The input images are 8-bit colour images; the colour format can be either RGB or YCbCr.
- The image scaling ratio is limited to downsizing within the range between 100% and 50%. This limitation is imposed to simplify the analysis in this thesis, but the approach remains applicable when adapted to hardware.
- The scaling factor is fixed to 2^4.

To implement image scaling using 8-bit unsigned integer adders and multipliers, the following steps are undertaken:

- Determine the scaling factor: The scaling factor, or grid size, is utilized to scale up floating point values and calculate the integer parts. The output is then divided by the scaling factor, and the nearest integer is taken as the final output. By avoiding floating-point operations, this approach simplifies hardware implementation. Choosing a larger scaling factor generally improves the accuracy of the scaling result, but it also increases the hardware requirements. Larger scaling factors necessitate higher-bit arithmetic circuits, adding complexity and resource demands to the implementation. Additionally, they require the storage of larger Lanczos kernels, which consume more memory and can be limiting in resource-constrained systems. To ensure a hardware-friendly design, a grid size of 16 is chosen in this thesis, as it is a power of 2 and enables efficient implementation using right-shift operations instead of division.
- Select the available scaled image size: The coordinates of the new pixels in the scaled image are interpolated according to the ratio of the new image versus the original image, as shown in (3.1) and (3.2). The new pixels' coordinates in the scaled image are interpolated based on the ratio of the

new image size to the original image size. The new image size is determined by applying an integer step size between 16 and 32, corresponding to scaling between 100% and 50%. The coordinates of the new pixels are calculated incrementally using the step size as shown in (3.3), ensuring that the new image size is an integer value. Since the new image size is often not a multiple of the scaling factor, an offset, as shown in (3.4), is introduced to prevent the loss of the last pixels during multiple scaling operations. This offset, which varies from 0 to 8 (less than or equal to half the scaling factor), is added at both the beginning and the end of the image. The calculation of the new image size is determined by the scaling factor, the original image size, the offset, and the step size, as shown in (3.5). It is important to note that this limitation of a new image of integer size applies solely for the purpose of simplification in this thesis, while real applications may employ other methods to accommodate different image sizes.

$$\text{Scaling ratio} = \frac{\text{New image size}}{\text{Old image size}} \tag{3.1}$$

$$\text{New pixels} = i \times \frac{1}{\text{scaling ratio}}, i = 0,\ 1,\ \ldots\text{New image size -1} \tag{3.2}$$

$$\text{Scaled new pixels} = i \times \text{step size}, i = 0,\ 1,\ \ldots\text{New image size -1} \tag{3.3}$$

$$\text{Offset} = (0, 1, \ldots \frac{\text{Scaling factor}}{2}) \tag{3.4}$$

$$\text{New image size - 1} = \frac{\text{Scaling factor} \times (\text{Ori image size - 1}) - 2 \times \text{Offset}}{\text{Step size}} \tag{3.5}$$

- Calculate the scaling coefficients: For the Lanczos resampling algorithm, the sinc coefficients are scaled from ±1 to ±255. This thesis uses a filter size of 4; since the grid size is 16, the number of Lanczos kernels is 4 × grid size. Resulting in a total of 64 precalculated filter coefficients stored in a lookup table. It is worth noting that when the negative coefficient in the interpolation formula is calculated, it is computed as a positive value. This means that the absolute value of the negative coefficient is also achieved using unsigned multipliers. The sign of the coefficient is added after the calculation to obtain the correct result.
- Resize the image: To derive the interpolated pixel values in the new image, it is necessary to obtain the floor integer from a floating point value, denoted as (2.32). Since all computations are performed using integers, the floor operation is computed as shown below in (3.6):

$$\text{Scaled floor} = \lfloor(\text{new pixel/scaling factor})\rfloor * \text{scaling factor} \tag{3.6}$$

3.2.3 Replacement with approximate multipliers

In implementing image scaling using approximate multipliers, only the multiplication operations involved in pixel value calculations during Lanczos resampling are replaced with approximate multipliers. Control logic and pixel location calculations, such as determining the source and destination pixel coordinates, are kept intact and implemented using exact adders.

The Lanczos resampling algorithm requires the calculation of weighted neighbouring pixels to determine the new pixel values in the scaled image. These weighted values involve multiplication operations between the pixel values and the corresponding filter coefficients. In the software implementation, these multiplication operations are replaced with approximate multipliers.

By replacing exact multiplication operations with their approximate counterparts, the implementation provides a targeted approach to evaluate the impact of approximate multipliers on the overall image scaling process. This allows a focused analysis of how the approximation affects the calculation of the pixel value and the image quality.

The software implementation can effectively calculate the image quality metrics using approximate multipliers. By isolating approximate multiplication operations, it becomes possible to evaluate the accuracy and fidelity of the scaled images while maintaining the integrity of the control logic and the calculation of the location of the pixels, which are crucial for achieving correct system behaviour.

Overall, this approach enables a controlled evaluation of the effects of approximate multipliers on image scaling, providing valuable insights into the trade-offs between power consumption and image quality in later usage in the thesis.

3.2.4 Relation between Arithmetic Accuracy and Application Specific Metrics

This thesis uses three widely used correlation coefficient methods to establish the correlation between Arithmetic Accuracy and Application Specific Metrics. These methods are Pearson's correlation coefficient, Spearman's, and Kendall's rank correlation coefficients, which have also been used in image quality assessment and are referenced from a widely cited research paper on finding the correlations between different image quality metrics [22]. Pearson's coefficient is ideal for assessing linear relationships between continuous variables, while Spearman's is suitable for studying monotonic relationships or ordinal data. Kendall's coefficient is also used to analyze ordinal data, account for ties, and focus on the order or ranking of variables. Since PSNR is in a logarithmic scale and does not present a linear relationship, Pearson's correlation is unsuitable, but the other two methods may be applied.

3.3 Approximate Arithmetic Circuits

The implementation and analysis of Approximate Arithmetic Circuits (AACs) for image scaling application involves - (a) evaluation and optimization of the RTL approximate circuit design by performing logical synthesis with the given technology and standard cell library, (b) performing power simulations on the optimized design to check the power savings that can be achieved using AAC, and (c) calculating the error metrics of various approximate circuits to find the correlation between power and error values to select the best suitable multiplier for our image application.

3.3.1 Synthesis

Designers can effectively tackle current design challenges using an Register-Transfer Level (RTL) synthesis tool, which enables concurrent optimization of timing, area, and power. The tool offers high accuracy in predicting post-layout area, timing, and power without requiring the wire load models. It generates a virtual design layout to accurately predict the net capacitances and adjust the net delays. With this method, there is no longer a need for optimistic wire load models in synthesis. This accurate prediction of net capacitances helps generate a netlist optimized for all design objectives. With the ability to address real design issues during synthesis, designers can obtain a better starting point for the subsequent placement and routing stages, ultimately resulting in improved overall design results. Consequently, the power consumption can be accurately analyzed using the netlist simulation results comparable to those obtained from execution on actual hardware.

Experimental Setup

After the RTL simulation is performed and verified, the design is synthesized for gate-level optimization with sub-10 nm technology. The synthesis process requires technology libraries and constraints to build a specific architecture from the cells in the library. To achieve the best possible hardware design at this stage, libraries with Low Voltage Threshold (LVT) characteristics are utilized for better performance, and the tool is set for a high level of optimization with a clock frequency of 500 MHz. The synthesized netlist is then subject to simulation for power analysis.

Synthesis is first performed on the design with one instance of each approximate adder and multiplier and then on the design with ten instances of adder and multiplier together. The netlist of both designs is subjected to power analysis to check the power consumption of each approximate circuit and verify if the multiple instances affect the power consumed by the adders and multipliers. Circuits with an acceptable power range are then selected for implementation in the image scaling application, and synthesis is carried out on the design for further power analysis.

The results of the synthesis stage are discussed in Chapter 4.

3.3.2 Power Simulation

System flow of scaling application

Power simulation is performed on a post-synthesis netlist estimating the power consumption of a sub-system. The power analysis includes average power, peak power, glitch power, clock network power, and dynamic and leakage power.

The power estimation tool can effectively simulate the power consumption from a gate-level netlist strongly correlated with the actual power usage observed during the implementation and signoff stages. By providing accurate power consumption estimates, designers can confidently analyze, explore, and optimize the design. This leads to improved power efficiency and shorter design cycles.

Experimental Setup

Several modes are available for power analysis, including average, time, and toggle-based modes. Of these, the time-based mode is considered the most accurate. During netlist simulation, an Fast Signal Database (FSDB) file is generated, which stores the simulation waveform data, recording the state and time of the nets. The switching activity of an FSDB file is used to measure accurate power consumption, glitch power, and state-dependent leakage power.

This thesis performs time-based power analysis by providing the switching activity at each time unit. The tool generates various summary reports that provide insights into different power consumption aspects, including hierarchical and summary reports. The hierarchical report offers a detailed analysis of power consumption for each sub-module in the design. In contrast, the summary report presents the total power consumption of the entire sub-system, including power groups like registers, clock networks, and sequential and combinational logic. Both reports include information about the dynamic power (including internal and switching power), leakage power, and glitch power, which are crucial for selecting the appropriate multiplier for further image analysis when comparing the powers of different implementations.

3.3.3 Calculations of error metrics and error plots

All input combinations are fed to the target circuit to evaluate approximate arithmetic circuits and calculate the error metrics. In the thesis, a flow to speed up this process is proposed. As illustrated in Figure 3.3, approximate circuits are implemented or imported into a C program, and the calculation is performed in C code. Since 8-bit unsigned adders and multipliers are of interest in this thesis, two input numbers are given with a nested loop from 0 to 255. After the calculations are completed, a Python script reads the results from text files and calculates the error metrics, plotting the result in charts. This flow significantly reduces the calculation time compared to only the implementation with Python script.

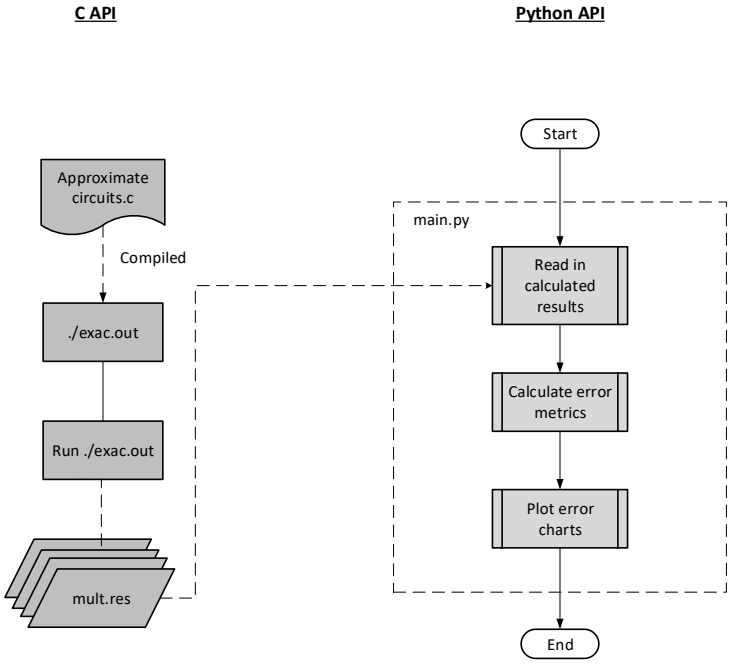


Figure 3.3: Flow of calculating error metrics of arithmetic circuits

Visualization of errors in approximate arithmetic circuits

Error metrics are a common method for evaluating errors in approximate circuits. However, in this thesis, we found a need to visualize approximate arithmetic circuits since error metrics are global metrics, and some local characteristics are lost in those metrics. In Table 3.1, eight 8-bit multipliers, including exact and approximate multipliers with their corresponding error metrics, are listed.

Table 3.1: Approximate multipliers and corresponding error metrics.

Index	Multiplier	MAE(%)	MRE(%)	WCE(%)	MSE	EP(%)
(a)	exact	0.00	0.00	0.00	0.00E+00	00
(b)	mul8u_UDM	1.38	3.28	22.05	6.46E+06	47
(c)	mul8u_197B	0.18	4.42	0.66	2.10E+04	98
(d)	mul8u_17C8	0.56	10.85	2.41	2.10E+05	99
(e)	mul8u_T83	2.15	39.78	8.21	3.09E+06	99
(f)	mul8u_17MN	8.01	59.69	27.24	4.28E+07	99
(g)	mul8u_1A0M	2.88	34.69	10.99	5.45E+06	99
(h)	mul8u_Z9D	4.84	15.66	49.22	3.36E+07	89

Based on the given error metrics, it is hard to interpret the characteristics of each multiplier. With the visualization of the approximate multipliers as Figure 3.4, we find it useful to identify the characteristic of the approximate multipliers more intuitively. In Figure 3.4, eight selected multipliers as given in Table 3.1 are plotted in rows of orders from (a) to (h). The figures for each column are plotted with different types of error plots. It is calculated and plotted as follows.

C_i represents the column i^{th} where each figure is calculated according to the equation below. a_m and b_n are the two input values given in sequence from 0 to 255. The symbol $*$ represents the multiplication calculated by the given multiplier, while $t_{m,n}$ is the accurate output calculated by the exact multipliers. The last column C6 is the error histogram of the corresponding multiplier. From these error plots, a clearer view is provided.

$$C1_{m,n} = \begin{pmatrix} (a_0 * b_0) & (a_1 * b_0) & \cdots & (a_{255} * b_0) \\ (a_0 * b_1) & (a_1 * b_1) & \cdots & (a_{255} * b_1) \\ \vdots & \vdots & \ddots & \vdots \\ (a_0 * b_{255}) & (a_1 * b_{255}) & \cdots & (a_{255} * b_{255}) \end{pmatrix} \tag{3.7}$$

$$C2_{m,n} = \begin{pmatrix} t_{0,0} - a_0 * b_0 & \cdots & t_{255,0} - a_{255} * b_0 \\ t_{0,1} - a_0 * b_1 & \cdots & t_{255,1} - a_{255} * b_1 \\ \vdots & \ddots & \vdots \\ t_{0,255} - a_0 * b_{255} & \cdots & t_{255,255} - a_{255} * b_{255} \end{pmatrix} \tag{3.8}$$

$$C3_{m,n} = max(C2_{m,n}, 0) \tag{3.9}$$

$$C4_{m,n} = min(C2_{m,n}, 0) \tag{3.10}$$

$$C5_{m,n} = C2_{m,n}/t_{0,0} \tag{3.11}$$

If the gradient of pixels from blue to green in the first column C1 is smoother, it indicates higher accuracy for the multiplier, as there are no abrupt pixel changes. The C2 column is a combination of the second and third columns. C3 and C4 are the plots of the error distance between the exact multiplier and the approximate multiplier of the target. Column C3 shows the difference when the output of the approximate multiplier produces a smaller value. In comparison, column C4 shows the difference when the output of the approximate multiplier produces a larger value. From these two columns, we can visually see that these approximate multipliers show different characteristics locally. Some multipliers are more accurate when the input values are smaller, while others perform the opposite. Some multipliers have errors spread evenly among all pixels. Column C5 shows a relative error corresponding to each output of the exact multiplier. The relative error is quite small in some multipliers to be visible in the error plot. The last columns provide an error histogram of how errors are distributed compared to an exact multiplier. The graphical representation in the last columns can be used to detect patterns in the distribution of errors, such as identifying whether errors follow a normal distribution, determining the range of the error distribution, and identifying the presence of outliers.

In practical implementation, the outputs of each multiplier in the image scaling process are typically scaled by a scaling factor and then shifted by 8 bits after the multiplication, as mentioned later in Section 3.2. Consequently, in Figure 3.4, all the pixel values in the error plots are shifted by 256. This adjustment makes the plots more relevant to the target application and better estimates the differences between the pixel values and the actual output. By shifting the values, we can more accurately assess how the pixel values deviate from the desired output in the image scaling process.

In conclusion, error plots can be useful in identifying areas where a multiplier is performing well or poorly. For example, suppose the error is highlighted higher within a specific range. In that case, it may suggest that this multiplier should be avoided in some applications requiring more calculations in that particular area. This information can help identify suitable multipliers for a target application.

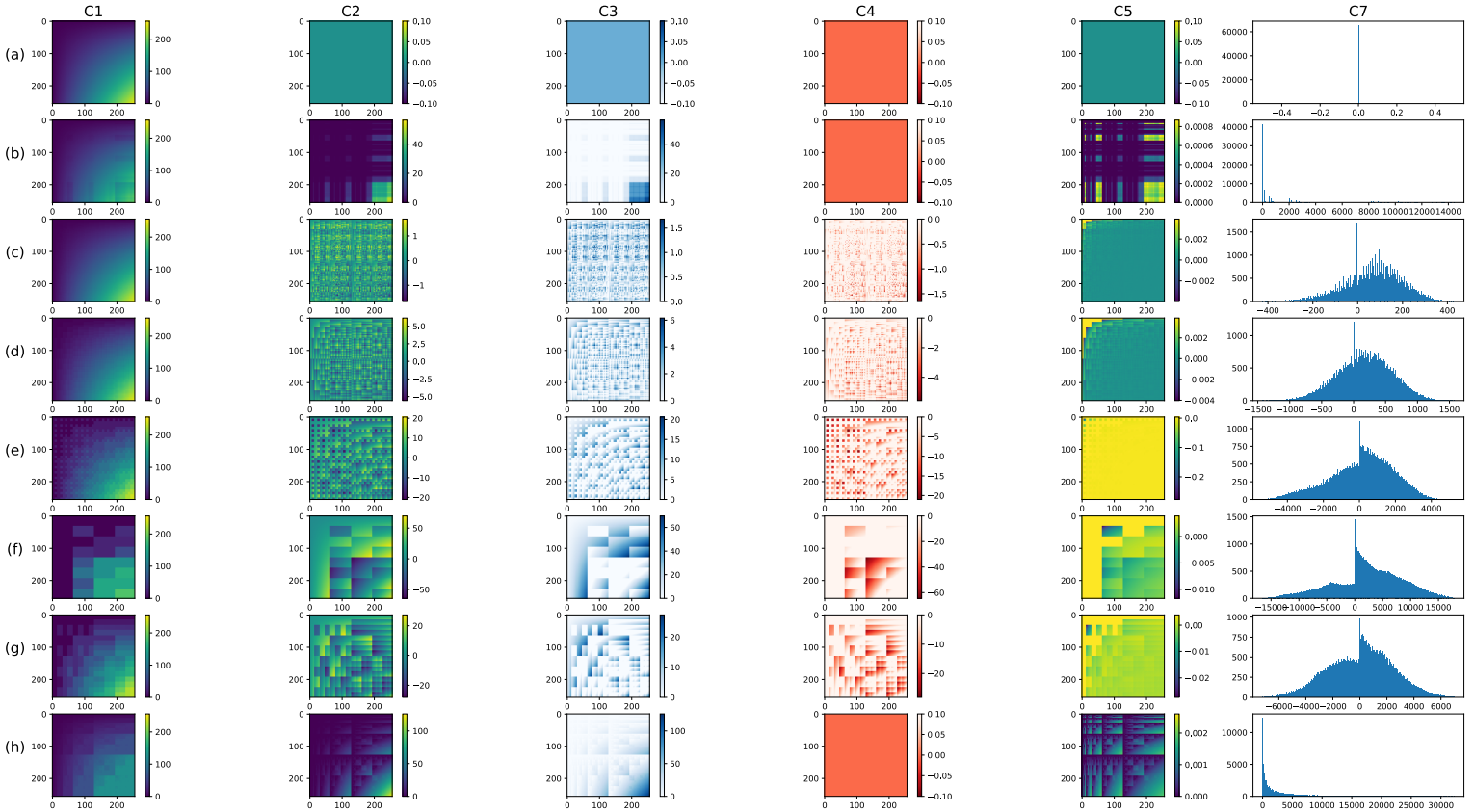


Figure 3.4: Visualization of errors in approximate multipliers with a shift of 256

3.4 System View

To evaluate the error tolerance of AACs applied in image applications, this thesis selects the Lanczos resampling algorithm as our target application.

3.4.1 Implementation of Lanczos scaling on hardware

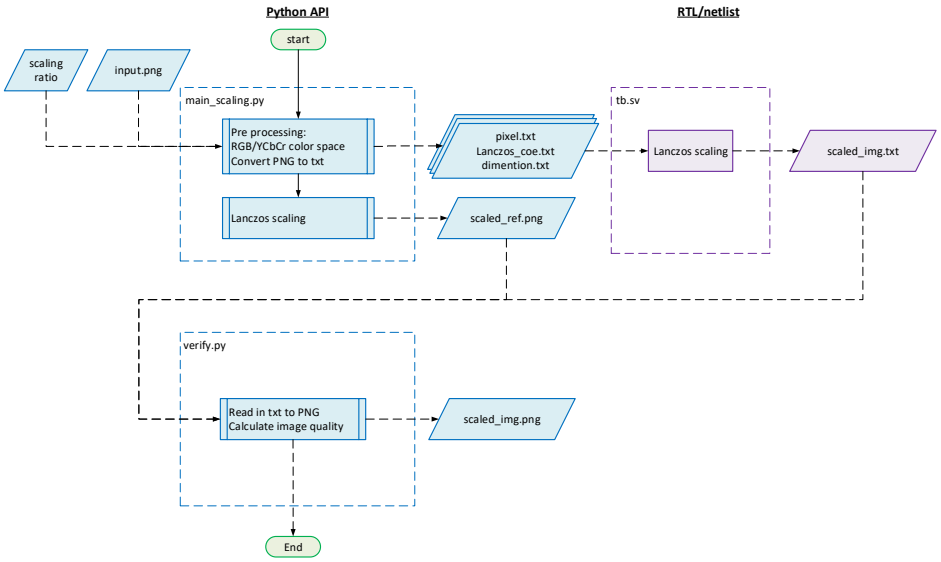


Figure 3.5: System diagram of scaling application

Block diagram of Lanczos scaling

The Lanczos scaling is implemented as Figure 3.5. We calculate the RGB/YCbCr image pixels and the Lanczos coefficients using the Python/C model. These values are then fed into the RTL design, where the scaling operation occurs. This scaling model is implemented in 4 pipeline stages as described in Figure 3.6. The pixel values are stored in memory, and 16 pixels are read at each clock cycle. There is a latency of four clock cycles with a throughput of 1.

The main components of the design are the Processing Element (PE) block and the accumulation block. The PE block includes the instantiation of a multiplier, either an exact or an approximate multiplier, which takes 8-bit pixel values as input and an 8-bit shifter to produce an 8-bit result, as shown in Figure 3.7. The accumulator block combines the four resulting 8-bit pixel values obtained from the Lanczos resampling process in the X and Y directions and checks if the value is in the pixel range of 0 to 255. If the value is outside this range, the block clamps the values to fit within the range. The result of the accumulator is the scaled value of

the pixels. All pixels are subjected to this process, and the scaled pixels are sent back to the Python-implemented module, which uses them to plot the final image.

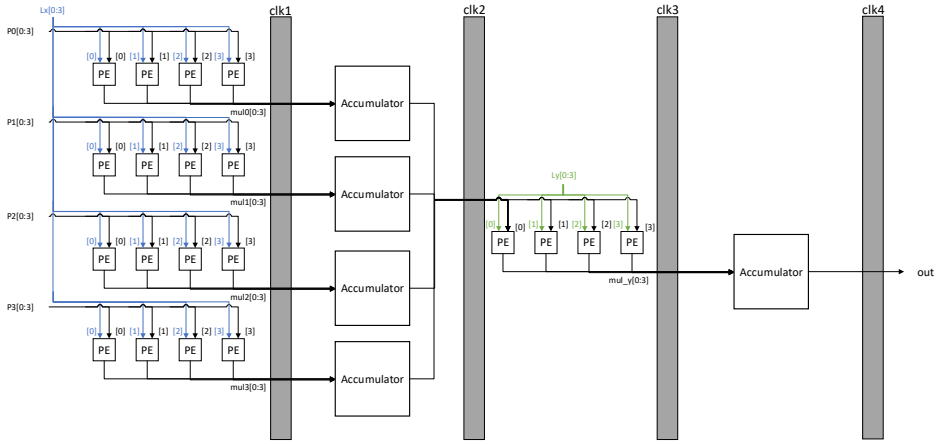


Figure 3.6: Block diagram of Lanczos scaling module

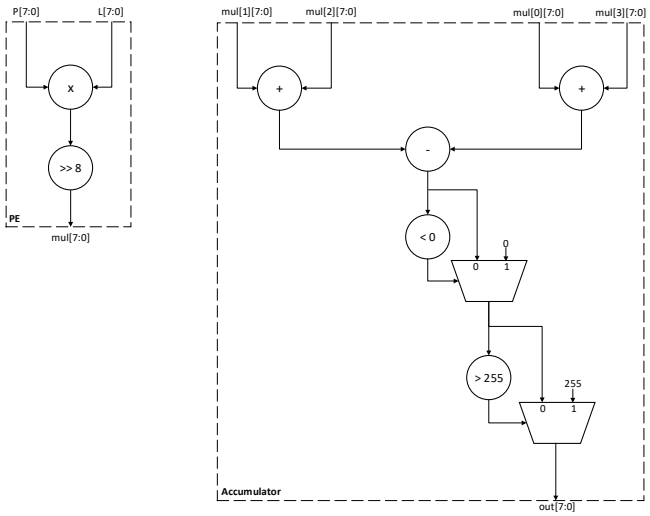


Figure 3.7: Block diagram of Lanczos scaling submodule

3.4.2 Power Calculation and Estimation Framework

The power analysis of a design for an image scaling application using various multipliers is performed. Table 3.2 displays the power ratios of each sub-module in the design when an exact multiplier is utilized. The findings indicate that the multipliers for the Lanczos scaling in the x direction consume the highest amount of power compared to the other sub-modules, as it requires 16 8-bit multipliers to perform computations per clock cycle.

Table 3.2: Sub-module power ratio of Lanczos scaling with an exact multiplier.

Sub-module	**Percentage(%)**
Register and others	24
Lanczos y	10
Lanczos x	55
Accumulate x	9
Accumulate y	2

The amount of power saved using approximate multipliers instead of exact multipliers can be estimated using (3.12). Table 3.3 provides the estimated power saving ratios of various approximate multipliers compared to exact multipliers, revealing a notable reduction in power consumption. Taking the exact multiplier as the reference, the approximate multiplier, such as 17C8 and 17MN, can save about 67% and 98% of the power, respectively, when compared to the exact multiplier. Therefore, a significant power saving is observed when various approximate multipliers are used.

However, it is crucial to consider the error characteristics of these multipliers when used in image applications to maintain acceptable image quality.

$$\text{Estimated power saving ratio} = \text{ratio}_{opt} \times (\text{power saving ratio}) + (1 - \text{ratio}_{opt}) \tag{3.12}$$

Table 3.3: Power saving ratio of approximate multipliers.

Multiplier	**Exact**	**UDM**	**197B**	**17C8**	**T83**	**Z9D**	**1A0M**	**17MN**
Ratio(%)	100	92	69	33	11	17	7	2

To further investigate, a comparison between the estimated and simulated ratios of these multipliers when used in image applications is made, and the results are discussed in Chapter 4.

Chapter 4

Results and Discussions

This chapter compares various AACs considering multiple aspects, including power consumption, image quality metrics, and error metrics. The objective is to identify approximate adders and multipliers that exhibit lower power consumption while maintaining acceptable error rate levels and image quality.

A selection of AACs is made to perform the comparison, and these circuits are evaluated using different test images for the specific application of image scaling. The power consumption of each AAC is measured, and error rates, as well as image quality metrics, are calculated.

4.1 Power Simulation of Approximate Arithmetic Circuits

In this thesis, a key aspect is evaluating the power consumption of AACs synthesized using the sub-10 nm technology. Evaluation involves the implementation of various approximate adders and multipliers discussed in Chapter 2 in Hardware Description Language (HDL) or importing circuits from EvoApproxLibs RTL source codes and synthesizing them using a synthesis tool in the given technology. Subsequently, the netlist obtained from the synthesis is used to estimate the power consumption.

To estimate the average power consumption, random inputs are provided to the netlist in each clock cycle, and a simulation is run for 1000 clock cycles at a clock frequency of 500 MHz. The power simulation tool then utilizes the generated FSDB file to calculate the average power consumption.

The power consumption of all the 8-bit approximate adders and multipliers optimized with various error metrics discussed in the thesis is calculated and compared against each other. This comparison allows one to assess the amount of power saved using AACs compared to the EACs.

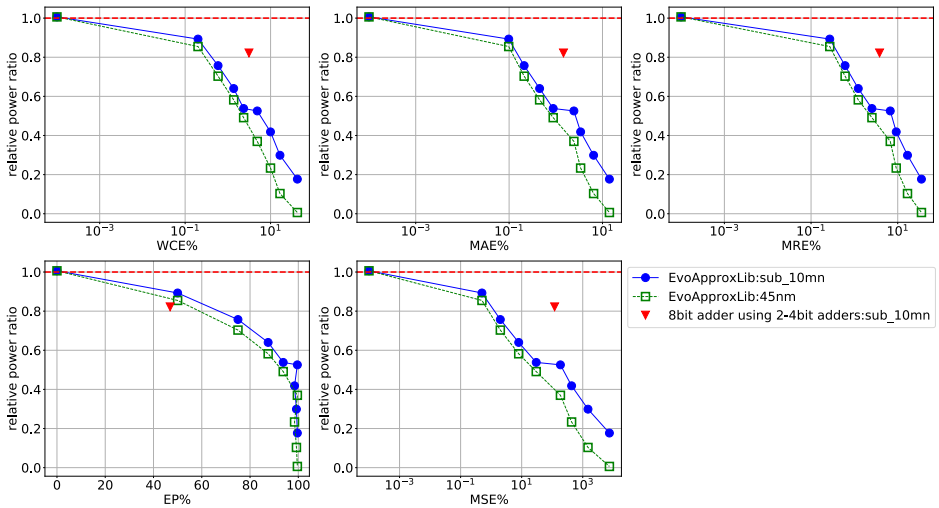
4.1.1 Power Simulation of Approximate Adders

Figures 4.1a to 4.1e represent the power ratios of different approximate adder circuits of the EvoapproxLib Library [5] that are optimized with different error metrics. The circuits are optimized with a 45 nm technology in the library, whereas sub-10 nm technology is used in this thesis for optimization. The exact adder is the baseline with a power ratio of 1, to which all other approximate adders are compared. The baseline is indicated with a red line across the exact adder circuit.

We can observe that the estimated result from the library and simulated results with sub-10 nm follow the same trend and have similar error metric values. However, the latter shows an increase in power consumption as the approximate adders are synthesized with a different technology than the one claimed by the EvoApproxLib library.

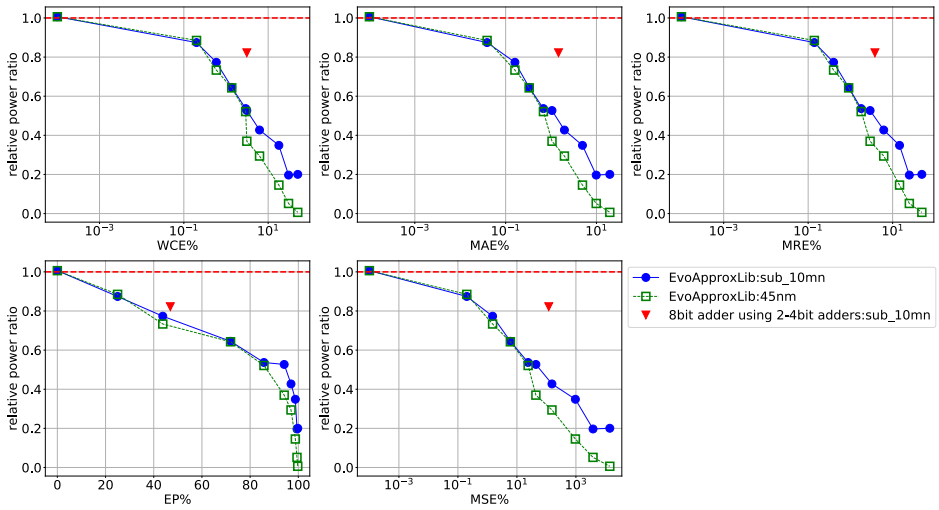
The plots also include an 8-bit segmented approximate adder designed using two 4-bit adders. But this approximate adder consumes much more power, almost 30% more than other circuits in EvoApproxLib, even though it has similar error values to the approximate adder from the library. The netlist report shows that one reason for this behaviour could be that the number of logic used in these circuits is more than that used in EvoApproxLib circuits.

From the graphs, we can observe a significant difference in the power consumption of the approximate adders as we move down the power ratio scale, as indicated in Figure 4.1a. The fact that circuits in this region have little to no logic in the design means that the synthesis tool adds several buffer cells for optimized design. This leads to more switching and high power consumption.

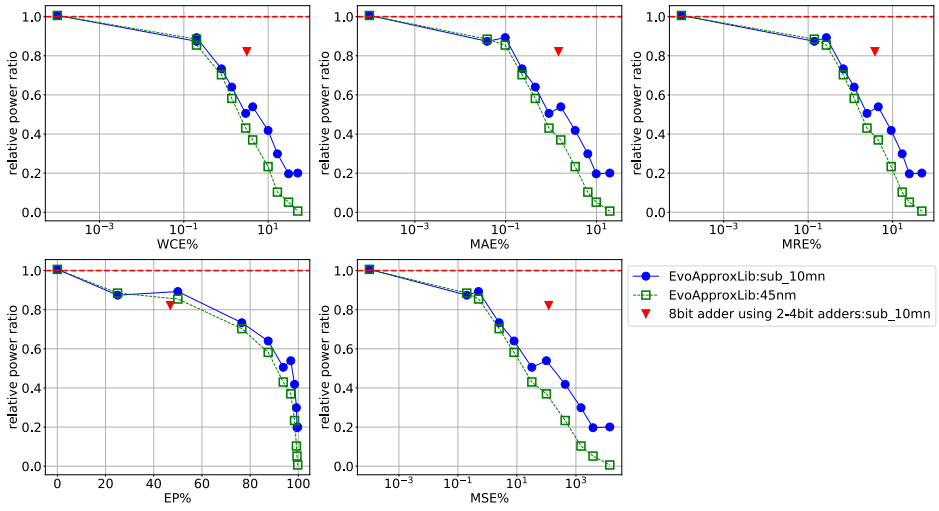


(a) Power simulation of approximate adders with WCE Optimization

Figure 4.1: Power simulation of approximate adders – EvoApproxLib versus power simulation with sub 10nm technology

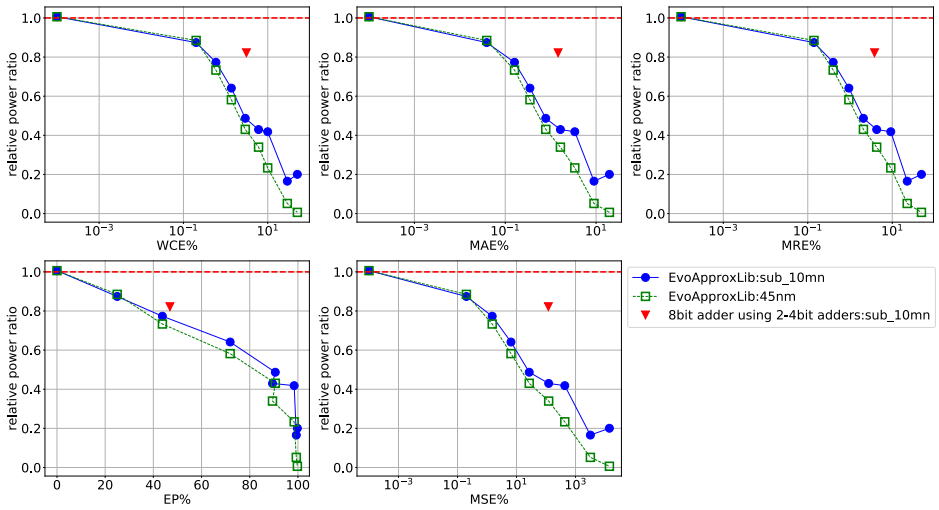


(b) Power simulation of approximate adders with MAE Optimization

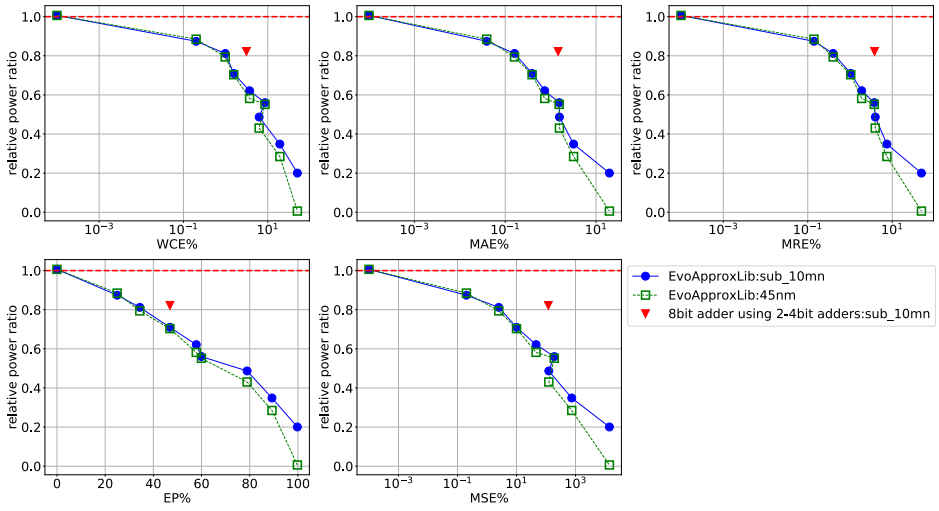


(c) Power simulation of approximate adders with MSE Optimization

Figure 4.1: Power simulation of approximate adders – EvoApproxLib versus power simulation with sub 10nm technology



(d) Power simulation of approximate adders with MRE Optimization



(e) Power simulation of approximate adders with EP Optimization

Figure 4.1: Power simulation of approximate adders – EvoApproxLib versus power simulation with sub 10nm technology

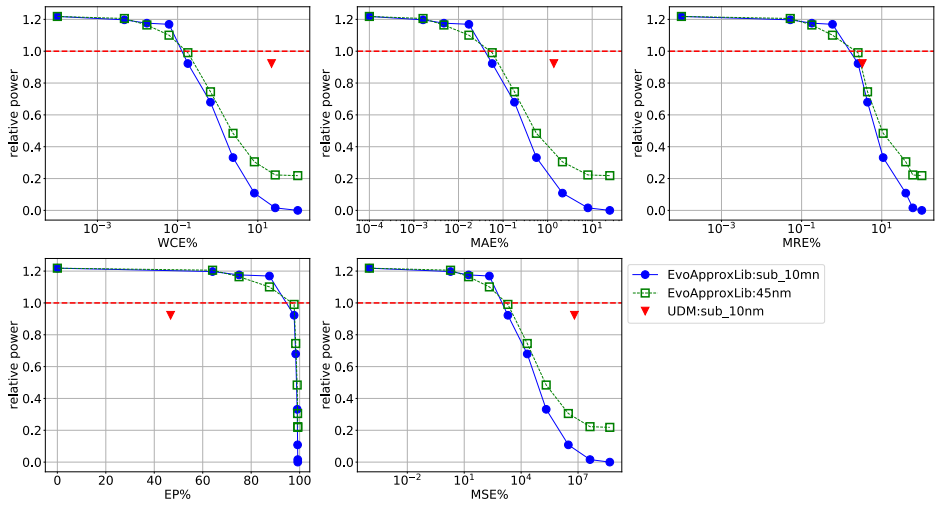
4.1.2 Power simulation of approximate multipliers

In the evaluation of the power ratios of different approximate multiplier circuits from the EvoApproxLib Library [5] optimized with different error metrics, Figures 4.2a to 4.2e provide insights into their power consumption. The baseline exact multiplier, which serves as the reference point, is implemented simply by utilizing a multiplication sign in the RTL code. This baseline exact multiplier is considered the benchmark, with a power ratio of 1, against which all other approximate

multipliers are compared.

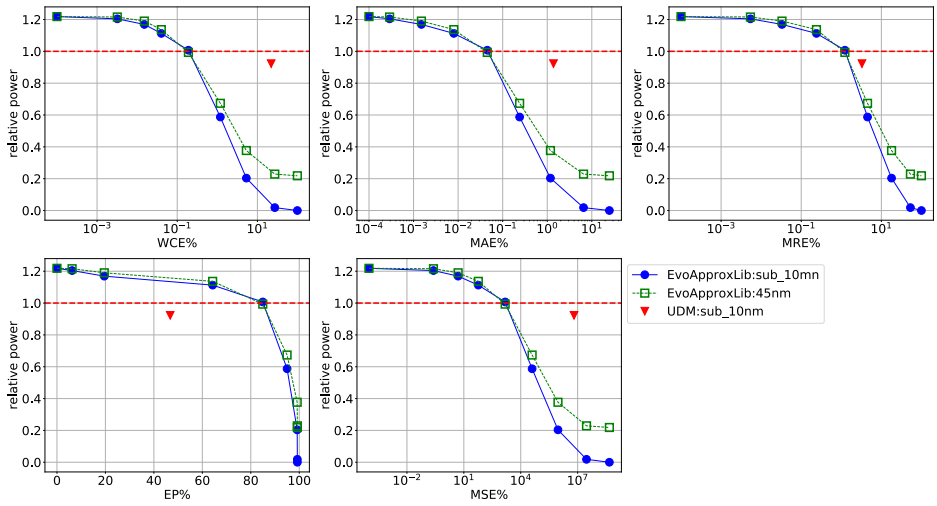
Interestingly, the exact multiplier implemented in the EvoApprox library consumes approximately 20% more power than the baseline exact multiplier. This discrepancy arises due to the complexity of the multiplier designs in the EvoApprox library, making it challenging for the synthesis tool to optimize the logic effectively. Consequently, the synthesis tool replaces the gates with available options from the standard cell library, assuming it to be an optimized design. However, this substitution process results in higher power consumption for all the approximate multipliers. It is worth noting that this behaviour is not observed in the case of approximate adders, as their designs are simpler and more amenable to effective synthesis optimization, leading to lower power consumption.

The approximate multipliers optimized with the EP error metric in Figure 4.2e show a slightly higher variation in power consumption when simulated with the given technology compared to the power claimed in the library. All other optimizations with other error metrics show a similar trend in power consumption as in the library. This is not the case with EP-optimized approximate adders in Figure 4.1e because the approximate multiplier has more complicated computations and logic than the approximate adders.

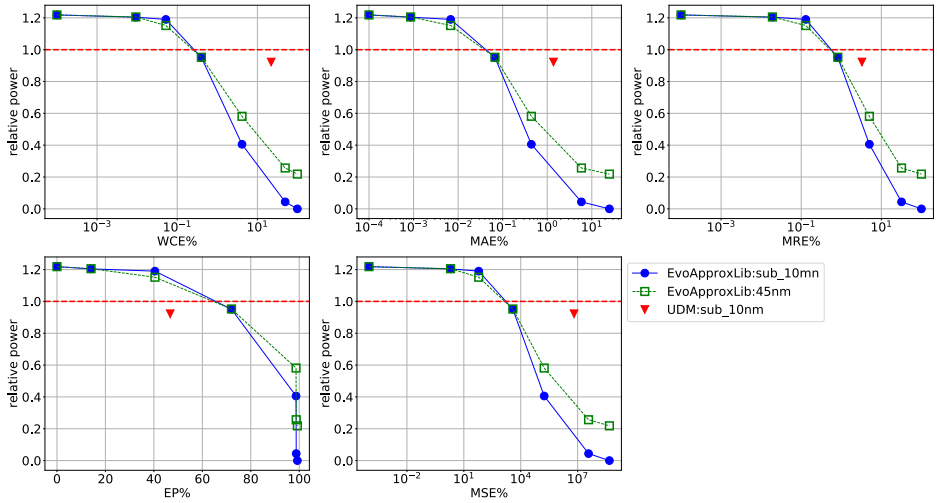


(a) Power simulation of approximate multipliers with WCE Optimization

Figure 4.2: Power simulation of approximate multipliers – EvoApproxLib versus power simulation with sub 10nm technology

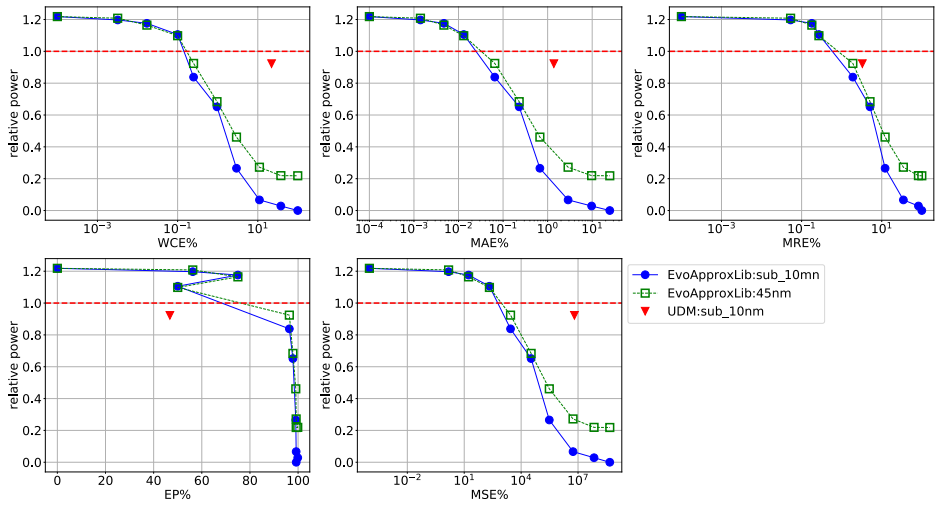


(b) Power simulation of approximate multipliers with MAE Optimization

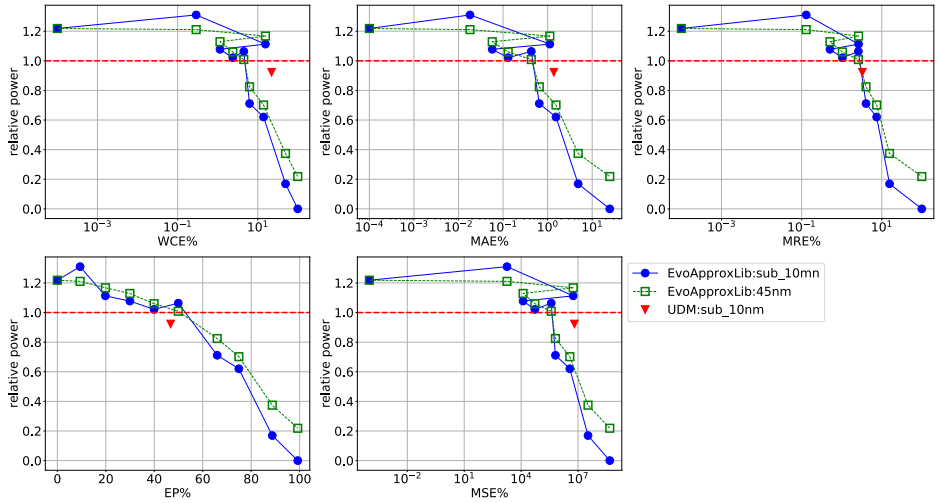


(c) Power simulation of approximate multipliers with MRE Optimization

Figure 4.2: Power simulation of approximate multipliers – EvoApproxLib versus power simulation with sub 10nm technology



(d) Power simulation of approximate multipliers with MSE Optimization



(e) Power simulation of approximate multipliers with EP Optimization

Figure 4.2: Power simulation of approximate multipliers – EvoApproxLib versus power simulation with sub 10nm technology

The comparison baseline is set as 1 for the exact multiplier, and all other approximate multipliers are evaluated relative to this baseline. Figure 4.3 provides a detailed analysis of the power consumption of different approximate multipliers optimized with MAE error metrics. In our image scaling application, we employ MAE as an error metric to select the best approximate multipliers for the image scaling application. We choose MAE because it is highly relevant to SSIM, which will be further discussed in Subsection 4.2.2.

Figure 4.3 highlights that some multipliers consume more power than the exact multiplier. From a careful analysis of the netlist report, it was found that the baseline that the library considers is their exact multiplier design, which, when compared with the exact multiplier of our implementation, used more logic gates. Ideally, both exact multiplier designs should consume a similar number of logic cells. This leads to the higher power consumption of a few approximate multipliers compared to the baseline we set from the given technology. A more detailed analysis of the netlist report is available in Subsection 4.1.2. Therefore, only multipliers below the power ratio 1 are considered for inclusion in image applications. In the following sections, the synthesis results of different approximate multipliers are compared with the exact multiplier to check how the optimization affects the number of logic gates in the design and hence, the overall power consumption. Later, we will also examine how image quality varies depending on the power consumption of the selected approximate multipliers.

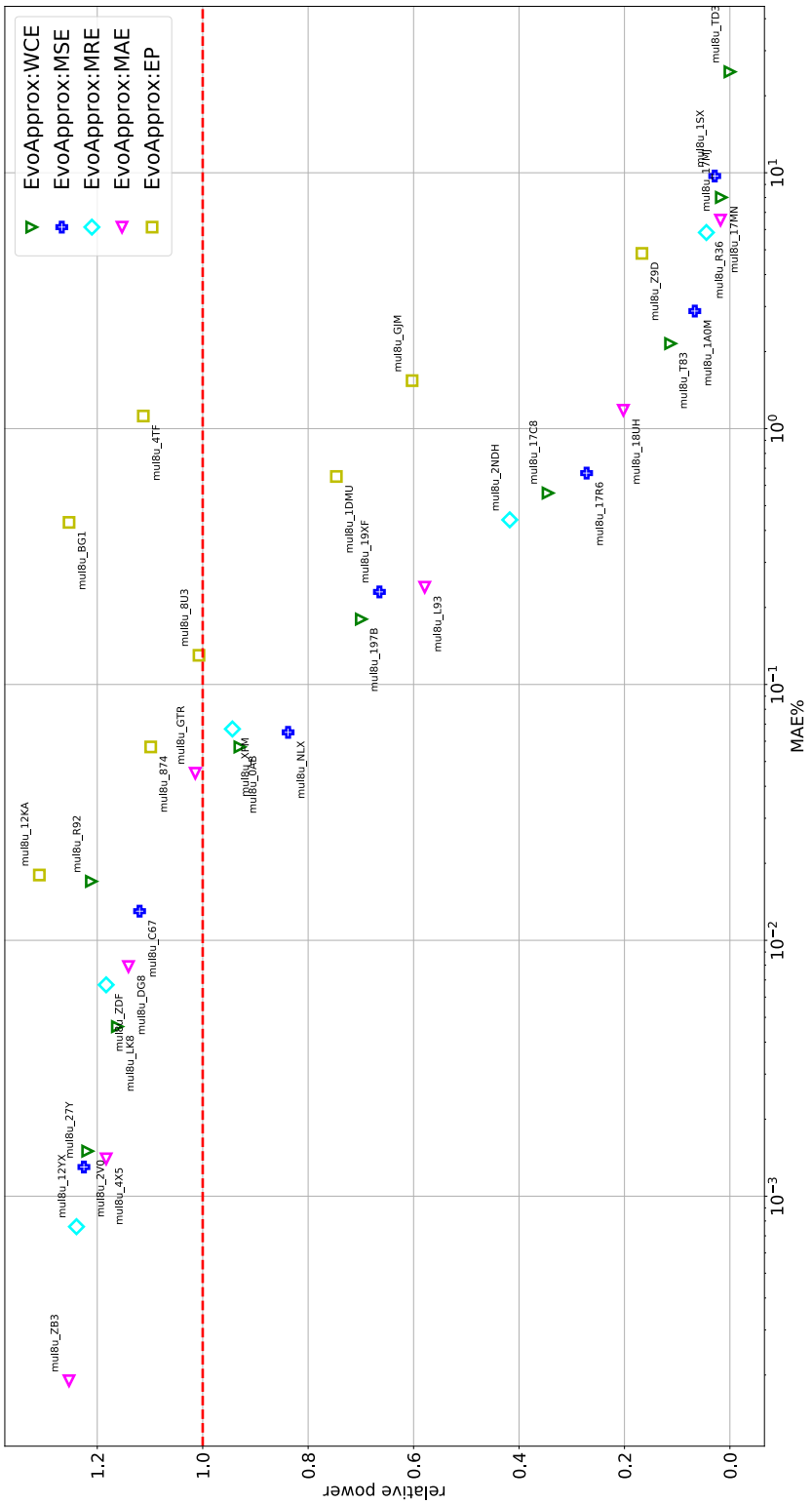


Figure 4.3: Power simulation of approximate multipliers – sub 10nm technology- MAE

Investigating Synthesis Report

Synthesis is performed on various approximate adders and multipliers from the EvoApproxLib library, and their power consumptions are compared with their respective power numbers claimed by the library. It was found that some circuits in the library did not demonstrate any reduction in power consumption, as claimed. Figure 4.3 shows that some approximate multiplier circuits consume more power than the exact multiplier. In particular, when analyzing a netlist report of two approximate multipliers with similar worst-case errors but significantly different power consumption, it was found that these approximate multipliers contained more logic cells than the exact multiplier, as shown in Table 4.1. Theoretically, the exact multiplier should require more logic cells, but the opposite was observed. Consequently, this discrepancy leads to increased cell switching and higher power consumption.

Table 4.1: Logic comparison between exact and approximate multipliers from netlist report

Cell Type	Exact	Exact mul_JJQ	Approximate mul_12KA	Approximate mul_NLX
Full Adder	46	33	30	46
Inverters	16	22	33	16
Other cells	81	216	211	86
Total	**143**	**271**	**274**	**148**

4.2 Evaluation of Image Quality and Error Metrics

This chapter presents the results of implemented Lanczos resampling with a selection of 8-bit unsigned multipliers. The Lanczos resampling process is performed on images in RGB and YCbCr colour spaces. The overall image quality evaluation of this study is based on commonly used image quality indexes such as MSE, PSNR, and SSIM. These metrics provide objective measures of image quality, with SSIM being particularly relevant as it incorporates aspects of HVS. In addition, an investigation is conducted to identify any correlation between image quality and error metrics specifically for the Lanczos resampling application.

Using the conclusions drawn from this analysis and the utilization of error plots, a guideline is proposed for selecting 8-bit unsigned multipliers. This guide aims to help decision-making when choosing appropriate multipliers that strike a balance between power savings and maintaining acceptable image quality in the context of Lanczos resampling.

4.2.1 Image Scaling Results

This section presents the scaling results obtained by implementing Lanczos resampling using a range of approximate multipliers with different power-saving ratios. Using a combination of C and Python models, we tested all approximate multipliers selected in this thesis to assess their effect on the quality of images scaled with Lanczos resampling, especially using SSIM as an image quality assessment score.

To present the findings, we selected approximate multipliers for each power-saving range. The selection was based on power simulations performed on the netlists of the approximate multipliers. For each power saving range of approximately 20%, an approximate multiplier with the highest SSIM was chosen within that range. However, it should be noted that there was no approximate multiplier available for the power savings ratio of around 50% after synthesis with our selected technology node. The selected approximate multipliers and the corresponding power savings number are in Table 3.3.

By focusing on a reduced set of approximate multipliers with different power-saving ratios, our objective was to analyze and highlight the impact of these ratios on the scaling results achieved through Lanczos resampling. This approach allows for a more focused investigation of how power-saving ratios influence image quality.

In this study, two input images are used. The first image is a standard test image called "Baboon". This image was chosen for its diverse colour range, including red, green, and blue variations, and for many details, such as the texture of the baboon's fur. The second image is a surveillance image captured by the authors at Lund University. Selecting this image provides a real-world use case scenario closely resembling the conditions faced by network cameras. Using these two testing images, we can observe how they are affected by the different error characteristics of the approximate multipliers.

The original size of both images is 512 x 512 pixels, and a consistent downscale is applied, resulting in a final size of 372 x 372 pixels. To ensure a fair comparison, the downscaling process incorporates a grid size of 16, a step size of 22 upscaled pixel units, and an offset of 7 upscaled pixel units. The results of Lanczos resampling are split into two sections, images in RGB and YCbCr colour spaces, to determine which colour space retains better image quality after scaling.

Image Scaling with RGB Color Space

In Figures 4.4 and 4.5, the individual results of the scaling using seven approximate multipliers are presented. Each multiplier is evaluated separately to assess its impact on the scaled output, compared to the output achieved with an exact multiplier. A zoomed-in section of each image is included in the figures to provide better visibility. Image quality evaluation is performed using MSE, PSNR, and SSIM metrics. Based on the results in Figures 4.4 and 4.5, together with the image quality scores presented in Tables 4.2 and 4.3 many observations can be made.

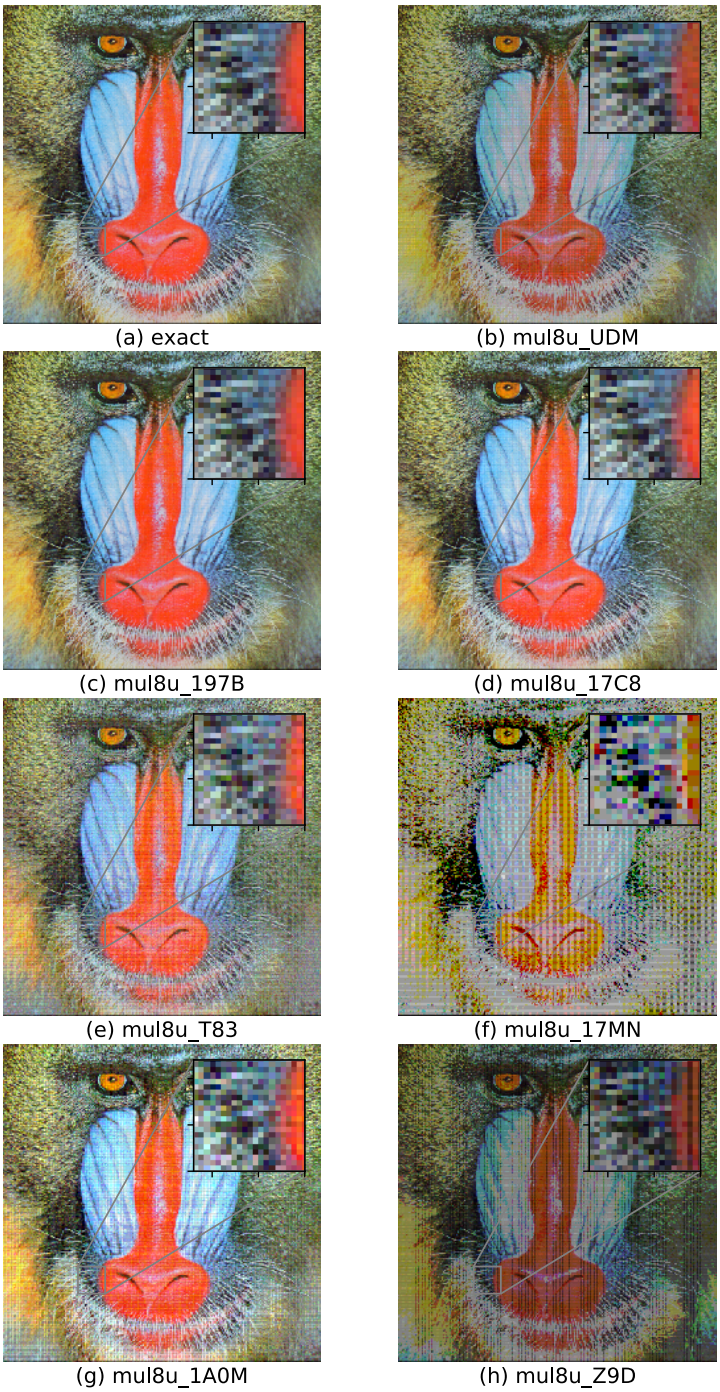
Table 4.2: Test image in RGB colour space: Lanczos resampling with approximate multipliers

Index	Multiplier Name	MSE	PSNR(dB)	SSIM
(a)	exact	0.0	inf	1.0
(b)	mul8u_UDM	48.621	31.26	0.864
(c)	mul8u_197B	0.798	49.11	0.993
(d)	mul8u_17C8	3.109	43.20	0.967
(e)	mul8u_T83	29.76	33.39	0.685
(f)	mul8u_17MN	58.202	30.48	0.416
(g)	mul8u_1A0M	20.663	34.98	0.717
(h)	mul8u_Z9D	112.221	27.63	0.572

Table 4.3: Survalliance image in RGB colour space: Lanczos resampling with approximate multipliers

Index	Multiplier Name	MSE	PSNR(dB)	SSIM
(a)	exact	0.0	inf	1.0
(b)	mul8u_UDM	62.669	30.16	0.784
(c)	mul8u_197B	0.839	48.89	0.983
(d)	mul8u_17C8	3.272	42.98	0.930
(e)	mul8u_T83	35.246	32.66	0.585
(f)	mul8u_17MN	80.633	29.07	0.366
(g)	mul8u_1A0M	23.254	34.47	0.607
(h)	mul8u_Z9D	108.284	27.79	0.494

- SSIM is a more useful metric in this application to evaluate image quality. Multipliers (c) and (d) achieve satisfactory image results, with SSIM scores greater than 0.9, PSNR values exceeding 40dB, and MSE within 5. Multipliers (b), (e), and (g) also yield acceptable image results, with SSIM scores ranging from 0.6 to 0.8. However, the MSE and PSNR values do not align consistently with SSIM in these cases. Multiplier (b) introduces more noise into the image, whereas multipliers (e) and (f) exhibit sharp lines in scaled results, making image quality look worse, as suggested by SSIM. Lastly, multiplier (f) and multiplier (h) display poor image quality in Figures 4.4 and 4.5, characterized by the presence of a significant number of unwanted lines in the images. Multiplier (h) produces a higher number of MSE, while SSIM is better than the multiplier (f) result. This stresses the importance of considering the characteristics of the HVS when evaluating image quality rather than relying solely on pixel-level errors. We can see that in this application, SSIM is a better metric to evaluate overall image quality.



(a) exact (b) mul8u_UDM

(c) mul8u_197B (d) mul8u_17C8

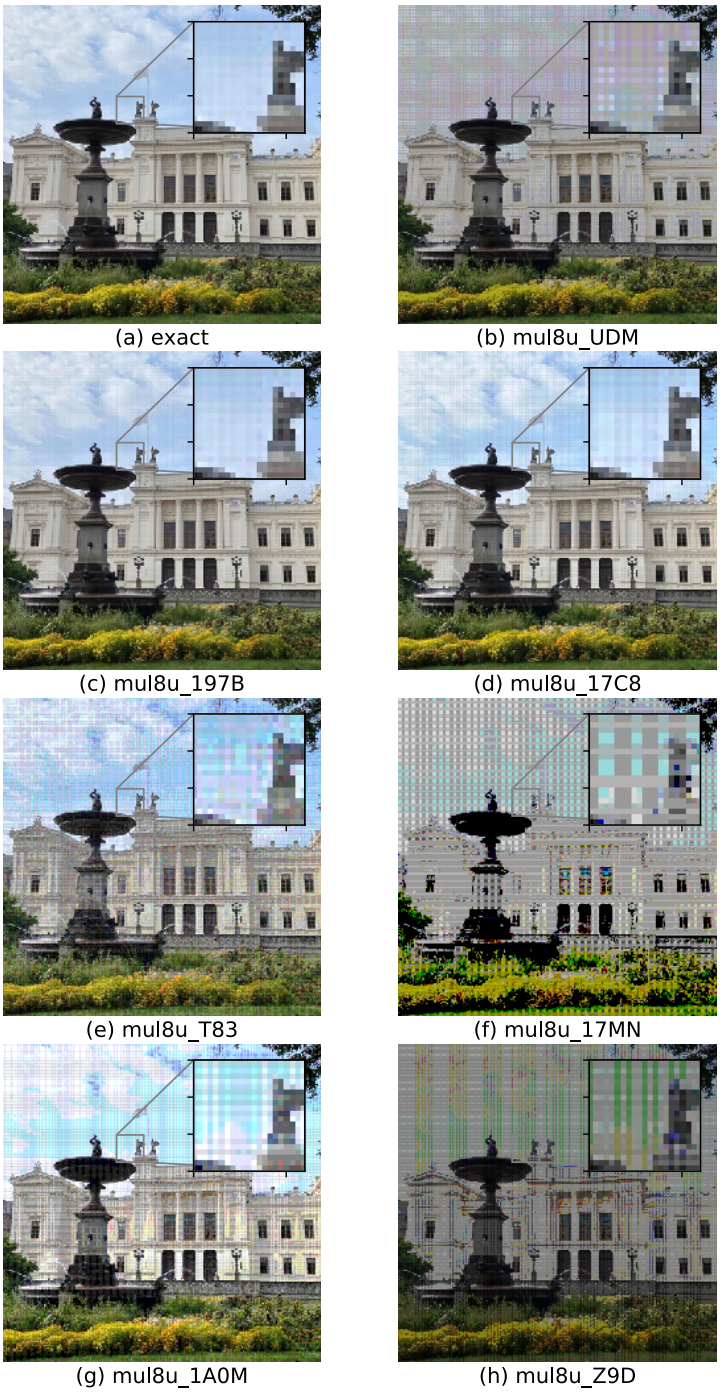
(e) mul8u_T83 (f) mul8u_17MN

(g) mul8u_1A0M (h) mul8u_Z9D

Figure 4.4: Test image in RGB colour space: Lanczos resampling with approximate multipliers

- The scaling results obtained using different multipliers show varying brightness. Multipliers (b) and (h) produce darker results compared to the multiplier (a) in Figures 4.4 and 4.5. On the contrary, multiplier (g) produces brighter results. Despite the brightness differences, the higher SSIM score of multiplier (b) suggests that image quality is not solely determined by brightness but also by image structure and contrast. The brightness observations align with the error plots in Figure 3.4, discussed in Section 3.3. The error plot visualizes errors in columns C3 and C4, which explain the brightness variations. Column C3 shows smaller errors than the exact multiplier, indicated by darker blue shades. In contrast, column C4 shows larger errors, indicated by darker red shades. Examining the error plot reveals the relationship between errors and brightness differences in the scaling results. Figures 4.4 and 4.5 show that images (b) and (h) have errors in C3 but not in C4, indicating that the multipliers used for these images produce outputs equal to or smaller than the exact multiplier. Consequently, these images appear darker. In contrast, image (g) employs a multiplier with errors in both C3 and C4, resulting in higher contrast than the exact multiplier.
- SSIM image quality presents a higher value when the error plots in C2 show that the maximum and minimum error values are lower. Image (f) also displays errors in both C3 and C4. However, the resulting image is unacceptable due to significantly higher error values, exceeding ± 60, as reflected in the low SSIM scores of around 0.3 and 0.4, depending on the input images.

An error plot is a valuable tool for visualizing and understanding the error characteristics among different multipliers and how they contribute to variations in brightness in the scaled images. In this particular application, we can conclude that SSIM is a better metric for evaluating image quality, and a multiplier with a smaller error range in the C2 plot is more desirable.



(a) exact (b) mul8u_UDM

(c) mul8u_197B (d) mul8u_17C8

(e) mul8u_T83 (f) mul8u_17MN

(g) mul8u_1A0M (h) mul8u_Z9D

Figure 4.5: Survalliance image in RGB colour space: Lanczos resampling with approximate multipliers

Image Scaling with YCbCr Color Space

Given that the human eye is more sensitive to luminance than chroma, our objective is to identify an AAC that produces better image quality when scaling the pixels in the luminance component (Y) while allowing more error tolerance when selecting approximate multipliers to use in the chroma components (Cb and Cr). Based on this idea, we can further evaluate the scaling process using a combination of different approximate multipliers.

Based on the results obtained from the image scaling using the RGB colour space, we can conclude that AACs 197B and 17C8 consistently maintain higher image quality after scaling. Consequently, in this section, we investigate the feasibility of utilizing these AACs specifically for the luminance component (Y) and further examine the approximate results achieved when combined with other chroma AACs.

First, we compare Lanczos scaling with exact multipliers for Y channels, while Cb and Cr colour channels use approximate multipliers, with Lanczos scaling with approximate multipliers for all three channels. From Figure 4.6 to Figure 4.8 together with Table 4.4 to Table 4.6, no obvious image quality drop is found in the test image. The same results are observed in the surveillance image from Figure 4.6 to Figure 4.8 and from Table 4.4 to Table 4.6. Second, we also observed that the SSIM in all of these tables appears almost all higher than 0.9, which is quite close to what we observed from bare eyes, as the overall image quality looks good even though in some of the images, the chroma is almost zero after being scaled with the multiplier (f) and (h). However, this proves our assumption that luminance has a greater impact on image quality. While it is interesting to notice here that MSE and PSNR metrics show a large amount of noise just by switching from exact multiplier to approximate multiplier (c) or to approximate multiplier (d) since these metrics calculate the error distances among pixels while ignoring HVS.

Based on different input images, such as test images and surveillance images, we can observe that, for different approximate multipliers, the performance of scaling different pixel values may differ. For example, in Figure 4.10 (e) and Figure 4.10 (g), although there is an obvious purple shade in the zoomed-in parts of the image, it still maintains good image quality.

To conclude, we observe that using a mixture of multiplier (c) or multiplier (d) together with some multipliers, such as (e) and (f), we can still achieve higher image quality where SSIM is greater than 0.9.

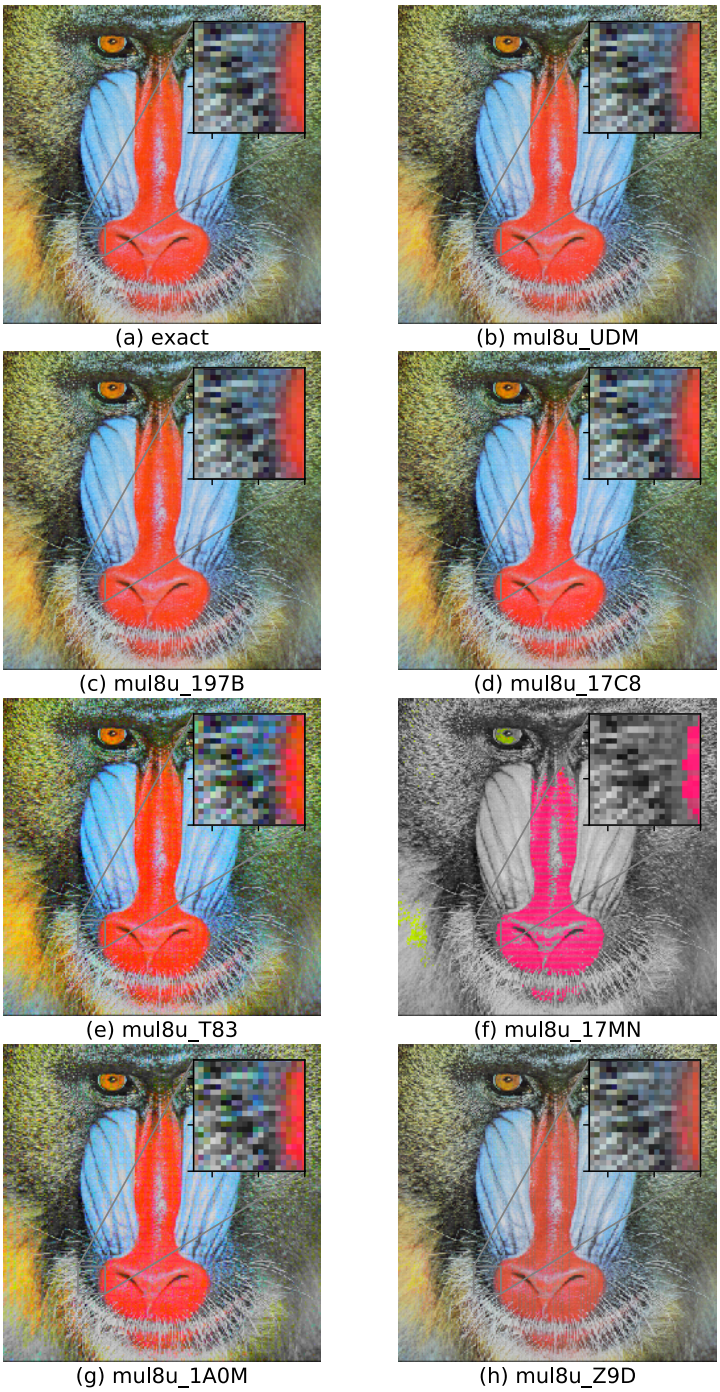


Figure 4.6: Test image in YCbCr colour space: Lanczos resampling with Y using an exact multiplier, CbCr using approximate multipliers (b) to (h)

Table 4.4: Test image in YCbCr colour space: Lanczos resampling with Y using an exact multiplier, CbCr using approximate multipliers (b) to (h)

Index	Multiplier Name	MSE	PSNR(dB)	SSIM
(a)	exact	0.013	66.99	1.0
(b)	mul8u_UDM	50.05	31.14	0.996
(c)	mul8u_197B	24.378	34.26	0.998
(d)	mul8u_17C8	107.891	27.8	0.991
(e)	mul8u_T83	890.421	18.63	0.911
(f)	mul8u_17MN	619.815	20.21	0.912
(g)	mul8u_1A0M	662.652	19.92	0.923
(h)	mul8u_Z9D	455.35	21.55	0.97

Table 4.5: Test image in YCbCr colour space: Lanczos resampling with Y using 197B approximate multiplier, CbCr using approximate multipliers (b)-(h)

Index	Multiplier Name	MSE	PSNR(dB)	SSIM
(a)	exact	70.343	29.66	0.995
(b)	mul8u_UDM	120.38	27.33	0.991
(c)	mul8u_197B	94.708	28.37	0.993
(d)	mul8u_17C8	178.221	25.62	0.986
(e)	mul8u_T83	960.751	18.3	0.906
(f)	mul8u_17MN	690.144	19.74	0.906
(g)	mul8u_1A0M	732.982	19.48	0.917
(h)	mul8u_Z9D	525.68	20.92	0.965

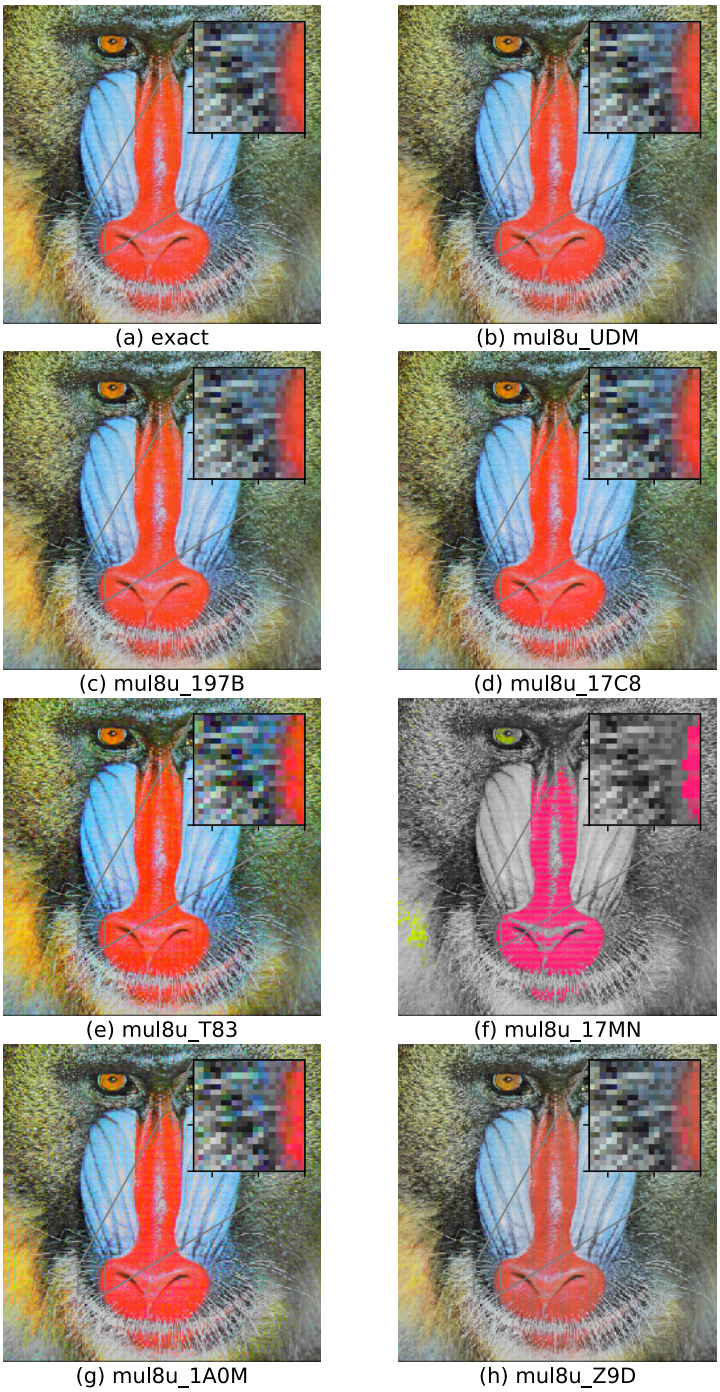
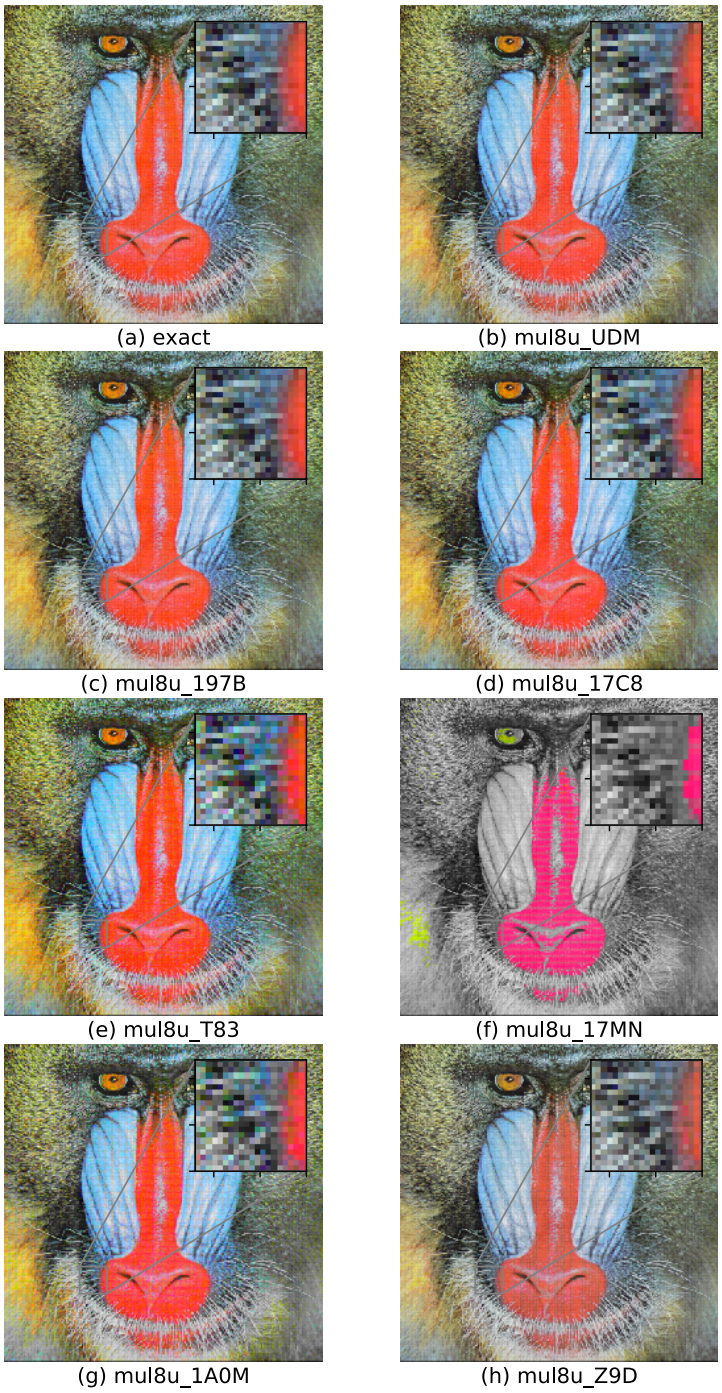


Figure 4.7: Test image in YCbCr colour space: Lanczos resampling with Y using 197B approximate multiplier, CbCr using approximate multipliers (b) to (h)



(a) exact (b) mul8u_UDM

(c) mul8u_197B (d) mul8u_17C8

(e) mul8u_T83 (f) mul8u_17MN

(g) mul8u_1A0M (h) mul8u_Z9D

Figure 4.8: Test image in YCbCr colour space: Lanczos resampling with Y using 17C8 approximate multiplier, CbCr using approximate multipliers (b) to (h)

Table 4.6: Test image in YCbCr colour space: Lanczos resampling with Y using 17C8 approximate multiplier, CbCr using approximate multipliers (b) to (h)

Index	Multiplier Name	MSE	PSNR(dB)	SSIM
(a)	exact	333.304	22.9	0.97
(b)	mul8u_UDM	383.341	22.29	0.966
(c)	mul8u_197B	357.669	22.6	0.968
(d)	mul8u_17C8	441.182	21.68	0.962
(e)	mul8u_T83	1223.712	17.25	0.882
(f)	mul8u_17MN	953.106	18.34	0.882
(g)	mul8u_1A0M	995.943	18.15	0.893
(h)	mul8u_Z9D	788.641	19.16	0.941

Table 4.7: Survalliance image in YCbCr colour space: Lanczos resampling with Y using an exact multiplier, CbCr using approximate multipliers (b) to (h)

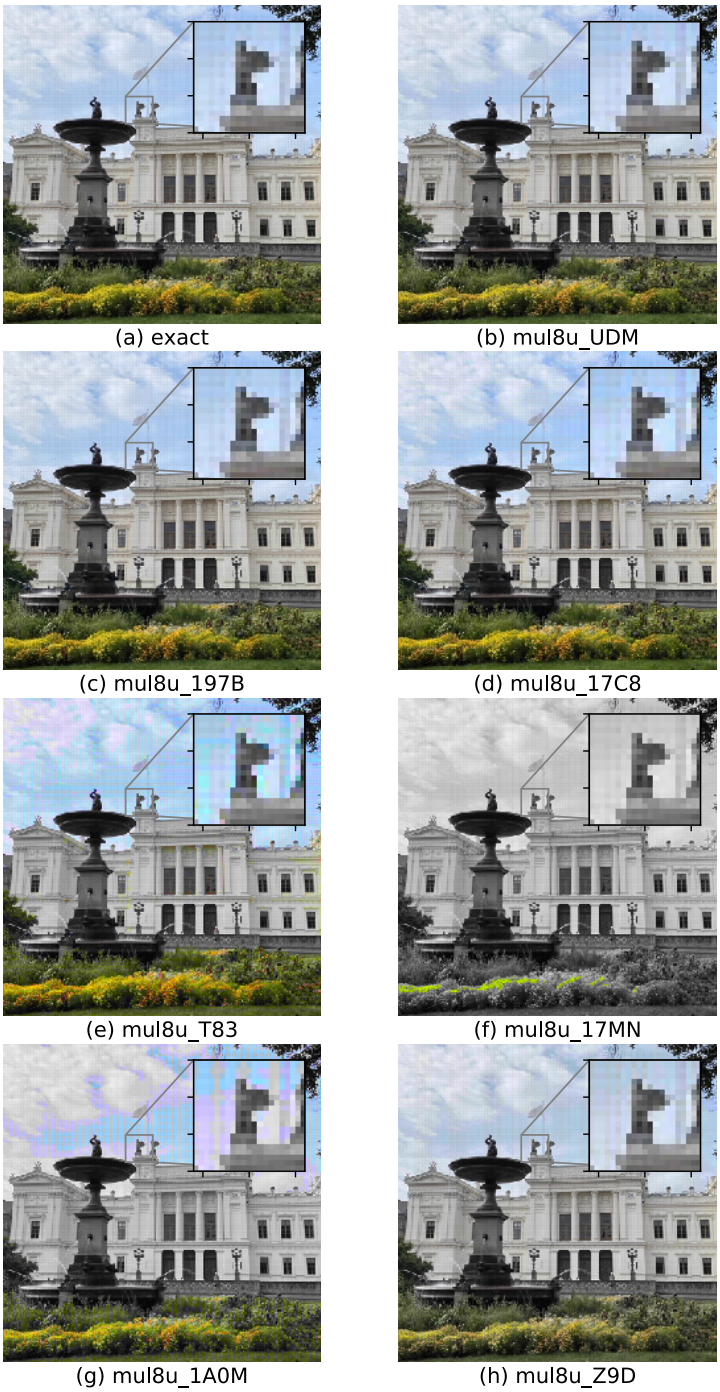
Index	Multiplier Name	MSE	PSNR(dB)	SSIM
(a)	exact	0.204	55.03	1.0
(b)	mul8u_UDM	14.224	36.6	0.998
(c)	mul8u_197B	18.95	35.35	0.997
(d)	mul8u_17C8	98.187	28.21	0.99
(e)	mul8u_T83	334.771	22.88	0.941
(f)	mul8u_17MN	675.689	19.83	0.985
(g)	mul8u_1A0M	638.927	20.08	0.952
(h)	mul8u_Z9D	284.455	23.59	0.989

Table 4.8: Survalliance image in YCbCr colour space: Lanczos resampling with Y using 197B approximate multiplier, CbCr using approximate multipliers (b) to (h)

Index	Multiplier Name	MSE	PSNR(dB)	SSIM
(a)	exact	74.61	29.4	0.987
(b)	mul8u_UDM	88.63	28.65	0.985
(c)	mul8u_197B	93.357	28.43	0.984
(d)	mul8u_17C8	172.593	25.76	0.977
(e)	mul8u_T83	409.178	22.01	0.927
(f)	mul8u_17MN	750.095	19.38	0.972
(g)	mul8u_1A0M	713.334	19.6	0.939
(h)	mul8u_Z9D	358.862	22.58	0.975

Table 4.9: Survalliance image in YCbCr colour space: Lanczos resampling with Y using 17C8 approximate multiplier, CbCr using approximate multipliers (b) to (h)

Index	Multiplier Name	MSE	PSNR(dB)	SSIM
(a)	exact	294.498	23.44	0.936
(b)	mul8u_UDM	308.518	23.24	0.934
(c)	mul8u_197B	313.244	23.17	0.933
(d)	mul8u_17C8	392.481	22.19	0.926
(e)	mul8u_T83	629.065	20.14	0.877
(f)	mul8u_17MN	969.983	18.26	0.921
(g)	mul8u_1A0M	933.221	18.43	0.888
(h)	mul8u_Z9D	578.749	20.51	0.925



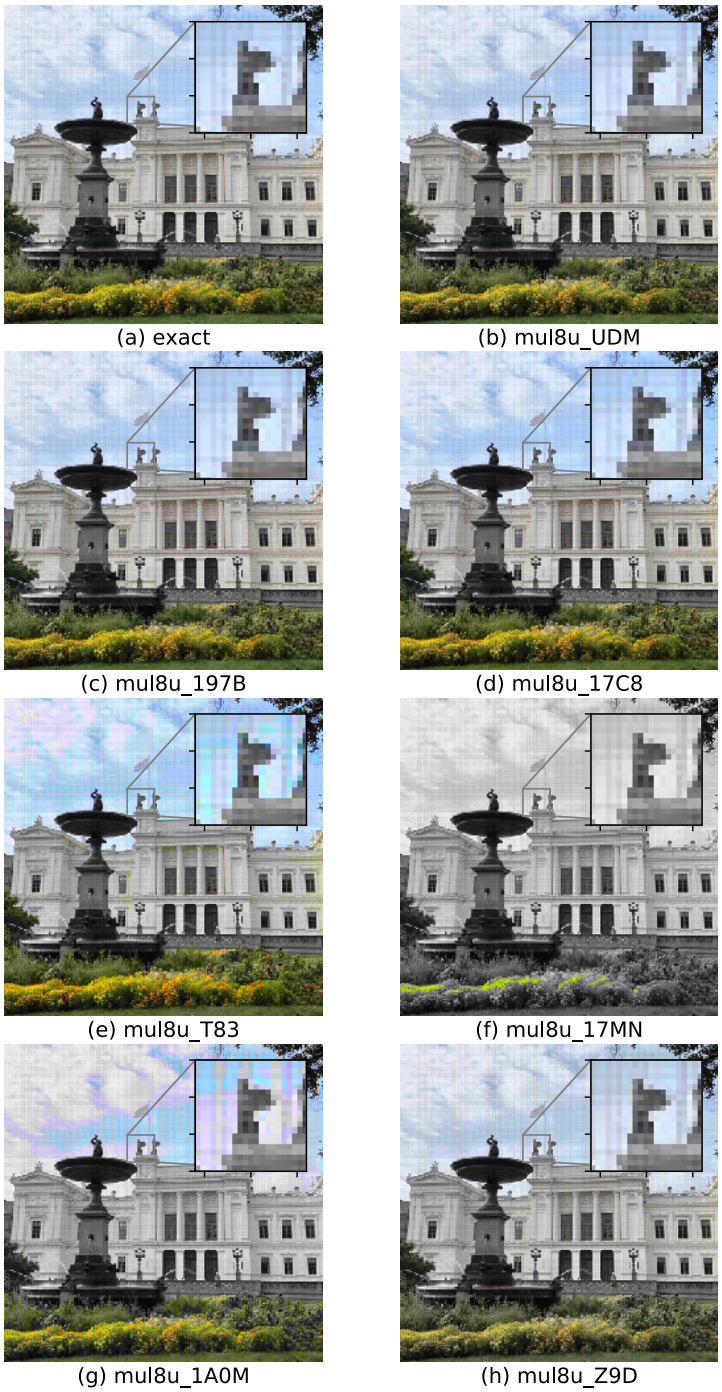
(a) exact (b) mul8u_UDM

(c) mul8u_197B (d) mul8u_17C8

(e) mul8u_T83 (f) mul8u_17MN

(g) mul8u_1A0M (h) mul8u_Z9D

Figure 4.9: Survalliance image in YCbCr colour space: Lanczos resampling with Y using 197B approximate multiplier, CbCr using approximate multipliers (b) to (h)



(a) exact (b) mul8u_UDM

(c) mul8u_197B (d) mul8u_17C8

(e) mul8u_T83 (f) mul8u_17MN

(g) mul8u_1A0M (h) mul8u_Z9D

Figure 4.10: Survalliance image in YCbCr colour space: Lanczos resampling with Y using 17C8 approximate multiplier, CbCr using approximate multipliers (b) to (h)

4.2.2 Arithmetic Accuracy versus Application Specific Metrics

In this section, we try to find a correlation between the error metrics and the SSIM image quality metric. For the YCbCr colour space, since there are several approximate multipliers, it is more difficult to evaluate. So we discuss only image scaling in RGB colour space. A total of 20 approximate multipliers show a power saving ratio below 1 in EvoApproxLib, which are discussed here. Three common coefficient correlation methods are used here, including Pearson's coefficient correlation, Spearman's coefficient correlation, and Kendall's coefficient correlation, to correlate error metrics.

We can see that MAE is the most relevant among all coefficient correlation methods in general. EP is the least relevant error metric.

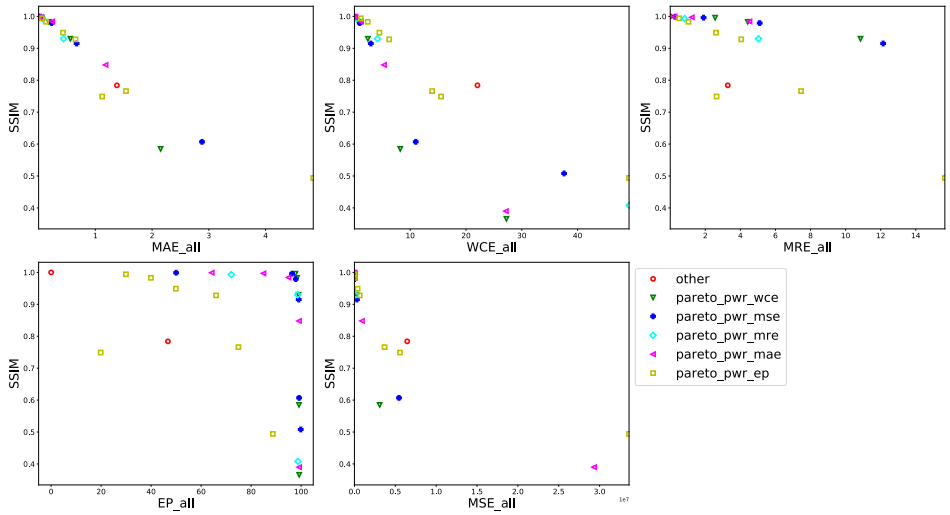


Figure 4.11: Surveillance image in RGB colour space: Lanczos resampling: SSIM versus error metrics

Table 4.10: Coefficient correlation versus image quality SSIM

Error metrics	Pearson's	Spearman's	Kendall's
MAE	-0.91	-0.99	-0.93
MRE	-0.84	-0.89	-0.73
MSE	-0.81	-0.97	-0.88
EP	-0.35	-0.58	-0.46
WCE	-0.88	-0.96	-0.85

Based on this conclusion, we can see that MAE is the most important error metric when evaluating the image quality of image scaling with approximate multipliers.

4.2.3 Hardware Selection

Based on the results of the previous section, it is clear that MAE is the most useful metric to identify approximate multipliers in the selected image application. The error plots, particularly Figure 3.4 where all output values are shifted by 256, serve as a valuable tool for visualizing the error characteristics among different approximate multipliers.

Figure 3.4 illustrates that the error distances for the approximate multipliers (c) and (d) are comparatively smaller, as indicated in C2. The approximate multiplier (c) shows a variation within the range of ± 2, whereas the approximate multiplier (d) shows a variation within the range of ± 6. Moreover, errors are uniformly distributed across all pixels.

Compared to these approximate multipliers, the multiplier represented by (f) should be avoided. Although its WCE is smaller than that of the approximate multiplier (h), as shown in Table 3.1, it presents significant variation between the pixel values when considering adjacent input combinations, as illustrated in Figure 3.4. On the other hand, MAE shows a consistent trend toward the final SSIM score.

Another illuminating comparison can be made between the approximate multipliers (e) and (g). By examining the error metrics, we notice that the MAE is similar for these two multipliers. However, the MSE reveals a more substantial difference. Furthermore, when considering the SSIM score of the scaled image, multiplier (g) demonstrates slightly superior performance compared to multiplier (e). This observation suggests that when the MAE values are close, the MRE becomes a more influential error metric in determining the overall performance.

In conclusion, error plots provide a deeper understanding of the meaning of error metrics and their impact on the scaled image results with Lanczos resampling. When considering the trade-off between power and error for image scaling, it is found that MAE has greater significance, while EP is the least relevant error metric.

4.3 Power Simulation of Image Scaling Application

Image scaling is performed on different image sets using the Lanczos resampling algorithm, followed by a power simulation of the design. The image set includes an image generated by random pixel values, a completely black colour image along with the test image of a Baboon standard test image in Figure 4.4, and a real image of the surveillance image shown in Figure 4.5 .

Power is calculated for each image in RGB colour format in the image set with an exact multiplier and approximate multiplier variants. Based on the acceptable range of image quality and error metrics, a set of approximate multipliers is chosen and used for image scaling evaluation. The overall power consumption is calculated when these approximate multipliers are used. The power ratios of each image concerning every multiplier are shown in Figure 4.12. We can observe a significant decrease in power with approximate multipliers compared to the exact multiplier.

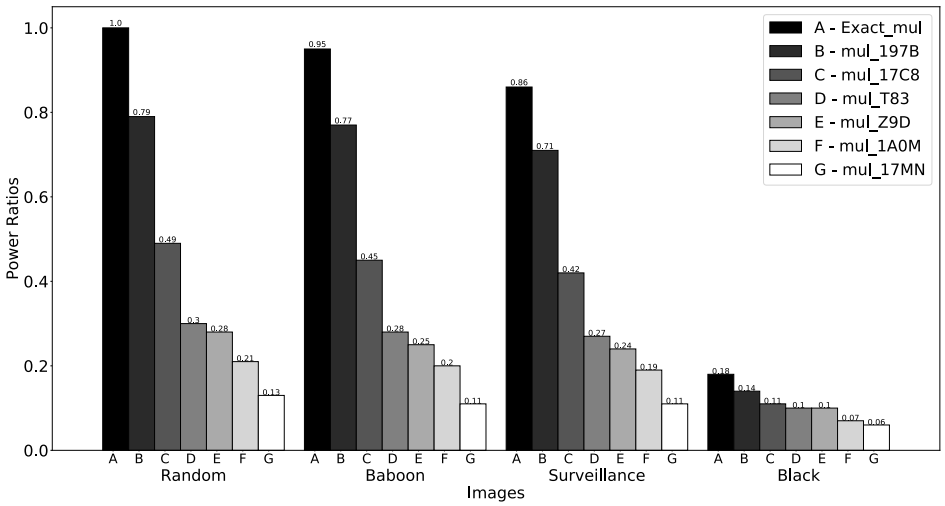


Figure 4.12: Comparison of power ratios of different approximate multipliers on test images

For performing a Laczos scaling, we consider a completely black image with all pixel values set to 0 as the optimal input scenario. This is because, in such an image, the majority of gates do not need to toggle, resulting in lower power consumption. On the other hand, an image with random pixel values requires a significantly higher number of toggles, leading to increased power consumption during the scaling process. By comparing the power consumption between the all-black image and an image with random pixels, we can effectively assess how the power consumption varies based on different input images.

We can conclude from the power simulation that the power consumption of the system varies depending on the image we choose. The arithmetic computations required depending on the pixel values have the highest impact on the power

consumption. Random input images can be considered a good worst-case example for power evaluation because they consist of random pixels, which creates a lot of switching activity and will consume a high amount of power. Real-world images, such as the surveillance and the Baboon image and random input images, both consume similar power with various approximate and exact multipliers. Therefore, we have done further power simulations using only a random input image, as this result is valid and acceptable for any type of image.

Using the method of comparing the power consumption of each multiplier and the SSIM of the image, a suitable approximate multiplier can be selected, providing better image quality and significant power savings.

Table 4.11: Power estimation versus power simulation of Lanczos scaling with difference approximate multipliers

Multiplier	Exact	197B	17C8	T83	Z9D	1A0M	17MN
Estimated ratio(%)	100	80	56	42	46	40	36
Simulated ratio(%)	100	79	49	30	27	21	13

Various modules in a Lanczos scaling application can have a different impact on total power consumption. In Figure 4.13, we can see the total power consumed by each sub-module in the design. As discussed in Section 3.4.1, the Lanczos scaling design includes several processing elements that perform maximum computations and thus consume maximum power, which can also be proved from Figure 4.13. The Lanczos computation in the X direction consumes more power than the Y direction, as it is subjected to more pixels for computation. Various approximate multipliers in these modules show a significant reduction in power usage in both the X and Y directions. We can also observe that the accumulator module consumes almost 10% of the total power. Although all the accumulators are fixed for all designs, depending on the approximate multiplier used, some bits can be redundant, and the synthesis tool optimizes the design of the accumulators and their related registers. This results in a reduction in the power of the accumulator stage and the registers.

In Figure 4.14, it is clear that dynamic power is more predominant in total power consumption, and static power consumption is less than 10 % in all multipliers. We can observe that the approximate multipliers play a crucial role in reducing dynamic power consumption compared to the exact multiplier.

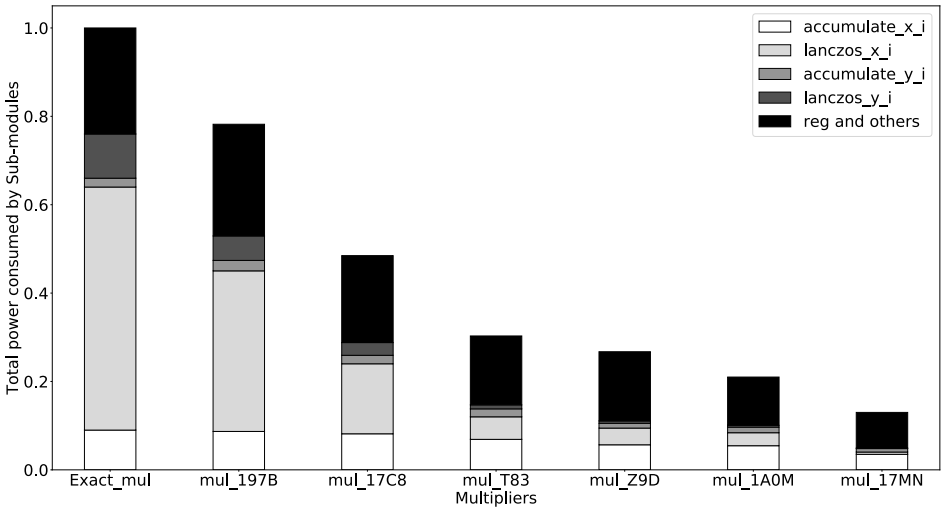


Figure 4.13: Total power consumed by each sub-module in the design when different approximate multipliers are used

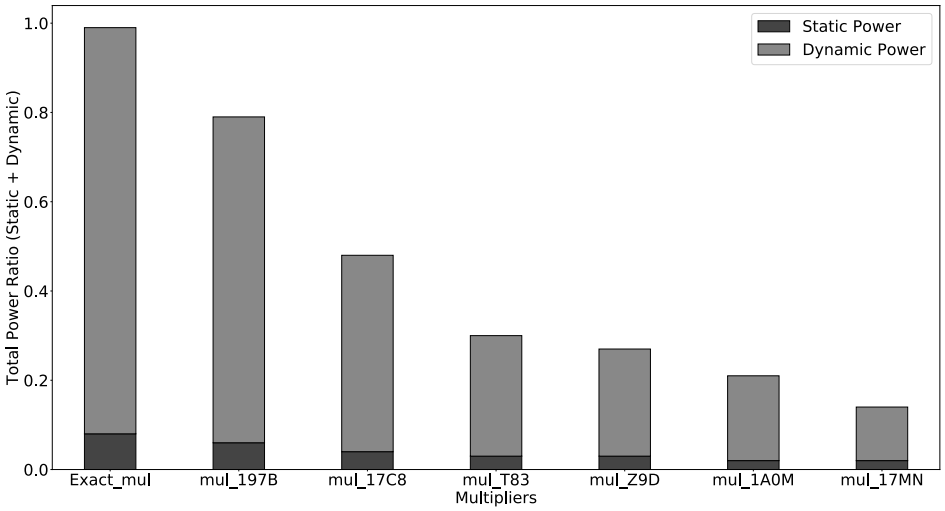


Figure 4.14: Power ratio of static and dynamic power consumption

Therefore, from the results of this thesis, it can be concluded that dynamic power plays an important role in overall power consumption and that using appropriate approximation techniques in design can significantly reduce overall system power.

Chapter 5

Conclusion and Future Works

A conclusion is presented in Section 5.1 and highlights the key findings of the thesis results. Section 5.2 discusses a potential research area that this thesis could inspire in the future.

5.1 Conclusion

Our results demonstrated the potential to incorporate AACs into the Lanczos resampling for image scaling in the selected manufacturing technology. This resulted in over 50% power-savings when the RGB scheme was used in the colour space while maintaining the high SSIM score of 0.9. Furthermore, our study suggests that there can be even greater power savings in the YCbCr colour space without significantly compromising image quality.

Through our study, various error metrics of approximate multiplier designs were investigated from the algorithm level, and MAE is concluded to be the most relevant to the image quality of the scaled images, while EP is the least relevant error metric. Additionally, the error plots serve as an effective visualization tool for understanding the error characteristics of the approximate multipliers, helping to select and evaluate suitable approximate multiplier designs. From the circuit level, we show that the power savings ratio also depends on the fabrication technology. Based on the conclusions obtained from the algorithm- and circuit-level analyses, we demonstrate that the YCbCr colour space has a better potential to apply AACs than the RGB colour space for scaling images with Lanczos resampling. It even allows for a mixture of different approximate multipliers, further improving the power-saving ratio while maintaining satisfactory image quality results.

In summary, this thesis highlights the importance of considering power consumption, arithmetic metrics, image quality metrics, and fabrication technology when using approximate multipliers in an image scaling application. It provides promising insights into the trade-offs between power savings and error in an error-tolerant application. It guides the selection of appropriate approximate multiplier designs and reduces the system's overall power consumption.

5.2 Future work

In terms of future work, several interesting directions related to using approximate components in image processing can be explored. First, investigating the selection of a mixture of approximate multipliers for different Lanczos coefficients holds the potential to further improve the overall power reduction as an extension of this thesis. By tailoring the selected approximate multipliers to match values of different Lanczos coefficients, achieving a better trade-off between power and accuracy may be possible during the resampling process.

Another area of interest is understanding how compression algorithms are affected by using approximate components. Exploring the impact of approximate multipliers on popular image compression techniques can provide valuable insights into the trade-offs between compression efficiency and the use of approximate components. This research can help optimize compression algorithms for images processed with approximate components, leading to improved compression ratios and reduced system power.

In addition, considering the intricate computations involved in CNNs, there is a promising potential in utilizing AACs to replace Multiply-Accumulate (MAC) units within the CNN module. Since CNN classifications are based on probabilities, a structural design that can tolerate errors could be deemed acceptable and potentially beneficial, such as adding some noise to the module [23].

Another interesting research avenue is training the CNN module by feeding in poorly scaled images using an approximate multiplier that consumes extremely low power. This approach aims to reduce the system's overall power consumption, as image scaling is commonly used as a preprocessing step in CNNs. By training the CNN on poorly scaled images, the network can potentially learn to adapt and perform well even with suboptimal scaling, leading to power savings for the overall system.

In general, future research in these areas can contribute to advancing the field of approximate computing in image processing.

References

[1] Vinay K. Chippa, Hrishikesh Jayakumar, Debabrata Mohapatra, Kaushik Roy, and Anand Raghunathan. Energy-efficient recognition and mining processor using scalable effort design. In *Proceedings of the IEEE 2013 Custom Integrated Circuits Conference*, pages 1–4, 2013.

[2] Vincent Camus, Jeremy Schlachter, Christian Enz, Michael Gautschi, and Frank K. Gurkaynak. Approximate 32-bit floating-point unit design with 53product reduction. In *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, pages 465–468, 2016.

[3] Honglan Jiang, Francisco Javier Hernandez Santiago, Hai Mo, Leibo Liu, and Jie Han. Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proceedings of the IEEE*, 108(12):2108–2135, 2020.

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

[5] Vojtech Mrazek, Radek Hrbacek, Zdenek Vasicek, and Lukas Sekanina. Evoapprox8b: library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 258–261, 2017.

[6] John Crowe and Barrie Hayes-Gill. 9 - choosing a means of implementation. In John Crowe and Barrie Hayes-Gill, editors, *Introduction to Digital Electronics*, pages 191–239. Newnes, Oxford, 1998.

[7] Muhammed Ibrahim and Ibrahim Hamarash. Dynamic voltage frequency scaling (dvfs) for microprocessors power and energy reduction. 12 2005.

[8] Bharadwaj Amrutur, Nandish Mehta, Satyam Dwivedi, and Ajit Gupte. Adaptative techniques to reduce power in digital circuits. *Journal of Low Power Electronics and Applications*, 1(2):261–276, 2011.

[9] J. Leijten, J. Meerbergen, and Jochen Jess. Analysis and reduction of glitches in synchronous networks. pages 398–403, 04 1995.

[10] Radek Hrbacek, Vojtech Mrazek, and Zdenek Vasicek. Automatic design of approximate circuits by means of multi-objective evolutionary algorithms.

In *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6, 2016.

[11] S. Mohan Das Arifa Parveen Pallevale, N. MD. Bilal. Implementation of approximate full adders using majority logic. pages 2–9, 12 2019.

[12] Padmanabhan Balasubramanian, Raunaq Nayar, Okkar Min, and Douglas L. Maskell. Approximator: A software tool for automatic generation of approximate arithmetic circuits. *Computers*, 11(1), 2022.

[13] Manickam Ramasamy, Narmadha Ganesamoorthy, and S. Deivasigamani. Carry based approximate full adder for low power approximate computing. pages 1–4, 06 2019.

[14] Honglan Jiang, Francisco Javier Hernandez Santiago, Hai Mo, Leibo Liu, and Jie Han. Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proceedings of the IEEE*, 108(12):2108–2135, 2020.

[15] Debabrata Mohapatra, Vinay K. Chippa, Anand Raghunathan, and Kaushik Roy. Design of voltage-scalable meta-functions for approximate computing. In *2011 Design, Automation & Test in Europe*, pages 1–6, 2011.

[16] Parag Kulkarni, Puneet Gupta, and Milos Ercegovac. Trading accuracy for power in a multiplier architecture. *J. Low Power Electronics*, 7:490–501, 12 2011.

[17] Radek Hrbacek, Vojtech Mrazek, and Zdenek Vasicek. Automatic design of approximate circuits by means of multi-objective evolutionary algorithms. pages 1–6, 04 2016.

[18] Renuka Deshpande, Lata Ragha, and Satyendra Sharma. Video quality assessment through psnr estimation for different compression standards. *Indonesian Journal of Electrical Engineering and Computer Science*, 11:918–924, 09 2018.

[19] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[20] Zhou Wang, Ligang Lu, and A.C. Bovik. Video quality assessment using structural distortion measurement. In *Proceedings. International Conference on Image Processing*, volume 3, pages III–III, 2002.

[21] Richard Gallagher and Zhaohui Chen. *Principles of Digital Image Processing: Core Algorithms*. Springer International Publishing, 2018.

[22] Nikolay Ponomarenko, Vladimir Lukin, Alexander Zelensky, Karen Egiazarian, Marco Carli, and Federica Battisti. Tid2008 - a database for evaluation of full-reference visual quality assessment metrics. *Advances of Modern Radioelectronics*, 10:30–45, 01 2009.

[23] L Hemanth Krishna, J Bhaskara Rao, Sk Ayesha, Sreehari Veeramachaneni, and Sk Noor Mahammad. Energy efficient approximate multiplier design for image/video processing applications. In *2021 IEEE International Symposium on Smart Electronic Systems (iSES)*, pages 210–215, 2021.

Printed by Tryckeriet i E-huset, Lund 2023

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2023-938
http://www.eit.lth.se