UNIVERSITY OF GENOVA

Polytechnic School

Doctoral Program in Science and Technology for Electronic and Telecommunication Engineering (XXXII cycle)

Energy-efficient embedded machine learning algorithms for smart sensing systems

Mario Osta

Supervisor: Prof. M. Valle

A thesis submitted for the degree of
Doctor of Philosophy

February 2020

ACKNOWLEDGMENTS

This Ph.D. journey is a combined effort of several individuals, including the scholars, family members who supported me, and friends who were always accessible through this journey.

First and foremost, I would like to express my gratitude to my marvelous thesis supervisor Prof. Maurizio Valle whose support, guidance during these three years has made this thesis possible. He gave me opportunities to improve my work through my learning path.

I am very grateful to my colleague Dr. Ali Ibrahim for his encouragement and extensive support through my Ph.D. journey. His evaluation and suggestions were essential to solving various bottlenecks in my research work innovatively. He influenced me to work hard and persist in my determination.

During my Ph.D., I had the opportunity to work at the Swiss Federal Institue of Technology in Zurich (ETH) as a visiting researcher. I am grateful to my internship mentor for making this research work successful.

I would like to thank my friends and lab mates at the COSMIC lab for the lively discussion and unforgettable adventures. I am also grateful to all my friends at the ETH IIS lab in Zurich.

Last but not least, I would also like to thank my parents, my sister, and my brother, for their advice and support throughout my life. This could not be achieved without them.

SUMMARY

Embedded autonomous electronic systems are required in numerous application domains such as Internet of Things (IoT), wearable devices, and biomedical systems. Embedded electronic systems usually host sensors, and each sensor hosts multiple input channels (e.g., tactile, vision), tightly coupled to the electronic computing unit (ECU). The ECU extracts information by often employing sophisticated methods, e.g., Machine Learning. However, embedding Machine Learning algorithms poses essential challenges in terms of hardware resources and energy consumption because of: 1) the high amount of data to be processed; 2) computationally demanding methods. Leveraging on the trade-off between quality requirements versus computational complexity and time latency could reduce the system complexity without affecting the performance. The objectives of the thesis are to develop: 1) energy-efficient arithmetic circuits outperforming state of the art solutions for embedded machine learning algorithms, 2) an energy-efficient embedded electronic system for the "electronic-skin" (e-skin) application. As such, this thesis exploits two main approaches:

Approximate Computing: In recent years, the approximate computing paradigm became a significant major field of research since it is able to enhance the energy efficiency and performance of digital systems. "Approximate Computing"(AC) turned out to be a practical approach to trade accuracy for better power, latency, and size [1],[2]. AC targets error-resilient applications and offers promising benefits by conserving some resources. Usually, approximate results are acceptable for many applications, e.g., tactile data processing [3], [4] image processing [5], and data mining [6]; thus, it is highly recommended to take advantage of energy reduction with minimal variation in performance [7]. In our work, we developed two approximate multipliers: 1) the first one is called “META” multiplier and is based on the Error Tolerant Adder (ETA), 2) the second one is called “Approximate Baugh-Wooley(BW)” multiplier where the approximations are implemented in the generation of the partial products. We showed that the proposed approximate arithmetic circuits could achieve a relevant reduction in power consumption and time delay around 80.4% and 24%, respectively, with respect to the exact BW multiplier. Next, to prove the feasibility of AC in real world applications, we explored the approximate multipliers on a case study as the e-skin application. The e-skin application is defined as multiple sensing components, including 1) structural materials, 2) signal processing, 3) data acquisition, and 4) data processing. Particularly, processing the originated data from the e-skin into low or high-level information is the main problem to be addressed by the embedded electronic system. Many studies have shown that Machine Learning is a promising approach in processing tactile data when classifying input touch modalities. In our work, we proposed a methodology for evaluating the behavior of the system when introducing approximate arithmetic circuits in the main stages (i.e., signal and data processing stages) of the system. Based on the proposed methodology, we first implemented the approximate multipliers on the low-pass Finite Impulse Response (FIR) filter in the signal processing stage of the application. We noticed that the FIR filter based on (Approx-BW) outperforms state of the art solutions, while respecting the tradeoff between accuracy and power consumption, with an SNR degradation of 1.39dB. Second, we implemented approximate adders and multipliers respectively into the Coordinate Rotational Digital Computer (CORDIC) and the Singular Value Decomposition (SVD)

circuits; since CORDIC and SVD take a significant part of the computationally expensive Machine Learning algorithms employed in tactile data processing. We showed benefits of up to 21% and 19% in power reduction at the cost of less than 5% accuracy loss for CORDIC and SVD circuits when scaling the number of approximated bits.

2) **Parallel Computing Platforms (PCP):** Exploiting parallel architectures for near-threshold computing based on multi-core clusters is a promising approach to improve the performance of smart sensing systems. In our work, we exploited a novel computing platform embedding a Parallel Ultra Low Power processor (PULP), called "Mr. Wolf," for the implementation of Machine Learning (ML) algorithms for touch modalities classification. First, we tested the ML algorithms at the software level; for RGB images as a case study and tactile dataset, we achieved accuracy respectively equal to 97% and 83.5%. After validating the effectiveness of the ML algorithm at the software level, we performed the on-board classification of two touch modalities, demonstrating the promising use of Mr. Wolf for smart sensing systems. Moreover, we proposed a memory management strategy for storing the needed amount of trained tensors (i.e., 50 trained tensors for each class) in the on-chip memory. We evaluated the execution cycles for Mr. Wolf using a single core, 2 cores, and 3 cores, taking advantage of the benefits of the parallelization. We presented a comparison with the popular low power ARM Cortex-M4F microcontroller employed, usually for battery-operated devices. We showed that the ML algorithm on the proposed platform runs 3.7 times faster than ARM Cortex M4F (STM32F40), consuming only 28 mW. The proposed platform achieves 15× better energy efficiency than the classification done on the STM32F40, consuming 81mJ per classification and 150 pJ per operation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

1.1 Introduction

Over the past decade, a new wave of intelligent computing systems driven by machine learning algorithms has been deployed in many applications such as the Internet of things and security systems. IoT devices ought to effectively decode a large amount of raw data coming from miniaturized sensors providing vital information. The extracted information is transmitted onto the wireless channel, and it is usually used to close the loop in control systems providing real-time response. Besides, the small size form of these embedded devices (e.g., wearable sensors) limits the battery size and hence the energy availability.

Furthermore, intelligent devices could move "closer to the sensor," thereby eliminating the latency of cloud access and reducing limitations in communication bandwidth by employing sophisticated methods (e.g., Machine learning). Machine learning is considered as dominant paradigms making intelligent tasks and providing information about the observed phenomenon. However, deploying Machine Learning algorithms in embedded devices poses several challenges in terms of hardware resources and energy consumption because of 1) the high amount of data to be processed significantly affecting the real-time functionality, and 2) the complex processing tasks involve computationally demanding methods imposing an additional burden in terms of power consumption. Therefore, aiming to implement powerful machine and deep learning algorithms within a power range of microwatt requires an essential improvement in processing energy efficiency.

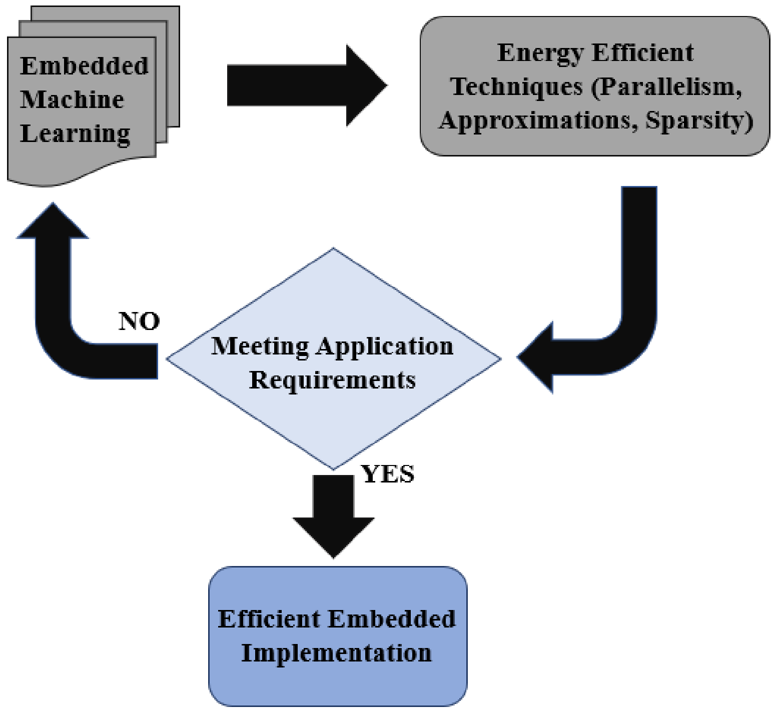

Embedded Machine Learning
Energy Efficient Techniques (Parallelism, Approximations, Sparsity)
NO
Meeting Application Requirements
YES
Efficient Embedded Implementation

Fig.1.1. Procedure for implementing an energy efficient embedded machine learning algorithm

Leveraging on the trade-off between quality versus computational complexity and latency would reduce the system complexity (see Fig.1.1) without affecting the system functionality. Such an approach can be achieved by several energy-efficient techniques (Parallelism and data reuse, Approximate Computing, Network Sparsity) on embedded machine learning systems. In this perspective, this chapter aims to present an overview of the energy-efficient implementation of machine learning algorithms on hardware platforms highlighting the main challenges when embedding such algorithms. Moreover, it reports the techniques that could be applied to improve energy efficiency, such as approximate computing, by reviewing the effective methods used to overcome the challenges at the circuit, architectural, and algorithmic levels. Further, it describes the main factors to be taken into consideration when choosing the appropriate platform. Lastly, this chapter describes the context of the work and the mains contributions achieved during the Ph.D. The overview work presented in this chapter was published in [129], [131].

1.2 Embedded Machine Learning

Machine learning has emerged in different scientific fields and everyday tasks in today's electronic systems and smartphones. ML paradigms have been effectively used to address standard regression and classification problems. Furthermore, deep learning methods represent state-of-the-art technology addressing the task of extracting structured information from complex domains such as object/face recognition, object tracking, etc. However, employing such paradigms for embedded platforms imposes challenges in terms of time latency, energy consumption, and storage.

1.2.1 Power/Energy Consumption

Energy consumption is the power consumed during the runtime of the algorithm when targeting embedded implementations. It is the energy consumed during the execution of the algorithm, basically composed of computing energy, IO operations, and necessary energy for memory storage. Energy efficiency is considered an essential metric, especially when dealing with applications such as portable, medical/biomedical IoT devices. Moreover, integrated circuits must embed ML instead of power-hungry FPGA-based microprocessors in order to meet the low power budget constraint in wearable or implantable devices [8],[9]. To emphasize the critical need of this metric, we take an example of the implemented tensorial SVM on the FPGA device for classifying different touch modalities, as shown in [10]. The proposed implementation is feasible for real-time classification while the amount of power consumed is 1.14 W. Similarly, as shown in

[8],[9] ML must be embedded into dedicated platforms in order to reduce the power envelope constraint in wearable devices to the range of mW. Therefore, the critical challenge is to improve power consumption while preserving the real-time constraints for longer battery life. Improving energy efficiency will provide longer battery life, and then more extended functionality within the same energy source.

1.2.2 Latency

Real-time operation is a principal requirement in many application domains for embedded ML[11]. Latency is defined as the time difference between the generated output data and the input data provided to the system. The time latency depends on the number of operations performed in a unit of time, such as operations per second (OPS) or floating-point operations per second (FLOPS). In many real-time applications such as deep brain stimulation[12]and vital sign monitoring[13], the response time of the machine learning algorithm must be adequately fast. Moreover, [14] and [15] ML/DL algorithms take more than 1 second to classify different objects. This fact highlights the latency problem faced in IoT devices when implementing ML/DL algorithms; since the application must meet real-time constraints.

1.2.3 Memory Storage

Memory is a big challenge in embedded machine learning and deep neural networks today[16],[17]. Memory subsystems expend significant time and energy in computing platforms due to the frequent high data transfer between processors and off-chip memory[18]. Researchers are combatting with the DRAM devices having limited capacity and memory bandwidth. The memory is employed for storing a high amount of training data, and parameters such as weights in DNN and support vectors in SVM. The higher the number of training data, the harder the memory challenge. For example, more than 900M operations of memory read, and writes are needed in [19].

1.2.4 Algorithm Complexity

Concerning the complexity of the algorithm, it depends on the number of mathematical operations and instructions executed by the algorithm in the embedded space. For instance, ANN requires energy consumption less than SVM since the ANN model takes five times fewer operations than SVM [20].

1.3 Machine Learning On Embedded Hardware Platforms

This section describes the steps to follow when implementing ML on embedded hardware platforms [21]. A model with few operations should be selected since some networks, e.g., AlexNet and GoogleNet[22], [23], are not designed for the embedded space. Then, a performance study must be applied on the selected ML algorithm, which depends on some characteristics such as: 1) the number of layers, 2) fps (frames per second) requirements, 3) the number of bits used in the algorithm, 4) memory storage and 5) number of operations. The final step is to look for processors supporting some significant features (e.g., quantization, sparsity, etc.). For example, different software architectures of CNN have been designed in[24] in order to find only one suitable architecture having the lowest complexity.

Moreover, the activation of the hidden layers with the parameters of the selected CNN, which will be implemented, must be regularly stored and accessed. Then, the low power microcontroller from the MSP430FR series is selected as the target platform. It is characterized by a ferromagnetic memory of 256 kB, which reduces power consumption by three times and increases the speed up of the data movement by 100x. Similarly, in[25], the DNPU processor characterized by a LUT-based multiplier dedicated to quantization is considered optimal for the implementation of CNN and RNN. Thus, reducing off-chip memory access and improving the energy efficiency consequently by 4.5x with a negligible accuracy loss. Therefore, by following the flow diagram, the selected processor will feed the requirements of the selected algorithm, leading to an efficient implementation.

1.4 Energy-Efficient Techniques For ML/DL Processing

Table 1.1.: Energy efficient embedded machine/deep learning algorithms on different hardware platforms

Energy Efficient Technique	**Design Approach**	**M.L and D.L algorithms**	**Hardware Platforms**	**Performance**	**Application**
Parallelism and data reuse	Intra-layer approach[14]	DCNN	Orlando SoC	Speed up: 14.21x	
	Parallelization on 8 cores[35]		PULP-Mr.Wolf	Energy: 83.2Uj Power: 10.4 mW	EEG
	Parallelization on 2 cores[36]	Tensorial SVM	PULP-Mr.Wolf	15× energy savings	E-skin
	Row stationary-Exploiting local data reuse[26]	CNN	Eyeriss Chip	1.4×-2.5× energy savings	Iot Devices
	Pipeline through HLS directives[27][28]	Linear SVM	FPGA	9.9×-speed up	Melanoma detection
		Decision Tree			Character recognition
	OpenCL (parallelism)[29]	KNN	DE5 FPGA	3× energy efficiency	KDD-CUP2004 Quantum physics set
Approximations	Algorithmic level[25], [43]	CNN and RNN	DNPU	4.5×-20×energy saving	
		ConvNet		100x-energy savings	Wearable devices
	Algorithmic, architecture and circuit levels [54]	Neural Networks	ASIC	5% till 87% power savings	CIFAR as benchmark
	Architecture and circuit levels [30],[54]	Neural Network	TSMC 65nm	43.9% till 62.5% energy savings	
		Convolution Network	ASIC-28nm silicone	2-9 TerraOps/w/s	Real-time embedded scene labeling
Network Sparsity	Skipping Sparse operations[31]	ConvNet	Envision Platform	10 TOPS/W-efficiency	Wearable devices
	Energy aware pruning [56]	CNN	ASIC	70%-power reduction	Wearable devices and Smartphones
	Width and Resolution reduction[32]	Mobilenets	ASIC	88% mult-add reduction 1% accuracy degradation	Image processing
	Removing zero operand multiplication [33]	DCNN	ASIC	1.24×-1.55× performance improvement	

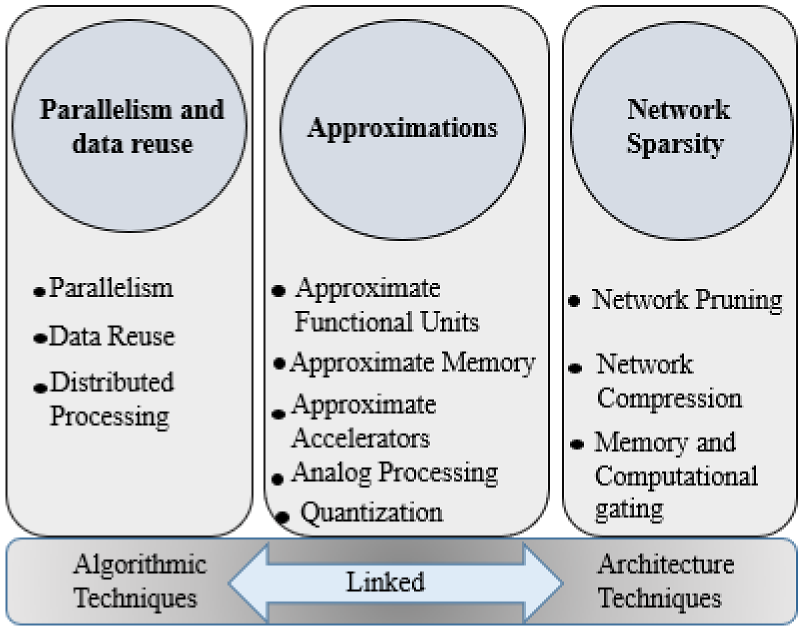


Fig. 1.2. An overview of some energy efficient techniques at algorithmic, architecture and circuit level for embedded Machine and Deep Learning Algorithms

Selecting the appropriate hardware platform can be incorporated with other optimization techniques that could be applied to embedded Machine learning implementations.

Table 1.1 reports some of the most significant energy-efficient techniques employed in the literature at different levels, for embedded ML/DL algorithms on different hardware platforms (FPGA, Parallel platforms, an ASIC). It highlights the use of some relevant techniques by presenting the followed design approach. Then an analysis of the impact of these techniques on different applications is reported. These techniques include (1) Parallelism and data reuse[26],[27],[28],[29] (2) Approximations[30], and (3) Network Sparsity[31],[32],[33] (see fig.1.3). However, the main challenge is to find the optimal way of choosing a technique or a combination of multiple techniques that can be implemented at different abstraction levels.[27] This can further reduce the power/energy consumption while obeying the target application requirements.

1.4.1 Parallelism and Data Reuse

Parallelism is one of the most used solutions to simplify this challenge. For instance, many computations within the same function of the algorithm can share common resources, as shown in [14], [34]and [35]. Authors in [14] took advantage of the multicore architecture offered by Orlando SoC to parallelize the convolution operations on tensors, achieving a speedup of 14.21x. In [34], [35], the PULP platform, an efficient ultra-low-power processor, is used for EMG signal and nano drones, where power consumption and energy efficiency have been reduced to below 64 mW by utilizing the available eight cores in [34]. Also, in [36], relatively low power consumption less than 28 mW and an energy efficiency improvement of 15x have been recorded when parallelizing the tensorial SVM on two cores of the PULP platform.

1.4.2 Approximations

Due to their inherent error resiliency, machine learning methods may scarify a part of their accuracy at the benefit of performance improvement. In this regard, recent researches have relied on approximate computing methods to address the embedded implementation of ML algorithms. Approximate computing methods have been applied at various levels of the system abstraction, i.e. algorithmic, architecture, and circuit levels. Table 1.2 reports some relevant works presented in the literature at different levels [37]. It highlights the used approximate computing technique and the followed design approach with the analysis of the impact of the results on the application. The analysis is basically focused on the gain in performance represented by the power and latency savings on the one hand, and by the accuracy degradation on the other hand. Approximate Computing (AxC) is the idea of a trade-off between accuracy and efficiency. It allows the system to expose inexactness to the application layer of an error-resilient system in return for conserving some resources. Therefore, selecting an AxC technique for an embedded machine learning algorithm requires careful analysis in order to maintain a low accuracy degradation with a noticeable reduction in power consumption.

1.4.2.1 Algorithmic Level

At the algorithmic level[38],[39], approximations are applied to the loops and functions constituting the program/software which describes the application. These approaches enable the approximation by reducing the number of iterations in an iterative algorithm. Focusing on the loops, loop perforation[40] trades accuracy for time latency/power

Table1.2. : Approximate computing methods at different abstraction levels

Level	**Technique**	**Design Approach**	**Application**	**Results (Energy Savings/Latency)**	**Accuracy Degradation**
Algorithmic	Synaptic Pruning and Quantization	Scale computational precision /apply on accelerator circuits [38]	Convolutional Neural Networks	Energy reduction up to 30×	No loss
		Prune the synaptic weights / reduce the bit width of the synapses [54]	DNN	80% energy saving	< 0.2%
	Approximating Networks	Approximate Neural Network by approximating neurons [39]	Neuromorphic Systems	1.14X-1.92X energy savings	< 0.5%
		Reduce the number of hidden layers and the number of neurons [54]	DNN	83.23% energy savings	0.178%
	Approximate processing	Skip reading specific rows in weight matrix of several neurons [6]	ANN	34.11% ~ 51.72% energy savings	< 5%
Architecture	Selective Approximation	QUORA vector processor with approximate processing elements [46]	Programmable processors	1.05X-1.7X energy savings	< 0.5%
	Scalable Hardware	Scaling number of bits in data path between MAC and FIFOs [47]	SVM	1.2X-2.2X energy savings	no loss
	Data Storage Approximation	Perform approximate storage on the unreliable cache sets [48]	MLC STTRAM	7%-19% energy savings	0.22% to 0.43%
Circuit	Approximate Multipliers	simplified shift and add operations [46]	DNN	18% till 27% power savings	< 0.4%
		inexact logic minimization approach [15]	Neural Network	43.9% till 62.5% energy savings	Mean square error from 0.14 to 0.2
	Approximate Adders	alternate circuits for Full Adder [37]	CNN	Reduction the area delay product by 50% for LOA	13%
		divides the p-bit addition(m+n=p)[55]	Neural Network		
	Approximate memory	-hybrid 8T-6T SRAM cell [54]	Deep fully connected network	Reduction of the operating voltage from 0.85 V to 0.8V	< 0.5%
		quality configurable memory array [49] [50]	8 machine- learning benchmarks applications	19.5% energy savings	Accuracy loss than 0.5%
		approximate memory compression [18]		1.28x energy savings 11.5% reduction in execution time	1.5% accuracy loss
	Quantization	substitute the floating point multiplications with lookup table search [51]	Voice recognition	3x energy savings 2.6x improvement in time delay	0.2% accuracy loss
		Lowering the precision of network weights [52]	Deep convolution network	Reduction of the compute requirement by ~3×	7% to 23.4%
		Reduced data precision [45] [53]	CNN and DNN		1%.

consumption by transforming loops to limit the execution of a subset of the iterations. The loop perforation is only applied to the tunable loops producing acceptable accuracy. Data Format Modification is another AxC technique that is widely used. It includes the transition from floating-point to fixed-point representation. For instance, data samples in voice applications are represented using 16-bit precision [41], while a 12-bit precision is often sufficient for image processing applications [42]. Authors in [43] embedded ConvNet in a wearable device while running a benchmark using a 1-9b fixed-point representation. An energy efficiency improvement of 100x with an accuracy loss of 1% has been recorded. As for fixed-point precision-based neural networks, a quantization methodology presented in [44] can be applied to find a suitable representation of each layer of the network [45] in order to maintain an acceptable accuracy.

1.4.2.2 Architecture Level

The goal of approximation at the architectural level [46],[47], is to use relaxed specifications on circuits able to support inexactness during execution and storage. Concerning data storage, a writing mechanism has been proposed recently [18], enabling the approximate data storage. Authors in [40] proposed a mechanism based on trading off accuracy/writing speed in multilevel cell accesses[48]. Approximate storage leverages the properties of a wide range of memories, such as spintronic memories [49] or solid-state memories.

1.4.2.3 Circuit Level

At this level[50],[51],[52],[53],[45] hardware designers focus on designs producing approximate results through synthesizing inexact circuits. Such paradigms have been used mainly to design arithmetic circuits [54]. The use of approximate arithmetic module blocks has been considered as a relevant solution aiming to develop energy-efficient and high-performance machine learning algorithms. For computation, multiplication operations are central arithmetic units characterized by intricate logic design. Therefore, a number of approximate multipliers for machine learning have been proposed in the literature [54],[6],[15], and [55]. In[54], the authors evaluated the use of ASM multiplier in a deep neural network. The conventional multiplication is substituted by simplified shift and add operations. The power consumption has been reduced by 18% to 27% at the cost of accuracy loss less than 0.4%; after carrying out the re-training to the network to compensate for the loss in accuracy added through the approximations. The approach adopted in[54] has been proposed in[15],where the energy efficiency is improved by 43.9% to 62.5% after implementing the inexact multiplier using the inexact logic minimization approach in a neural network. But the mean square error reported has been increased from 0.14 to 0.2. However, the degradation in quality shown is higher (5%) after implementing another architecture of an approximate multiplier in an artificial neural network. Through ApproxANN, the approximation is applied for both computation and memory accesses. The proposed multiplier has a tunable output of (n + k) bits, where n represents the bit-width of input data. The results reported a reduction in terms of power, around 45.9% for the MNIST application having an MP_{24} configuration.

Besides the multiplication operation, addition block is considered as a fundamental block having significant influences over the performances of the system. Therefore, in[37], authors have selected five approximate adders configurations from the approximate IMPACT adder configurations. After evaluating every adder configuration, the results indicate that the average of the accuracy for all the fives adders configurations in a deep CNN architecture based on the LeNet-5 is around 87%. On the other side, the performances could be improved, as shown in[55].Authors have proposed the use of Lower-part-OR Adder(LOA) and BAM multiplier in a neural network for face recognition applications. The architecture Lower-part-OR Adder (LOA) is based on dividing a p-bit addition into two m-bit and n-bit smaller parts (m+n=p) . While the structure of the BAM multiplier is similar to that of an array multiplier. The area delay product of the model has been decreased around 50% after combining the approximate adder and multiplier into the model.

1.4.3 Network Sparsity

For an embedded deep neural network, there is a probability that some weight values are equal to zero. This presents a large sparsity in the network, thus assisting in improving energy efficiency. After skipping the unnecessary sparse operations in [43],[17], the ConvNet processor archives an efficiency up to 10 TOPS/W at the same throughput. Another promising solution towards efficient neural networks is pruning the layers of a CNN with the most power requirements [56]. This method achieved a 70% reduction in power consumption, surpassing the previous efforts done in reducing the model size of a CNN.

1.5 Context of the work

Energy-efficient circuits have become a substantial need for designing embedded computing systems for such application domains as the Internet of Things (IoT), Wearable Devices, and biomedical applications. In the “Prosthetics” application [57], a dedicated portable electronic system is needed for developing wearable devices. Portable prosthetic systems contain autonomous and networked sensors; each sensor hosts multiple input tactile sensors tightly coupled to the embedded processing unit and power supply [58]. The embedded processing unit locally extracts meaningful information by employing sophisticated methods, e.g., Machine Learning [3], which deals with large dimensions of datasets. Nevertheless, this imposes challenges on the real-time operation and adds a burden regarding power consumption. The demands of the electronic-skin are not satisfied since, as shown by [59], the estimated energy/power is not feasible, i.e., 100 pJ/op, time latency (i.e., around 7 s) is very high and the computational load is of about 1.2 GOPS. Therefore, since the implementation of tactile data decoding algorithms for touch modalities classification requires a high amount of power consumption [10],then our primary goal in the thesis is to implement energy-efficient techniques for embedded machine learning algorithms used for tactile data processing in the e-skin .

1.6 Thesis contributions

The contributions in this thesis, are summarized as follow:

- Two signed approximate multipliers have been designed and implemented. The first one is a rounder multiplier called the “META” multiplier [60] and is based on the Error Tolerant Adder (ETA), which has been implemented instead of the exact adder. While the second one is called “Approximate Baugh-Wooley(BW)” multiplier[61] and is based on the architecture of the exact Baugh-Wooley multiplier, where the approximation is enabled after introducing the approximate adder in the computation of the partial products . The proposed circuits have been implemented using RTL description in VHDL Hardware Description Language for Virtex-7 xc7vx485tffg1157-1 FPGA device. 10000 inputs have been uniformly selected and simulated in order to compute the accuracy metrics of the designs. The relative error (RE) and the mean relative error (MRE) metrics have been calculated to assess the performance of the approximate multipliers. Results show that approximate-BW is the most efficient design between the approximate multipliers achieving a relevant reduction in power consumption and time delay around 80.4% and 24% respectively with respect to the exact BW multiplier and an improvement of power consumption reduction by 68.1% with respect to other state of the art solutions.

- Implementing approximate arithmetic circuits into the Coordinate Rotational Digital Computer (CORDIC)[62] algorithm and the SVD[63] circuits in order to reduce the power consumption of ML algorithms; since CORDIC and SVD circuits take part of the real-time ML algorithm for tactile data processing. Approximate CORDIC and approximate SVD have been implemented using RTL description in VHDL Hardware Description Language for Virtex-7 xc7vx485tffg1157-1 FPGA device. ETA and LOA have been implemented instead of the entirely precise adder (RCA) into the CORDIC algorithm aiming to study the performance of CORDIC in terms of slices, power and time latency after employing approximate circuits. ETA and LOA in the CORDIC design allow respectively a dynamic power consumption saving up to 13% and 21% with respect to CORDIC-RCA. While for the (SVD) singular Value Decomposition, the Approximate BW multiplier has been implemented in Post rotation and Pre rotation blocks of the SVD after scaling the approximation in the multiplier by increasing the number of the approximate bits from (8 to 28). Results show that the power consumption of the SVD is reduced by 19% with a negligible accuracy loss.

- Implementing an energy-efficient smart system for tactile sensing based on a RISC-V parallel ultra-low-power platform (PULP). The PULP processor, called Mr. Wolf[64], performs the on-board classification of different touch modalities. This demonstrates the promising use of on-board classification for emerging robot and prosthetic applications. Experimental results demonstrate the effectiveness of the platform on improving energy efficiency and the accuracy of the classification. A memory management strategy has been proposed in order to store 50 trained for each class in L2 memory (512 kB). We evaluated the execution cycles for Mr. Wolf using a single core, 2 cores, and 3 cores to evaluate the benefits of the parallelization. The three SVDs blocks have been executed in parallel on three different cores on the Wolf SoC. A 3.72× speed-up can be achieved after executing SVD (A) ,SVD (B), and SVD(C) blocks on three different cores in parallel. We demonstrated that the algorithm on the proposed platform outperforms ARM Cortex M4F (STM32F40) and by 15 times in terms of energy efficiency, without exceeding the power envelope of a 28mW.

1.7 Organization of the thesis document

The thesis is organized as follows. Chapter 2 describes the architecture of the exact and approximate adders and multipliers presented in state of the art. Then, it presents the architectures of the two new proposed approximate multipliers (META and approximate Baugh-Wooley multipliers). After, an assessment study of the new approximate multipliers has been done in terms of performance, accuracy, delay, area, and power consumption with respect to state of the art.

Chapter 3 assesses the impact of the new approximate multipliers designs as well as some of the most relevant state of the art approximate multipliers on tactile digital signal processing. The quality is measured in terms of different metrics, mainly: SNR degradation, power consumption, and time delay. On the other hand, approximate computing techniques have been applied on the machine learning algorithm employed for the tactile data processing; by implementing approximate adders and multipliers into the Coordinate Rotational Digital Circuits (CORDIC) and the Singular Value Decomposition (SVD) algorithms which take a significant part of the real-time ML algorithm.

Chapter 4 presents the machine learning algorithms employed for tensorial tactile data processing. Moreover, it discusses the reasons behind following the mentioned approach. Then, the SVM based tensor kernel algorithm is implemented in C language in order to validate the effectiveness of the algorithm when classifying images (as a case study) and touch modalities.

Chapter 5 describes the hardware implementation of the SVM based tensor kernel approach on a novel computing platform embedding a Parallel Ultra Low Power processor (PULP), called “Mr. Wolf” for the aim to reach an embedded low power implementation for wearable devices. The classification based on Support Vector Machine (SVM) runs directly on PULP classifying two touch modalities (finger sliding and washer rolling) outperforming ARM Cortex M4 in terms of power consumption and energy efficiency.

CHAPTER 2. LOW POWER APPROXIMATE ARITHMETIC CIRCUITS.

2.1 Introduction

Energy-efficiency has become a paramount concern in designing computing systems. The ever-increasing demand for higher computing power represents a driving force toward ultra-low power design strategies. Low power consumption is the most critical design goal for a wide range of electronic systems, including smart self-powered sensing systems for such application domains as the Internet of Things (IoT), Wearable Devices and Robotics. To improve energy efficiency, at different layers of the system stack, researchers have developed different optimizations methods.

In recent years, several techniques at the circuit and system level have been proposed to address this issue. One of these techniques is “approximate computing,” which turned out to be a practical approach providing a tradeoff between accuracy and power saving to improve performance and energy efficiency [1],[2]. Approximate results are usually acceptable for many applications requiring tactile data processing [3], [4] image processing [5], and data mining [6]. Thus, it is highly recommended to take advantage of energy reduction with minimal variation in performance [7]. Recently, approximations have been used in computing units of embedded systems, especially graphics processing units (GPUs) and field-programmable arrays (FPGAs) [65]. Computing units, e.g., embedded digital signal processing (DSP) systems, are considered critical components of modern electronic embedded devices [55]. Among arithmetic DSP operations, multiplication is considered as a complex block consuming a high amount of power with significant time latency when compared to other operations. Decreasing the complexity of multiplication blocks may reduce the power consumption of DSP systems. In this perspective, the proposed work employs approximate computing techniques for the arithmetic units, i.e., adders and multipliers for energy-efficient data processing units. In this chapter, we present an overview of entirely precise and approximate adders and multipliers circuits. Then we describe the architectures of the two new proposed approximate multipliers (META and approximate Baugh-Wooley multiplier). After, an assessment study of the proposed approximate multipliers has been done in terms of performance, accuracy, delay, area, and power consumption with respect to state of the art. Therefore, the main goal is to implement efficient hardware architectures of approximate multipliers providing low power consumption. The results presented in this chapter were published in [60],[61],[130].

2.2 Background on Adders and Multipliers

This section presents the architecture of some relevant exact arithmetic circuits in state of the art.

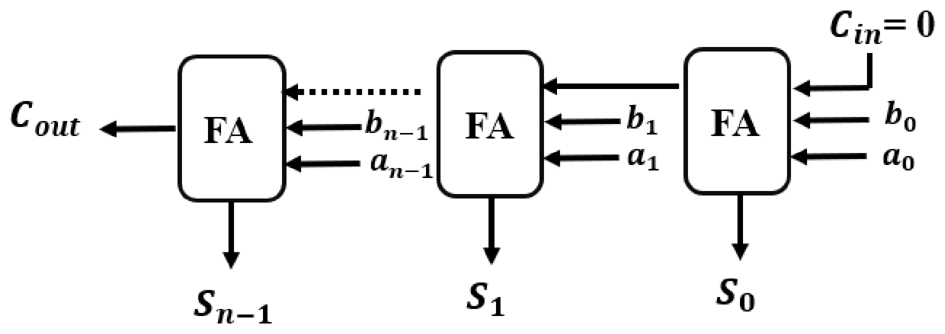


Fig.2.1. Ripple-carry adder

2.2.1 Representation of signed integers

In this section, we will discuss the representation of signed integers (positive and negative). Usually, two representations are presented such as: the sign and magnitude representation and the true-and-complement representation.

2.2.1.1 Sign-and-Magnitude (SM) Representation

In the SM system, the signed integer x is represented by a pair of (a_1, a_m), where $a_1$is the sign and a_m is the magnitude. The values of two sign $(+, -)$ are usually represented by a binary variable; where the integer 1 corresponds to – and 0 corresponds to $+$. When representing the positive integers, the magnitude could be represented in any system. In case of conventional radix-r system, the range of signed integers is presented as equation below:

$$0 \leq a_m \leq r^n - 1 \tag{2.1}$$

2.2.1.2 True-and -Complement (TC) Representation

No separation is applied between the representation of the sign and the magnitude in the TC system. But in this system, all the signed integer is represented by a positive integer. Therefore, the signed integer x is represented by positive integer called a_R which is expressed as below:

$$a_R = a \ mod \ C \tag{2.2}$$

Where the positive integer C is called the complementation constant. For max$|a| < C,$ the following system could be derived:

$$a_R = \begin{cases} a & \text{if } a \geq 0 \\ C - |a| = C + a & \text{if } a \geq 0 \end{cases} \tag{2.3}$$

However, the region for $a > 0$ should not overlap with the region for $a < 0$. This requires that:

$$\max|a| < \frac{C}{2} \tag{2.4}$$

In this case, the following system could be derived:

$$a = \begin{cases} a_R & a_R < C/2 \\ a_R - C & a_R \geq 0 \end{cases} \tag{2.5}$$

When $a_R = C/2$ is representable, it is usually assigned to $a = -C/2$, making the representation asymmetrical. So, the true forms indication corresponds to the positive integer's representation, while the complement forms correspond to negative integer's representation.

2.2.2 Basic Adders

2.2.2.1 Ripple-Carry Adder

Ripple-Carry Adder (RCA) [66] is a well-known circuit used to compute the addition of two binary numbers in many arithmetic circuits. RCA adds sequentially the bits having the same significance and the carry-bit from the previous stage using a full adder (FA), then propagates the carry-bit to the following stage, as shown in Fig. 2.1. This adder could be employed for adding both unsigned and two's complement numbers. However, the main drawback of RCA is that the worse-case delay is intended to be proportional to the word length. Moreover, since the full adder cells are supposed to wait for the correct carry input, thus many glitches will be produced from the RCA. Nevertheless, this drawback could be improved if the delay of the carry bit is smaller than that of the sum bit.

2.2.2.2 Carry-Chain Adder

Fig.2.2. shows the structure of an *n*-digit adder having a separate carry calculation. The *generate* and the *propagate* functions are calculated through the G-P (generate-propagate) cell based on the following equations:

$$g(i) = \begin{cases} 1 & \text{if } a(i) + b(i) > B - 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

$$p(i) = \begin{cases} 1 & \text{if } a(i) + b(i) = B - 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

Where $a(i)$ and $b(i)$ are the inputs signals, and B represents the Base.

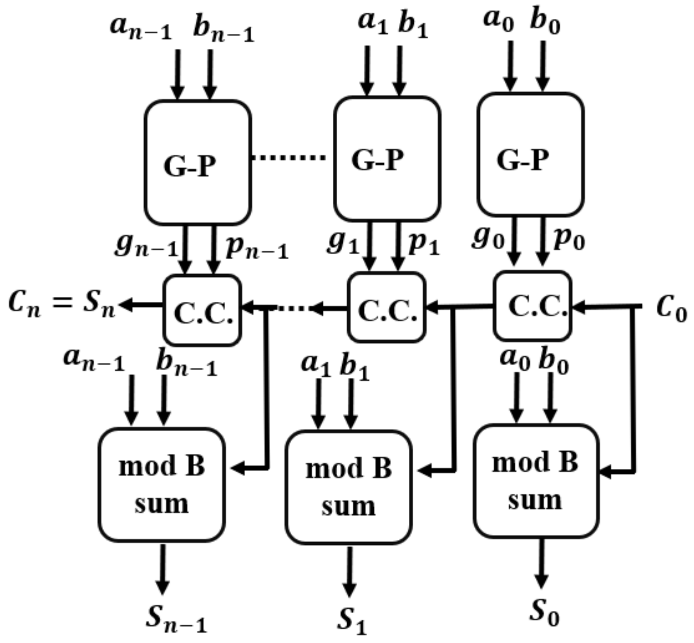


Fig.2.2. