DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
CHENNAI - 600036

ON THE USE OF APPROXIMATE MULTIPLIER IN ERROR-TOLERANT APPLICATIONS

A Project Report

Submitted by

KISHORE VANKUDOTHU

In the partial fulfilment of requirements

For the award of the degree

Of

MASTER OF TECHNOLOGY

June 18, 2021

DEDICATION

To my beloved family

CERTIFICATE

This is to undertake that the Project report titled **ON THE USE OF APPROXIMATE MULTIPLIER IN ERROR-TOLERANT APPLICATIONS**, submitted by me to the Indian Institute of Technology Madras for the award of M.Tech, is a bonafide record of the research work done by me under the supervision of Dr. Vinita Vasudevan. In whole or in parts, the contents of this Project report have not been submitted to any other Institute or University for the award of any degree or diploma.

Place: Chennai 600 036

Date: 18th June 2021

KISHORE VANKUDOTHU
EE19M048

Prof. Vinita Vasudevan
Project Guide

ACKNOWLEDGEMENTS

First and foremost, I want to express my gratitude to Dr.Vinita Vasudevan, Professor in the Department of Electrical Engineering at IIT Madras, who has supported me throughout the project and provided me with valuable guidance and assistance with tremendous patience over the last twelve months. Whenever I had problems with the project, she would give me suggestions by conducting weekly meets and sending emails. Prof. Vinita Vasudevan had such a significant impact on me with her hard work, dedication to perfection and excellence, and enthusiasm for research, which will also significantly impact my future work in the industry. Thanks a lot for being so patient in explaining things and thank you for being strict, which made me give the best out of me.

I would like to express my sincere gratitude to Janaki Madam for providing all facilities to do this project.

I wish to thank my friends for helping me get through the difficult times and for all the emotional support, entertainment, and care they provided during my MTech. Finally, I love to dedicate this work to my family members for their sacrifice, cooperation, support, affection, and patience shown during the course work, making it possible to complete this work in time.

Thank you all VANKUDOTHU KISHORE

ABSTRACT

Approximate computing is a growing topic of study that involves compromising an application's accuracy in order to make it more efficient. It involves a deliberate effort to make an application inaccurate in order to save cost and resources. Approximation techniques may include the use of arithmetic circuits such as approximate adders and approximate multipliers. As the multiplier is more computationally intensive than the adder, therefore approximate multiplier is considered for this study.

Approximate multiplier have been explored as a possible solution for error-tolerant applications to trade off accuracy for gains in other circuit-based metrics, such as area, power, and delay. Existing approximate multiplier designs have improved many of these operating features significantly. This report presents a comparative evaluation of existing approximate multipliers. Multipliers are evaluated based on Error metrics such as the Mean Error Distance (MED), Normalized Mean Error Distance (NMED), and Mean-Squared Error (MSE) which are obtained using these monte carlo simulations.

In order to assess the effectiveness of these approximate multipliers, few error-tolerant applications are considered. In error-tolerant applications, linear image processing applications such as image sharpening and smoothing are used to demonstrate the effectiveness of the approximation multipliers. The effect of introducing an approximate multiplier in a non linear system is evaluated using an adaptive LMS filter is examined with reference to an accurate multiplier. The filter was used for system identification. Finally, use of approximate multipliers in Neural Network was examined.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology Madras
DSP	Digital Signal Processing
MED	Mean Error Distance
NMED	Normalized Mean Error Distance
MSE	Mean Squared Error
PSNR	Peak Signal-to-Noise Ratio
ETM	Error Tolerant Multiplier
SSM	Static Segment Multiplier
AWTM	Apprximate Wallace Tree Multiplier
R4ABM	Radix-4 Approximate Booth Multiplier
LMS	Least Mean Square
ApproxANN	Approximate Artificial Neural Network
FIR	Finite Impulse Response
IIR	Infinite Impulse Response
ANN	Artificial Neural Network

CHAPTER 1

INTRODUCTION

1.1 Motivation

The motivation for approximation computing stems from the fact that the level of accuracy offered by the computer system in many applications, such as multimedia, signal processing, and machine learning, is greater than the degree of accuracy required by the end-user. As a result, rather than doing exact computations and consuming excessive resources, a small approximation can be used to reduce resources while maintaining acceptable service quality.

In many systems loss of precision is inherent for example noisy components/inputs. In Approximation, on the other hand, errors are deliberately introduced to allow for resource savings at the expense of imprecise results. There are some applications that are suitable to loosing accuracy due to humans inability to distinguish between accurate and inaccurate computation. Multimedia applications, for example, have a lot of room for approximation. Even if the results are incorrect, human senses may be unable to detect the difference, and the service quality may be acceptable. Because the majority of these applications use computationally demanding DSP blocks, adopting approximate hardware would save a significant amount of resources. Approximate computing is motivated by all of these applications.

1.2 Objective

The objective of this work is to have a look at the existing designs of approximate multipliers in the literature and to provide a comparative evaluation of their error characteristics in terms of their error metrics. Approximate multipliers are considered in error-tolerant applications to assess their performance in such applications.

1.3 Problem Statement

To evaluate the effectiveness of using approximate multipliers in linear and nonlinear systems. The systems considered were image sharpening, LMS adaptive filter for system identification, and neural networks.

CHAPTER 2

LITERATURE SURVEY

A digital processor's core components are arithmetic circuits (e.g., adders and multipliers). This survey aims to showcase the current landscape of approximate multipliers, including a solid overview of approximate multipliers in literature.
Approximate multiplier designs primarily employ three approximation techniques: i) approximation in partial product generation, ii) truncation in the partial product tree, and iii) accumulating partial products with approximate adders and or compressors.

2.1 Approximation in Generating Partial Products

The Underdesigned Multiplier (UDM)

By changing one entry in a 2x2 multiplier's K-Map, Kulkarni *et al.* (2011) proposed an approximate 2x2 multiplier block. Larger Underdesigned Multipliers (UDMs) can be produced using the 2x2 block by recursive multiplication. While the adder tree remains accurate, this multiplier design creates error in generating partial products.

2.2 Approximation in the Partial Product Tree

Truncation Multiplier

The area requirement and the power consumption of an array multiplier can be lowered by estimating the least significant columns of the multiplier and adding a correction constant. The least significant columns are set to 0 without generating partial products. This approach was suggested by King and Swartzlander (1997) looking into two types of errors: reduction and rounding error. The correction constant is calculated by calculating expected rounding and reduction errors.

Broken-Array Multiplier (BAM)

The Broken-Array Multiplier (BAM) is a bio-inspired inaccurate multiplier proposed in Mahdiani *et al.* (2009)]. BAM works by eliminating certain lines of carry-save adder cells both horizontally and vertically in the carry-save adder tree.

Error-Tolerant Multiplier (ETM)

The Error-Tolerant Multiplier (ETM) proposed by Kyaw *et al.* (2010) is split into two parts: one is multiplication part and another is non-multiplication part, a control block is used to deal with two cases: i) if the product of the MSBs is zero, then a standard (accurate) multiplier is used to process the LSBs, and ii) if the product of the MSBs is non-zero, a standard multiplier is used to multiply the MSBs while a simple approximate multiplier is used to process the LSBs.

Static Segment Multiplier (SSM)

A similar partition approach (ETM) was used to propose the static segment multiplier (SSM) proposed by Narayanamoorthy *et al.* (2014). In contrast to ETM, the SSM does not apply any approximation to the LSBs. Depending on whether each operand's MSBs are all zeros, the MSBs or LSBs of each operand are accurately multiplied.

Approximate Wallace Tree Multiplier (AWTM)

Bhardwaj *et al.* (2014) proposed a bit-width aware approximate multiplication and a carry-in prediction method are used to create a power and area-efficient Approximate Wallace Tree Multiplier (AWTM). Four $n/2$-bit sub-multipliers implement an n-bit AWTM, with the most significant $n/2$-bit sub-multiplier being further implemented by four $n/4$-bit sub-multipliers. The AWTM is configured into four different modes by the number of approximate $n/4$-bit sub-multipliers in the most significant $n/2$-bit sub-multiplier. The approximate partial products are then accumulated by a Wallace tree.

Radix-4 Approximate Booth Multiplier (R4ABM)

Liu (2014) proposed two approximate Booth encoders and analyzed for error-tolerant computing. Approximate Booth multipliers are designed based on approximate radix-4 modified booth encoding algorithms. A Booth multiplication takes place in three parts: partial product generation using Booth encoder, partial product accumulation and final product generation using fast adder. Here few LSBs of partial products are generated by these booth encoders and MSBs partial products are generated by exact modified booth encoder.

2.3 Using Approximate Adders in the Partial Product Tree

Approximate Multiplier with Configurable Partial Error Recovery (AM)

Liu (2014) presented a basic, yet fast approximate multiplier built using fast approximate adder. This newly constructed adder can process data in parallel by breaking the carry propagation chain (and thus, introducing an error). This adder computes the sum while also generating an error signal; this feature is intended to lower the multiplier's final result error. In the approximation multiplier, a basic tree of approximate adders is employed for partial product accumulation, and error signals are used to correct for inaccuracies for improved accuracy.

Inaccurate Counter based Multiplier (ICM)

Lin and Lin (2013) proposed inaccurate counter based multiplier (ICM), an approximate (4:2) counter is proposed for an inaccurate 4-bit Wallace multiplier. The carry and sum of the counter are approximated as "10" (for "100") when all input signals are '1'. The inaccurate 4-bit multiplier is then utilised to build larger multipliers that include error detection and correction circuits.

Approximate Compressor based Multiplier (ACM)

In the compressor based multiplier, accurate (3:2) and (4:2) compressors are improved to speed up the partial product accumulation stage proposed by Baran *et al.* (2010). Better energy and delay characteristics for a multiplier can be obtained by employing improved compressors. To further reduce delay and power, two approximate (4:2) compressor designs (AC1 and AC2) are presented in Momeni *et al.* (2014).

CHAPTER 3

Evaluation of Approximate Multipliers

Arithmetic circuits (e.g., adders and multipliers) are key components of a DSP processor. This chapter reviews existing approximate multiplier designs. The key approximate arithmetic computing design problem is to develop an efficient low precision computing unit and sometimes an error compensation unit to significantly reduce energy, delay, and or area overhead while achieving a low degree of error. While the area and energy consumption of a given multiplier can often be easily estimated, the key challenge is developing insights on the error to optimize the error compensation scheme, which is focused on in the following section.
The organization of this chapter is as follows. Section 3.1 describes the error metrics for characterizing the performance of the approximate multiplier. Section 3.2 deals with the review of approximate multipliers in the literature.

