Rounding Technique Analysis for Power-Area & Energy Efficient Approximate
Multiplier Design

by

Priyanka Bhushan Lohray, B.Tech, M.Tech

A Thesis

In

Electrical Engineering

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

Approved

Dr. Stephen Bayne
Chair of Committee

Dr. Tooraj Nikoubin

Dr. Brian Nutter

Mark Sheridan
Dean of the Graduate School

August, 2020

Copyright 2020, Priyanka Bhushan Lohray

ACKNOWLEDGMENTS

The process of earning a Master's degree involves long, arduous, dedication and encouragement, and this cannot be done single handedly. First and foremost, I would like to deeply thank my mentor Dr. Tooraj Nikoubin for constant advise, support and encouragement during ups and downs. I am truly grateful to Dr. Nikoubin for giving me an opportunity to start with an individual study in order to access my research ability. Secondly, I would like thank Dr. Stephen Bayne and Dr. Brian Nutter for agreeing to be in my committee and investing some time on me during their busy schedule. Their advice, feedback and insight were influential and essential throughout my thesis writing.

I would certainly be remiss to not mention and sincerely thank Dr. Hamed Sari-Sarraf for identifying my passion towards research and allowing me into his additional projects that gave me an immense knowledge and stability during research thought process. I am thankful to Dr. Tanja Karp for helping me with all my questions with and without related to thesis. Dr. Karp was always happy to discuss and bring me to all my answers with detailed explanation.

I would also like to thank Dr. Richard Porter, Julia Siu and Hisatoshi Fujino from International Affairs for helping me to smoothly drive through towards my graduation.

I would also like to thank my husband Bharath Lohray who had confidence in me more than myself all through my studies and thesis, which strongly encouraged me to show my strongest ability.

I am thankful to all my friends, professors, family members, who have constantly support, provided good advice, believed in me during all this time.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	ii
ABSTRACT	v
LIST OF TABLES	vi
LIST OF FIGURES	vii
1. INTRODUCTION.....	1
Data Explosion – Exponential Growth of Data.....	1
Digital Computation.....	2
Data Approximation.....	2
Multiplier	2
2. LITERATURE SURVEY	4
Previous Works	4
TOSAM: An Energy-Efficient Truncation- and Rounding-Based Scalable Approximate Multiplier	6
Approximate Hybrid High Radix Encoding for Energy-Efficient Inexact Multipliers	7
DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications	8
3. BASE WORK ON APPROXIMATE MULTIPLIER	9
Design	9
Block Diagram	9
Input Rounding	11
Partial Product Reduction	12
Rounding Error Analysis	14
4. ROUNDING TECHNIQUE ANALYSIS.....	16
Rounding 2-Bit.....	16
16-Bit	16
32-Bit	17
Rounding 3-bit	17
16-Bit	17
32-Bit	17
Rounding 5-bit	18
16-Bit	18

32-Bit	18
5. RECONFIGURABLE SVM REALIZATION AND ANALYSIS	19
SVM Design.....	19
Linearly Non-Separable SVM	20
Non-Linear SVM (Kernel).....	20
Quadprog Programming.....	20
Error Analysis	21
SVM Output Analysis.....	21
Different Rounding Techniques.....	23
Overfitting and Underfitting	23
Different Data Sets.....	25
The Complexity Analysis.....	27
6. OPTIMIZED APPROXIMATE MULTIPLIER DESIGN	28
Block Diagram	28
Hardware Design.....	29
Rounding Techniques.....	30
7. SIMULATIONS AND TEST RESULTS	31
Hardware Characteristics and Results.....	31
Accuracy Analysis	39
8. APPLICATIONS	43
9. CONCLUSIONS	46
BIBLIOGRAPHY	47

ABSTRACT

Approximate computing precipitates a paradigm shift in energy-efficient systems design and operation, based on the idea that we are hindering computer systems' efficiency by demanding too much accuracy from them. Interestingly, perfect answers are unnecessary in a large number of application domains, such as DSP, statistics, and machine learning. Approximate computing is suited for efficient data processing and error resilient applications, such as signal and image processing, computer vision, machine learning, data mining etc. Approximate computing reduces accuracy, which is acceptable as a cost of changing other circuit characteristics, depends on the application. Desirable accuracy is the threshold point for controlling the tradeoff between accuracy and circuit characteristics under the control of the circuit designer.

In this work, the rounding technique is introduced as an efficient method for controlling this trade off. In this regard, multiplier circuits as a critical building block for computing in most of the processors have been considered for the evaluation of the rounding technique efficiency. The impact of the rounding method is investigated by comparison of circuit characteristics for three multipliers. These three multipliers are the conventional Wallace tree accurate multiplier, DRUM [1] the recently proposed approximate multiplier, and the round based approximate multiplier proposed in this work.

Simulation results for three selected technologies show significant improvement in the circuit characteristics in terms of power, area, speed, and energy for the proposed multiplier in comparison with its counterparts. Input data rounding pattern and the probability of the repetition for rounded values has been introduced as two essential items to control the level of the accuracy for each range of the data with minimum cost on the hardware. The proposed approximate multiplier algorithms are shown with Support Vector Machine classifier, which is therefore an attractive option for low power mobile applications.

LIST OF TABLES

Table 1: Comparison with features	4
Table 2: Inputs and rounded values of conventional multiplier.....	5
Table 3: Bit position strategy	11
Table 4:Data with number of occurrences for whole range	14
Table 5: Overall Error Analysis	22
Table 6: Time complexity	26
Table 7: Area-Power-Delay comparison for 16-bits & 32-bits.....	32
Table 8: Area-Power-Delay comparison for 16-bits.....	34
Table 9: Area-Power-Delay comparison for 32-bits.....	36
Table 10: PSNR and SSIM for 16-bits and 32-bits.....	40

LIST OF FIGURES

Figure 1 Exponential growth of data.....	1
Figure 2: Block Diagram and Pseudo-code for 8-bit multiplier	3
Figure 3: 16-bit Block diagram of algorithm	10
Figure 4: Position error curve (16-bit) and an example	11
Figure 5: Multiplier Design (left) with an example (right) (16-bit).....	13
Figure 6: Rounding Pattern and Probability of repetition (16-bit).....	14
Figure 7: Rounding 16-bit data with 2-bit each time	16
Figure 8: Rounding 32-bit data with 2-bit each time	17
Figure 9: Rounding 16-bit data with 3-bit each time	17
Figure 10: Rounding 32-bit data with 3-bit each time	17
Figure 11: Rounding 16-bit data with 5-bit each time	18
Figure 12: Rounding 32-bit data with 5-bit each time	18
Figure 13: SVM Algorithm.....	19
Figure 14: 25 examples of $n*n$ values	22
Figure 15: Absolute Error (left) and Error Percent (right).....	22
Figure 16: SVM for linearly and non-linearly (RBF kernel)	24
Figure 17: SVM outputs for different rounding techniques.....	25
Figure 18: SVM outputs for Overfitting and Underfitting.....	26
Figure 19: Complexity Analysis	26
Figure 20: Optimized block diagram	28
Figure 21: Hardware Design of optimized multiplier.....	29
Figure 22: Rounding 2-bit function	30
Figure 23: Rounding 3-bit function	30
Figure 24: Rounding 3-bit function	30
Figure 25 (a) (b)(c)(d): Characteristic comparison for 16 and 32-bit	33
Figure 26(a)to(d): Characteristic comparison of proposed & references.....	38
Figure 27: Accuracy analysis for 16-bit values.	39
Figure 28: Complexity Analysis	42
Figure 29: Image sharpening using accurate multiplier.....	43

Figure 30: Image sharpening using approximate multiplier	44
Figure 31: Image sharpening using accurate multiplier	44
Figure 32: Image sharpening using approximate multiplier	45

CHAPTER 1

INTRODUCTION

Data Explosion – Exponential Growth of Data

Many sources predict exponential data growth toward 2020 and beyond. Yet they are all in broad agreement that the size of the digital universe will double every two years at least, a 50-fold growth from 2010 to 2020. Human- and machine-generated data is experiencing an overall 10x faster growth rate than traditional business data, and machine data is increasing even more rapidly at 50x the growth rate.

Figure 1 Exponential growth of data

The acquisition and analysis of data and its subsequent transformation into actionable insight is a complex workflow which extends beyond data centers, to the edge, and into the cloud in a seamless hybrid environment. The utilization of edge devices, in situ-computation and analysis, centralized storage and analysis, and deep learning methodologies that accelerate data processing at scale require a new technological approach. Historically, data processing and analytics systems had specialized features for business analytics and high-performance computing (HPC) workloads. Yet with the advent of big data and industry standard x86-based computing, we are seeing a convergence in big compute, big data, and IoT for analytics. IDC research categorizes this convergence as high-performance data analytics (HPDA).

Digital Computation

The advent of Big Data delivers a cost-effective prospect for improved decision-making in critical development areas such as healthcare, economic productivity and security. At the same time, the well-known caveats of the Big Data debate, such as privacy concerns and human resource scarcity, are aggravated in developing countries by long-standing structural shortages in the areas of infrastructure, economic resources and institutions. The result is a new kind of digital divide: a divide in the use of data-based knowledge to inform intelligent decision-making.

Data Approximation

In mathematics, Data Approximation is the central theme of approximation theory. The emergence of Big Data presents both opportunities and challenges. On one hand, big data introduces more information about the unknowns and, in principle, allows us to create more accurate models. On the other hand, data storage and processing become highly challenging. Moreover, data often contain certain corruption errors, in addition to the standard noise errors.

Multiplier

Approximate computing is a computation technique which returns a possibly inaccurate result rather than a guaranteed accurate result and [2] can be used for applications where an approximate result is sufficient for the application's purpose. One example of such situation is a search engine where no exact answer may exist for a certain search query, and, hence, many answers may be acceptable. Approximate computing is based on the observation that in many scenarios, although performing exact computation requires large amount of resources, allowing bounded approximation can provide disproportionate gains in performance and energy, while still achieving acceptable result accuracy.

Multipliers are fundamental subsystems for microprocessors, digital signal processors, and embedded systems with applications ranging from filtering to convolutional neural networks. Unfortunately, multipliers are characterized by complex logic design and constitute one of the most energy-hungry digital blocks. Therefore, approximate multiplier design has become an important research subject in recent years. A multiplier includes a few basic blocks: partial products generation, partial products reduction and carry-propagate addition. Approximations can be introduced in any of these blocks. For example, truncation of the partial products is a well-established approximation technique in which some of the partial products are not formed and the truncation error is reduced with the help of suitable correction functions.

Figure 2: Block Diagram and Pseudo-code for 8-bit multiplier

Suppose we want to multiply A (8 bits) and B (8 bits), and store result at S. S is required to be 16-bit output, with higher part SH and lower part SL. Initially place B in SL (in C type operation $s=0|B$) and run the while loop of shifting S and incrementing i. S₀ is the right most bit, and if it is set then add will be activated.

CHAPTER 2

LITERATURE SURVEY

Previous Works

Approximations in circuit deliver lower dynamic and leakage power dissipation, and they can be introduced at various design abstraction levels: transistor, logic, architecture, and algorithm. Table 1 presents an overview of the most commonly used techniques, briefly describing methodology and negative features. Table 2 shows how big or small would be the difference of results in accuracy with rounding of inputs [3].

Table 1: Comparison with features

Technique	Methodology	Negative features
Voltage over-scaling	Lowering the supply voltage below the nominal value	Most significant bits (MSBs) are affected by timing errors, and it has unexpected errors.
Over – clocking	Inserting timing violation by increasing the clock frequency	It has unexpected errors and is not bounded.
Logic – simplification	Use inaccurate circuits of reduced complexity.	It provides limited gains in energy consumption.
Perforation	Omitting generation of partial products	It may generate bounded yet large errors if not tuned appropriately.
Truncation	Eliminating least significant bits (LSBs) from partial product columns	There is no significant reduction in the critical paths.

Table 2: Inputs and rounded values of conventional multiplier

<i>Inputs</i>	<i>Rounded value Ar</i>	<i>Rounded value Br</i>	<i>Result</i>
A=B	Larger value	larger value	Smaller value than the exact value
A>B	Large value	Larger value	Smaller value than the exact value
A<B	Larger value	Larger value	Larger value than the exact value

Approximating input value before multiplication were rarely altered since it inclines towards major error gap from accurate values. There are two different ways of implementation according to requirement on need for accuracy.

1. Rounding values
2. Truncating values

Rounding values changes input by converting into nearest values with designed algorithms. Corresponding error metrics are also calculated to tolerate errors and not compromise quality of result. In RoBA multiplier [2], the input operands were rounded to the nearest power of two where the output was produced by some shift, add, and subtraction operations. In this structure, the number of elements that needed to be added to generate result was reduced, compared to the exact multiplier, leading to better energy and speed. As another approach, to improve the speed and area of the multiplier, the least significant bits of the partial products were eliminated [4].

Truncation is another method where inputs are made shorter by excluding zeros from MSB which do not add any value to input. “Finding leading one” followed by truncation is one of the methods widely used. Approximating input data requires major responsibility in maintaining the accuracy. In [5] the power reduction of a Floating Point multiplier is accomplished by truncating the least significant mantissa bits of the input data; however, when this reduction is more than 66%, a tolerable accuracy is not preserved. An algorithmic-level simplification for Floating Point multiplication is proposed in [6]; for a 32-bit FLP multiplier, the 24×24 -bit mantissa multiplication is

replaced by a 25×25 -bit addition. In the dynamic segment method (DSM) structure [7], the input operands were truncated to m -bits based on the position of their leading one bit where a fixed-width multiplication was performed on the truncated values. This way of truncation made the generated output always less than the exact one, making the mean relative error (MRE) negative. This is an undesired feature due to the fact that for the approximate arithmetic units with the Gaussian error distribution, it is better to have the mean error close to zero for having a higher signal-to-noise ratio (SNR) when dealing with digital signal processing applications [8].

Floating-point numbers provide a wider dynamic range of representable real numbers compared with fixed point representations. However, floating point hardware incurs substantial power consumption. There are many applications that require a wide range of numbers and can be tolerant to some errors; so inexact floating point computation is also viable [9]. Floating point multiplication is typically one of the most frequent power consuming operations. so, the focus of [10] is to design approximate floating point multipliers with low power and moderate errors with new metrics for evaluating the reliability and power efficiency of inexact designs. Although inexact floating-point arithmetic circuits have been extensively studied [11], floating point arithmetic circuits have not been fully considered for inexact computing. Secondly recent research in inexact fixed-point multipliers has shown that an inexact 8×8 -bit multiplier with an average normalized mean error distance (NMED) of 0.18 achieves up to 37% improvement in power consumption [12].

TOSAM: An Energy-Efficient Truncation- and Rounding-Based Scalable Approximate Multiplier

This paper presents an approximation technique for decreasing the number of partial products. In TOSAM (Truncation- and Rounding-Based Scalable Approximate Multiplier) [13], input operands are truncated to h and t bits according to the position of their leading one bit, where these truncated values are employed for the multiplication and addition operations. In addition, to reduce the error resulting from the truncation operation, they found the approximate amount of the truncated values by rounding them.

These simplifications results in lower accuracy and performance compared to those of the state-of-the-art approximate multipliers. The calculation core of the TOSAM performs multiplication and addition operations on truncated and rounded numbers and the result. Truncate and rounding both add errors and seem not ideal for accuracy efficiency.

In this paper, a low-energy and area-efficient approximate multiplier is suggested in which the input operands were truncated with two different lengths, t and h , and then rounded to the nearest odd numbers to reduce the error resulting from the truncation operation. TOSAM multiplier claims to be as scalable and outperformed other approximate multipliers in terms of speed, area, and energy. Comparing with the exact multiplier, the speed, area, and energy improvements of the proposed multiplier became better as the multiplier width increased. That was due to the simple and scalable calculation core of the proposed multiplier.

Approximate Hybrid High Radix Encoding for Energy-Efficient Inexact Multipliers

This paper is targeting the design of inexact multipliers by applying approximations on the partial product generation. Here, the most significant bits (MSBs) of the multiplicand are encoded using radix-4 encoding, whereas the k least significant bits (LSBs) are encoded using a radix- 2^k (with $k \geq 4$). To simplify the increased complexity induced by the hybrid encoding, the circuit for generating the partial products is approximated by altering its truth table accordingly. Hence, the number of the partial products decreases significantly, and simpler tree architectures are used for their accumulation, reducing the multiplier's energy consumption, area, and critical path delay. Due to different radix encoding, time required and error incurred are higher than other algorithms.

The error of the proposed technique follows a Gaussian distribution with near zero average. Compared with state-of-the-art inexact multipliers, the proposed ones constitute a better approximate design alternative, outperforming them in both energy

consumption and accuracy. Furthermore, the efficiency of the proposed multipliers showed by applying them to applications.

DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications

This paper focuses on approximate multiplier design. Multipliers are particularly attractive to approximate due to their high cost in terms of power and silicon area, and their prominent utilization in the emerging application domains. The dynamic nature of DRUM (Dynamic Range Unbiased Multiplier) [1] enables the multiplier to “zoom in” on the most significant digits of the numbers, all while making the multiplier scalable and configurable. DRUM is also a type of approximation that truncates the input before multiplication and that induces huge error when compared to design.

Furthermore, its unbiased nature leads to distinct errors when used for computations within applications. DRUM’s (Dynamic Range Unbiased Multiplier) error and power consumption characteristics were analyzed, both analytically and empirically. Paper also evaluated the design against three different applications in a comprehensive study. Paper demonstrated that the quality of application is not notably affected, and the degradation is negligible. For example, on the image filtering application, with an application SNR of 91 dB, this design achieves power savings of 58%.

CHAPTER 3

BASE WORK ON APPROXIMATE MULTIPLIER

Design

This thesis proposes as a new rounding technique method for the input block prior to partial product generation. Accuracy curve as a criterion plays a critical role in controlling and minimizing the error range depending on applications. Different algorithms are implemented on different levels of the multiplier blocks. The input block applies rounding technique for both 16-bit and 32-bit based on accuracy levels. Generated partial products are divided into either active or inactive partial products. Inactive partial products are all zeros and hence are not needed to be considered in the reduction compressor process. This reduces the number of partial products for compression significantly. Compression of active partial products which takes most of the time, has been diminished and as positive impact on the circuit delay. The compressed output block contains both accurate compressor block and OR gates based approximate compressors to preserve its accuracy and minimize the hardware.

Block Diagram

The main idea behind the base approximate multiplier is to make use of rounded input for multiplication. The base algorithm applies a rounding technique before passing the data to the partial product generation. Fig. 1 shows the design chart for realization of the base method for the approximate multiplier design. Among the two inputs (Multiplicand and Multiplier), the Multiplier is rounded first by passing through rounding block. Before the multiplication operation starts, the sign bit of both inputs is stored, and the output sign of the multiplication result based on the inputs signs are determined. At the last stage, the proper sign is applied to the result. In an event of multiplying negative numbers, the respective input blocks are converted into their 2's complement. In conventional multipliers, with N -bit input, $N \times N$ partial products (partial products) are generated. But in the rounding technique, the partial products generated are the combination of active and inactive partial products. The active partial

products have "1" as the coefficient on the Multiplier. Therefore, inactive partial products are the lines with whole 0's with no necessity to cover them in the reduction process.

Figure 3: 16-bit Block diagram of algorithm

Input Rounding

Rounding input data requires major responsibility in maintaining the accuracy. With a basic intuition, it can be stated that, rounding lower bits results in less error compared to rounding higher bits. Thus, the base algorithm has assigned rounding weights with respect to the bit position value. From fig. 4 and table. 2, base method shows 16-bit position error curve that gives less weight to lower bits and more weight to higher bits. There is a small error gap between accurate bit position and rounded bit position. For every accurate bit, there is a corresponding rounded bit value assigned as seen in table. 1. Error gap reduces as the bit position value increases. Fig. 4 gives an example where 'A' and 'B' are inputs, and input 'B' is rounded to get 'Br'. "Rounding Technique" basically checks for a '1' in the 'X' bit position and assigns '1' to the respective 'Y' bit position with or without a small error.

Figure 4: Position error curve (16-bit) and an example

Table 3: Bit position strategy

Accurate Bit Position	Approximate Bit Position	Accurate Bit Position	Approximate Bit Position	Accurate Bit Position	Approximate Bit Position
bit0	bit1	bit5	bit4	bit10	bit10
bit1	bit1	bit6	bit7	bit11	bit10
bit2	bit1	bit7	bit7	bit12	bit13
bit3	bit4	bit8	bit7	bit13	bit13
bit4	bit4	bit9	bit10	bit14	bit15

Partial Product Reduction

Partial products reduction is a stage where partial products are compressed using different kinds of compressors. Base algorithm gives a flexibility on reducing number of partial product rows. For an instance, the 16-bit design shown in Fig. 5 reduces partial products to 6 row, which are identified as active partial products. With n -bit inputs are multiplication, $N \times N$ partial products are generated. In terms of computation complexity, as the number of bits increases, the length of Partial products increases with $O(N^2)$. Base algorithm provides computation complexity $\leq O(N \times 6)$ for 16-bit and $\leq O(N \times 13)$ for 32-bit. For better understanding, along with design, an example is explained with input values A , B and Br (from fig. 5(right)). Multiplier input 'B' is first rounded to 'Br'. Inputs are then multiplied to get $N \times N$ partial products. Due to rounding of multiplier input, $N \times N$ partial products is a combination of active and inactive partial products. Multiplier with '1' as coefficient, after rounding, causes a complete row of Multiplicand as a result as illustrated in fig. 6. Therefore, inactive partial products are the whole zero values lines, which had "0" as the coefficient on the Multiplier. Thus, there is no need to cover them in the reduction process. In fact, inactive partial products could only increase hardware. This has led us to first eliminate all inactive partial products before packing. This approach plays an important role in reducing power, area and time usage, in turn increasing efficiency. The active partial products are compressed and packed using three stages of compression. In the 1st stage, partial products are compressed using full adders and half adders. An output of 1st stage compression is further compressed using a 4:2 compressor when inputs are 16-bit whereas for 32-bit, 9:2 compressor. Fig. 5 (right) illustrates corresponding operations on an example. An output of the 2nd stage compressor is finally packed using OR gates to get a final product. Conventionally, full adders are used instead of OR. The very idea of using OR gates instead of full adders is to reduce area and energy usage noticeably.

Figure 5: Multiplier Design (left) with an example (right) (16-bit).

Rounding Error Analysis

Figure 6: Rounding Pattern and Probability of repetition (16-bit)

Table 4: Data with number of occurrences for whole range

Rounded value (DEC)	# of Occurrence of Rounded value	Rounded value (DEC)	# of Occurrence of Rounded value	Rounded value (DEC)	# of Occurrence of Rounded value
0	1	8338	1029	33936	1029
2	7	9216	21	33938	7203
16	7	9218	147	40960	9
18	49	9232	147	40962	63
128	7	9234	1029	40976	63
130	49	9344	147	40978	441
144	49	9346	1029	41088	63
146	343	9360	1029	41090	441
1024	7	9362	7203	41104	441
1026	49	32768	3	41106	3087
1040	49	32770	21	41984	63
1042	343	32784	21	41986	441
1152	49	32786	147	42000	441
1154	343	32896	21	42002	3087
1168	343	32898	147	42112	441
1170	2401	32912	147	42114	3087

Table 4 Continued

8192	3	32914	1029	42128	3087
8194	21	33792	21	42130	21608
8208	21	33794	147		
8210	147	33808	147		
8320	21	33810	1029		
8322	147	33920	147		
8336	147	33922	1029		

As seen in previous sections, only one Input (Multiplier) is rounded. As an instance let us consider 16-bit multiplier to analyze rounding error. From given inputs Multiplicand and Multiplier, Multiplier input is rounded. For 16-bit values ranging from 0 to 65535, a set of rounded values are obtained using rounding algorithm as discussed in fig. 4. Fig. 6 plots the relationship between rounded values and its occurrences. The step design obtained shows how close the rounded values are with each other. Most places, step sizes are small which infers as lesser error and hence more accuracy. Only in one case, we see higher step size which may incur slightly higher error. This shows that efficient data processing is done before the multiplication to mainly work on this area with additional hardware to obtain better accuracy. Thus, with this thorough analysis, a process. We then calculated the probability of occurrence of rounded values for numbers ranging from 0 to 65535. Red lines in fig. 6 shows, very good analogy on how rounded values are distributed, with probabilities. Area until 9360 rounded values have lower probability, which gives higher accuracy. Also, area from rounded value 40960 onwards also have lesser probability. Area at the center, with higher probability gives an opportunity to change rounding pattern to get better accuracy. Further studies and research will be mainly focusing on these areas to optimize algorithm with the expense of extra hardware. Table 3 provides number of occurrences of rounded values against input values.

CHAPTER 4

ROUNDING TECHNIQUE ANALYSIS

As part of analysis, there are 3 types of rounding techniques designed and analyzed thoroughly. Rounding 2-bit, 3-bit and 5-bit. As seen in figures 7 through 12 below, ‘x’ number of bits, where ‘x’ can be 2-bit, 3-bit or 5-bit, are chosen, and then majority function is applied. Based on majority function, if the number of ‘1’s is more than 50% of bits, then majority function returns 1. There are values defined to be assigned if majority function is 1, otherwise, same input in kept. This leads to a smaller number of 1’s in output of rounding function. Intuitively, we can say that, as number of bits increases for rounding, error increases. Rounding techniques are used based on applications. This is applied to both 16-bit and 32-bit.

For instance, for 3-bit rounding, 3 bits is given as input, number if number of 1’s ≥ 2 then majority function is one by outputting 4-bit output as 1000. In order to keep same length of input, additional bits between majority function outputs are overlapped using OR gates which incurs less error (as seen in the figures below). Similar analogy is applied to 2-bits and 5-bits.

Rounding 2-Bit

16-Bit

Figure 7: Rounding 16-bit data with 2-bit each time

32-Bit

Figure 8: Rounding 32-bit data with 2-bit each time

Rounding 3-bit

16-Bit

Figure 9: Rounding 16-bit data with 3-bit each time

32-Bit

Figure 10: Rounding 32-bit data with 3-bit each time

Rounding 5-bit

16-Bit

Figure 11: Rounding 16-bit data with 5-bit each time

32-Bit

Figure 12: Rounding 32-bit data with 5-bit each time

CHAPTER 5

RECONFIGURABLE SVM REALIZATION AND ANALYSIS

SVM Design

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where each class lies in one side. Fig. 13 shows the analogy of SVM (Support Vector Machine) for two class classification. As part of this project, two types of SVM are been implemented using MATLAB's `quadprog ()` function

1. Linearly non-separable SVM
2. Non-Linear SVM (Kernel SVM)

Linearly Non-Separable SVM

Why is it interesting to extend the SVM optimization problem to handle non-linearly separable training data? Most of the applications in which SVMs are used in computer vision require a more powerful tool than a simple linear classifier. This stems from the fact that in these tasks the training data can be rarely separated using a hyperplane. Thus, this method will help ease up the decision boundary by have two hyperplanes on either side.

Non-Linear SVM (Kernel)

This can be done using Mercer's theorem, which is a representation of a symmetric positive-definite function on a square as a sum of a convergent sequence of product functions. Inputs are first mapped to kernel and then used for classification. The kernel used here is 'Radial Basis Function'. The most common form of radial basis function is a Gaussian distribution, calculated as:

$$K(\vec{x}, \vec{z}) = e^{-(\vec{x}-\vec{z})^2 / (2\sigma^2)}$$

A radial basis function (rbf) is equivalent to mapping the data into an infinite dimensional Hilbert space.

Quadprog Programming

Quadprog finds a minimum for a problem. If the specified input bounds for a problem are inconsistent, the output x is x0 and the output fval is [].

Quadprog resets components of x0 that violate the bounds $lb \leq x \leq ub$ to the interior of the box defined by the bounds. quadprog does not change components that respect the bounds. A formula specified as $[x, fval, exitflag, output, lambda] = \text{quadprog}(H, f, A, b, Aeq, beq, lb, ub, x0)$ is used.

Error Analysis

Multiplier design has an additional block (rounding block) which rounds the input with three algorithms. Three algorithms are rounding 2-bits, 3-bits and 5-bits. Fig. 14 shows the outputs of 25 examples and its multiplier output of three rounding techniques in comparison with accurate design. It is clearly shown that absolute error of three rounding techniques are well within 10^a where ($a = -8$). Fig 15 also shows the “error percentage” of 25 examples. Thorough error analysis is done by running 1 to 255 values for 16-bit floating point numbers and 1 to 65535 values for 32-bit floating point numbers. Table. 4 shows that 16-bit with rounding 3-bit at a time has lowest absolute error and error percentage. Similarly, for 32-bit rounding 5-bit at a time has lowest error compared to others.

SVM Output Analysis

Support Vector Machine is implemented in MATLAB for both linearly non-separable and non-linearly separable algorithms on randomly generated floating-point data. The first figure in fig.16 shows the given data, which are non-separable. 2nd, 3rd and 4th figure in fig.16 shows the result of linearly non-separable algorithm with regularized parameter $C = 1, 10$ and 100 , respectively. Similarly, 5th, 6th and 7th figures show the results of nonlinearly separable kernel based SVM with $C = 1, 10$ and 100 , respectively. As C increases, distance between hyperplane decreases. This tends to have lesser support vectors, as seen in fig.16.

Figure 14: 25 examples of $n \times n$ values

Figure 15: Absolute Error (left) and Error Percent (right)

Table 5: Overall Error Analysis

	16 bit (FP – 1 to 255)			32-bit (FP = 1 to 65535)		
	Round2	Round3	Round5	Round2	Round3	Round5
Max Absolute Error	0.13	0.011	0.027	0.13	0.14	0.10
Max Error Percent	9.5	8.5	9.85	9.9	10	10

Different Rounding Techniques

Linear SVM is implemented with three different rounding techniques in approximate multiplier instead of accurate multiplier in MATLAB. Fig. 6 shows the outputs using both accurate multiplier and approximate multiplier (with three different rounding techniques). As seen in fig. 17, the number of support vectors remained the same, but distance between hyperplanes are changed when compared between accurate and approximate SVM.

Overfitting and Underfitting

The cause of poor performance in machine learning is often either overfitting or underfitting the data. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data are picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize. Underfitting is often not discussed, as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting. In this paper, since SVM is analyzed thoroughly with approximate multiplier, only training is done. Thus, to measure overfitting and underfitting, standard deviation is increase and decreased for underfitting and overfitting, respectively. Fig.18 shows the outputs of SVM when overfitting with $\sigma = 7$ and underfitting with $\sigma = 1$. Number of support vectors are different with approximate and accurate multipliers yet not much different in outputs.

Figure 16: SVM for linearly and non-linearly (RBF kernel)

Figure 17: SVM outputs for different rounding techniques.

Different Data Sets

SVM algorithm is further analyzed using different sets of data with different ranges. Fig. 8 shows two different outputs with small and large data sets for both accurate and approximate SVM's. Distance between hyperplane for accurate and approximate are pretty much same when seen in both small range and large range of data sets. Similarly, support vectors are also nearer.

Figure 18: SVM outputs for Overfitting and Underfitting

Table 6: Time complexity

Num.	Methods	Complexity (Big-O)
1	parse_command_line	$O(1)+O(n)$
2	read_problem	$O(1)+2*O(m*n)$
3	svm_predict (main) dan predict	$O(1)+O(n)+O(n*m)$
4	svm_check_parameter	$O(1)+O(m*n)+O(n^2)$
5	svm_train	$O(1)+9*O(n)+2*O(m*n)+6*O(n^2*m)$
6	svm_save_model	$O(1)+5*O(n)+2*O(m*n)$
7	svm_load_model	$O(1)+2*O(n)+2*O(m*n)$
8	svm_check_probability_model	$O(1)$
9	svm_predict_probability	$O(1)+3*O(n)+O(n^2)$
10	svm_predict	$O(1)+4*O(n)+2*O(n^3)$
11	Approximate multiplier	$O(k*n)$

Figure 19: Complexity Analysis

The Complexity Analysis

Complexity takes as a significant variable, so in this case n^2 is more significant than the n . Complexity for `svm_train`, `svm_predict` and `svm` can be seen in fig 19. The table in fig.19 shows total complexity for all methods in SVM. It also shows some methods in LibSVM where `svm_predict` provide the highest complexity, is $O(n^3)$. Actually, `svm_train` also have the same complexity with `svm_predict`, but in this case we limit our computation only to main routine without any further search of the sub-routine, and then some routines in `svm_train` are considered as $O(1)$. Our analysis shows that SVM algorithm requires three nested loop so that the complexity of SVM is $O(n^3)$. This analysis is in accordance to the standards of SVM complexity. Approximate computing's big O notation is calculated based on partial product rows used for computation. Active partial product row is now reduced to 'k' for n-bit values, thus $O(k*n)$.

CHAPTER 6

OPTIMIZED APPROXIMATE MULTIPLIER DESIGN

Block Diagram

Figure 20: Optimized block diagram

With thorough analysis on previous algorithms, an optimized algorithm was designed to wisely and optimistically trade-off between hardware characteristics and accuracy. As seen in fig. 20, signs of two inputs A and B are determined first. Then one of the inputs is processed for rounding. In order to get lower error, rounding ‘multiplier’ was better. Furthermore, input to rounding function is reduced by finding the leading one from the left because 0’s from the left means, that the number is small. There is no need to process all n-bit if number is small. In this way, accuracy and hardware characteristics are positively obtained based on inputs. Truncated input is rounded using rounding algorithms discussed in previous chapters. Rounding reduces number of 1’s in

“multiplier” (B) input drastically, which in-turn reduces partial product rows. This gives an advantage on have lesser hardware for compressors and does not compromise with accuracy. After padding back to required input size and compressing, output is now ready as product of two numbers. Sign bit is then put back based on input sign bits.

Hardware Design

Figure 21: Hardware Design of optimized multiplier

Fig. 21 demonstrates hardware design of optimized approximate multiplier for 16-bit using rounding 2-bit function. Similarly, other rounding techniques 3-bit and 5-bit rounding functions can be designed. Above design is independent of number of bits for the input. As stated in block diagram, 2, n-bit input are first separated with its sign bit by doing XOR with 16'h8000. Multiplier input (B) is then rounded using rounding function (as explained in below “Rounding Techniques”). B input now has few 1’s, which in-turn helps in having fewer partial product rows. Based on ‘1’ position in B, A value is added incrementally to Sum_AB and output using multiplexer, which has enable pin as B input bits. Once sum of partial product is calculated, sign bit is checked of inputs, and result is presented as product.

Rounding Techniques

Rounding 2-bit Function

Figure 22: Rounding 2-bit function

Rounding 3-bit Function

Figure 23: Rounding 3-bit function

Rounding 5-bit Function

Figure 24: Rounding 3-bit function

Fig 22 to 24 show the Rounding function for 2-bit or 3-bit or 5-bit as an input. Majority function is decided by using AND between trail of bits, and then majority bit is using as enable for multiplexer to give output same as input if majority bit is 0 or value shown in fig if majority bit is 1. In any case, output is of fewer 1's in trail of bits.

CHAPTER 7

SIMULATIONS AND TEST RESULTS

Hardware Characteristics and Results

Part of this thesis is being published in IEEE CCWC 2019, where Power, Area, and Delay are calculated using design compiler on three different technologies (32 nm, 45 nm, and 90 nm) for both 16-bit and 32-bit. Fig. 25 clearly shows that the proposed algorithm is better than other algorithms. It is compared with DRUM [1] and accurate multiplication using 32 nm, 45 nm, and 90 nm. Table.7 shows the Area-Power-Delay comparable readings using design compiler.

The result seen in table.7 reveals that area usage is reduced by 45% (85%) for 16-bit (32-bit) inputs. Power usage has reduced significantly to 30% (5%) when compared to accurate 100% (100%) and DRUM for [1] 16-bit (32-bit). The results also indicate that, delay is as low as 60% (60%) when compared to accurate 100% (100%) for 16-bit (32-bit). DRUM [1] seems to have much higher delay than accurate, thus it's not comparable to proposed algorithm.

Among all the data collected, the power is in order of milliwatt (mW), the delay in order of nano (ns), area in the order of micrometer square (μm^2), and the PDP in order of picojoules (pJ). In a similar fashion, optimized algorithms rounding 2-bit, 3-bit and 5-bit with RoundMul2, RoundMul3 and RoundMul5 respectively, are compared with CCWC2019 (own base algorithm), DRUM [1] and TOSAM [13] by obtaining hardware characteristics using design compiler for 32 nm, 45 nm and 90 nm architectures.

Table. 8 and 9 shows the reading of power, delay, area and PDP for both 16-bit and 32-bit. Both actual values and its norm are calculated. Area is in μm^2 , power is in mW, delay is in ns and PDP is in pJ as noted.

Fig 26 shows graphical representation of hardware characteristics for area(a), delay(b), power(c) and PDP(d). Optimized algorithms are mainly focused to get approximate values as near to accurate value as possible keeping hardware characteristics well within limit. As seen in area plot, area consumed by optimized algorithms, RoundMul2, RoundMul3 and RoundMul5 are less than accurate whereas power and PDP fall much less than accurate. When compared to other reference algorithms, values are very competitive in both 16-bit and 32-bit. When it comes to delay, 32-bit seems to have stabilized delay when compared to accurate, whereas DRUM seems to have higher delay in both 16-bit and 32-bit.

Table 7: Area-Power-Delay comparison for 16-bits & 32-bits

		16-bit						32-bit					
		32 nm		45 nm		90 nm		32 nm		45 nm		90 nm	
		μm^2	Norm	μm^2	Norm	μm^2	Norm	μm^2	Norm	μm^2	Norm	μm^2	Norm
Area	Accurate	3130.29	100.00	4830.97	100.00	11669.29	100.00	8725.01	100.00	13427.14	100.00	42684.82	100.00
	DRUM	1795.78	57.37	2756.19	57.05	6762.70	57.95	3342.50	38.31	5143.53	38.31	12451.74	29.17
	CCWC 2019	1844.83	58.93	2618.22	54.20	6453.96	55.31	1349.25	15.46	1924.60	14.33	5679.82	13.31
		mW	Norm	mW	Norm	mW	Norm	mW	Norm	mW	Norm	mW	Norm
Power	Accurate	0.34	100.00	3.53	100.00	6.70	100.00	1.23	100.00	13.09	100.00	26.28	100.00
	DRUM	0.15	45.77	1.34	37.92	2.32	34.63	2.32	18.93	1.74	13.34	2.93	11.18
	CCWC 2019	0.12	35.45	1.03	29.36	2.12	31.72	0.07	5.40	0.44	3.42	1.03	3.94
		ns	Norm	ns	Norm	ns	Norm	ns	Norm	ns	Norm	ns	Norm
Delay	Accurate	4.32	100.00	3.67	100.00	11.60	100.00	7.22	100.00	5.65	100.00	37.11	100.00
	DRUM	5.76	133.33	4.48	122.07	13.99	120.60	9.72	134.63	7.91	140.00	24.27	65.40
	CCWC 2019	2.85	65.97	2.17	59.13	6.77	58.36	6.66	92.24	5.20	92.04	21.79	58.72
		pJ	Norm	pJ	Norm	pJ	Norm	pJ	Norm	pJ	Norm	pJ	Norm
PDP	Accurate	1.46	100.00	12.96	100.00	77.72	100.00	8.88	100.00	73.96	100.00	975.25	100.00
	DRUM	0.86	61.03	6.00	46.29	32.46	41.77	22.55	25.49	13.76	18.68	71.11	7.31
	CCWC2019	0.34	23.39	2.24	17.36	14.35	18.51	0.44	4.98	2.29	3.15	22.44	2.31

(a) Area Comparison

(b) Delay Comparison

(c) Power Comparison

(d) PDP Comparison

Figure 25 (a) (b)(c)(d): Characteristic comparison for 16 and 32-bit

Table 8: Area-Power-Delay comparison for 16-bits

		16-bit					
		32 nm		45 nm		90 nm	
		μm^2	Norm	μm^2	Norm	μm^2	Norm
Area	Accurate	3130.29	100.00	4830.97	100.00	11669.29	100.00
	RoundMul2	2893.17	92.43	4768.55	98.71	11170.71	95.73
	RoundMul3	2752.63	87.94	4496.83	93.08	10635.26	91.14
	RoundMul5	2864.45	91.51	4760.57	98.54	11048.14	94.68
	DRUM	1818.14	57.37	2756.19	57.05	6762.70	57.95
	CCWC2019	1844.83	58.93	2618.22	54.20	6453.96	55.31
	TOSAM_3_7	1240.22	39.62	1765.50	36.55	4565.60	39.12
		mW	Norm	Mw	Norm	mW	Norm
Power	Accurate	338.82	100.00	3534.20	100.00	6709.20	100.00
	RoundMul2	189.34	55.88	613.10	17.35	2963.60	44.17
	RoundMul3	229.06	67.61	2156.60	61.02	4007.00	59.72
	RoundMul5	163.54	48.27	1070.70	30.30	2335.50	34.81
	DRUM	157.13	45.77	1339.50	37.92	2323.70	34.63
	CCWC2019	120.12	35.45	1037.80	29.36	2128.00	31.72
	TOSAM_3_7	95.86	28.29	838.80	23.73	1468.10	21.88

Table 8 Continued

		ns	Norm	Ns	Norm	ns	Norm
Delay	Accurate	4.32	100.00	3.67	100.00	11.60	100.00
	RoundMul2	5.21	120.60	4.09	111.44	14.11	121.64
	RoundMul3	3.60	83.33	2.81	76.57	11.13	95.95
	RoundMul5	5.49	127.08	4.51	122.89	14.40	124.14
	DRUM	5.76	133.33	4.48	122.07	13.99	120.60
	CCWC2019	2.85	65.97	2.17	59.13	6.77	58.36
	TOSAM_3_7	4.07	94.21	2.93	79.84	7.72	66.55
		<i>pJ</i>	<i>Norm</i>	<i>pJ</i>	<i>Norm</i>	<i>pJ</i>	<i>Norm</i>
PDP	Accurate	1463.70	100.00	12970.51	100.00	77826.72	100.00
	RoundMul2	986.46	67.40	2507.58	19.33	41816.40	53.73
	RoundMul3	824.62	56.34	6060.05	46.72	44597.91	57.30
	RoundMul5	897.83	61.34	4828.86	37.23	33631.20	43.21
	DRUM	905.07	61.03	6000.96	46.29	32508.56	41.77
	CCWC2019	342.34	23.39	2252.03	17.36	14406.56	18.51
	TOSAM_3_7	390.15	26.66	2457.68	18.95	11333.73	14.56

Table 9: Area-Power-Delay comparison for 32-bits

		32-bit					
		32 nm		45 nm		90 nm	
		μm^2	Norm	μm^2	Norm	μm^2	Norm
Area	Accurate	8725.01	100.00	13427.14	100.00	42684.82	100.00
	RoundMul2	10216.33	117.09	15497.22	115.42	46334.36	108.55
	RoundMul3	10175.16	116.62	15441.84	115.00	46158.33	108.14
	RoundMul5	10150	116.33	15380.36	114.55	46049.58	107.88
	DRUM	3364.86	38.57	5143.52	38.31	12451.73	29.17
	CCWC2019	1349.25	15.46	1924.60	14.33	5679.82	13.31
	TOSAM_3_7	1258.26	14.42	1931.16	14.38	4585.88	10.74
		mW	Norm	mW	Norm	mW	Norm
Power	Accurate	1230.2	100.00	13099.5	100.00	26287	100.00
	RoundMul2	1004.9	81.69	9056.2	69.13	18829	71.63
	RoundMul3	1003.9	81.60	9078.2	69.30	18830	71.63
	RoundMul5	1004.1	81.62	9078.4	69.30	18834	71.65
	DRUM	234.71	19.08	1747.20	13.34	2939.50	11.18
	CCWC2019	66.41	5.40	448.20	3.42	1036.10	3.94
	TOSAM_3_7	92.12	7.49	848.90	6.48	1379.00	5.25

Table 9 Continued

		<i>ns</i>	<i>Norm</i>	<i>Ns</i>	<i>Norm</i>	<i>ns</i>	<i>Norm</i>
Delay	Accurate	7.22	100.00	5.65	100.00	37.11	100.00
	RoundMul2	7.04	97.51	5.42	95.93	36.04	97.12
	RoundMul3	7.04	97.51	5.42	95.93	36.04	97.12
	RoundMul5	7.04	97.51	5.42	95.93	36.04	97.12
	DRUM	9.69	134.21	7.91	140.00	24.27	65.40
	CCWC2019	6.66	92.24	5.20	92.04	21.79	58.72
	TOSAM_3_7	1.90	26.32	1.56	27.61	5.24	14.12
		<i>pJ</i>	<i>Norm</i>	<i>pJ</i>	<i>Norm</i>	<i>pJ</i>	<i>Norm</i>
PDP	Accurate	8882.044	100.00	74012.18	100.00	975510.6	100.00
	RoundMul2	7074.496	79.65	49084.6	66.32	678597.2	69.56
	RoundMul3	7067.456	79.57	49203.84	66.48	678633.2	69.57
	RoundMul5	7068.864	79.59	49204.93	66.48	678777.4	69.58
	DRUM	2274.34	25.61	13820.35	18.67	71341.67	7.31
	CCWC2019	442.2773	4.98	2330.64	3.15	22576.62	2.31
	TOSAM_3_7	175.028	1.97	1324.284	1.79	7225.96	0.74

(a) Area Comparison

(b) Delay Comparison

(c) Power Comparison

(d) PDP Comparison

Figure 26(a)to(d): Characteristic comparison of proposed & references

Accuracy Analysis

Figure 27: Accuracy analysis for 16-bit values.

Fig.27 demonstrates accuracy curves for a few 16-bit examples ranging from small values to larger. By thoroughly analyzing error potential areas as discussed during fig.6, three optimized algorithms were introduced as RoundMul2, RoundMul3 and RoundMul5, which reduced error gap from accurate values. As seen in fig.27, accuracy curve clearly shows that the optimized algorithm's curve goes hand in hand with accuracy curve, whereas both DRUM and TOSAM seem to have large gaps for few ranges of values.

Table 10: PSNR and SSIM for 16-bits and 32-bits

PSNR		Cameraman			Lena		
	Sharpening levels	0.8	1.0	1.2	0.8	1.0	1.2
8-bit	Rounding2	28.44	27.97	27.53	28.95	28.43	27.93
	Rounding3	43.54	43.01	42.51	39.41	38.73	38.09
	Rounding5	44.75	44.56	44.42	44.96	44.67	44.38
16-bit	Rounding2	28.44	27.97	27.53	28.95	28.43	27.93
	Rounding3	26.37	25.98	25.61	28.41	27.91	27.43
	Rounding5	27.41	26.73	26.10	27.33	26.65	26.01
32-bit	Rounding2	28.44	27.97	27.53	28.95	28.43	27.93
	Rounding3	26.37	25.98	25.61	28.41	27.91	27.43
	Rounding5	27.41	26.73	26.10	27.33	26.65	26.01

Table 10 continued

SSIM		Cameraman			Lena		
	Sharpening levels	0.8	1.0	1.2	0.8	1.0	1.2
8-bit	Rounding2	0.91	0.90	0.90	0.84	0.83	0.82
	Rounding3	0.99	0.99	0.99	0.98	0.98	0.98
	Rounding5	0.99	0.99	0.99	0.99	0.99	0.99
16-bit	Rounding2	0.91	0.90	0.90	0.84	0.83	0.82
	Rounding3	0.84	0.83	0.82	0.84	0.82	0.81
	Rounding5	0.78	0.76	0.75	0.72	0.71	0.69
32-bit	Rounding2	0.91	0.90	0.90	0.84	0.83	0.82
	Rounding3	0.84	0.83	0.82	0.84	0.82	0.81
	Rounding5	0.78	0.76	0.75	0.72	0.71	0.69

Table 10 shows the PSNR and SSIM of Cameraman and Lena images after sharpening them with different levels and using approximate multiplier for computation. SSIM seems to have better values in all except for Rounding 5-bit for 16-bit and 32-bit, which is intuitive. Whereas PSNR seems to have distinct values that are not easy to distinguish for good analysis. Image quality does not completely depend on these parameters. Accuracy is analyzed by running with different examples. Less accuracy is seen when number is small (lower bits are mostly one's) but high accuracy with higher value numbers (higher bits are mostly one's). Figure 14, 15 and 16 show the relationship between accuracy and energy consumed between different algorithms.

The three figure emphasizes accuracy w.r.t lower bits, middle bit and higher bits. EDP-RTAM (Efficient Data Processing with Rounding Technique for Approximate Multipliers) has better accuracy with middle and higher bits. For conventional multiplication (accurate), percent is 100% and EDP-RTAM and DRUM is compared in relative to Accurate. As seen in figures below, EDP-RTAM has lower accuracy compared to DRUM but acceptable, whereas EDP-RTAM consumes much less energy than DRUM, with acceptable accuracy.

Figure 28: Complexity Analysis

CHAPTER 8

APPLICATIONS

Optimized algorithms are exercised and analyzed by re-writing in MATLAB as function and in-cooperating in image processing and pattern recognition algorithms. Applications used are sharpening of image (Image processing application) and SVM (Pattern Recognition algorithm). SVM application was discussed in detail in a separate chapter with name “Reconfigurable and Realization of SVM”. Fig.28 to 31 show the images when sharpened with accurate and optimized approximate multiplier. Visual difference is bare minimum. Two images were used, Lena and Cameraman, with sharpening amount as 0.8, 1.0 and 1.2. Sharpening an image increases the contrast between bright and dark regions to bring out features. The sharpening process is basically the application of a high pass filter to an image.

Figure 29: Image sharpening using accurate multiplier

Figure 30: Image sharpening using approximate multiplier

Figure 31: Image sharpening using accurate multiplier

Figure 32: Image sharpening using approximate multiplier

CHAPTER 9

CONCLUSIONS

The proposed algorithm proves superior in terms of power-area, delay and PDP efficiency when compared to other algorithms for both signed and unsigned data (16-bit and 32-bit). Primary investigation of rounding technique on approximate multiplier was done by having one method of rounding pattern with fixed active partial product rows. With its rounding pattern, we see potential areas of less accuracy and areas with better accuracy corresponding to probability of rounding value. Based on accuracy required, rounding patterns are changed with a little extra expense of hardware. Rounding patterns can be modified to have fixed or dynamic partial product rows and yet have fewer active partial product rows for compression. The proposed algorithm can be used in the wide range of applications in image processing, machine learning and signal processing. Thus, different weights based on the bit position of '1' play an important role to keep accuracy relatively near to the conventional method.

Optimized approximate multiplier proves to be best in terms of power, area and delay efficiency when compared to other algorithms. Accuracy is compromised in a few zones but not everywhere. Proposed algorithm can be used in vast area of Image processing, Machine learning applications, and many more. Accuracy difference is not very much visible in lower valued number. As compared to previous work, in this case DRUM [1], Power consumption is 3 times lesser, area usage is 10% - 30% lesser, and delay very negligible comparatively. With flexible reduction of partial products, the proposed algorithm produces great hardware characteristics, when compared to DRUM [1].

BIBLIOGRAPHY

- [1] S. Hashemi, R. I. Bahar and S. Reda, "DRUM: A Dynamic Range Unbiased Multiplier for approximate applications," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 418-425, 2015.
- [2] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha and M. Pedram, "RoBA multiplier: A rounding-based approximate multiplier for highspeed yet energy-efficient digital signal processing," *IEEE Transaction Very Large-Scale Integer. (VLSI) System.*, vol. 25, no. 2, pp. 393-401, 2017.
- [3] L. Vasileios, Z. Georgios and X. Sotirios, "Walking through the Energy-Error Pareto Frontier of Approximate Multipliers," *IEEE Micro*, vol. 38, no. 4, pp. 40-49, August 2018.
- [4] H.-J. Ko and S.-F. Hsiao, "Design and Application of Faithfully Rounded and Truncated Multipliers With Combined Deletion, Reduction, Truncation, and Rounding," *IEEE Transactions on Circuits and Systems II*, vol. 58, no. 5, pp. 304-308, 2011.
- [5] A. Gupta, S. Mandavalli, V. J. Mooney, K.-V. Ling, A. Basu, H. Johan and B. Tandianus, "Low power probabilistic floating point multiplier design," in *2011 IEEE Computer Society Annual Symposium on VLSI*, Chennai, India, 2011.
- [6] H. Zhang, M. Putic and J. Lach, "Low power GPGPU computation with imprecise hardware," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, San Fransisco, USA, 2014.
- [7] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 6, pp. 1180-1184, 2015.
- [8] A. Lingamneni, A. Basu, C. Enz, K. V. Palem and C. Piguet, "Improving energy gains of inexact DSP hardware through reciprocative error compensation," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, USA, 2013.
- [9] W. Liu, L. Chen, C. Wang, M. O'Neill and F. Lombardi, "Design and analysis of inexact floating-point adders," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 308-314, 2016.

- [10] J. Liang, J. Han and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders.," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1760-1771, 2013.
- [11] A. Momeni, J. Han, P. Montuschi and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984-994, 2015.
- [12] Z. Vasicek and L. Sekanina, "A global postsynthesis optimization method for combinational circuits," in *2011 Design, Automation & Test in Europe*, Grenoble, France, 2011.
- [13] S. Vahdat, M. Kamal, A. Afzali-Kusha and M. Pedram, "TOSAM: An Energy-Efficient Truncation- and Rounding-Based Scalable Approximate Multiplier," *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*, vol. 27, no. 5, pp. 1-13, 2019.
- [14] M. V. Leussen, L. W. J. Huiskens, H. Jiao and J. P. D. Gyvez, "Reconfigurable Support Vector Machine Classifier with Approximate," *IEEE Computer Society Annual Symposium on VLSI*, pp. 13-18, 2017.
- [15] S. Vahdat, M. Kamal, A. Afzali-Kusha and M. Pedram, "LETAM," *Computers and Electrical Engineering, Science Direct*, vol. 63, pp. 1-17, 2017.
- [16] E. team. [Online]. Available: <https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/>.
- [17] [Online]. Available: https://en.wikipedia.org/wiki/Approximate_computing.
- [18] D. A. Sahu, "CS223 : Hardware Lab Jan 2013- Apr 2013," [Online]. Available: <https://www.iitg.ac.in/asahu/cs223-13/>.
- [19] V. Leon, G. Zervakis, D. Soudris and K. Pekmestzi, "Approximate Hybrid High Radix Encoding for Energy-Efficient Inexact Multipliers," *IEEE Transactions*, vol. 26, no. 3, pp. 421-430, March 2018.
- [20] M. Nazemi and M. Pedram, "Deploying Customized Data Representation and Approximate Computing in Machine Learning Applications," *ACM Digital Library*, no. 48, pp. 1-6, July 2018.
- [21] M. S. Ansari, H. Jiang, B. F. Cockburn and J. Han, "Low-Power Approximate Multipliers Using Encoded Partial Products and Approximate Compressors," *IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS*, vol. 8, no. 3, pp. 404-416, 2018.

- [22] P. Suhasini and Dr.J.Selvakumar, "Modified Rounding Based Approximate Multiplier (MROBA) and MAC Unit Design for Digital Signal Processing," *International Journal of Pure and Applied Mathematics*, vol. 118, no. 18, pp. 1539-1545, 2018.