Evaluation of Approximate Adders using BDDs and satisfiability

A Project Report

submitted by

NAVANEETH A

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

June 2021

THESIS CERTIFICATE

This is to certify that the project report titled **Evaluation of Approximate Adders using BDDs and satisfiability**, submitted by **Navaneeth A**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof Vinita Vasudevan
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: June 26, 2021

ACKNOWLEDGEMENTS

I would like to express my profound gratitude to my project guide Prof. Vinita Vasudevan for the continuous support of my MTech Project. This work would not have been completed without the suggestions from ma'am.

Also I would like to thank Koneru Basava Naga Girish for helping me setup the remote system and Janaki M for providing the vpn access and helping me install the required packages in the remote system. I thank my parents and sister for all their support.

ABSTRACT

This work evaluates various approximate adders. The approximate adders are designed by modifying the accurate adder design in order to obtain improvements in power consumption, area and the delay. The approximate adders are used in the applications which require high speed, power efficiency, and also can tolerate error. Applications in image processing such as Image smoothing and canny-edge detection are implemented in Python by replacing accurate adders by approximate adders.

To compare the performance of approximate adders various approaches for the error calculations are used. Characterizing an approximate adder by performing error calculation using all possible input combinations is a time consuming job. Monte-carlo simulations takes a small number of inputs from the set of all possible input combinations to calculate errors. As the Monte-carlo simulations does not guarantee accurate values, methods such as Binary Decision Diagrams(BDDs) and satisfiability are used to for the exact error analysis. Methods to compute exact errors are implemented in Python and compared the results with Monte-carlo simulations. The exact values of error metrics such as Mean absolute error(MAE) and Mean squared error(MSE) are found for various approximate adders when the probability of occurrence of a bit(static probability) is not 0.5.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS **i**

ABSTRACT **iii**

LIST OF TABLES **viii**

LIST OF FIGURES **xi**

ABBREVIATIONS **xiii**

1 INTRODUCTION **1**

1.1 Motivation 1

1.2 Objective 1

2 Review of Approximate Adders **3**

2.1 Error Tolerant Adder (ETA) 3

2.1.1 Type 1 (ETA1) 3

2.1.2 Type 2 (ETA2) 4

2.2 Lower-part OR Adder (LOA) 5

2.3 Approximate Mirror Adders 5

2.3.1 Type 1 (AMA1) 6

2.3.2 Type 2 (AMA2) 6

2.3.3 Type 3 (AMA3) 7

2.3.4 Type 4 (AMA4) 7

2.3.5 Type 5 (AMA5) 9

2.4 Approximate XOR/XNOR based adder 9

2.4.1 Approximate XOR-based adder (AXA1) 10

2.4.2 Approximate XNOR-based adder (AXA2) 10

2.4.3 Approximate XNOR-based adder (AXA3) 11

2.5 Approximate Carry Skip Adder 12

2.6 Consistent Carry Approximate Adder (CCA) 12

3 Applications of approximate adders **15**

3.1 Image smoothing 15

3.1.1 Gaussian Filter 15

3.1.2 Results 17

3.2 Canny Edge Detector 22

3.2.1 Edge Detection 22

3.2.2 Results after gradient filter 23

4 Adder Comparison **27**

4.1 Error Metrics 27

4.1.1 Error Rate (ER) 27

4.1.2 Error Distance (ED) 27

4.1.3 Hamming Distance (HD) 28

4.1.4 Mean Hamming Distance (MHD) 28

4.1.5 Mean Absolute Error (MAE) 28

4.1.6 Worst-case arithmetic error (WCE) 28

4.1.7 Mean Squared Error (MSE) 29

4.2 Error Metrics computation 29

4.2.1 Binary Decision Diagram and Boolean Satisfiability 29

4.2.2 Miter circuit 29

4.2.3 Computation of the Error Metrics using satisfiability(Vasicek, 2019) 30

4.2.4 Computation of Error Metrics - Simplified Algorithm(Vasicek, 2019) 32

4.3 PyEDA 33

4.4 Modification and Results 34

4.4.1 Mean Absolute Error 34

4.4.2 Mean Squared Error 46

4.4.3 Worst-case error 57

4.4.4 Error Rate 58

4.4.5 Time take for computation 59

LIST OF TABLES

2.1 Truth Table for Mirror Adder 5

2.2 Truth Table for AMA1 6

2.3 Truth Table for AMA2 7

2.4 Truth Table for AMA3 8

2.5 Truth Table for AMA4 8

2.6 When $S = A$ and $C_{out} = A$ 9

2.7 When $S = B$ and $C_{out} = A$ 9

2.8 Truth Table for AXA1 10

2.9 Truth Table for AXA2 11

2.10 Truth Table for AXA3 11

4.1 MAE for 10 bit LOA with 5 bit approx. 34

4.2 MAE for 10 bit ETA1 with 5 bit approx. 35

4.3 MAE for 10 bit AMA5 with 5 bit approx. 35

4.4 MAE for 10 bit AMA4 with 5 bit approx. 36

4.5 MAE for 10 bit AMA3 with 5 bit approx. 36

4.6 MAE for 10 bit AMA2 with 5 bit approx. 37

4.7 MAE for 10 bit AMA1 with 5 bit approx. 37

4.8 MAE for 10 bit AXA1 with 5 bit approx. 38

4.9 MAE for 10 bit AXA2 with 5 bit approx. 38

4.10 MAE for 10 bit AXA3 with 5 bit approx. 39

4.11 MAE for 16 bit LOA with 10 bit approx. 41

4.12 MAE for 16 bit ETA1 with 10 bit approx. 41

4.13 MAE for 16 bit AMA5 with 10 bit approx. 42

4.14 MAE for 16 bit AMA4 with 10 bit approx. 42

4.15 MAE for 16 bit AMA3 with 10 bit approx. 43

4.16 MAE for 16 bit AMA2 with 10 bit approx. 43

4.17 MAE for 16 bit AMA1 with 10 bit approx. 44

4.18 MAE for 16 bit AXA1 with 10 bit approx. 44
4.19 MAE for 16 bit AXA2 with 10 bit approx. 45
4.20 MAE for 16 bit AXA3 with 10 bit approx. 45
4.21 MSE for 10 bit LOA with 5 bit approx. 46
4.22 MSE for 10 bit ETA1 with 5 bit approx. 46
4.23 MSE for 10 bit AMA5 with 5 bit approx. 47
4.24 MSE for 10 bit AMA4 with 5 bit approx. 47
4.25 MSE for 10 bit AMA3 with 5 bit approx. 48
4.26 MSE for 10 bit AMA2 with 5 bit approx. 48
4.27 MSE for 10 bit AMA1 with 5 bit approx. 49
4.28 MSE for 10 bit AXA1 with 5 bit approx. 49
4.29 MSE for 10 bit AXA2 with 5 bit approx. 50
4.30 MSE for 10 bit AXA3 with 5 bit approx. 50
4.31 MSE for 16 bit LOA with 10 bit approx. 52
4.32 MSE for 16 bit ETA1 with 10 bit approx. 52
4.33 MSE for 16 bit AMA5 with 10 bit approx. 53
4.34 MSE for 16 bit AMA4 with 10 bit approx. 53
4.35 MSE for 16 bit AMA3 with 10 bit approx. 54
4.36 MSE for 16 bit AMA2 with 10 bit approx. 54
4.37 MSE for 16 bit AMA1 with 10 bit approx. 55
4.38 MSE for 16 bit AXA1 with 10 bit approx. 55
4.39 MSE for 16 bit AXA2 with 10 bit approx. 56
4.40 MSE for 16 bit AXA3 with 10 bit approx. 56
4.41 WCE for 10 bit adder with 5 bit approximate bits 57
4.42 WCE for 16 bit adder with 10 bit approximate bits 57
4.43 ER for 10 bit adder with 5 bit approximate bits 58
4.44 ER for 16 bit adder with 10 bit approximate bits 58
4.45 Time taken for 16 bit adder with 10 approximate bits 59
4.46 Time taken for 16 bit adder with 10 approximate bits 59

LIST OF FIGURES

2.1 Working of ETA 1 3

2.2 disadvantage of ETA 1 3

2.3 ETA 2 4

2.4 ETA 2m 4

2.5 LOA 5

2.6 Conventional Mirror Adder 5

2.7 AMA1 6

2.8 AMA2 7

2.9 AMA3 8

2.10 AMA4 8

2.11 Accurate full adder with 10 transistors 9

2.12 AXA1 10

2.13 AXA2 11

2.14 AXA3 11

2.15 CSA 12

2.16 CCA 13

3.1 3x3 BOX filter 15

3.2 Gaussian kernels Example 16

3.3 3×3 Gaussian filter implementation with $\sigma = 1$ 16

3.4 5×5 Gaussian filter implementation $\sigma = 1.4$ 17

3.5 ETA1 Gaussian filter, PSNR = 29.3100 dB 17

3.6 ETA2 Gaussian Filter, PSNR = 28.7048 dB 17

3.7 AMA1 Gaussian filter, PSNR = 28.2628 dB 18

3.8 AMA2 Gaussian Filter, PSNR = 35.4950 dB 18

3.9 AMA3 Gaussian filter, PSNR = 31.5288 dB 18

3.10 AMA4 Gaussian Filter, PSNR = 34.3298 dB 18

3.11 AMA5 Gaussian filter, PSNR = 35.5404 dB 18

3.12 CSA Gaussian Filter, PSNR = 28.5429 dB 18
3.13 CCA-0 Gaussian filter, PSNR = 27.6150 dB 19
3.14 CCA-1 Gaussian Filter, PSNR = 27.4202 dB 19
3.15 LOA Gaussian filter, PSNR = 35.6971 dB 19
3.16 AXA1 Gaussian Filter, PSNR = 35.4191 dB 19
3.17 AXA2 Gaussian filter, PSNR = 34.3604 dB 19
3.18 AXA3 Gaussian Filter, PSNR = 30.1345 dB 19
3.19 ETA1 Gaussian filter, PSNR = 29.2185 dB 20
3.20 ETA2 Gaussian Filter, PSNR = 28.0433 dB 20
3.21 AMA1 Gaussian filter, PSNR = 33.5836 dB 20
3.22 AMA2 Gaussian Filter, PSNR = 33.2358 dB 20
3.23 AMA3 Gaussian filter, PSNR = 34.0434 dB 20
3.24 AMA4 Gaussian Filter, PSNR = 33.0829 dB 20
3.25 AMA5 Gaussian filter, PSNR = 33.1825 dB 21
3.26 CSA Gaussian Filter, PSNR = 27.8665 dB 21
3.27 LOA Gaussian filter, PSNR = 33.2448 dB 21
3.28 AXA1 Gaussian Filter, PSNR = 33.3785 dB 21
3.29 AXA2 Gaussian filter, PSNR = 33.0235 dB 21
3.30 AXA3 Gaussian Filter, PSNR = 30.800 dB 21
3.31 Sobel operator 23
3.32 Gradient filter implementation 23
3.33 ETA1 gradient filter 24
3.34 ETA2 gradient filter 24
3.35 AMA1 gradient filter 24
3.36 AMA2 gradient filter 24
3.37 AMA3 gradient filter 24
3.38 AMA4 gradient filter 24
3.39 AMA5 gradient filter 25
3.40 AXA1 gradient filter 25
3.41 AXA2 gradient filter 25
3.42 AXA3 gradient filter 25
3.43 LOA gradient filter 25

3.44 CSA gradient filter 25
3.45 CCA0 gradient filter 26
3.46 CCA1 gradient filter 26

4.1 Miter Circuits for analysis of a) error rate b)average hamming distance 30
4.2 Miter Circuits for analysis of a) Absolute ED b)ED c) Squared ED . 30
4.3 Variation of MAE with static probability - 10 bit adder 5 bit apprx. . 39
4.4 Variation of MSE with static probability - 10 bit adder 5 bit apprx . 51

ABBREVIATIONS

MSB	Most Significant Bit
LSB	Least Significant Bit
RCA	Ripple Carry Adder
ETA	Error Tolerant Adder
AMA	Approximate Mirror Adder
AXA	Approximate Xor/Xnor based adder
LOA	Lower-part OR Adder
CSA	Approximate carry skip adder
CCA	Consistent carry approximate adder
MAE	Mean Absolute Error
MSE	Mean Squared Error
WCE	Worst Case Error
ED	Error Distance
MED	Mean Error Distance
ER	Error Rate
BDD	Binary Decision Diagram

CHAPTER 1

INTRODUCTION

1.1 Motivation

In this time period the demands of a high speed, power efficient processing systems are increasing. As the number and the complexity of the computation increases day by day, the power usage by the respective processing units also increases proportionally. Depending on the application, we can implement several methods to reduce power usage by these processors by compromising the output.

For applications such as image processing which does not require an accurate result since human eye can not differentiate small errors, approximate circuits can be introduced. Approximate circuits are designed by slightly modifying the accurate circuits, thus introducing an error. Use of approximate circuits results in lower energy, lower area or improvement in the delay.

Some adders, such as two-part segmented adders are designed for low power consumption through circuit modification. In some other adders the carry is allowed to propagate only few bits so that the critical path delay is reduced.

So if the accurate circuit is replaced with approximate circuit and the whole system is run at the same frequency, the supply voltage can be reduced, thus the power consumption get reduced.

1.2 Objective

The approximate adders are designed by modifying different accurate adder circuits. Depending on the accurate adder design which is used to design the approximate adder, approximate adders are divided into groups.

1. Segmented adders divide the inputs into blocks and does addition in each block depending on the design.

2. Carry select adders are also similar to segmented adders, but several signals are commonly used between the bits inside a particular block.
3. Approximate full adders are modification of a full adder design by removing the transistors from the full adder design.

The objective of this project is to find different types of approximate adders and implement these designs in Python. Analyze the adders using Monte-carlo simulations to find the errors such as Mean absolute error(MAE), Mean squared error(MSE) etc.
Find applications of these approximate circuits and implement them in Python.
Use Binary Decision Diagram (BDD) and satisfiability for the exact analysis of these approximate circuits. Find exact values of MAE, MSE, WCE, ER and compare them with Monte-carlo simulations. Also find these error metrics when the probability of occurrence of a bit(static probability) is not 0.5 .

CHAPTER 2

Review of Approximate Adders

2.1 Error Tolerant Adder (ETA)

2.1.1 Type 1 (ETA1)

In the Error Tolerant Adder type 1(Zhu *et al.*, 2009) the inputs are divided into two parts; accurate part and approximate part. The accurate part consists of the higher order bits in the input whereas the approximate part contains the remaining lower order bits in the input. In the accurate part normal addition is carried out using carry-in as 0. Also the addition is carried out from the LSB of accurate part to the MSB of the accurate part.

In the approximate part, the addition operation is carried out from the MSB of the approximate part to the LSB of the approximate part. The addition carried out is a half-adder addition. The addition operation is carried out till there is an input combination of '1,1' for the half adder. From that bit onwards, till the LSB, all bits are made to one. This process eliminates the carry propagation path in the approximate part of the adder. Both approximate and accurate additions are carried out simultaneously so that the time taken for the computation get reduced.

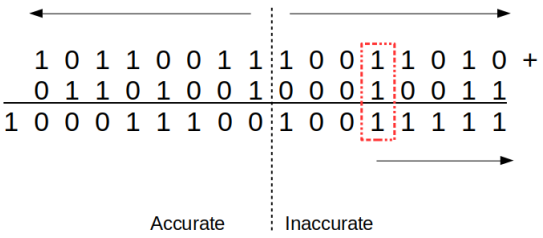


Figure 2.1: Working of ETA 1

If we fix the number of approximate bits for a particular application, then for the inputs having lower magnitude, ETA1 is less accurate because the addition operation is happening only in the approximate part. So the ETA1 yield a result which is far from the correct value.

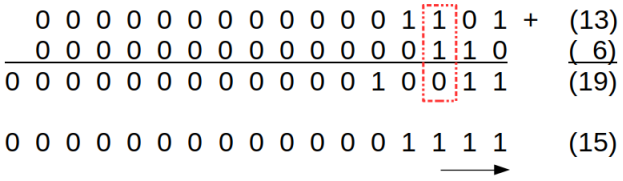


Figure 2.2: disadvantage of ETA 1

2.1.2 Type 2 (ETA2)

Error Tolerant Adder type 2(Zhu *et al.*, 2009) is a segmented adder, so it does not eliminate the carry propagation path entirely. N bit adder is divided to M blocks, where $M \geq 2$.

Each block contains N/M bits. Each block has a carry generation and sum generation. Carry generation is done by considering carry-in as zero ie; the carry generation does not depend on the previous blocks. Carry generated from the i^{th} block is propagated to the $(i+1)^{th}$ block. so, the carry propagation path only includes one carry generation block and one sum generation block instead of propagating along the entire adder structure. The carry propagation length is 2N/M. For the first sum generation block, the carry-in is considered as zero.

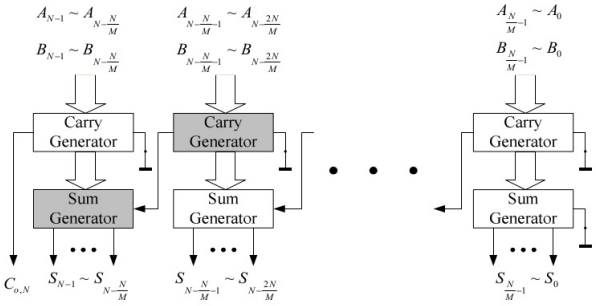


Figure 2.3: ETA 2

Type 2m (ETA 2m)

In the MSB of the ETA2 adder, the output depends only on the bits in the previous block, so ETA2 can give less accuracy if the number of bits in the input is higher.

So the ETA2 is modified(Zhu *et al.*, 2009) to give better accuracy. Three blocks which are producing MSBs in the result are allowed to propagate the carry. ie; these blocks are cascaded. so, The carry propagation path includes three carry generation blocks and one sum generation block.

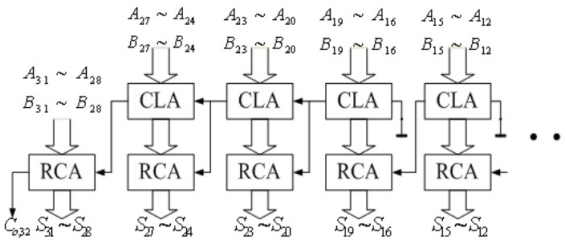


Figure 2.4: ETA 2m

2.2 Lower-part OR Adder (LOA)

Lower-part OR adder(Mahdiani *et al.*, 2010) also divides the inputs into accurate and approximate parts. In the approximate part the addition is carried out by taking the OR operation of the input bits from A with the corresponding input bit from B. The carry-out from the approximate part to the accurate part is found by the AND operation of MSBs in the approximate part inputs. In the accurate part normal RCA addition is carried out using the carry out from approximate part as the carry in.

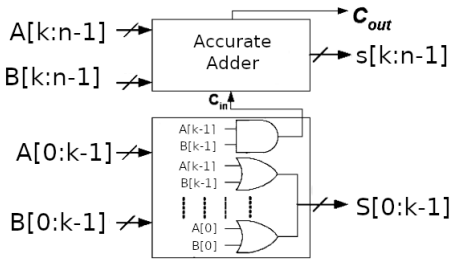


Figure 2.5: LOA

2.3 Approximate Mirror Adders

Approximate mirror adders(Gupta *et al.*, 2013) are modification of the conventional 24-FETs Mirror adders. MOSFETs are removed from the conventional Mirror adders in such a way that it ensures there is no direct path from VDD/GND to the Cout or Sum. This method is chosen because the mirror adder is not based on complementary CMOS logic.

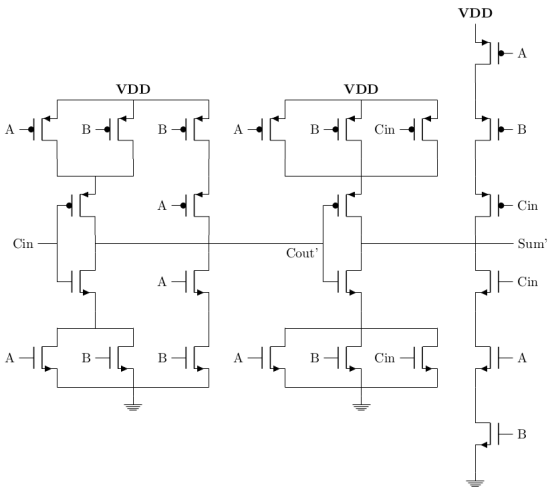


Figure 2.6: Conventional Mirror Adder

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2.1: Truth Table for Mirror Adder

2.3.1 Type 1 (AMA1)

This design has eight fewer transistors compared to the conventional mirror adder. Here the sum and carry are defined as

$$S = \bar{A}\bar{B}C_{in} + ABC_{in}$$
$$C_{out} = ABC_{in} + AB\bar{C_{in}} + A\bar{B}C_{in} + \bar{A}B\bar{C_{in}}$$

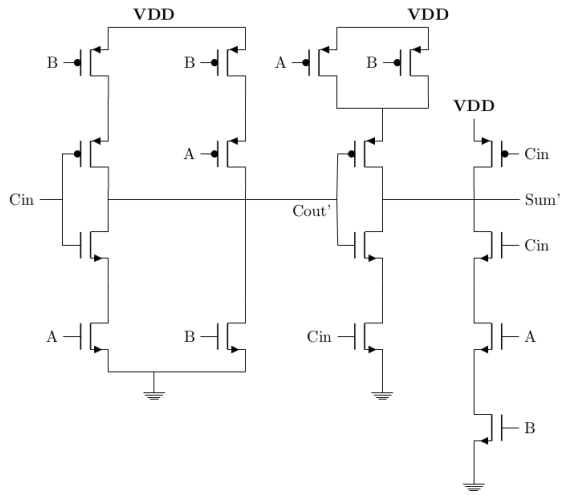


Figure 2.7: AMA1

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2.2: Truth Table for AMA1

There is one error in the Cout output and two errors in the Sum.

2.3.2 Type 2 (AMA2)

For the sum bit in the conventional mirror adder, for six out of 8 combinations, it is equal to the complement of the carry out bit. so, for AMA2 carry out is found accurately where sum is approximated to the complement of the carry out. In the design a buffer is included between C'_{out} and sum to reduce the capacitance at the sum node.

$$S = \overline{C_{out}}$$
$$C_{out} = AB + \bar{A}BC_{in} + A\bar{B}C_{in}$$

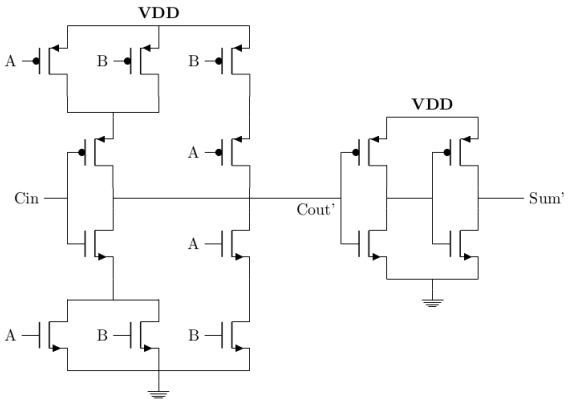


Figure 2.8: AMA2

A	B	Cin	S	Cout
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Table 2.3: Truth Table for AMA2

There is no error in the Cout output but there are two errors in the Sum.

2.3.3 Type 3 (AMA3)

This adder is a combination of AMA1 and AMA2. Carry out is approximately calculated using the equation for AMA1 and Sum is approximated as the complement of the Cout.

$$S = \overline{C_{out}}$$

$$C_{out} = ABC_{in} + AB\bar{C_{in}} + A\bar{B}C_{in} + \bar{A}BC_{in}$$

There is one error in the Cout output and three errors in the Sum.

2.3.4 Type 4 (AMA4)

For six out of the eight combinations the Cout is equal to the input A. Also for six out of the eight combinations Cout is equal to input B. For this adder the Cout is approximated

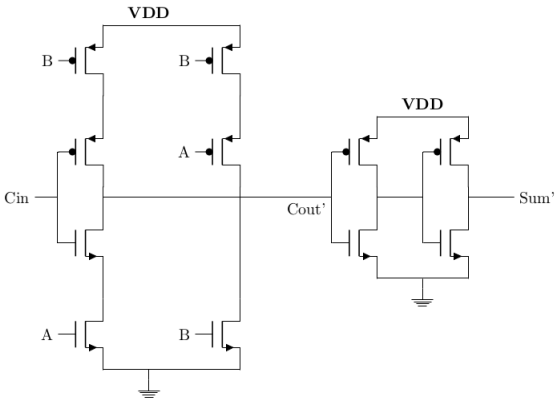


Figure 2.9: AMA3

A	B	Cin	S	Cout
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Table 2.4: Truth Table for AMA3

to input A. Sum is calculated using the equation used for AMA1.

$$S = \bar{A}\bar{B}C_{in} + ABC_{in}$$

$$C_{out} = A$$

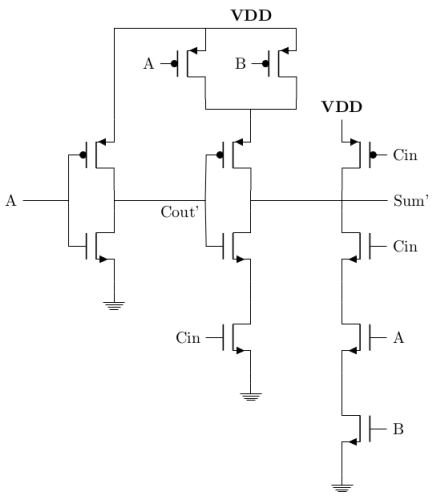


Figure 2.10: AMA4

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2.5: Truth Table for AMA4

There are two errors in the Cout output and three errors in the Sum.

2.3.5 Type 5 (AMA5)

In this adder one more error is introduced to the Sum compared to the AMA4. Carry out is approximated similar to the AMA4 ($C_{out} = A$). Sum is also approximated to one of the inputs. so there are two combinations; $S = A$ or $S = B$

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Table 2.6: When $S = A$ and $C_{out} = A$

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Table 2.7: When $S = B$ and $C_{out} = A$

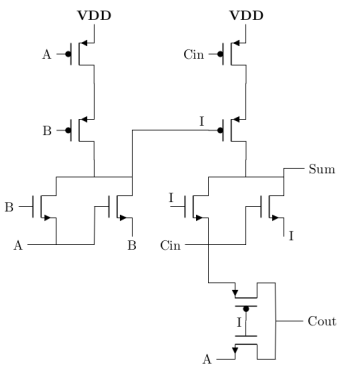
When Sum and C_{out} are equal, the number of combinations where actual values are produced for both Sum and C_{out} are only two. But when they are different it is four.

$$\therefore \quad Sum = B$$
$$C_{out} = A$$

There are 2 errors in the C_{out} and four errors in the Sum.

2.4 Approximate XOR/XNOR based adder

The approximate XOR/XNOR based adders(Yang *et al.*, 2013) are modifications of the 10-transistor full adders. There are one approximate XOR-based adder (AXA 1) and two approximate XNOR-based adders (AXA2 & AXA3)



2.4.1 Approximate XOR-based adder (AXA1)

In this adder the XOR operation is done using an inverter which is connected to first input and two pass transistors which are connected to second input. There are 4 errors in both sum and carry out of 8 combinations.

$$S = C_{in}$$
$$Cout = \overline{(A \oplus B)C_{in} + \bar{A}\bar{B}}$$

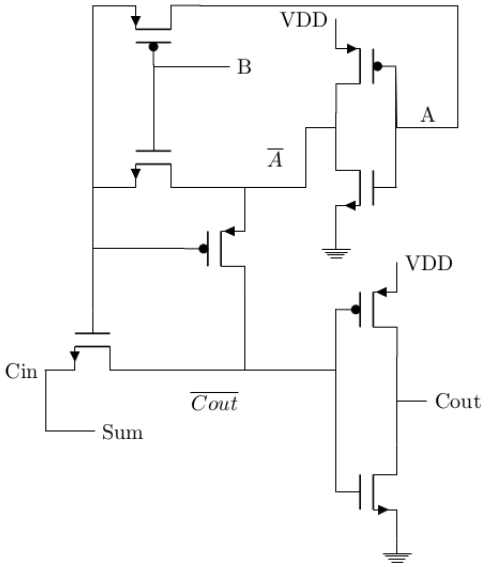


Figure 2.12: AXA1

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

Table 2.8: Truth Table for AXA1

2.4.2 Approximate XNOR-based adder (AXA2)

This adder consists of 6 transistors. ie; 4 transistors to implement the XNOR operation and a pass transistor block. Here the sum bit has 4 errors, but carry out is accurate for all input combinations.

$$S = \overline{A \oplus B}$$
$$Cout = (A \oplus B)C_{in} + AB$$

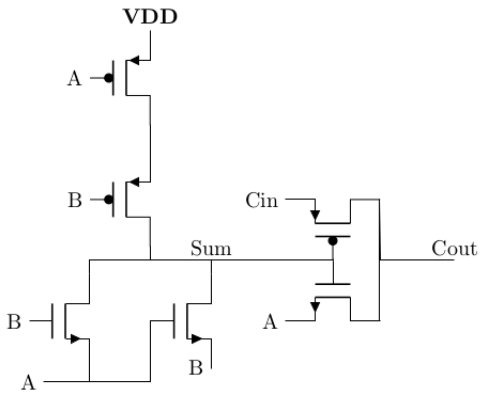


Figure 2.13: AXA2

A	B	Cin	S	Cout
0	0	0	1	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Table 2.9: Truth Table for AXA2

2.4.3 Approximate XNOR-based adder (AXA3)

This adder is the modification of the AXA2. There are two more transistors at the pass transistor block to increase the accuracy of the sum bit. So four transistors for the XNOR operation and four transistors for the pass transistor block. So sum bit has two errors while carry out has no errors.

$$S = \overline{A \oplus B} C_{in}$$

$$Cout = (A \oplus B) C_{in} + AB$$

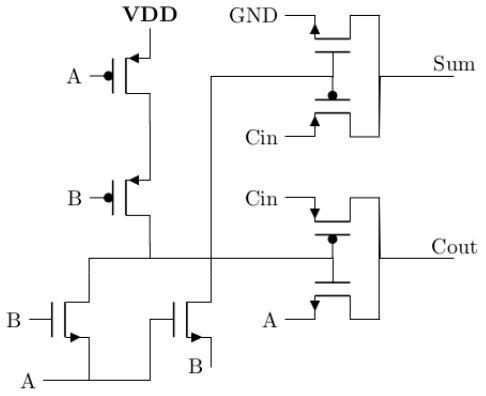


Figure 2.14: AXA3

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2.10: Truth Table for AXA3

2.5 Approximate Carry Skip Adder

Working of the CSA(Kim *et al.*, 2013) is similar to a Carry Skip Adder (segmented adder). But, the maximum possible distance a carry can propagate is two blocks. There are k bits in each block.

There is a carry generation and sum generation in each block. Carry generated from $(i-1)^{th}$ block is propagated to the i^{th} block and $(i+1)^{th}$ block. Also carry generated from i^{th} block also propagates to $(i+1)^{th}$ block and $(i+2)^{th}$ block. A multiplexer is used to determine which carry will be taken as the carry-in of the $(i+1)^{th}$ block.

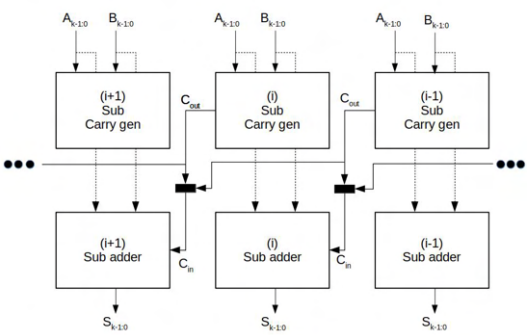


Figure 2.15: CSA

$$C_{in}^{i+1} = \overline{P_{k-1:0}^{i}} C_{out}^{i} + P_{k-1:0}^{i} C_{out}^{i-1}$$

$$\text{where } P_{k-1:0}^{i} = \prod_{j=0}^{k-1} p_j^i$$

$$\text{and } C_{out}^{i} = g_{k-1:0}^{i} + g_{k-2:0}^{i} P_{k-1:0}^{i} + \cdots + g_0^i \prod_{j=0}^{k-1} p_j^i$$

Where

$$g_i = a_i b_i, \quad \text{is the generate signal}$$

$$p_i = a_i \oplus b_i, \quad \text{is the propagate signal}$$

2.6 Consistent Carry Approximate Adder (CCA)

Consistent Carry Approximate Adder(Li and Zhou, 2014) is a segmented adder. The adder is divided into blocks of size k/2. All the bits in a block share the same carry-in for the addition operation. So carry propagation is not happening inside a particular block.

In each block there are two sum-blocks which are having carry-in as 0 and 1 respectively. These sum-blocks are implemented using traditional adders. So the output sum from the block is chosen using a multiplexer which is having a select signal(C_i) defined as

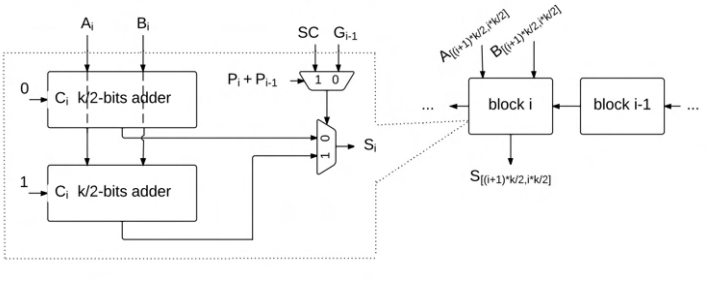


Figure 2.16: CCA

$$C_i = (P_i + P_{i-1})SC + \overline{(P_i + P_{i-1})}G_{i-1}$$

Where P_i, G_i are propagate and generate signal of the block i. SC is the global speculative carry. SC can be either 0 or 1. So depending on the value of SC there are two consistent carry approximate adders, CCA0 and CCA1.

CHAPTER 3

Applications of approximate adders

3.1 Image smoothing

3.1.1 Gaussian Filter

When a picture is taken, light fluctuations, sensor noise etc can introduce error in the image. Image smoothing is a method to reduce the noise in an image. This method improves the quality of the images.
In the smoothing operation value of the current pixel is replaced by averaging the local values, because the noise is assumed to be a zero mean Gaussian noise.

The simplest method to do image smoothing is the use of box filter. All weights in the filter are equal. It does the mean filtering, but it reduces the image details.

Box filter

$1/9 \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

Figure 3.1: 3x3 BOX filter

So to improve the details of the image, **Gaussian filter**(Oliveira Julio *et al.*, 2015) is used. this filter does the image sharpening by doing the weighted averaging. This filter uses a 2-D Gaussian distribution with mean $\mu = 0$.

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

Also the distribution is confined within $\pm 3\sigma$

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

3 x 3 Gaussian Kernal

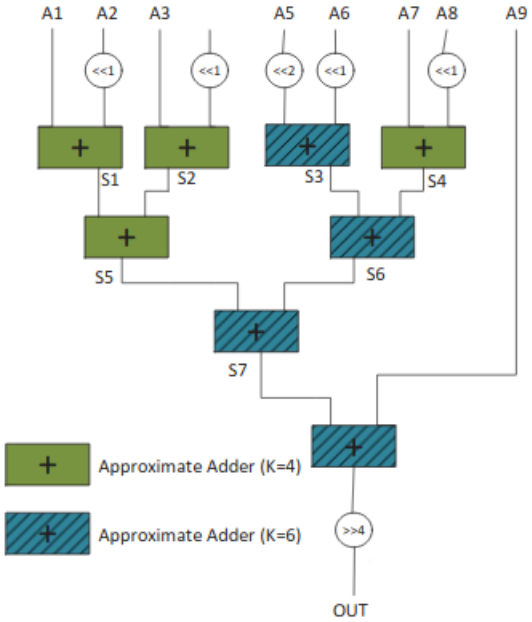
$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

5 x 5 Gaussian Kernal

Figure 3.2: Gaussian kernels Example

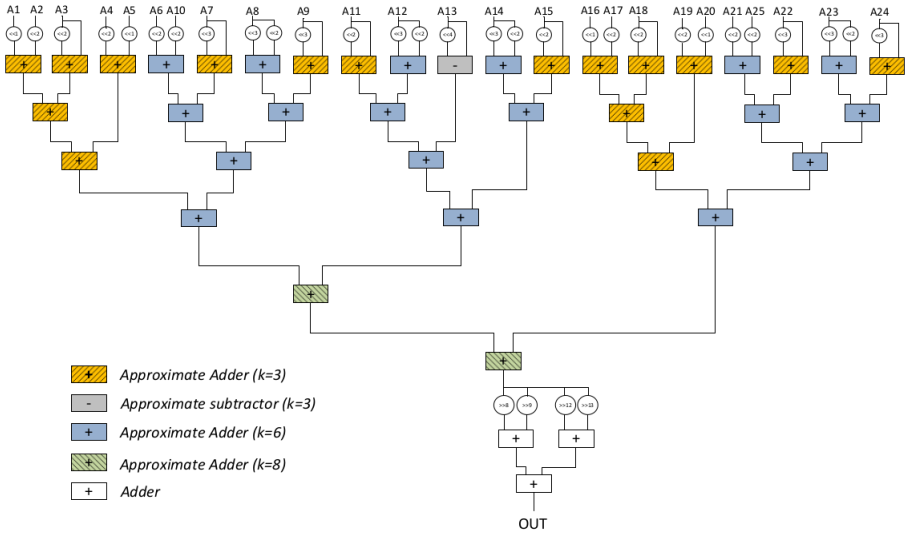
These kernels can be implemented using approximate adders.



(Oliveira Julio *et al.*, 2015)

Figure 3.3: 3×3 Gaussian filter implementation with $\sigma = 1$

In the implementation, the approximate adders which are having similar magnitudes in the input operands are grouped. Approximate adders which are having higher magnitude input operands are chosen to be less accurate, ie; number of approximate bits are higher. Approximate adders which are having lower magnitude input operands are chosen to be more accurate, ie; number of approximate bits are lower. So in the 3×3 Gaussian filter implementation, approximate adder with 6 approximate bits are used for higher magnitude input operands and approximate adder with 4 approximate bits are used for lower magnitude input operands.



(de Oliveira *et al.*, 2016)

Figure 3.4: 5×5 Gaussian filter implementation $\sigma = 1.4$

3.1.2 Results

The adders and filters are implemented in Python. The filters are applied on a 512×512 lena image. The image is converted to an array of pixels. Then the Gaussian filter is applied and converted the filtered array back to an image.

3×3 Gaussian filter

Figure 3.5: ETA1 Gaussian filter, PSNR = 29.3100 dB

Figure 3.6: ETA2 Gaussian Filter, PSNR = 28.7048 dB

Figure 3.7: AMA1 Gaussian filter, PSNR = 28.2628 dB

Figure 3.8: AMA2 Gaussian Filter, PSNR = 35.4950 dB

Figure 3.9: AMA3 Gaussian filter, PSNR = 31.5288 dB

Figure 3.10: AMA4 Gaussian Filter, PSNR = 34.3298 dB

Figure 3.11: AMA5 Gaussian filter, PSNR = 35.5404 dB

Figure 3.12: CSA Gaussian Filter, PSNR = 28.5429 dB

Figure 3.13: CCA-0 Gaussian filter, PSNR = 27.6150 dB

Figure 3.14: CCA-1 Gaussian Filter, PSNR = 27.4202 dB

Figure 3.15: LOA Gaussian filter, PSNR = 35.6971 dB

Figure 3.16: AXA1 Gaussian Filter, PSNR = 35.4191 dB

Figure 3.17: AXA2 Gaussian filter, PSNR = 34.3604 dB

Figure 3.18: AXA3 Gaussian Filter, PSNR = 30.1345 dB

5×5 **Gaussian filter**

Figure 3.19: ETA1 Gaussian filter, PSNR = 29.2185 dB

Figure 3.20: ETA2 Gaussian Filter, PSNR = 28.0433 dB

Figure 3.21: AMA1 Gaussian filter, PSNR = 33.5836 dB

Figure 3.22: AMA2 Gaussian Filter, PSNR = 33.2358 dB

Figure 3.23: AMA3 Gaussian filter, PSNR = 34.0434 dB

Figure 3.24: AMA4 Gaussian Filter, PSNR = 33.0829 dB

Figure 3.25: AMA5 Gaussian filter, PSNR = 33.1825 dB

Figure 3.26: CSA Gaussian Filter, PSNR = 27.8665 dB

Figure 3.27: LOA Gaussian filter, PSNR = 33.2448 dB

Figure 3.28: AXA1 Gaussian Filter, PSNR = 33.3785 dB

Figure 3.29: AXA2 Gaussian filter, PSNR = 33.0235 dB

Figure 3.30: AXA3 Gaussian Filter, PSNR = 30.800 dB

Conclusion

For the approximate adders ETA2[3.6], AMA1[3.7], CCA0[3.13], CCA1[3.14] and CSA[3.12], the value of a pixel after Gaussian filtering is higher than the value obtained when accurate adders are used for the filtering. So the filtered image has more pixels having a value closer to white.
The approximate adders AMA2[3.8], AMA4[3.10], AMA5[3.11], AXA1[3.16], AXA2[3.17] and LOA[3.15] produces a value which is closer to the value obtained when accurate adders are used for the filtering. So the PSNR values for these adders are higher compared to other approximate adders.

3.2 Canny Edge Detector

3.2.1 Edge Detection

This operation is used to detect the edges in an image. Canny edge detection(de Oliveira *et al.*, 2016) is a multi-stage process consisting of 5 steps.

1. Gaussian Filter - Image smoothing to reduce the noise
2. Gradient Filter - Detects the edges in the image using Sobel operator
3. Non Maximum Suppression - Thin the edges which are detected in the previous step
4. Double Threshold - Categorize the pixels into strong, weak and non-relevant according to the predefined threshold values.
5. Edge Detection By Hysteresis - groups the pixels from weak pixels to strong/non-relavant according to it's neighboring pixels and map the strong pixels to maximum value and non-relavant pixels to minimum value.

For the first two stages, where filters are used, approximate adders can be used instead of accurate adders to improve the time/power usage for the computation.
For the Gradient filtering operation Sobel operator is used. There are two sobel operators which are along x and y directions. The mean of the values along x and y is taken to compute the value of the pixel.

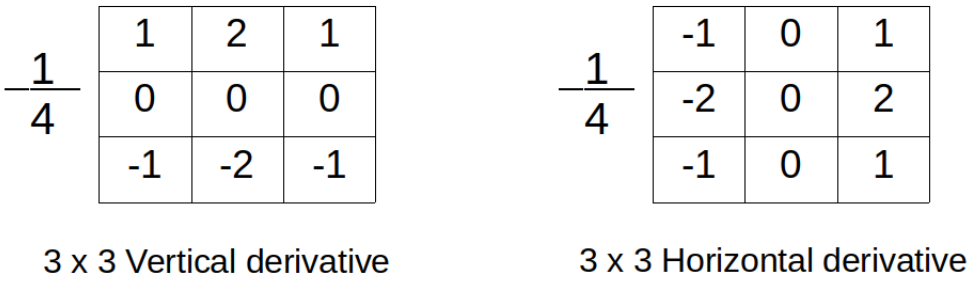
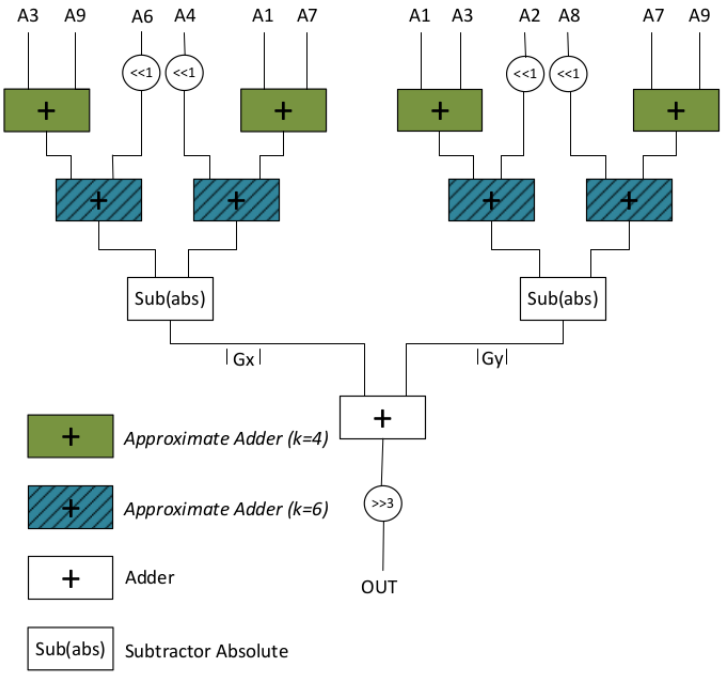


Figure 3.31: Sobel operator



(de Oliveira *et al.*, 2016)

Figure 3.32: Gradient filter implementation

Similar to the Gaussian filter, the adders are categorized according to the magnitude of input operands. Bit shifting is used instead of multiplication.

3.2.2 Results after gradient filter

The filter is implemented in Python. The output from the Gaussian filter is used as the input to this program.

Figure 3.33: ETA1 gradient filter

Figure 3.34: ETA2 gradient filter

Figure 3.35: AMA1 gradient filter

Figure 3.36: AMA2 gradient filter

Figure 3.37: AMA3 gradient filter

Figure 3.38: AMA4 gradient filter

Figure 3.39: AMA5 gradient filter

Figure 3.40: AXA1 gradient filter

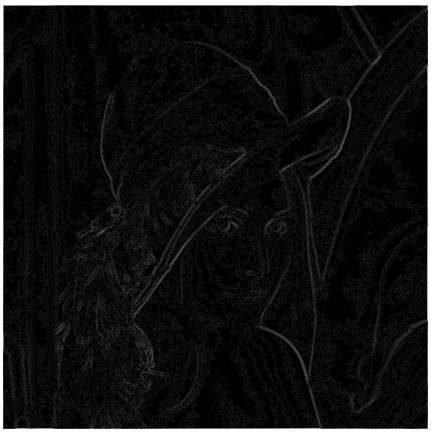


Figure 3.41: AXA2 gradient filter

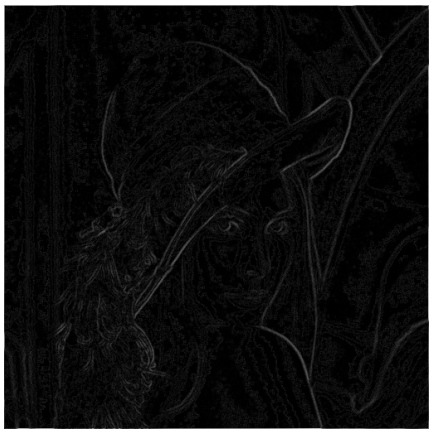


Figure 3.42: AXA3 gradient filter

Figure 3.43: LOA gradient filter

Figure 3.44: CSA gradient filter

Figure 3.45: CCA0 gradient filter

Figure 3.46: CCA1 gradient filter

Conclusion

Approximate adders behave similar to the Gaussian filtering. Approximate adders which produces a value higher than the value obtained when accurate adders are used, produces a filtered image which has more edges than other approximate adders. These images have more number of pixels which are closer to white.

CHAPTER 4

Adder Comparison

4.1 Error Metrics

To evaluate the accuracy and compare the approximate circuits, different error measures(Vasicek, 2019)(Jiang *et al.*, 2015) are implemented. Simulation based error measurement is implemented in Python.

4.1.1 Error Rate (ER)

Error rate/ error probability is the measure of probability of occurrence of an error in an approximate circuit.
For a Monte-carlo simulation error rate is found as

$$e_{prob} = \frac{1}{N} Count(f \neq \hat{f})$$

where N is the number of simulations. f is the accurate output and $\hat{f}$ is the approximate output
For simulation based approach, error rate is found as

$$e_{prob}(f, \hat{f}) = \frac{1}{2^{2n}} \sum_{\forall x \in \mathbb{B}^n} [\![f(x) \neq \hat{f}(x)]\!]$$

where n is the number of bits in the input.

4.1.2 Error Distance (ED)

Error distance is the absolute difference between the output from accurate circuit and output from approximate circuit.

$$ED = |f - \hat{f}|$$

4.1.3 Hamming Distance (HD)

Hamming distance is the number of bits in a particular approximate output which are different from the corresponding accurate output.
For an m bit output, hamming distance is

$$HD = \sum_{i=0}^{m-1} f_i(x) \oplus \hat{f}_i(x)$$

4.1.4 Mean Hamming Distance (MHD)

Average of hamming distance across all possible input combinations

$$e_{mhd}(f, \hat{f}) = \frac{1}{2^{2n}} \sum_{\forall x \in \mathbb{B}^n} \sum_{i=0}^{m-1} f_i(x) \oplus \hat{f}_i(x)$$

4.1.5 Mean Absolute Error (MAE)

Average-case arithmetic error or mean absolute error is the mean of the absolute difference between the outputs from the accurate circuit and approximate circuit.
For a Monte-carlo simulation MAE is found as

$$e_{mae} = \frac{1}{n} \sum_{i=0}^{n-1} |f_i - \hat{f}_i|$$

For a simulation based approach, MAE is found as

$$e_{mae}(f, \hat{f}) = \frac{1}{2^{2n}} \sum_{\forall x \in \mathbb{B}^n} |nat(f(x)) - nat(\hat{f}(x))|$$

where $nat(f(x))$ is the decimal value of f

4.1.6 Worst-case arithmetic error (WCE)

Error magnitude/ worst-case error is the maximum possible error an approximate circuit can produce.

$$e_{wce}(f, \hat{f}) = \max_{\forall x \in \mathbb{B}^n} |nat(f(x)) - nat(\hat{f}(x))|$$

4.1.7 Mean Squared Error (MSE)

Mean squared error is the mean of sum of squared difference between the accurate output and approximate output

$$e_{mse}(f, \hat{f}) = \frac{1}{2^{2n}} \sum_{\forall x \in \mathbb{B}^n} \Big(nat(f(x)) - nat(\hat{f}(x)) \Big)^2$$

4.2 Error Metrics computation

4.2.1 Binary Decision Diagram and Boolean Satisfiability

Binary Decision Diagrams are rooted, directed acyclic graphs which are used to represent a Boolean function similar to a binary tree. The BDD are canonical (ROBDD) if the order of variables is fixed. In the BDD every node is labeled with a variable, and each node has a high child and low child. The BDD terminate either at 1 or 0 (leaf nodes).

Boolean Satisfiability problem, which is applicable to a Boolean function $g : \mathbb{B}^n \rightarrow \mathbb{B}$, is used to find an input assignment a which satisfies the condition $g(a) = 1$ or inform there are no such input a exists. Here the assignment a is called satisfiable assignment.

SATOne(g) is a function which finds a single satisfiable assignment.

SATCount(g) is a function which finds the total number possible satisfiable assignments for the function g.

The SATOne and SATCount functions can be obtained using ROBDDs. When a path is traced from leaf node 1 to the root node, a possible input combination which satisfies $g(a) = 1$ can be found. If all such paths are found, we can compute SATCount.

4.2.2 Miter circuit

To compute the difference between the accurate output and the approximate circuit, a miter circuit is implemented. The miter circuit is represented as a CNF or as an ROBDD and further analyzed.

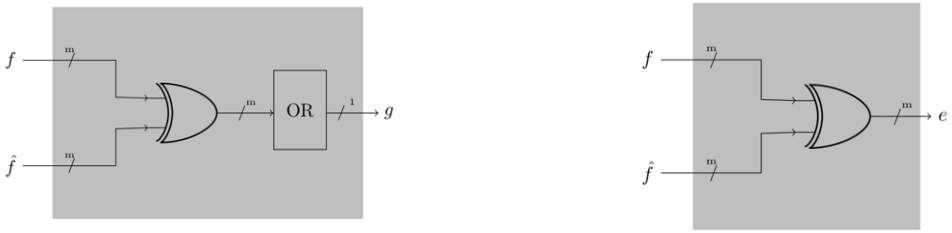


Figure 4.1: Miter Circuits for analysis of a) error rate b)average hamming distance

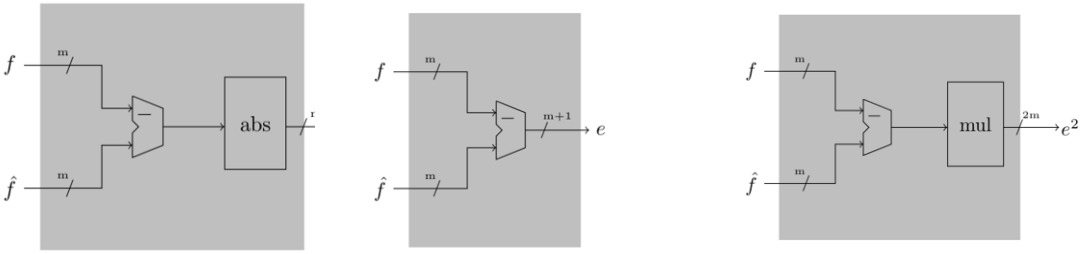


Figure 4.2: Miter Circuits for analysis of a) Absolute ED b)ED c) Squared ED (Vasicek, 2019)

4.2.3 Computation of the Error Metrics using satisfiability(Vasicek, 2019)

Error Probability

$$
\begin{aligned}
e_{prob}(f,\hat{f}) &= \frac{1}{2^{2n}} \sum_{\forall x \in \mathbb{B}^n} [\![f(x) \neq \hat{f}(x)]\!] \\
&= \frac{1}{2^{2n}} \sum_{\forall x \in \mathbb{B}^n} \left(\bigvee_{0 \leq i \leq m} f_i(x) \oplus \hat{f}_i(x) \right) \\
&= \frac{1}{2^{2n}} SATCount \left(\bigvee_{0 \leq i \leq m} f_i(x) \oplus \hat{f}_i(x) \right)
\end{aligned}
$$

Average Hamming Distance

$$
\begin{aligned}
e_{mhd}(f,\hat{f}) &= \frac{1}{2^{2n}} \sum_{\forall x \in \mathbb{B}^n} \left(\sum_{i=0}^{m-1} f_i(x) \oplus \hat{f}_i(x) \right) \\
&= \frac{1}{2^{2n}} \sum_{i=0}^{m-1} \left(\sum_{\forall x \in \mathbb{B}^n} f_i(x) \oplus \hat{f}_i(x) \right) \\
&= \frac{1}{2^{2n}} \sum_{i=0}^{m-1} SATCount(f_i(x) \oplus \hat{f}_i(x))
\end{aligned}
$$

Mean Absolute Error

$$
\begin{aligned}
\text{Let } D(x) &= |nat(f(x)) - nat(\hat{f}(x))| \\
d &= nat^{-1}(D) \\
\text{Then } e_{mae}(f,\hat{f}) &= \frac{1}{2^{2n}} \sum_{\forall x \in \mathbb{B}^n} D(x) \\
&= \frac{1}{2^{2n}} \sum_{\forall x \in \mathbb{B}^n} \left(\sum_{i=0}^{m-1} 2^i d_i(x) \right) \\
&= \frac{1}{2^{2n}} \sum_{i=0}^{m-1} \left(2^i \sum_{\forall x \in \mathbb{B}^n} d_i(x) \right) \\
&= \sum_{i=0}^{m-1} 2^{i-2n} \cdot SATCount(d_i)
\end{aligned}
$$

Mean Squared Error

$$
\begin{aligned}
e_{mse}(f,\hat{f}) =& \frac{1}{2^{2n}} \sum_{\forall x \in \mathbb{B}^n} \left(-2^m + \sum_{i=0}^{m-1} 2^i e_i \right)^2 \\
=& \frac{1}{2^{2n}} \sum_{\forall x \in \mathbb{B}^n} \left(\sum_{i=0}^{m} 2^{2i} e_i + \sum_{i,j=0;j>i}^{m-1} 2^{1+i+j} e_i e_j - \sum_{i=0}^{m-1} 2^{1+i+m} e_i e_m \right) \\
=& \frac{1}{2^{2n}} \left[\sum_{i=0}^{m} 2^{2i} SATCount(e_i) - \sum_{i=0}^{m-1} 2^{1+i+m} SATCount(e_i \wedge e_m) \right. \\
& \left. + \sum_{i,j=0;j>i}^{m-1} 2^{1+i+j} SATCount(e_i \wedge e_j) \right]
\end{aligned}
$$

4.2.4 Computation of Error Metrics - Simplified Algorithm(Vasicek, 2019)

Mean Absolute Error

Algorithm 1: Computation of Mean Absolute Error

Input: approximation miter with signed output (e)

Output: mean absolute error (e_{mae})

$c \leftarrow SATCount(e_m)$, $\epsilon \leftarrow c$;

for $i \in \{0, 1, ..., m-1\}$ **do**

  if *c>0* **then**

    $\epsilon \leftarrow \epsilon + 2^i SATCount(e_i \oplus e_m)$;

  else

    $\epsilon \leftarrow \epsilon + 2^i SATCount(e_i)$;

  end

end

return $2^{-2n}\epsilon$

Mean squared Error

Algorithm 2: Computation of Mean Squared Error

Input: approximation miter with signed output (e)

Output: mean squared error (e_{sae})

$\epsilon \leftarrow 0$;

for $i \in \{0, 1, ..., m\}$ **do**

  $c \leftarrow SATCount(e_i)$, $\epsilon \leftarrow \epsilon + 2^{2i}c$;

  if *c>0* **then**

    for $j \in \{i+1, ...m\}$ **do**

      $c \leftarrow SATCount(e_i \wedge e_j)$;

      if *j=m* **then**

        $c = -c$

      $\epsilon = \epsilon + 2^{i+j+1}c$

    end

end

return $2^{-2n}\epsilon$

Worst-case arithmetic Error

Algorithm 3: worst-Case Error Analysis

Input: approximation miter with signed output (e)

Output: maximum absolute arithmetic error (e_{wce})

$\epsilon \leftarrow 0$, $\mu \leftarrow true$, $sgn \leftarrow SATOne(e_m) \neq \phi$

$d \leftarrow$ **if** *sgn* **then** $e \oplus e_m$ **else** e;

;

for $i \in \{m-1, m-2, ..., 0\}$ **do**

  if $(\gamma \leftarrow SATOne(\mu \wedge d_i)) \neq \phi$ **then**

    $\mu \leftarrow \mu \wedge d_i$;

    $\epsilon \leftarrow \epsilon + 2^i$;

end

if $d_m(\gamma) \vee SATOne(\mu \wedge e_m)$ **then**

  $\epsilon \leftarrow \epsilon + 1$

return ϵ

4.3 PyEDA

The algorithms discussed in the previous section are implemented in Python. For the implementation an electronic design automation Python library called PyEDA is used. All the approximate adders which can be divided into accurate and approximate parts are implemented using PyEDA. Each bit in the inputs to the adders are in the form of variables. Each bit in the output obtained from these adders are in the form of an expression. Accurate sum calculation is done using inbuilt method(ripple carry adder) in the PyEDA.

The output from accurate adder and approximate adders are sign extended to calculate the miter circuit output. The BDDs are constructed according to the algorithm to calculate error metrics using inbuilt method in PyEDA. An expression can be converted to a BDD using this method. The BDD constructed is used to find SATCount and SATOne functions. Using the output from the SATCount and SATOne functions, the algorithms to calculate the error metrics are implemented.

4.4 Modification and Results

In the PyEDA the construction of BDDs takes longer as the number of bits in the input and the number of approximate bits increases. Hence, for the adders which are in the form of approximate part and accurate part, the algorithms can be modified so that it requires only fewer number of BDD constructions for the computation of error metrics. For an n bit adder with k approximate bits, the approximate sum and approximate sum are modified as

modified Approximate sum = approximate sum[0:k-1] + carry from approximate part to accurate part

modified Accurate sum = accurate sum[0:k-1] + carry from (k-1)th bit to (k)th bit

Also the codes are modified to find the error metric when the static probability of the bits are not 0.5.

The results are done using the algorithms and compared the results with Monte-carlo simulation.

4.4.1 Mean Absolute Error

10 Bit adder with 5 bit approximate

1. **LOA**

Monte carlo - 5.877865(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	0.0773125
0.1	0.307
0.15	0.6823125
0.2	1.192
0.25	1.8203125
0.3	2.547
0.35	3.3473125
0.4	4.192
0.45	5.0473125
0.5	5.875

Static Probability	MAE
0.55	6.6323125
0.6	7.272
0.65	7.7423125
0.7	7.987
0.75	7.9453125
0.8	7.552
0.85	6.7373125
0.9	5.427
0.95	3.5423125

Table 4.1: MAE for 10 bit LOA with 5 bit approx.

2. **ETA1**

Monte carlo - 6.333775(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	0.018927665
0.10	0.100691148
0.15	0.280640557
0.20	0.590539878
0.25	1.05777836
0.30	1.70479289
0.35	2.54871489
0.40	3.60124457
0.45	4.86874507
0.50	6.35253906

Static Probability	MAE
0.55	8.04938138
0.60	9.95207363
0.65	12.0501803
0.70	14.3308019
0.75	16.7793589
0.80	19.3803403
0.85	22.1179756
0.90	24.9767942
0.95	27.9420499

Table 4.2: MAE for 10 bit ETA1 with 5 bit approx.

3. **AMA5**

Monte carlo - 8.003684(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	1.475
0.1	2.8
0.15	3.975
0.2	5.
0.25	5.875
0.3	6.6
0.35	7.175
0.4	7.6
0.45	7.875
0.5	8.

Static Probability	MAE
0.55	7.975
0.6	7.8
0.65	7.475
0.7	7.
0.75	6.375
0.8	5.6
0.85	4.675
0.9	3.6
0.95	2.375

Table 4.3: MAE for 10 bit AMA5 with 5 bit approx.

4. **AMA4**

Monte carlo - 8.253698(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	2.65720632
0.1	4.58919496
0.15	5.98379178
0.2	6.97483919
0.25	7.65776825
0.3	8.10064558
0.35	8.35189641
0.4	8.4456321
0.45	8.40528745
0.5	8.24609375

Static Probability	MAE
0.55	7.97677267
0.6	7.60072765
0.65	7.11692807
0.7	6.5206212
0.75	5.80396271
0.8	4.95662121
0.85	3.96638273
0.9	2.81974845
0.95	1.50248068

Table 4.4: MAE for 10 bit AMA4 with 5 bit approx.

5. **AMA3**

Monte carlo - 8.896184(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	27.95651912
0.1	24.93868278
0.15	21.98070928
0.2	19.13680712
0.25	16.47842216
0.3	14.08915405
0.35	12.0578917
0.4	10.47086008
0.45	9.4033696
0.5	8.91210938

Static Probability	MAE
0.55	9.02882029
0.6	9.75611576
0.65	11.06608239
0.7	12.90208161
0.75	15.18388176
0.8	17.81587016
0.85	20.69762172
0.9	23.7355267
0.95	26.85349985

Table 4.5: MAE for 10 bit AMA3 with 5 bit approx.

6. **AMA2**

Monte carlo - 7.377785(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	27.90330897
0.1	24.81995794
0.15	21.77128551
0.2	18.79873597
0.25	15.96740913
0.3	13.36514544
0.35	11.09685415
0.4	9.27470377
0.45	8.00548825
0.5	7.37695312

Static Probability	MAE
0.55	7.44510907
0.6	8.22457078
0.65	9.68373124
0.7	11.74611174
0.75	14.2985096
0.8	17.20559524
0.85	20.32938199
0.9	23.55050109
0.95	26.78645612

Table 4.6: MAE for 10 bit AMA2 with 5 bit approx.

7. **AMA1**

Monte carlo - 4.560227(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	2.6460223
0.1	4.50907787
0.15	5.74174338
0.2	6.46199788
0.25	6.76483727
0.3	6.73049488
0.35	6.42989201
0.4	5.92787988
0.45	5.28477582
0.5	4.55664062

Static Probability	MAE
0.55	3.79469155
0.6	3.04419348
0.65	2.34312125
0.7	1.72083568
0.75	1.19696617
0.8	0.78064108
0.85	0.4701542
0.9	0.25310027
0.95	0.10695338

Table 4.7: MAE for 10 bit AMA1 with 5 bit approx.

8. **AXA1**

Monte carlo - 8.534331(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	2.67385366
0.1	4.61297275
0.15	5.98657064
0.2	6.94086451
0.25	7.59164429
0.3	8.02466952
0.35	8.30035322
0.4	8.45991322
0.45	8.53095628
0.5	8.53125

Static Probability	MAE
0.55	8.47017661
0.6	8.34799411
0.65	8.15349693
0.7	7.86091387
0.75	7.42684937
0.8	6.78770688
0.85	5.8572759
0.9	4.52295778
0.95	2.63739541

Table 4.8: MAE for 10 bit AXA1 with 5 bit approx.

9. **AXA2**

Monte carlo - 10.074008(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	25.77913143
0.1	21.93037771
0.15	19.17051921
0.2	17.21740421
0.25	15.80187988
0.3	14.68270261
0.35	13.66158201
0.4	12.59531131
0.45	11.40239284
0.5	10.0625

Static Probability	MAE
0.55	8.60834396
0.6	7.11085435
0.65	5.65985548
0.7	4.34344309
0.75	3.2298584
0.8	2.35563653
0.85	1.72299207
0.9	1.30761547
0.95	1.07510814

Table 4.9: MAE for 10 bit AXA2 with 5 bit approx.

10. **AXA3**

Monte carlo - 8.99608(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	2.93718855
0.1	5.51536036
0.15	7.68143314
0.2	9.38314383
0.25	10.5769043
0.3	11.2366335
0.35	11.36183758
0.4	10.98323927
0.45	10.16460592
0.5	9.0

Static Probability	MAE
0.55	7.60638661
0.6	6.11228836
0.65	4.6438888
0.7	3.31056015
0.75	2.19213867
0.8	1.33030134
0.85	0.72601963
0.9	0.34418903
0.95	0.12506636

Table 4.10: MAE for 10 bit AXA3 with 5 bit approx.

Variation of MAE with static probability

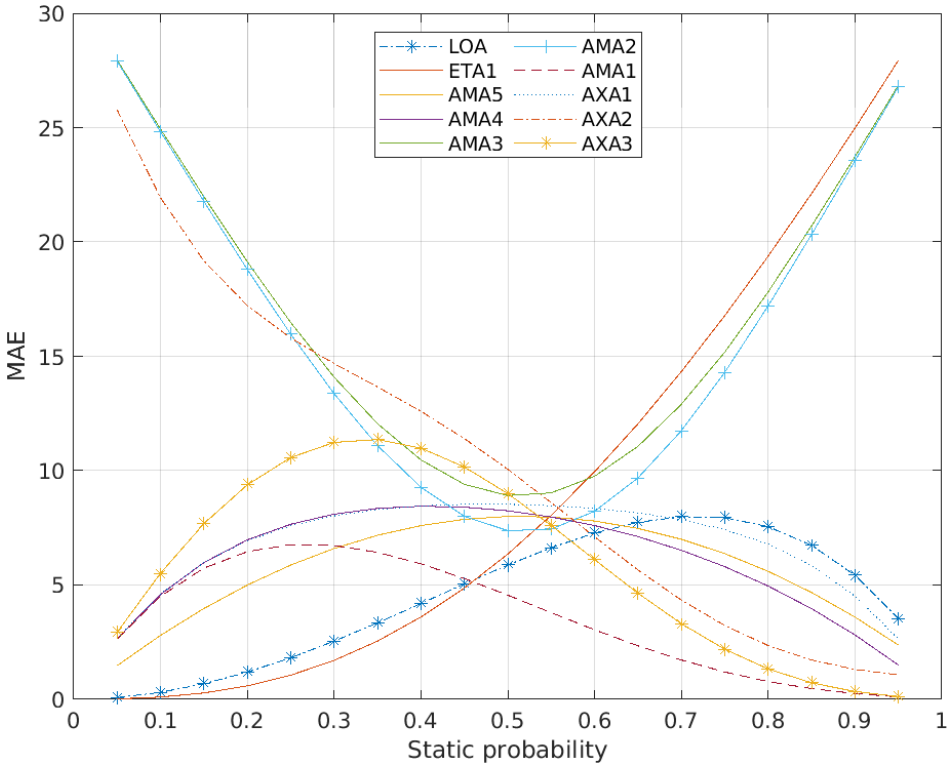


Figure 4.3: Variation of MAE with static probability - 10 bit adder 5 bit apprx.

For the adders as the static probability increases, the chance of occurrence of an input with higher magnitude increases. As the probability of occurrence of 1 increases,

so does the probability that in the accurate sum calculation, the carry from (bit k) to (bit k+1) becomes 1.

For LOA[4.1] the carry from approximate part to accurate part is 0 for 3 out of 4 cases. So this introduces error in the accurate part of the LOA if the actual carry is supposed to be 1. So as the static probability increases, the error increases, so the MAE increases. But as the static probability increases beyond a point, the chance of occurrence of (1,1) input combination at the MSB of the approximate part increases, so the carry from approximate part to the accurate part becomes 1. Thus reducing the magnitude of error.

Similarly, for ETA1[4.2], there is no carry from approximate part to the accurate part. Thus, as the static probability increases, the chance of occurrence of higher magnitude inputs increases, so the chance of occurrence of error in the accurate part of ETA1 increases, so the magnitude of MAE increases.

For smaller static probability, the probability of the input being a smaller magnitude number becomes higher. So the probability that the carry from approximate part to the accurate part being a 0 becomes higher. For AMA4[4.4] and AMA5[4.3], the carry from approximate part to the accurate part is the k^{th} bit in the first input. So the carry from approximate part to the accurate part becomes 0 for most cases. So the magnitude of error becomes less in the accurate part of the adder. As the static probability increases, the probability of carry becoming a 1 also increases. So the error increases. But beyond 0.5, the probability of accurate carry from k^{th} bit to the $k+1^{th}$ bit becoming a 1 increases. Similarly, probability of the carry from approximate part to the accurate part of AMA4/AMA5 becoming a 1 also increases. So after 0.5, the error decreases.

16 bit adder with 10 bit approximate

1. **LOA**

Monte carlo - 192.195679(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	2.5511125
0.10	10.1278000
0.15	22.5001125
0.20	39.2848000
0.25	59.9453125
0.30	83.7918000
0.35	109.981112
0.40	137.516800
0.45	165.249113
0.50	191.875000

Static Probability	MAE
0.55	215.938112
0.60	235.828800
0.65	249.784113
0.70	255.887800
0.75	252.070312
0.80	236.108800
0.85	205.627112
0.90	158.095800
0.95	90.8321125

Table 4.11: MAE for 16 bit LOA with 10 bit approx.

2. **ETA1**

Monte carlo - 204.608021(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	0.27736694
0.10	2.10242626
0.15	6.89711761
0.20	15.9447184
0.25	30.3399915
0.30	50.9552166
0.35	78.4235044
0.40	113.138824
0.45	155.270495
0.50	204.788737

Static Probability	MAE
0.55	261.497298
0.60	325.069232
0.65	395.082421
0.70	471.052277
0.75	552.459990
0.80	638.775609
0.85	729.475907
0.90	824.057459
0.95	922.045598

Table 4.12: MAE for 16 bit ETA1 with 10 bit approx.

3. **AMA5**

Monte carlo - 255.706835(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	48.595
0.10	92.080
0.15	130.455
0.20	163.720
0.25	191.875
0.30	214.920
0.35	232.855
0.40	245.680
0.45	253.395
0.50	256.000

Static Probability	MAE
0.55	253.495
0.60	245.880
0.65	233.155
0.70	215.320
0.75	192.375
0.80	164.320
0.85	131.155
0.90	92.880
0.95	49.495

Table 4.13: MAE for 16 bit AMA5 with 10 bit approx.

4. **AMA4**

Monte carlo - 265.117801(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	86.593868
0.10	148.437421
0.15	192.708603
0.20	224.089250
0.25	245.739037
0.30	259.845357
0.35	267.943671
0.40	271.109708
0.45	270.077892
0.50	265.316263

Static Probability	MAE
0.55	257.075634
0.60	245.423934
0.65	230.273073
0.70	211.403656
0.75	188.491918
0.80	161.143041
0.85	128.935358
0.90	91.481167
0.95	48.512015

Table 4.14: MAE for 16 bit AMA4 with 10 bit approx.

5. **AMA3**

Monte carlo - 285.24595(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	920.781140
0.10	818.953295
0.15	718.573911
0.20	621.519083
0.25	530.475625
0.30	448.791426
0.35	380.168083
0.40	328.219820
0.45	295.968517
0.50	285.378716

Static Probability	MAE
0.55	297.042305
0.60	330.092897
0.65	382.371656
0.70	450.798952
0.75	531.854168
0.80	622.047940
0.85	718.291502
0.90	818.114367
0.95	919.732074

Table 4.15: MAE for 16 bit AMA3 with 10 bit approx.

6. **AMA2**

Monte carlo - 239.320233(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	920.709396
0.10	818.544323
0.15	716.957821
0.20	617.037901
0.25	520.835050
0.30	431.605800
0.35	353.793114
0.40	292.584550
0.45	253.029036
0.50	238.910971

Static Probability	MAE
0.55	251.759678
0.60	290.391313
0.65	351.195693
0.70	429.069545
0.75	518.623534
0.80	615.207150
0.85	715.445800
0.90	817.263914
0.95	919.592146

Table 4.16: MAE for 16 bit AMA2 with 10 bit approx.

7. **AMA1**

Monte carlo - 137.99217(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	86.2046327
0.10	145.695101
0.15	184.502522
0.20	206.777495
0.25	215.610035
0.30	213.505355
0.35	202.661796
0.40	185.126635
0.45	162.869562
0.50	137.796412

Static Probability	MAE
0.55	111.718596
0.60	86.2919121
0.65	62.9388207
0.70	42.7689184
0.75	26.5117094
0.80	14.4732791
0.85	6.52408713
0.90	2.11942597
0.95	0.34813864

Table 4.17: MAE for 16 bit AMA1 with 10 bit approx.

8. **AXA1**

Monte carlo - 273.095917(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	87.7287540
0.10	150.236069
0.15	193.676976
0.20	223.318303
0.25	243.189753
0.30	256.235075
0.35	264.546600
0.40	269.566880
0.45	272.232643
0.50	273.066406

Static Probability	MAE
0.55	272.229474
0.60	269.544799
0.65	264.486982
0.70	256.129948
0.75	243.045150
0.80	223.149111
0.85	193.503797
0.90	150.090161
0.95	87.655077

Table 4.18: MAE for 16 bit AXA1 with 10 bit approx.

9. **AXA2**

Monte carlo - 299.627428(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	845.1588320
0.10	714.154924
0.15	620.395694
0.20	554.210081
0.25	506.180388
0.30	467.666238
0.35	431.481454
0.40	392.578980
0.45	348.508605
0.50	299.416016

Static Probability	MAE
0.55	247.489103
0.60	195.981263
0.65	148.132586
0.70	106.351038
0.75	71.8753504
0.80	44.9035278
0.85	24.9823542
0.90	11.4177890
0.95	3.5672407

Table 4.19: MAE for 16 bit AXA2 with 10 bit approx.

10. **AXA3**

Monte carlo - 258.41887(Assumes input static probability is 0.5)

Static Probability	MAE
0.05	96.9168571
0.10	181.897820
0.15	253.002225
0.20	308.155214
0.25	345.404053
0.30	363.324706
0.35	361.535964
0.40	341.188310
0.45	305.227550
0.50	258.250000

Static Probability	MAE
0.55	205.890537
0.60	153.873698
0.65	107.015877
0.70	68.5011781
0.75	39.6364768
0.80	20.0834265
0.85	8.3825635
0.90	2.5234197
0.95	0.3854268

Table 4.20: MAE for 16 bit AXA3 with 10 bit approx.

4.4.2 Mean Squared Error

10 bit adder with 5 bit approximate

1. **LOA**

Monte carlo - 64.048423(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	0.8503750
0.10	3.376000
0.15	7.500375
0.20	13.096000
0.25	19.984375
0.30	27.936000
0.35	36.670375
0.40	45.856000
0.45	55.110375
0.50	64.000000

Static Probability	MSE
0.55	72.040375
0.60	78.696000
0.65	83.380375
0.70	85.4560000
0.75	84.2343750
0.80	78.9760000
0.85	68.8903750
0.90	53.1360000
0.95	30.8203750

Table 4.21: MSE for 10 bit LOA with 5 bit approx.

2. **ETA1**

Monte carlo - 91.238492(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	0.05909313
0.10	0.46668226
0.15	1.66388787
0.20	4.21260995
0.25	8.78554440
0.30	16.1545775
0.35	27.1778344
0.40	42.7856810
0.45	63.9659936
0.50	91.7490234

Static Probability	MSE
0.55	127.192185
0.60	171.365096
0.65	225.335183
0.70	290.154161
0.75	366.845641
0.80	456.394128
0.85	559.735586
0.90	677.749727
0.95	811.254102

Table 4.22: MSE for 10 bit ETA1 with 5 bit approx.

3. **AMA5**

Monte carlo - 85.605859(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	16.2
0.10	30.7
0.15	43.5
0.20	54.6
0.25	64.0
0.30	71.7
0.35	77.7
0.40	82.0
0.45	84.6
0.50	85.5

Static Probability	MSE
0.55	84.7
0.60	82.2
0.65	78.0
0.70	72.1
0.75	64.5
0.80	55.2
0.85	44.2
0.90	31.5
0.95	17.1

Table 4.23: MSE for 10 bit AMA5 with 5 bit approx.

4. **AMA4**

Monte carlo - 120.784091(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	30.859544
0.10	55.856304
0.15	75.826622
0.20	91.485696
0.25	103.429688
0.30	112.140336
0.35	117.992102
0.40	121.261824
0.45	122.140903
0.50	120.750000

Static Probability	MSE
0.55	117.156253
0.60	111.393024
0.65	103.482151
0.70	93.458736
0.75	81.398438
0.80	67.447296
0.85	51.854072
0.90	35.005104
0.95	17.461694

Table 4.24: MSE for 10 bit AMA4 with 5 bit approx.

5. **AMA3**

Monte carlo - 138.48023(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	812.996564
0.10	679.128998
0.15	559.498818
0.20	454.257878
0.25	363.607269
0.30	287.792629
0.35	227.094846
0.40	181.816059
0.45	152.261543
0.50	138.718750

Static Probability	MSE
0.55	141.435204
0.60	160.597293
0.65	196.312175
0.70	248.594952
0.75	317.363129
0.80	402.439903
0.85	503.567273
0.90	620.429097
0.95	752.683156

Table 4.25: MSE for 10 bit AMA3 with 5 bit approx.

6. **AMA2**

Monte carlo - 112.999564(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	810.076846
0.10	673.435713
0.15	551.048503
0.20	443.001922
0.25	349.497803
0.30	270.842857
0.35	207.429662
0.40	159.710032
0.45	128.161762
0.50	113.250000

Static Probability	MSE
0.55	115.385044
0.60	134.879197
0.65	171.906201
0.70	226.467791
0.75	298.372803
0.80	387.235100
0.85	492.497132
0.90	613.486196
0.95	749.510317

Table 4.26: MSE for 10 bit AMA2 with 5 bit approx.

7. **AMA1**

Monte carlo - 58.605067(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	30.700630
0.10	54.665665
0.15	72.092098
0.20	83.327963
0.25	88.880859
0.30	89.412605
0.35	85.719407
0.40	78.698699
0.45	69.305228
0.50	58.500000

Static Probability	MSE
0.55	47.196390
0.60	36.208048
0.65	26.203184
0.70	17.669412
0.75	10.892578
0.80	5.951853
0.85	2.731888
0.90	0.950983
0.95	0.201973

Table 4.27: MSE for 10 bit AMA1 with 5 bit approx.

8. **AXA1**

Monte carlo - 102.593797(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	31.328592
0.10	56.161860
0.15	74.446149
0.20	87.088920
0.25	95.276367
0.30	100.137803
0.35	102.621571
0.40	103.480734
0.45	103.297209
0.50	102.500000

Static Probability	MSE
0.55	101.356337
0.60	99.933533
0.65	98.043801
0.70	95.193861
0.75	90.565430
0.80	83.051378
0.85	71.365015
0.90	54.226279
0.95	30.608263

Table 4.28: MSE for 10 bit AXA1 with 5 bit approx.

9. **AXA2**

Monte carlo - 186.205952(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	744.655706
0.10	584.588087
0.15	470.180115
0.20	390.932935
0.25	336.847656
0.30	298.902484
0.35	269.534124
0.40	243.025962
0.45	215.720071
0.50	186.000000

Static Probability	MSE
0.55	154.030507
0.60	121.283339
0.65	89.918876
0.70	62.126179
0.75	39.542969
0.80	22.876369
0.85	11.819246
0.90	5.299695
0.95	2.006960

Table 4.29: MSE for 10 bit AXA2 with 5 bit approx.

10. **AXA3**

Monte carlo - 178.479214(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	37.881233
0.10	80.418781
0.15	123.228109
0.20	162.104267
0.25	193.347656
0.30	214.109850
0.35	222.692505
0.40	218.736925
0.45	203.257955
0.50	178.500000

Static Probability	MSE
0.55	147.621425
0.60	114.242796
0.65	81.920692
0.70	53.628594
0.75	31.335938
0.80	15.772226
0.85	6.441505
0.90	1.909816
0.95	0.320927

Table 4.30: MSE for 10 bit AXA3 with 5 bit approx.

Variation of MSE with static probability

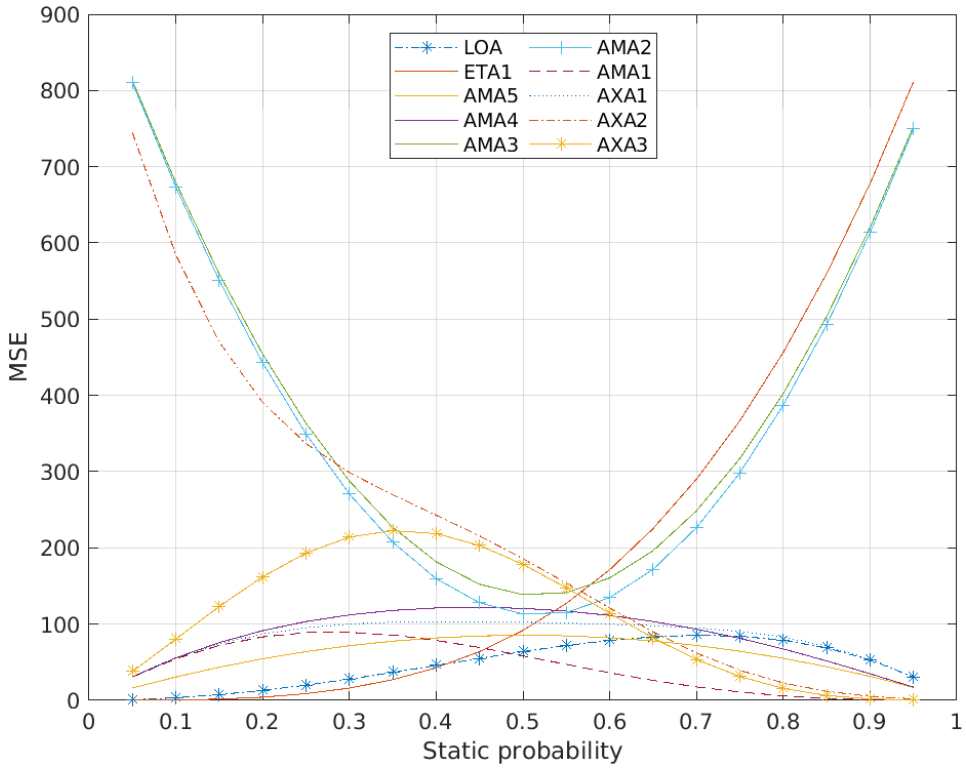


Figure 4.4: Variation of MSE with static probability - 10 bit adder 5 bit apprx

MSE variation with the static probability is similar to the variation of MAE with static probability. For ETA1[4.22] the MSE value increases as the static probability increases. For LOA[4.21] the value increases till certain point and decreases after that point. For AMA4[4.24] and AMA5[4.23] the value increases till 0.5 and decreases after that point.

16 bit adder with 10 bit approximate

1. **LOA**

Monte carlo - 65538.052788(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	871.62798
0.10	3460.29760
0.15	7687.36597
0.20	13421.7616
0.25	20479.9844
0.30	28626.1056
0.35	37571.7680
0.40	46976.1856
0.45	56446.1440
0.50	65536.0000

Static Probability	MSE
0.55	73747.6820
0.60	80530.6896
0.65	85282.0940
0.70	87346.5376
0.75	86016.2344
0.80	80530.9696
0.85	70078.1000
0.90	53792.5536
0.95	30756.8300

Table 4.31: MSE for 16 bit LOA with 10 bit approx.

2. **ETA1**

Monte carlo - 94142.625488(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	36.85699
0.10	351.66077
0.15	1375.54881
0.20	3698.18895
0.25	8054.97398
0.30	15311.0996
0.35	26443.1697
0.40	42519.0480
0.45	64676.7064
0.50	94102.8120

Static Probability	MSE
0.55	132011.745
0.60	179625.658
0.65	238156.092
0.70	308787.530
0.75	392663.165
0.80	490873.032
0.85	604444.532
0.90	734335.309
0.95	881428.330

Table 4.32: MSE for 16 bit ETA1 with 10 bit approx.

3. **AMA5**

Monte carlo - 87472.094592(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	16602.44
0.10	31457.26
0.15	44564.46
0.20	55924.04
0.25	65536.00
0.30	73400.34
0.35	79517.06
0.40	83886.16
0.45	86507.64
0.50	87381.50

Static Probability	MSE
0.55	86507.74
0.60	83886.36
0.65	79517.36
0.70	73400.74
0.75	65536.50
0.80	55924.64
0.85	44565.16
0.90	31458.06
0.95	16603.34

Table 4.33: MSE for 16 bit AMA5 with 10 bit approx.

4. **AMA4**

Monte carlo - 125681.09539(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	31627.5207
0.10	57248.7314
0.15	77740.9117
0.20	93858.6906
0.25	106230.961
0.30	115361.681
0.35	121634.559
0.40	125321.629
0.45	126595.711
0.50	125546.750

Static Probability	MSE
0.55	122202.056
0.60	116550.417
0.65	108570.103
0.70	98260.7590
0.75	85679.1797
0.80	70978.9722
0.85	54454.1052
0.90	36586.3442
0.95	18096.5728

Table 4.34: MSE for 16 bit AMA4 with 10 bit approx.

5. **AMA3**

Monte carlo - 141317.738755(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	880146.951
0.10	729634.752
0.15	595260.431
0.20	477380.518
0.25	376424.355
0.30	292866.913
0.35	227191.314
0.40	179844.534
0.45	151191.742
0.50	141475.929

Static Probability	MSE
0.55	150789.424
0.60	179062.381
0.65	226070.509
0.70	291460.929
0.75	374791.707
0.80	475578.422
0.85	593340.419
0.90	727640.453
0.95	878113.599

Table 4.35: MSE for 16 bit AMA3 with 10 bit approx.

6. **AMA2**

Monte carlo - 116440.565018(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	879984.186
0.10	728924.475
0.15	593252.021
0.20	473065.987
0.25	368739.161
0.30	280963.153
0.35	210724.954
0.40	159189.640
0.45	127495.518
0.50	116507.500

Static Probability	MSE
0.55	126602.883
0.60	157564.543
0.65	208624.973
0.70	278651.228
0.75	366408.321
0.80	470811.508
0.85	591090.135
0.90	726831.622
0.95	877928.612

Table 4.36: MSE for 16 bit AMA2 with 10 bit approx.

7. **AMA1**

Monte carlo - 58248.262005(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	31461.843
0.10	56000.850
0.15	73805.949
0.20	85217.733
0.25	90742.503
0.30	91054.297
0.35	86981.091
0.40	79473.064
0.45	69553.661
0.50	58257.500

Static Probability	MSE
0.55	46562.280
0.60	35323.596
0.65	25221.561
0.70	16726.027
0.75	10083.759
0.80	5326.93204
0.85	2298.95875
0.90	691.64628
0.95	87.26888

Table 4.37: MSE for 16 bit AMA1 with 10 bit approx.

8. **AXA1**

Monte carlo - 104851.335645(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	32410.885
0.10	58140.145
0.15	76821.780
0.20	89396.026
0.25	97203.302
0.30	101588.753
0.35	103733.106
0.40	104583.034
0.45	104821.964
0.50	104857.500

Static Probability	MSE
0.55	104819.156
0.60	104563.686
0.65	103682.097
0.70	101502.729
0.75	97093.739
0.80	89282.459
0.85	76723.921
0.90	58074.174
0.95	32385.432

Table 4.38: MSE for 16 bit AXA1 with 10 bit approx.

9. **AXA2**

Monte carlo - 174905.973824(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	803267.507
0.10	623317.298
0.15	494927.054
0.20	406369.339
0.25	346280.404
0.30	304196.724
0.35	271248.113
0.40	240859.092
0.45	209217.595
0.50	175274.000

Static Probability	MSE
0.55	140174.145
0.60	106259.189
0.65	75960.9113
0.70	50963.2388
0.75	31857.0463
0.80	18271.7484
0.85	9274.15417
0.90	3788.71197
0.95	896.79798

Table 4.39: MSE for 16 bit AXA2 with 10 bit approx.

10. **AXA3**

Monte carlo - 161651.4032(Assumes input static probability is 0.5)

Static Probability	MSE
0.05	39378.090
0.10	84333.899
0.15	129834.386
0.20	170882.788
0.25	202899.440
0.30	222249.188
0.35	226832.038
0.40	216562.668
0.45	193519.493
0.50	161603.500

Static Probability	MSE
0.55	125716.333
0.60	90675.842
0.65	60221.8505
0.70	36437.9259
0.75	19728.7225
0.80	9248.1542
0.85	3510.95516
0.90	923.03438
0.95	101.18438

Table 4.40: MSE for 16 bit AXA3 with 10 bit approx.

4.4.3 Worst-case error

Worst-case error does not depend on the static probability.

Adder	WCE	Monte-carlo
LOA	16	16
ETA1	31	31
AMA5	16	16
AMA4	31	31
AMA3	31	31
AMA2	31	31
AMA1	31	31
AXA1	21	21
AXA2	31	31
AXA3	31	31

Table 4.41: WCE for 10 bit adder with 5 bit approximate bits

Adder	WCE	Monte-carlo
LOA	512	512
ETA1	1023	1023
AMA5	512	512
AMA4	1023	1023
AMA3	1023	1023
AMA2	1023	1023
AMA1	1023	1023
AXA1	682	682
AXA2	1023	1023
AXA3	1023	1023

Table 4.42: WCE for 16 bit adder with 10 bit approximate bits

4.4.4 Error Rate

Probability of occurrence of an error.

Adder	ER	Monte-carlo
LOA	0.7626953125	0.762548
ETA1	0.7626953125	0.76222
AMA5	0.96875	0.968749
AMA4	0.9130859375	0.913877
AMA3	0.9130859375	0.913673
AMA2	0.7626953125	0.761777
AMA1	0.7734375	0.774242
AXA1	0.96875	0.968495
AXA2	1.0	1.0
AXA3	0.7734375	0.774396

Table 4.43: ER for 10 bit adder with 5 bit approximate bits

Adder	ER	Monte-carlo
LOA	0.94368649	0.943718
ETA1	0.94368649	0.943345
AMA5	0.99902344	0.999038
AMA4	0.98956108	0.989658
AMA3	0.98956108	0.989608
AMA2	0.94368649	0.943698
AMA1	0.89736938	0.897563
AXA1	0.99902344	0.999047
AXA2	1.0	1.0
AXA3	0.89736938	0.897338

Table 4.44: ER for 16 bit adder with 10 bit approximate bits

4.4.5 Time take for computation

The simulations are done on a 64-bit system with Intel® Core™ i7-7700 processor having maximum clock frequency of 4200MHz along with 31GB of RAM.

MAE Computation

Adder	Time taken (s)
AMA5	7193.11
AMA4	4793.55
AMA2	10287.35
AMA1	7439.13
ETA1	4443.88
LOA	4677.37
AXA1	11389.94
AXA2	9358.27
AXA3	7612.77

Table 4.45: Time taken for 16 bit adder with 10 approximate bits

MSE Computation

Adder	Time taken (s)
AMA5	6639.41
AMA4	5214.50
AMA3	8000.28
AMA2	11559.13
AMA1	9200.69
ETA1	6055.76
LOA	4871.58
AXA1	13529.24
AXA2	10642.74
AXA3	10504.45

Table 4.46: Time taken for 16 bit adder with 10 approximate bits

REFERENCES

1. **de Oliveira, J.**, **L. Soares**, **E. Costa**, and **S. Bampi**, Exploiting approximate adder circuits for power-efficient gaussian and gradient filters for canny edge detector algorithm. *In 2016 IEEE 7th Latin American Symposium on Circuits Systems (LASCAS)*. 2016.

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

3. **Jiang, H.**, **J. Han**, and **F. Lombardi**, A comparative review and evaluation of approximate adders. *In Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. 2015.

4. **Kim, Y.**, **Y. Zhang**, and **P. Li**, An energy efficient approximate adder with carry skip for error resilient neuromorphic vlsi systems. *In 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2013.

5. **Li, L.** and **H. Zhou**, On error modeling and analysis of approximate adders. *In 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2014.

6. **Mahdiani, H. R.**, **A. Ahmadi**, **S. M. Fakhraie**, and **C. Lucas** (2010). Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, **57**(4), 850–862.

7. **Oliveira Julio, R.**, **L. B. Soares**, **E. A. C. Costa**, and **S. Bampi**, Energy-efficient gaussian filter for image processing using approximate adder circuits. *In 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. 2015.

8. **Vasicek, Z.** (2019). Formal methods for exact analysis of approximate circuits. *IEEE Access*, **7**, 177309–177331.

9. **Yang, Z.**, **A. Jain**, **J. Liang**, **J. Han**, and **F. Lombardi**, Approximate xor/xnor-based adders for inexact computing. *In 2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013)*. 2013.

10. **Zhu, N.**, **W. L. Goh**, and **K. S. Yeo**, An enhanced low-power high-speed adder for error-tolerant application. *In Proceedings of the 2009 12th International Symposium on Integrated Circuits*. 2009.