3.1 Error Metrics

In this section, an analytical framework is presented for assessing the arithmetic multiplier accuracy, i.e., the Mean Error Distance (MED) and Normalized Mean Error Distance (NMED), and Mean Squared Error (MSE) of the approximate multipliers. These results are then used to estimate the Peak Signal-to-Noise Ratio (PSNR) in image processing for image sharpening application discussed in the next chapter.
The *Error Distance* (ED) and the *Mean Error Distance* (MED) and *Mean Squared Error* are proposed by Liang *et al.* (2012) to evaluate the arithmetic performance of approximate circuits.

3.1.1 Error Distance

For an *n*-bit approximate multiplier, the ED is defined as the absolute value of the difference between the approximate (a) and accurate value *b*, i.e.,

$$ED(a,b) = |a-b| = \left|\sum_{i=0}^{n-1} a[i]2^i - \sum_{j=0}^{n-1} b[j]2^j\right| \tag{3.1}$$

3.1.2 Mean Error Distance

The MED (d_m) is defined as the average ED for a given set of input vectors, i.e.,

$$d_m = \frac{1}{N}\sum_{i=0}^{N-1} ED[i] \tag{3.2}$$

Where N is the number of inputs.

3.1.3 Normalized Mean Error Distance

The MED increases exponentially when the number of approximate bits increased. Therefore, MED is an unfair metric when a comparison is made between multiplier with different sizes. To overcome this limitation, a normalized MED, referred to as a normalized error distance, is defined as follows:

$$d_n = \frac{d_m}{D} \tag{3.3}$$

Where d_m is the MED, and D is the maximum value of error.

3.1.4 Mean-Squared Error

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

$$MSE = \frac{1}{N}\sum_{i=0}^{N-1}(ED[i])^2 \tag{3.4}$$

3.2 Approximate Multipliers

Three design methodologies are applicable to approximate a multiplier: i) approximation in generating the partial product, ii) approximation (including truncation) in partial product tree, and iii) using approximate designs of adders, counters, and or compressors to accumulate the partial products.

3.2.1 Error-Tolerant Multiplier (ETM)

Multiplication Algorithm for ETM

Error Tolerant Multiplier proposed by Kyaw *et al.* (2010). This multiplication algorithm is can be explained via an example shown in Fig. 3.1. Firstly, The input operands are split into a multiplication part that includes higher order bits and a non-multiplication part which is made up of the remaining lower order bits. In the example inputs operands divided into two equal parts, and each of which contains 6 input bits. It is called ETM-6 (6 indicated number of bits in the non-multiplication part).

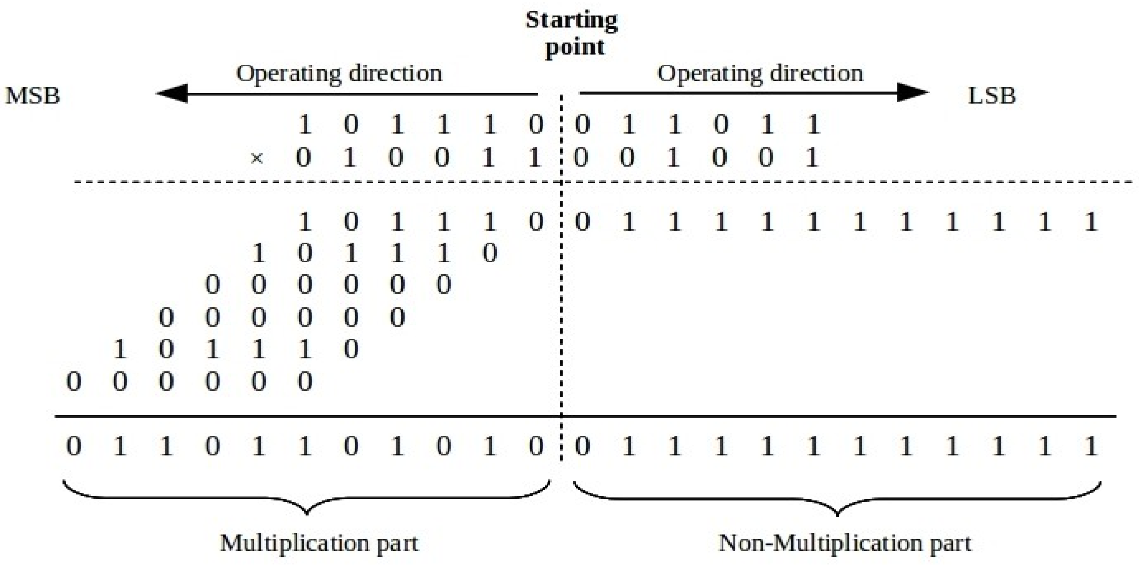


Fig. 3.1: Multiplication algorithm of ETM Kyaw *et al.* (2010)

A special scheme is applied to non-multiplication part - no partial products generated and carry propagation has been removed. From MSB to LSB of non-multiplication part checked and if either or both of the two operands are "1", the checking is brought to an end and from that bit onwards, all product bit positions are set to "1". If both operand bits occurs to be "0", the corresponding product bit set to "0".

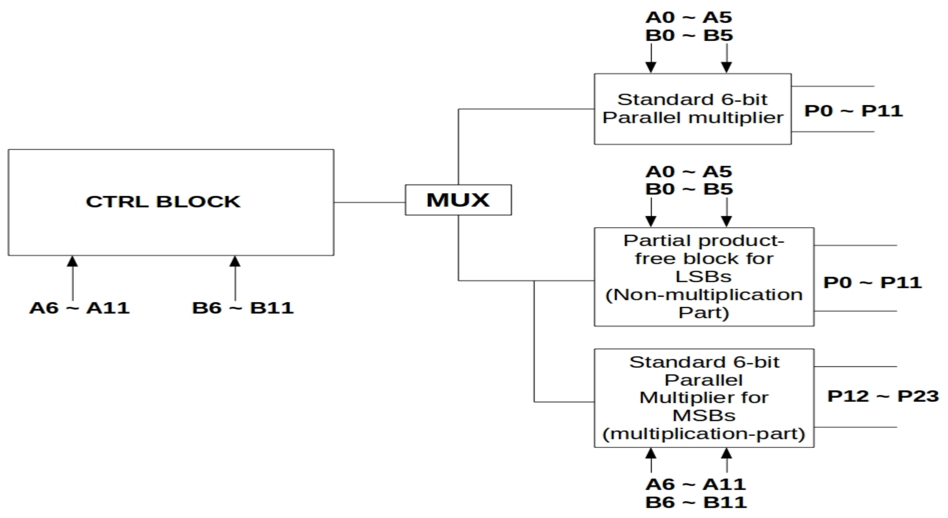

A0 ~ A5
B0 ~ B5
Standard 6-bit Parallel multiplier
P0 ~ P11
CTRL BLOCK
MUX
A0 ~ A5
B0 ~ B5
Partial product-free block for LSBs (Non-multiplication Part)
P0 ~ P11
A6 ~ A11
B6 ~ B11
Standard 6-bit Parallel Multiplier for MSBs (multiplication-part)
P12 ~ P23
A6 ~ A11
B6 ~ B11

Fig. 3.2: Architecture of a 12-bit Error-Tolerant Multiplier (ETM) Kyaw *et al.* (2010)

The ETM is divided into a multiplication section for the MSBs and a non-multiplication section for the LSBs. A NOR gate based control unit is used to deal with two scenarios: i) if the product of MSBs is zero, then non-multiplication part LSBs are accurately multiplied without any approximation, and ii) if the product of MSBs is nonzero, the non-multiplication section is used as an approximate multiplier to process the LSBs, while the multiplication section is activated to multiply the MSBs.

ETM error analysis

The 8×8 and 16×16 bit ETM multiplier was simulated using Monte Carlo Simulation with 10^6 random input combinations, uniformly distributed between 0 to $2^{n-1} - 1$ was used. The program was written in python and run on Intel(R) Core(TM) i7-7700 CPU with 32GB RAM machine. Table 3.1 and Table 3.2 shows MED, NMED and MSE as a function of number of approximate bits. The error grows non-linearly with number of approximate bits.

No. of approximate bits	MED	NMED	$\log_{10}$(MSE)
1	123.96	1.90×10^{-3}	4.45
2	365.11	5.61×10^{-3}	5.32
3	822.20	1.26×10^{-2}	6.00
4	1644.59	2.52×10^{-2}	6.61
5	2962.32	4.55×10^{-2}	7.14
6	4620.64	7.10×10^{-2}	7.56
7	5847.21	8.99×10^{-2}	7.83

Table 3.1: 8×8 bit ETM error analysis as a function of the number of approximate bits.

No. of approximate bits	MED	NMED	$\log_{10}$(MSE)
1	32787.15	7.63×10^{-6}	9.29
2	98365.51	2.29×10^{-5}	10.17
3	229173.78	5.33×10^{-5}	10.88
4	491020.61	1.14×10^{-4}	11.53
5	1015564.23	2.36×10^{-4}	12.16
6	2058723.93	4.79×10^{-4}	12.77
7	4145772.08	9.65×10^{-4}	13.37
8	8286424.02	1.92×10^{-3}	13.98
9	16468685.48	3.83×10^{-3}	14.57
10	32392373.86	7.54×10^{-3}	15.17

Table 3.2: 16×16 bit ETM error as a function of number of approximate bits.

3.2.2 Static Segment Multiplier (SSM)

Narayanamoorthy *et al.* (2014) proposed an approximate multiplication technique that takes *m* consecutive bits (i.e., *m*-bit segment) from each *n*-bit operand, where *m* is equal to or greater than *n/2*. An *m*-bit segment can start from one of two or three fixed bit positions depending on where the leading "1" bit is located for a positive number. This technique can provide much higher accuracy than one simply truncating the LSBs, because this technique can capture more noteworthy bits.

×					01xx	xxxx	xxxx	xxxx
					0001	xxxx	xxxx	xxxx
=	▨	▨	▨	▨	0000	0000	0000	0000
×					01xx	xxxx	xxxx	xxxx
					0000	0000	01xxx	xxxx
=	0000	0000	▨	▨	▨	▨	0000	0000
×					0000	0000	01xx	xxxx
					01xx	xxxx	xxxx	xxxx
=	0000	0000	▨	▨	▨	▨	0000	0000
×					0000	0000	01xx	xxxx
					0000	0000	01xx	xxxx
=	0000	0000	0000	0000	▨	▨	▨	▨

Fig. 3.3: Examples of 16×16 multiplications based on 8-b segments with two possible starting bit positions for 8-b segments. The shaded cells represent 8-b segments and the aligned position of 8×8 multiplication results Narayanamoorthy *et al.* (2014)

With two *m*-bit segments from two *n*-bit operands, we can perform a multiplication using an $m \times m$ multiplier instead of $n \times n$. Furthermore, an $m \times m$ bit consumes much lesser energy compared to an $n \times n$ bit multiplier, because the complexity of multiplier increased quadratically with *n*. The segment for each operand is taken from one of the two possible segments in an *n*-bit operand, a *2m*-bit result can be expanded to a 2n-bit result by left-shifting the *2m*-bit result by one of three possible shift amounts: 1) no shift when both segments are from the lower *m*-bit segments; 2) *(n–m)* shift when two segments are from the upper and lower ones, respectively; and 3) $2 \times (n - m)$ shift when both segments are from the upper ones, as shown in Fig. 3.3. Table 3.3 shows MED, NMED, and MSE as a function of number of m-bit segments for a 8×8 bit multiplier. Table 3.4 shows MED, NMED, and MSE as function of number of m-bit segments for a 16×16 bit multiplier.

SSM error analysis

No. of bits in the segment	MED	NMED	$\log_{10}$(MSE)
4	1783.25	2.74×10^{-2}	6.67
5	857.21	1.13×10^{-2}	6.04
6	373.63	5.74×10^{-3}	5.33
7	125.56	1.93×10^{-3}	4.46

Table 3.3: 8×8 bit SSM error analysis as a function of m-bit segment multiplier.

No. of bits in the segment	MED	NMED	$\log_{10}$(MSE)
8	8329640.00	1.93×10^{-3}	13.98
9	4154038.50	9.67×10^{-4}	13.38
10	2063377.18	4.80×10^{-4}	12.77
11	1014671.27	2.36×10^{-4}	12.16
12	491350.76	1.14×10^{-4}	11.53
13	229437.64	5.34×10^{-5}	10.88
14	98357.14	2.29×10^{-5}	10.17
15	32764.98	7.62×10^{-6}	9.29

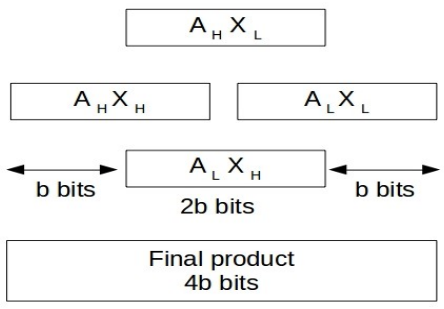
Table 3.4: 16×16 bit SSM error as a function of m-bit segment multiplier.

3.2.3 Approximate Wallace Tree Multiplier (AWTM)

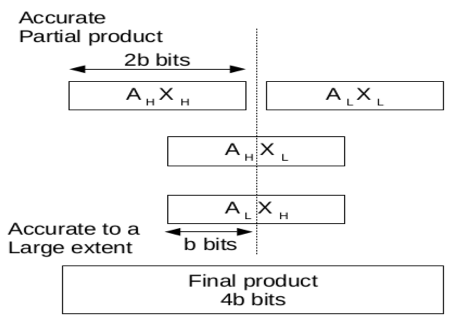
Bhardwaj *et al.* (2014) proposed a power and area-efficient approximate Wallace tree multiplier (AWTM) is based on a bit-width aware approximate multiplication and a carry-in prediction method. An *2b*-bit AWTM is implemented by four *b*-bit sub-multipliers, and the most significant *b*-bit sub-multiplier is further implemented by four *b*/2-bit sub-multipliers.

Recursive Multiplication

Multiplication of an multiplier and an multiplicand can be recursively done by smaller-size multiplications, each of these can be performed in same clock cycle. Let *A* be the multiplicand and X be the multiplier and both are 2b bits each. Now A and X can be written as $A = A_H A_L$ and $X = X_H X_L$ where A_H, A_L, X_H, and X_L are of b bits each. The $2b \times 2b$ multiplication is is performed recursively as shown in Fig. 3.4a.



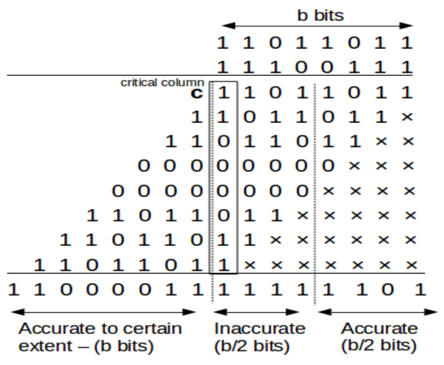
(a) Recursive Multiplication



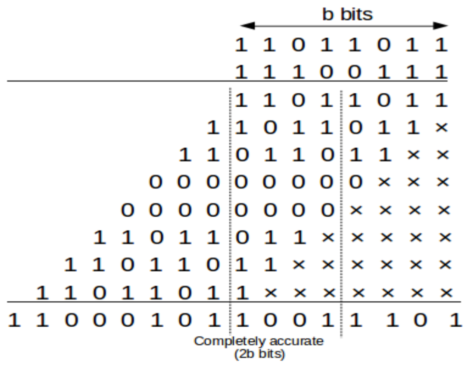
(b) Approximate Multiplication

AWTM Approximate Multiplication

In order to get higher accuracy multiplier $A_H X_H$ made accurate and $A_L X_H$, $A_H X_L$, $A_L X_L$ as $b \times b$ approximate multiplier. For $b \times b$ approximate multiplier upper b bits are accurate to high extent, to make upper *2b* bits of final *4b* bit product achieve high accuracy. An example of $A_L X_L$ $b \times b$ multiplier shown in Fig. 3.5a. The carry-in C is computed by OR of inputs from critical column.



(a) Approximate $A_L X_L$



(b) Accurate $A_L X_L$

In this approximate multiplication, again we divide the $b \times b$ accurate multiplier $A_H X_H$ into 4 smaller $b/2$ multiplier. This is done because, when $A_H X_H$ performed in parallel with $A_H X_L$, $A_L X_H$ and, $A_L X_L$ the critical path is still determined by the accurate multiplier $b \times b$ ($A_H X_H$) multiplier. Therefore, recursively reducing this $A_H X_H$ as $b/2 \times b/2$ will make approximate $b \times b$ multiplier decide the critical path.

Accuracy Configuration Modes

We can vary the accuracy of AWTM by varying the number of multipliers that are accurate in $A_H X_H$ multiplication. But in any case $A_{HH} X_{HH}$ kept accurate, so that accuracy doesn't fall below a certain level. This is so because accuracy of AWTM can be adjusted according to error tolerance of the application. For different modes of operation of AWTM are shown Table 3.5. Where *'A'* stands for accurate multiplier and *I* stands for inaccurate multiplier.

Multiplier	Mode	$A_{HH}X_{HH}$	$A_{HH}X_{HL}$	$A_{HL}X_{HH}$	$A_{HL}X_{HL}$
AWTM-4	4	A	I	I	I
AWTM-4	3	A	A	I	I
AWTM-2	2	A	A	A	I
AWTM-1	1	A	A	A	A

Table 3.5: Modes of operation for accuracy configuration multiplier

AWTM error analysis

8-bit AWTM				**16-bit AWTM**			
Mode	MED	NMED	$\log_{10}$(MSE)	Mode	MED	NMED	$\log_{10}$(MSE)
AWTM-1	390.74	0.5920	5.23	AWTM-1	66377.08	0.1335	9.82
AWTM-2	1091.04	0.6478	6.10	AWTM-2	834475.84	0.4544	11.91
AWTM-3	3905.03	0.6756	7.21	AWTM-3	13716722.41	0.5872	14.34
AWTM-4	6717.02	0.6801	7.67	AWTM-4	25552180.30	0.6040	14.86

Table 3.6: AWTM error analysis as a function of mode of operation of $A_H X_H$ multiplier.

Table 3.6 shows that MED, NMED, and log_{10}(MSE) as a function of mode of operation of $A_H X_H$ multiplier.

3.2.4 Approximate Radix-4 Booth Multiplier (R4ABM)

Liu *et al.* (2017) proposed two approximate Booth encoders and analyzed for error-tolerant computing. Approximate Booth multipliers are designed based on approximate radix-4 modified booth encoding algorithms. A Booth multiplication takes place in three parts: partial product generation using Booth encoder, partial product accumulation and final product generation using fast adder.

Booth encoders has been proposed to enhance the performance of two's complement binary numbers multiplication. It has been further improved radix-4 Booth encoding scheme. Booth encoder plays an vital role in Booth multiplication by reducing the partial product rows by half. Consider a N-bit multiplication with a multiplicand *A* and a multiplier *B* in two's complement given as follows:

$$A = -a_{N-1}2^{N-1} + \sum_{i=0}^{N-2} a_i 2^i \tag{3.5}$$

$$B = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i \tag{3.6}$$

In the Booth encoder, each group is decoded by selecting the partial products as *-2A*, *-A*, 0, *A*, *2A*. The output (partial products, pp_{ij}) of booth encoder is given by:

$$pp_{ij} = (b_{2i} \oplus b_{2i-1})(b_{2i+1} \oplus a_j) + \overline{(b_{2i} \oplus b_{2i-1})}(b_{2i+1} \oplus b_{2i})(b_{2i+1} \oplus a_{j-1}) \tag{3.7}$$

Approximate Radix-4 Booth Encoding Method-1

Table 3.7 shows the K-map of the first approximation radix-4 Booth encoding (R4ABE1) method, where circled 0 represents an entry where a 1 is substituted by a 0. To simplify the Booth encoding, just four entries are changed; the aim for the initial approximate design is to make the truth table as symmetrical as possible while introducing a tiny inaccuracy. As a result, the R4ABE1 design has the advantage of causing a very minor error because only four entries are updated; but, all modifications change a 1 to a 0, therefore the estimated product's absolute value is always smaller than its precise counterpart.

$b_{2i+1}b_{2i}b_{2i-1}$ / $a_j a_{j-1}$	000	001	011	010	110	111	101	100
00	0	0	0	0	**1**	0	**1**	$\textcircled{0}$
01	0	0	$\textcircled{0}$	0	**1**	0	**1**	0
11	0	**1**	$\textcircled{0}$	**1**	0	0	0	0
11	0	**1**	0	**1**	0	0	0	$\textcircled{0}$

Table 3.7: KMAP of ABE1

So from the modified truth table the output of R4ABE1 is given as follows:

$$pp_{ij} = a_j\overline{b_{2i+1}}\,\overline{b_{2i}}b_{2i-1} + a_j\overline{b_{2i+1}}b_{2i}\overline{b_{2i-1}} + \overline{a_j}b_{2i+1}b_{2i}\overline{b_{2i-1}} + \overline{a_j}b_{2i+1}\overline{b_{2i}}b_{2i-1} \tag{3.8}$$

$$pp_{ij} = (b_{2i} \oplus b_{2i-1})(b_{2i+1} \oplus a_j) \tag{3.9}$$

R4ABE1 can significantly lower both the complexity and the critical path latency of Booth encoding when compared to the precise MBE (Eq. 3.7).

Approximate Radix-4 Booth Encoding Method-2

Table 3.8 shows the truth table for the second approximate radix-4 Booth encoding (R4ABE2) technique, where circled "1" signifies a 0 entry that has been substituted by a where 1; eight K-map elements have been updated to simplify the Booth encoding logic. The method for R4ABE2 is to have as few prime implicants (indicated by rectangle) as possible, in addition to having a symmetric truth table with a smaller error. Although the error created by R4ABE2 is approximately double that of R4ABE1, the modification is accomplished by changing not just a 1 to a 0, but also a 0 to a 1. As a result, the approximate product can be larger or smaller than the precise product, and inaccuracies in the partial product reduction process might complement each other. As a result, when employing R4ABE2 in a Booth multiplier, the error may not be as significant as when using R4ABE1.

$b_{2i+1}b_{2i}b_{2i-1}$ / a_ja_{j-1}	000	001	011	010	110	111	101	100
00	0	0	0	0	**1**	$\textcircled{1}$	**1**	**1**
01	0	0	$\textcircled{0}$	0	**1**	$\textcircled{1}$	**1**	$\textcircled{1}$
11	$\textcircled{1}$	**1**	**1**	**1**	0	0	0	0
11	$\textcircled{1}$	**1**	$\textcircled{1}$	**1**	0	0	0	$\textcircled{0}$

Table 3.8: KMAP of ABE2

So from the modified truth table the output of R4ABE2 is given as follows:

$$pp_{ij} = a_j\overline{b_{2i+1}} + \overline{a_j}b_{2i+1} = b_{2i+1} \oplus a_j \tag{3.10}$$

When compared to R4ABE1, R4ABE2 reduces the complexity and critical path latency even more.

Design of Approximate Booth Multiplier

The approximate Booth encoders, namely R4ABE1 and R4ABE2, are employed in the first phase of the approximate Booth multiplier to generate inexact partial products. As a result, an approximation factor *p* (*p*=1, 2,..., 2N) is defined as the number of least significant partial product columns formed by the approximate Booth encoders. The exact adders are used to add up the approximate partial products.

1. R4ABE1 (to generate the *p* least significant partial product columns) and the regular approximate partial product array are used in the first approximation radix-4 Booth multiplier (R4ABM1). The *2N-p* most significant partial product columns are generated using the exact MBE. Both approximation and exact partial products are accumulated using accurate adders.

2. The second approximation radix-4 Booth multiplier (R4ABM2), like R4ABM1, makes use of R4ABE2 (to generate the *p* least significant partial product columns), the regular approximate partial product array, and precise adders.

Results of R4ABM1

Table 3.9 shows MED, NMED, and log_{10}(MSE) as a function of approximate bits for R4ABM1 multiplier.

8-bit Approximate Booth Multipliers				16-bit Approximate Booth Multipliers			
p	MED	NMED	$\log_{10}$(MSE)	*p*	MED	NMED	$\log_{10}$(MSE)
2	127.41	0.00245	5.07	2	64467.08	0.0008275	8.12
4	669.73	0.0062	5.91	4	96318.86	0.005214	10.12
6	1668.82	0.0122	6.94	6	267812.32	0.009872	11.34
8	2422.18	0.0369	8.13	8	474612.54	0.01007	12.86

Table 3.9: R4ABM1 error analysis.

Results of R4ABM2

Table 3.10 shows MED, NMED, and log_{10}(MSE) as a function of approximate bits for R4ABM2 multiplier.

8-bit Approximate Booth Multipliers				16-bit Approximate Booth Multipliers			
p	MED	NMED	$\log_{10}$(MSE)	*p*	MED	NMED	$\log_{10}$(MSE)
2	119.28	0.0024	3.62	2	66382.95	0.002335	9.06
4	975.47	0.0104	4.21	4	110871.43	0.009278	10.41
6	3107.41	0.0239	5.65	6	307943.31	0.01673	12.04
8	5697.56	0.0616	6.08	8	602354.62	0.04896	13.53

Table 3.10: R4ABM2 error analysis.

3.2.5 Approximate Multiplier with Configurable Partial Error Recovery (AM)

Liu *et al.* (2014) proposed an approximate multiplier using a simple, yet fast approximate adder. By breaking the carry propagation chain, this newly built adder can process data in parallel (and thus, introducing an error). Its critical path delay is shorter than that of a traditional one-bit full adder. This adder computes the sum and generates an error signal at the same time; this feature is used to reduce the error in the multiplier's final result. A simple tree of approximate adders is employed for partial product accumulation in the approximation multiplier, and error signals are used to correct for the inaccuracy for improved accuracy.

Approximate Adder

This adder works with a collection of inputs that have been pre-processed. The interchangeability of bits with the same weights in different addends is the basis for input pre-processing (IPP). Consider the following two sets of inputs for a 4-bit adder: i)$A = 1010$, $B = 0101$ and ii)$A = 1111$, $B = 0000$. Clearly, adding i) and ii) yields the same result. Because the corresponding bits in the two operands are interchangeable, the two input bits $A_iB_i = 01$ are identical to $A_iB_i = 10$ (with i being the bit index).

$$\dot{A_i} = A_i + B_i \tag{3.11}$$

$$\dot{B_i} = A_iB_i \tag{3.12}$$

By breaking the carry propagation chain, this adder can process data in parallel. When $\dot{B_i} = 1$, $\dot{A_{i+1}} = 1$, and $\dot{B_{i+1}} = 1$, a carry propagation chain begins at the i-th bit. S_{i+1} is 0 with an exact adder, and the carry is propagated to the higher bit. In the suggested approximate adder, however, S_{i+1} is set to 1 and an error signal $E_{i+1} = 1$ is created. As a result of this the carry signal is unable to propagate to higher bits. By doing so, only the generate signal produces a carry signal, i.e. $C_i = 1$ only when $\dot{B_i} = 1$, and it only propagates to the next higher bit, i.e. the (i+1)-th position.

$\dot{B}_i\dot{B}_{i-1}$	00	01	10	11
$\dot{A}_i$	$\dot{A}_i$	$\dot{A}_i$	1	1
$C_{i-1}/\dot{B}_{i-1}$	0	1	0	1
S_i	$\dot{A}_i$	1	0	1
E_i	0	$\dot{A}_i$	0	0

Table 3.11: Truth table of approximate adder

The logic function of Table 3.11 given by

$$S_i = \dot{B}_{i-1} + \overline{\dot{B}_i}\dot{A}_i \tag{3.13}$$

$$E_i = \overline{\dot{B}_i}\dot{B}_{i-1}\dot{A}_i \tag{3.14}$$

Replacing $\dot{A}_i$ and $\dot{B}_i$ with 3.11 and 3.12, the logic functions with respect to original inputs given by

$$S_i = (A_i \oplus B_i) + A_{i-1}B_{i-1} \tag{3.15}$$

$$E_i = (A_i \oplus B_i)A_{i-1}B_{i-1} \tag{3.16}$$

Partial Product Accumulation

The ease with which approximate adders can be used in partial product accumulation is a key characteristic of this approximate multiplier.

Error Reduction

The error signals can be added using precise adders, and the accumulated error can thus fully compensate for the final product. To reduce complexity, an approximate error accumulation is introduced. The sum of the errors for a single bit is approximated using an OR gate.

$$E_i = E1_iORE2_iOR...OREm_i \tag{3.17}$$

An accumulated error vector is added to the adder tree output using a conventional adder to lessen the error (e.g. a carry look-ahead adder). To further reduce the total complexity, only a few (e.g. k) MSBs of the error signals are used to compensate the outputs. In an 8×8 adder tree, there are a total of 7 error vectors formed by the tree of 7 approximate adders. Because the MSBs of some vectors are less significant than the least significant bits of the k MSBs, not all of the bits in the 7 vectors must be added. In the case of Fig. 3.6, 4 MSBs (i.e. the 11-14th bits) are taken into account for error recovery, resulting in 4 error vectors (i.e. the error vectors of adders A3, A4, A6 and A7).

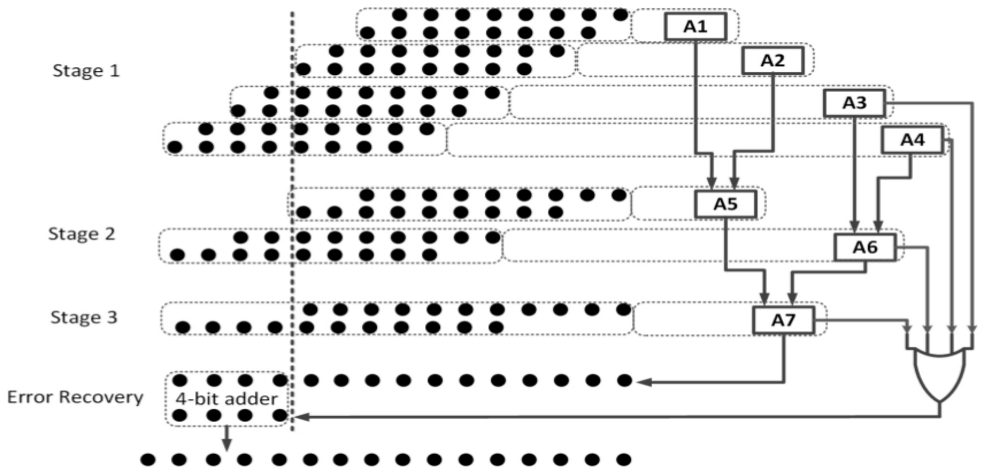


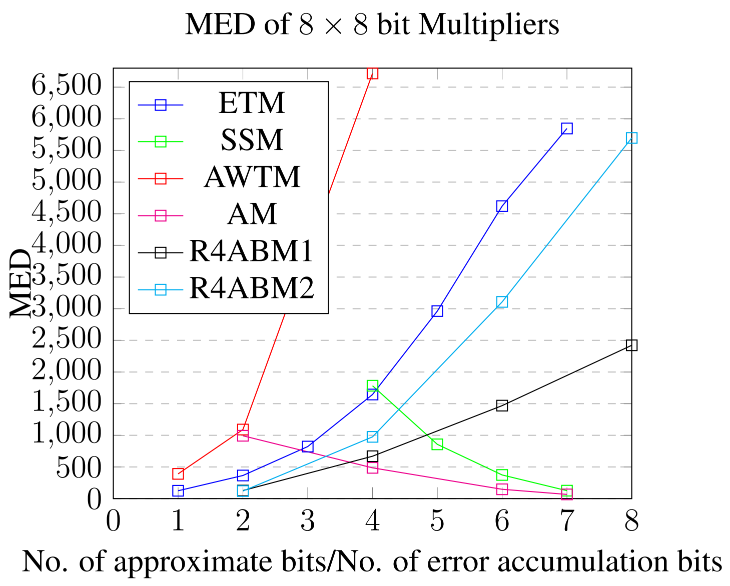
Fig. 3.6: An approximate multiplier with OR-gate based partial error recovery using 4 MSBs of the error vector Liu *et al.* (2014).

Table 3.12 shows MED, NMED, and MSE as a function number of bits for error compensation.

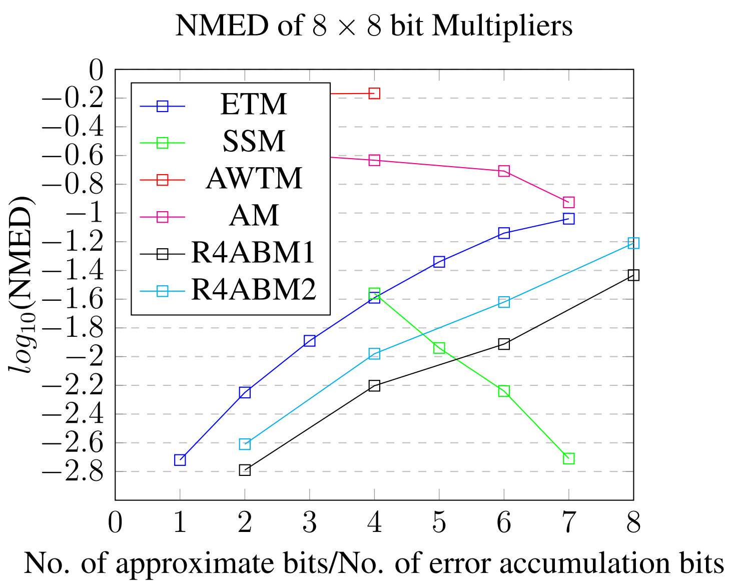
8-bit AM				**16-bit AM**			
k	MED	NMED	MSE ($\log_{10}$)	k	MED	NMED	MSE ($\log_{10}$)
2	992.80	0.2682	6.55	2	70001875.52	0.3851	16.22
4	485.98	0.2330	5.75	4	35391438.34	0.3550	15.44
6	147.27	0.1959	4.65	6	12574934.09	0.3360	14.43
7	66.52	0.1186	3.98	8	3788425.16	0.2979	13.33
				10	1045270.54	0.2476	12.18
				12	270007.69	0.1904	10.99
				14	64850.57	01224.	9.75

Table 3.12: AM error analysis as a function of number of bits for error compensation.

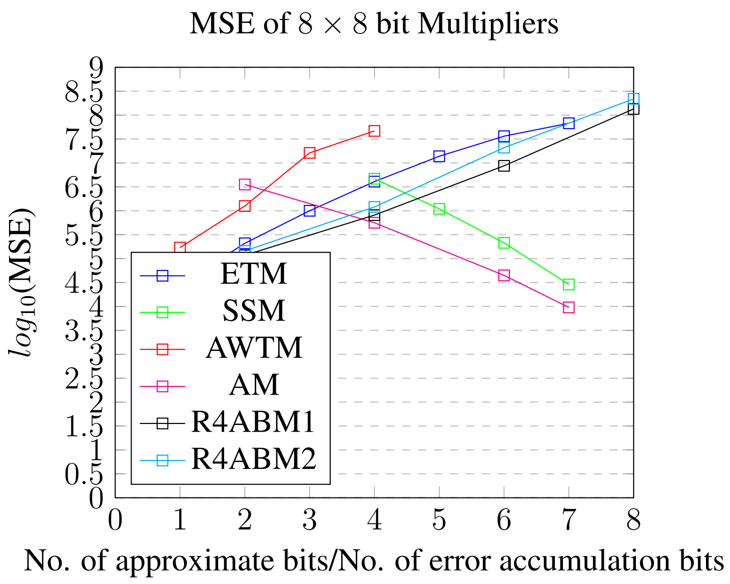
MED plot of 8×8 **bit multipliers:**



NMED plot of 8×8 bit multipliers



MSE plot of 8×8 bit multipliers



Clearly, the plot shows that the error grows non-linearly with number of approximate bits.

CHAPTER 4

APPLICATIONS

In this chapter, Image sharpening, Least Mean Square (LMS) adaptive filter, and Approximate Artificial Neural Network (ApproxANN) for handwritten digit recognition are the applications implemented with approximate multiplier used in place of an exact multiplier is illustrated.

4.1 Image Processing Application: Image Sharpening

Digital images are often degraded by noise during the image acquisition and transmission stage. So as a pre-processing step, the noise should be removed without degrading image details such as edges and textures. Noise smoothing aims at identifying noisy pixels and modifying their intensity values using some prescribed rules. Gaussian filter is one such filter that is a widely used filter for smoothing noise from digital images. It is used as an initial step for many edge detection algorithms such as Canny and Marr and Hildreth (Canny, 1986).

The filters to blur an image are the average filter and the low pass filter. The average filters replaces the pixel value by the average of neighborhood pixel intensity, but cases side effects, blurring the edge. Some pixels are more important, as the low pass filter provides the average weight value, not the average value. Instead, the importance of other pixels is sacrificed. Before detecting the object, blurring the image is used as a preprocessing to eliminate the useless detail and connect the unnatural part.

Approximate circuits can be used in error-tolerant applications such as image processing; image sharpening and smoothing applications are studied next. Since multiplication is the arithmetic operation under investigation; approximate multipliers replace an accurate multiplier. All other operations such as addition are kept accurate.

4.1.1 Image Sharpening

The sharpening algorithm of Lau *et al.* (2009) is simulated using both exact and approximate multipliers. Let *I* be the image to be enhanced. First, Gaussian smoothing is performed on *I*. This is done by convolving *I* with the following matrix in the spatial domain.

$$\text{G} = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

The Gaussian smoothing produces the following image:

$$R(x,y) = \frac{1}{273}\sum_{i=-2}^{2}\sum_{j=-2}^{2} G(i+3,j+3)I(x-i,y-j) \tag{4.1}$$

The factor $\frac{1}{273}$ is for normalization. For approximate smoothing, the multiplication in the convolution is performed by an approximate multiplier, and the other operations, such as addition and division, are all deterministic. Finally, the sharpened image *S* is obtained by *S = 2I - R*. Multiplication of *I* by 2 can be performed by bit-shifting, and hence multiplication is not required to obtain *S* from *R*.

In the results shown in Figure. 4.1, approximate multipliers with different numbers of approximate bits are evaluated, and an improvement in performance is achieved when the number of approximate bits is less or the number of error reduction bits is increased. The degradation of quality is evident for ETM-6 i.e., when the approximate number of bits increased to 6 bits, the output quality is bad. The performance of AWTM-4 and AWTM-3 is good, whereas AWTM-2 and AWTM-1 are bad. This is because for these multiplier upper half of the bits are accurate to a large extent.

4.1.2 Peak Signal-to-Noise Ratio (PSNR)

PSNR is widely used in many DSP applications (such as image processing) as an important figure of merit. In image processing, if **I** is the noise-free image and **K** is the noisy image, the PSNR is defined as :

$$PSNR = 20 \log \frac{MAX_I}{\sqrt{MSE_K}} \tag{4.2}$$

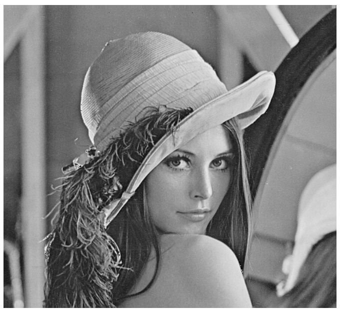
where MAX_I is the maximum possible pixel value of image **I** and MSE is the mean squared error defined as

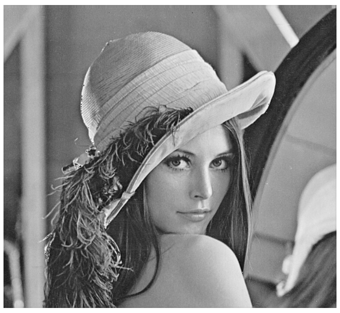
$$MSE_K = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(j,k) - K(j,k)]^2 \tag{4.3}$$

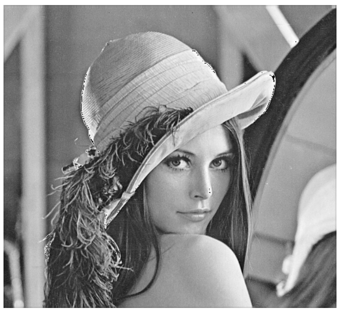
When approximate circuits are used for image processing, **I** can be the resulting image using an exact computation, while **K** is the image obtained by approximate computing.

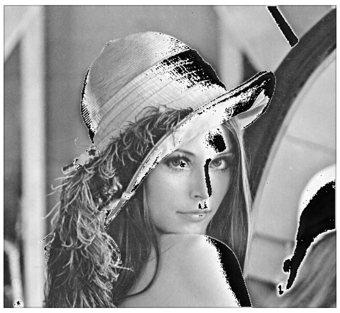
The considered approximate multipliers are applied to image sharpening as an application. Some of the sharpened images are shown in Fig. 4.1. The quality of sharpened by ETM-6, AWTM-3, AWTM-4 are not very high dues to their low accuracy (i.e., by large NMEDs and MREDs). Likewise, The multipliers with higher accuracy (i.e., with smaller NMEDs and MREDs), AM2-13, AM1-13, ETM-2, AWTM-1, show better image sharpening. As it is difficult to distinguish the quality, so PSNR is computed as shown in Table 4.1.

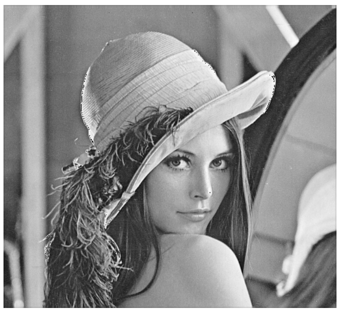
(a) Original

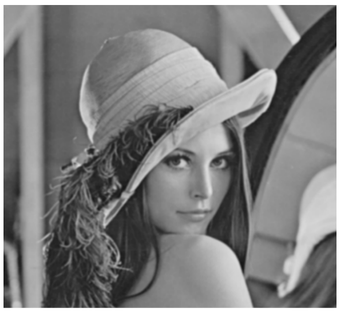

(b) Accurate

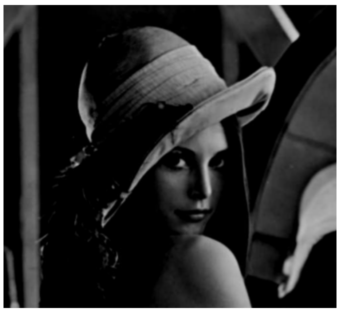

(c) ETM-2

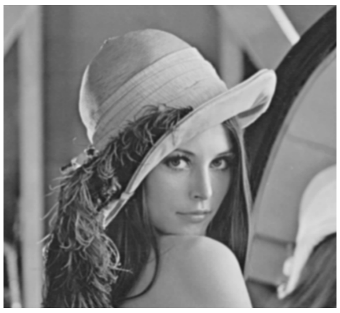

(d) ETM-4


(e) ETM-6


(f) SSM-6


(g) AWTM-1


(h) AWTM-4


(i) AM2-13

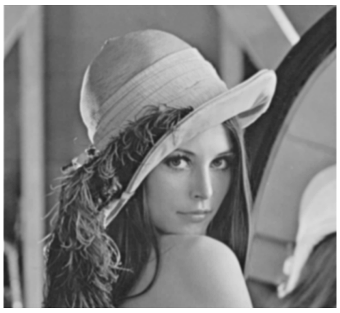

(j) AM1-13

Fig. 4.1: Image sharpened using different multipliers.

Multiplier	PSNR (*dB*)
AM-15	53.23
AM-13	44.57
R4ABM1-4	41.76
R4ABM2-4	38.92
AWTM-1	38.06
ETM-2	34.76
SSM-6	30.01
ETM-4	28.85
AM-10	27.46
AWTM-2	26.86
AWTM-3	14.76
AWTM-4	8.89

Table 4.1: PSNR of the sharpened images using approximate multipliers

4.2 Approximate Multiplier in LMS Adaptive Filters

In this section, approximate LMS adaptive filters are explored by employing approximate multipliers. A system identification scenario is adopted to assess the algorithm behavior. Adaptive filtering based on the Widrow-Hoff LMS algorithm finds application in DSP for channel adaptive equalization, system identification, adaptive noise cancellation, etc. The LMS algorithm exhibits numerical stability, satisfactory convergence error and, computational simplicity. An LMS adaptive filter comprises an FIR filter (due to its inherent stability) whose coefficients are updated by the LMS algorithm (Fig. 4.2).

The LMS is a recursive algorithm to minimize the MSE between the FIR filter output and the desired signal. The minimization requires MSE gradient computation, which, in the LMS algorithm, is computed in an approximated way, causing the so-called gradient noise [Haykin (2008)]. Therefore, the LMS algorithm is characterized by an inherent grade of noise and constitutes a fertile ground to employ approximate

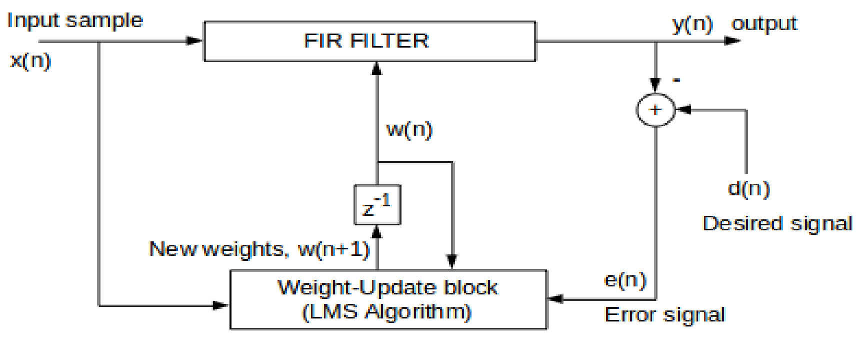


Fig. 4.2: Adaptive LMS filter Esposito *et al.* (2018)

hardware circuits.

4.2.1 Review of The LMS Algorithm

Consider an *M*-tap FIR filter, with weights $w_k(n), k\epsilon 0, 1, ..., M-1$ and inputs $x(n) = [x(n), x(n-1), ..., x(n-M+1)]$. The output at each n can be written as:

$$y(n) = \sum_{k=0}^{M-1} w_k(n).x(n-k) \tag{4.4}$$

Note that the difference between non-adaptive filters and adaptive filters is weights are a function of time instant n. In this filter (Fig. 4.2), the difference between the actual filter output *y(n)* and the desired signal *d(n)* defines the error signal *e(n)*:

$$e(n) = d(n) - y(n) \tag{4.5}$$

Error signal *e(n)* is used by LMS in finding the updated filter tap weights $w_k(n)$, as follows

$$w_k(n+1) = w_k(n) - \mu.e(n).x(n-k) \tag{4.6}$$

where μ is the step size parameter determines the trade-off between convergence speed and convergence error of the algorithm. Fig. 4.3 shows an adaptive LMS filter from a circuital perspective.

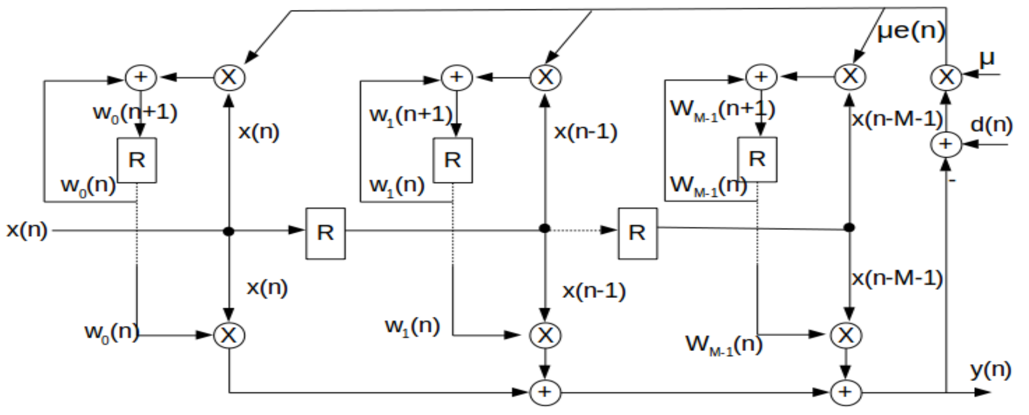


Fig. 4.3: Adaptive LMS filter-hardware overview.

4.2.2 Application scenario: System Identification

In this section, the effect of introducing an approximate multiplier in an adapative LMS filter is examined with reference to an accurate multiplier. To assess the performance of the LMS algorithm, when approximations are introduced, a system identification application is considered.

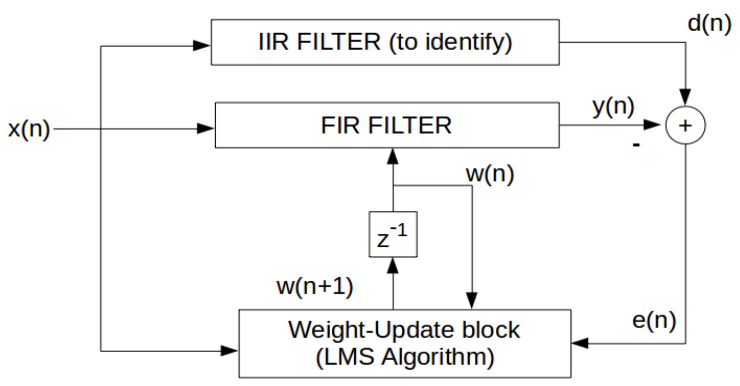
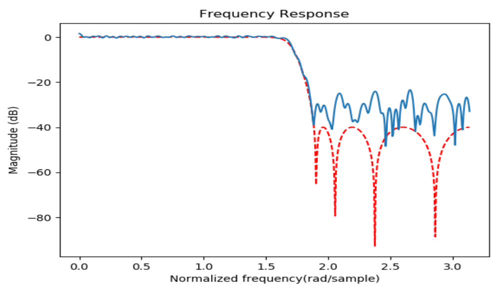


Fig. 4.4: System Identification [Esposito *et al.* (2018)].

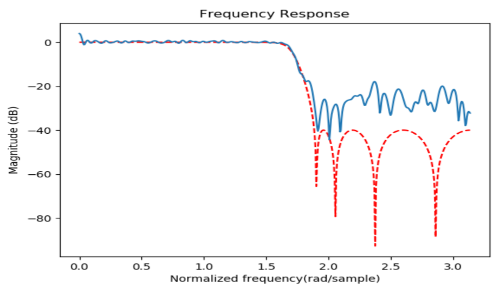
In Fig. 4.4 system identification scenario is shown; the system to identify is an IIR filter. In this case the LMS algorithm minimizes error *e(n)* by adjusting FIR filter coefficients $w_k(n)$ to mimic IIR impulse response. In this case, an IIR 6th order Butterworth low-pass filter with stop-band attenuation of -40dB and with a cutoff frequency of 0.6π. The IIR filter identified using an LMS FIR filter that uses 125 taps.

Exact and approximate implementation of adaptive LMS filters of Fig. 4.4 in 16-bit fixed-point representation was implemented in python and evaluated the convergence performance of the algorithm in terms of MSE. The input to IIR filter and adaptive LMS filter is a white Gaussian signal with zero mean and 0.3 standard deviation (so the input x(n) has a range [-1,1)).

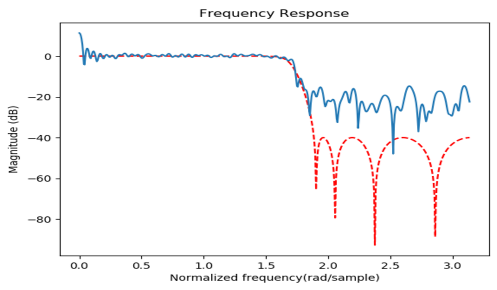
In Fig. 4.5 shows that magnitude response for system identification scenario with one of the multiplier. The acronym **k** after ETM is the number of approximate bits for a 16-bit multiplier. The 125-Tap FIR filter able to converge to IIR filter up to 8 approximate bits. Beyond this, did not converge. Also reported the MSE at the output of the filter in Table 4.2.



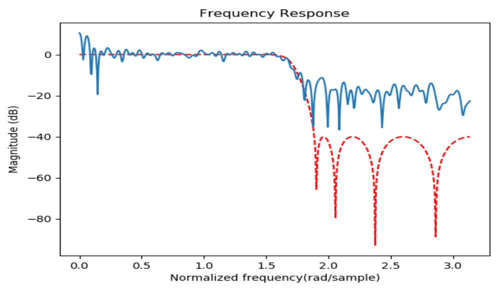
(a) ETM-2



(b) ETM-4



(c) ETM-6



(d) ETM-8

Fig. 4.5: Magnitude response of IIR filter and magnitude response of 125-Tap FIR filter implemented with ETM.

Multiplier	MSE
ETM-2	6.1341×10^{-3}
ETM-4	6.8145×10^{-3}
ETM-6	8.4312×10^{-3}
ETM-8	6.0229×10^{-2}
SSM-10	6.1960×10^{-3}
SSM-8	7.0972×10^{-3}
AWTM-4	5.9563×10^{-3}
AWTM-3	6.0836×10^{-3}
AWTM-2	9.3842×10^{-3}
AWTM-1	8.9884×10^{-2}
AM-10	6.0862×10^{-3}
AM-8	6.1817×10^{-3}

Table 4.2: MSE between the desired signal and output signal for system identification scenario in Fig. 4.4

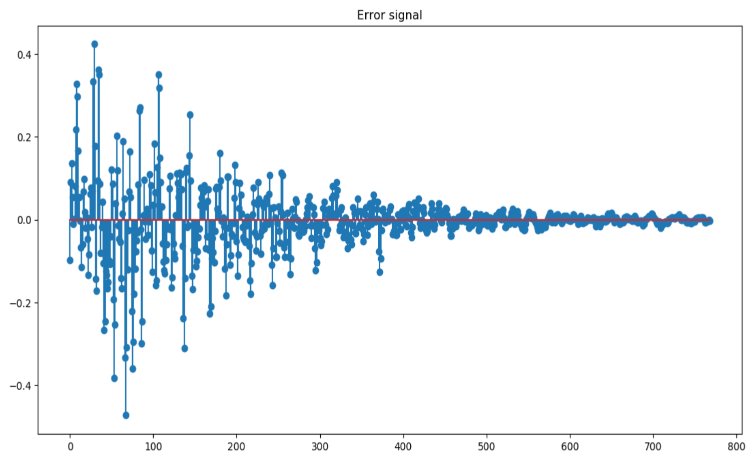


Fig. 4.6: Error convergence for ETM-2 multiplier as a function of number of input sequences applied.

4.3 ApproxANN: Approximate Artificial Neural Network

Artificial Neural Networks (ANNs) are one of the most well-established machine learning techniques with a wide range of applications in recognition, mining, and synthesis. ANNs are computationally intensive and has a large amount of neurons. Most of these applications are error-tolerant with noisy input datasets and or involving human interfaces with limited perceptual capability. Therefore approximate computing has been advocated for these applications. So, by relaxing computational exactness for neurons in ANNS, we can still achieve minor loss in the accuracy at the application level while gaining significant saving in power and delay.

4.3.1 Artificial Neural Network

ANNs are nothing but a system of interconnected computational nodes called *neurons*, as shown in Fig. 4.7. Each neuron generates a single output by operating on an input vector, denoted by *x* (b_0 is the bias). The input vector associated with a weight vector (denoted by *w*), indicates each entry's numerical importance. Mathematically the neurons will first compute a weighted sum and then performs a non-linear activation (e.g., sigmoid function) on the weighted sum to generate output *a*.

$$output = f(\sum_{i=1}^{d} x_i.w_i + b_0) = f(w^t.x) \tag{4.7}$$

$$f(z) = \frac{1}{1+e^{-z}}, \tag{4.8}$$

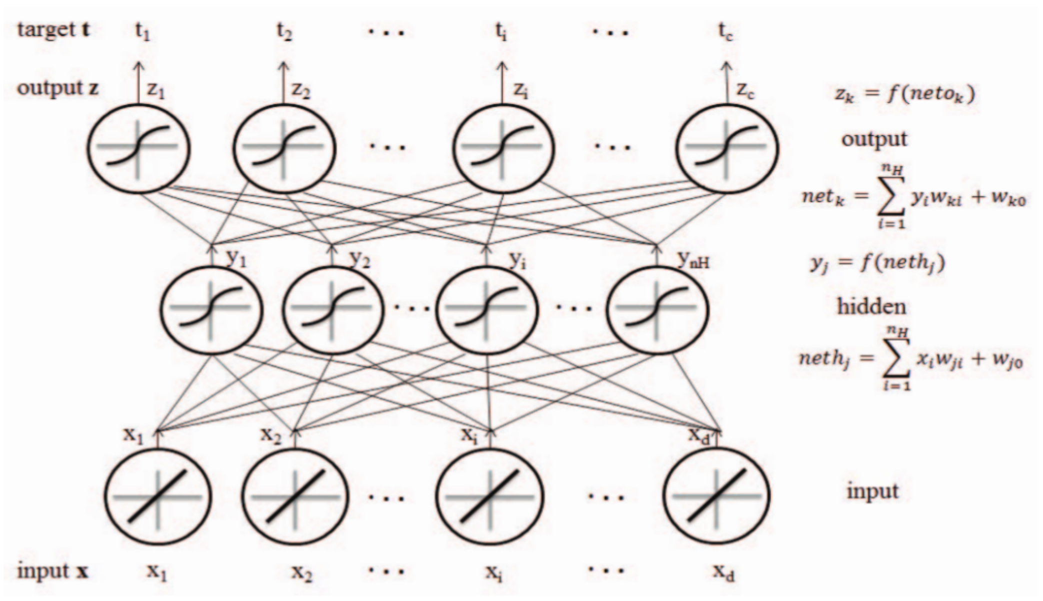


Fig. 4.7: Artificial Neural Network.

4.3.2 Neuron Criticality Analysis

Zhang *et al.* (2015) proposed efficient theoretical-based criticality analysis for neurons in output and hidden layers. We can say a neuron is critical if a small change in input of this neuron introduces a large final output quality degradation; otherwise, it is resilient. To support criticality analysis, we have the notation as follows. Hidden layer output y_j is given by

$$y_j = f(neth_j) \tag{4.9}$$

$$neth_j = \sum_{i=1}^{x_d} x_i.w_{ji} + w_{j0} \tag{4.10}$$

The output of output layer z_k is given by

$$z_k = f(neto_k) \tag{4.11}$$

$$net_k = \sum_{i=1}^{n_H} y_i.w_{ki} + w_{k0} \tag{4.12}$$

The final network's cost function given by

$$E = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t - z\|^2 \tag{4.13}$$

Where c is the number of neurons in the output layer and t is desired output from the network.

The i-th neuron's criticality, denoted by nc_i (includes nco_i and nch_i for output and hidden neurons, respectively, can be represented by the derivative of networks cost function *E* with respect to net_i, as illustrated by Equation below

$$nc_i = \frac{\partial E}{\partial net_i} \tag{4.14}$$

For the neurons in the output layer:

$$nco_k = \frac{\partial E}{\partial neto_k} = \frac{\partial \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2}{\partial z_k} . \frac{\partial z_k}{\partial neto_k} = -(t_k - z_k).f^{'}(neto_k) \tag{4.15}$$

For the neurons in the hidden layers:

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j}[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2] = -\sum_{k=1}^{c}(t_k - z_k).\frac{\partial z_k}{\partial neto_k}.\frac{\partial neto_k}{\partial y_j} = -\sum_{k=1}^{c} nco_k w_{kj} \tag{4.16}$$

$$nch_j = \frac{\partial E}{\partial y_j}.\frac{\partial y_j}{\partial neth_j} = -f^{'}(neth_j).\sum_{k=1}^{c} nco_k w_{kj} \tag{4.17}$$

So from Eq. 4.17 and Eq. 4.15, we can calculate the criticality factor of neurons in each hidden layer and output layer after the training phase fixes all weights and biases, even for a large-scale neural network.

A neuron is less critical if the accuracy requirement relaxation leads to lower final quality degradation. So a less critical neuron will have the higher priority to be approximated. Based on this analysis, we will sort all the neurons in the hidden layer and output layer in ascending order and finally get the criticality ranking vector $s = s_1, s_2, ..., s_n$ for a given network. An entry s_m indicates that neuron has m-th priority to be approximated.

4.3.3 ApproxANN for Handwritten Digit Recognition: MNIST-I Architecture with Sigmoid Activation Function

Approximate ANN for handwritten digit recognition is implemented using the MNIST database (LeCun and Cortes, 2010) and a three-layer neural network is used. Training data for the network will consist of 28 by 28 pixel images of scanned handwritten digits, so the input layer contains 784=28×28 neurons.

Number of neurons in the input layer: 784
Number of neurons in the hidden layer-1: 30
Number of neurons in the output layer: 10
Activation function: Sigmoid
Baseline network accuracy: **95.46%**
Number of neurons approximated: 25% (In the hidden layer)

Results with 14 $\times$ 8 bit multiplier:

First, the network is trained with an exact multiplier (Full weight considered). Then Approximate neurons are introduced to an already-trained network. The accuracy will be low. So again, the network is retrained with approximate multipliers in place so that it can learn from these errors for better accuracy.
Retrained with training set. After each epoch, tested for accuracy and and choose the weights with highest accuracy.
Baseline Network Accuracy – **95.46**%
Accurate 14×8 Multiplier Accuracy - 95.28% (epoch-6)
Truncated-18: For a 14×8 bit multiplier, 18 LSBs in the output set to 0.
1s in LSBs-18: For a 14×8 bit multiplier, 18 LSBs in the output set to 1.

Truncation-18	94.61% (Epoch-5)	1s in LSBs-18	92.59% (Epoch-6)
Truncation-16	94.70% (Epoch-5)	1s in LSBs-16	94.37% (Epoch-5)
Truncation-14	94.73% (Epoch-4)	1s in LSBs-14	94.38% (Epoch-3)

Results with 8 $\times$ 8 bit multiplier:

Baseline Network Accuracy – **95.46**%

Accurate 8 $\times$ 8 Multiplier Accuracy - 95.22% (epoch-6)

Truncated-12: For an 8 $\times$ 8 bit multiplier, 12 LSBs in the output set to 0.

1s in LSBs-12: For an 8 $\times$ 8 bit multiplier, 12 LSBs in the output set to 1.

Truncation-12	94.12% (Epoch-4)	1s in LSBs-12	92.01% (Epoch-5)
Truncation-10	94.30% (Epoch-1)	1s in LSBs-10	93.75% (Epoch-5)
Truncation-8	94.44% (Epoch-3)	1s in LSBs-8	94.31% (Epoch-2)

Clearly, for large number of approximate bits zeros in LSB is better than ones in LSB.

Results with 6 $\times$ 6 bit multiplier:

Baseline Network Accuracy – **95.46**%

Accurate 6 $\times$ 6 Multiplier Accuracy - 95.16% (epoch-5)

Truncated-8: For a 6 $\times$ 6 bit multiplier, 8 LSBs in the output set to 0.

1s in LSBs-8: For a 6 $\times$ 6 bit multiplier, 8 LSBs in the output set to 1.

Truncation-8	94.23% (Epoch-4)	1s in LSBs-18	91.63% (Epoch-1)
Truncation-6	94.49% (Epoch-5)	1s in LSBs-16	93.76% (Epoch-3)
Truncation-4	94.32% (Epoch-4)	1s in LSBs-14	94.39% (Epoch-4)

4.3.4 ApproxANN for Handwritten Digit Recognition: MNIST-I Architecture with Relu Activation Function

Number of neurons in the input layer: 784

Number of neurons in the hidden layer-1: 30

Number of neurons in the output layer: 10

Activation function: Relu

Baseline network accuracy: **96.75%**

Number of neurons approximated: 25% (In the hidden layer)

Results with 14 $\times$ 8 bit multiplier:

Baseline Network Accuracy – **96.75**%

Accurate 14×8 Multiplier Accuracy - 96.63% (epoch-5)

Truncated-18: For a 14×8 bit multiplier, 18 LSBs in the output set to 0.

1s in LSBs-18: For a 14×8 bit multiplier, 18 LSBs in the output set to 1.

Truncation-18	96.11% (Epoch-2)	1s in LSBs-18	88.58% (Epoch-1)
Truncation-16	96.47% (Epoch-3)	1s in LSBs-16	91.67% (Epoch-4)
Truncation-14	96.53% (Epoch-3)	1s in LSBs-14	95.43% (Epoch-2)

Results with 10 $\times$ 8 bit multiplier:

Baseline Network Accuracy – **96.75**%

Accurate 10×8 Multiplier Accuracy - 96.60% (epoch-8)

Truncated-14: For a 10×8 bit multiplier, 14 LSBs in the output set to 0.

1s in LSBs-14: For a 10×8 bit multiplier, 14 LSBs in the output set to 1.

Truncation-14	96.40% (Epoch-6)	1s in LSBs-14	73.98% (Epoch-3)
Truncation-12	96.48% (Epoch-3)	1s in LSBs-12	92.58% (Epoch-1)
Truncation-10	96.58% (Epoch-5)	1s in LSBs-10	96.35% (Epoch-5)

Results with 8 $\times$ 8 bit multiplier:

Baseline Network Accuracy – **96.75**%

Accurate 8×8 Multiplier Accuracy - 96.59% (epoch-7)

Truncated-12: For a 8×8 bit multiplier, 12 LSBs in the output set to 0.

1s in LSBs-12: For a 8×8 bit multiplier, 12 LSBs in the output set to 1.

Truncation-12	96.16% (Epoch-4)	1s in LSBs-12	79.11% (Epoch-6)
Truncation-10	96.51% (Epoch-2)	1s in LSBs-10	86.87% (Epoch-5)
Truncation-8	96.56% (Epoch-5)	1s in LSBs-8	95.41% (Epoch-3)

4.3.5 ApproxANN for Handwritten Digit Recognition: MNIST-I Architecture with Relu Activation Function (all neurons approximated)

Number of neurons in the input layer: 784

Number of neurons in the hidden layer-1: 30

Number of neurons in the output layer: 10

Activation function: Relu

Baseline network accuracy: **96.75%**

Number of neurons approximated: 40 (all including output neurons)

Results with 14 $\times$ 8 bit multiplier:

Baseline Network Accuracy – **96.75%**

Accurate 10×8 Multiplier Accuracy - 96.61% (epoch-5)

Truncated-18: For a 14×8 bit multiplier, 18 LSBs in the output set to 0.

1s in LSBs-18: For a 14×8 bit multiplier, 18 LSBs in the output set to 1.

Truncation-18	87.07% (Epoch-2)	1s in LSBs-18	66.58% (Epoch-4)
Truncation-16	90.13% (Epoch-3)	1s in LSBs-16	88.29% (Epoch-4)
Truncation-14	92.68% (Epoch-6)	1s in LSBs-14	89.44% (Epoch-2)

Results with 10 $\times$ 8 bit multiplier:

Baseline Network Accuracy – **96.75%**

Accurate 10×8 Multiplier Accuracy - 96.53% (epoch-7)

Truncated-14: For a 10×8 bit multiplier, 14 LSBs in the output set to 0.

1s in LSBs-14: For a 10×8 bit multiplier, 14 LSBs in the output set to 1.

Truncation-14	86.72% (Epoch-6)	1s in LSBs-14	71.28% (Epoch-3)
Truncation-12	89.68% (Epoch-4)	1s in LSBs-12	82.53% (Epoch-1)
Truncation-10	91.11% (Epoch-5)	1s in LSBs-10	86.35% (Epoch-4)

Results with 8 $\times$ 8 bit multiplier:

Baseline Network Accuracy – **96.75%**

Accurate 8×8 Multiplier Accuracy - 96.43%

Truncation-12: For a 8×8 bit multiplier ,12 LSBs in the output set to 0.

1s in LSBs-12: For a 8×8 bit multiplier, 12 LSBs in the output set to 1.

Truncation-12	85.16% (Epoch-4)	1s in LSBs-12	72.11% (Epoch-6)
Truncation-10	88.51% (Epoch-4)	1s in LSBs-10	83.87% (Epoch-5)
Truncation-8	90.06% (Epoch-4)	1s in LSBs-8	87.41% (Epoch-6)

4.3.6 ApproxANN for Handwritten Digit Recognition: MNIST-II Architecture

Number of neurons in the input layer: 784

Number of neurons in the hidden layer-1: 1020

Number of neurons in the hidden layer-2: 1020

Number of neurons in the output layer: 10

Activation function: Relu

Baseline network accuracy: **97.42%**

Number of neurons approximated: 25% (In the hidden layer-1)

Number of neurons approximated: 25% (In the hidden layer-2)

- Network Accuracy with 14×8 bit accurate multipliers : 97.34% (Epoch - 6)
  Truncation-18: For a 14×8 bit multiplier, 18 LSBs in the output set to 0.

Truncation-18	96.83% (Epoch-3)
Truncation-16	96.97% (Epoch-2)
Truncation-14	97.18% (Epoch-5)

- Network Accuracy with 10×8 bit accurate multipliers : 97.13% (Epoch - 4)
  Truncation-14: For a 10×8 bit multiplier, 14 LSBs in the output set to 0.

Truncation-14	96.86% (Epoch-6)
Truncation-12	96.93% (Epoch-3)
Truncation-10	97.06% (Epoch-1)

- Network Accuracy with 8×8 bit accurate multipliers : 97.04% (Epoch - 5)
  Truncation-12: For a 8×8 bit multiplier, 12 LSBs in the output set to 0.

Truncation-12	96.68% (Epoch-5)
Truncation-10	96.85% (Epoch-2)
Truncation-8	96.97% (Epoch-4)

It is shown that many of these applications are error-tolerant, where approximate computing is naturally used to achieve significant energy savings with minor loss in accuracy.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Error-tolerant applications were implemented using approximate multipliers. The experimental results show that the output quality is within tolerable limits. Intuitively there was an improvement in the area, power, and delay metrics of the approximate application over the accurate application. However, the output quality keeps reducing with the increasing number of approximated bits. So, the output quality required should be carefully weighed against the amount of area and power savings and the delay reduction. The results for error-tolerant applications have shown a lot of scope for approximation in these applications to get better performance for the application.

The current work has been done using the same approximation in the application, i.e., the same number of approximate bits for all multipliers. The future scope of the work is to find out the less critical adders and or multipliers that contribute less to output quality degradation. A multiplier and or adder is less critical if the accuracy requirement relaxation leads to less final quality degradation. Those less critical adders and or multipliers have to be approximated with the higher number of approximate bits to make the application highly cost-effective.

REFERENCES

1. **Baran, D.**, **M. Aktan**, and **V. G. Oklobdzija** (2010). Energy efficient implementation of parallel cmos multipliers with improved compressors. *In Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*.

2. **Bhardwaj, K.**, **P. S. Mane**, and **J. Henkel** (2014). Power-and area-efficient approximate wallace tree multiplier for error-resilient systems. *In Fifteenth International Symposium on Quality Electronic Design*. IEEE.

3. **Canny, J.** (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6), 679–698.

4. **Esposito, D.**, **G. Di Meo**, **D. De Caro**, **N. Petra**, **E. Napoli**, and **A. G. Strollo** (2018). On the use of approximate multipliers in lms adaptive filters. *In 2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE.

5. **Haykin, S. S.** (2008). *Adaptive filter theory*. Pearson Education India.

6. **King, E. J.** and **E. Swartzlander** (1997). Data-dependent truncation scheme for parallel multipliers. *In Conference Record of the Thirty-First Asilomar Conference on Signals, Systems and Computers (Cat. No. 97CB36136)*, volume 2. IEEE.

7. **Kulkarni, P.**, **P. Gupta**, and **M. Ercegovac** (2011). Trading accuracy for power with an underdesigned multiplier architecture. *In 2011 24th Internatioal Conference on VLSI Design*. IEEE.

8. **Kyaw, K. Y.**, **W. L. Goh**, and **K. S. Yeo** (2010). Low-power high-speed multiplier for error-tolerant application. *In 2010 IEEE international conference of electron devices and solid-state circuits (EDSSC)*. IEEE.

9. **Lau, M. S.**, **K.-V. Ling**, and **Y.-C. Chu** (2009). Energy-aware probabilistic multiplier: design and analysis. *In Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems*.

10. **LeCun, Y.** and **C. Cortes** (2010). MNIST handwritten digit database. URL `http://yann.lecun.com/exdb/mnist/`.

11. **Liang, J.**, **J. Han**, and **F. Lombardi** (2012). New metrics for the reliability of approximate and probabilistic adders. *IEEE Transactions on computers*, **62**(9), 1760–1771.

12. **Lin, C.-H.** and **C. Lin** (2013). High accuracy approximate multiplier with error correction. *In 2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE.

13. **Liu, C.** (2014). Design and analysis of approximate adders and multipliers.

14. **Liu, C.**, **J. Han**, and **F. Lombardi** (2014). A low-power, high-performance approximate multiplier with configurable partial error recovery. *In 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE.

15. **Liu, W.**, **L. Qian**, **C. Wang**, **H. Jiang**, **J. Han**, and **F. Lombardi** (2017). Design of approximate radix-4 booth multipliers for error-tolerant computing. *IEEE Transactions on Computers*, **66**(8), 1435–1441.

16. **Mahdiani, H. R.**, **A. Ahmadi**, **S. M. Fakhraie**, and **C. Lucas** (2009). 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.

17. **Momeni, A.**, **J. Han**, **P. Montuschi**, and **F. Lombardi** (2014). Design and analysis of approximate compressors for multiplication. *IEEE Transactions on Computers*, **64**(4), 984–994.

18. **Narayanamoorthy, S.**, **H. A. Moghaddam**, **Z. Liu**, **T. Park**, and **N. S. Kim** (2014). Energy-efficient approximate multiplication for digital signal processing and classification applications. *IEEE transactions on very large scale integration (VLSI) systems*, **23**(6), 1180–1184.

19. **Zhang, Q.**, **T. Wang**, **Y. Tian**, **F. Yuan**, and **Q. Xu** (2015). Approxann: An approximate computing framework for artificial neural network. *In 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE.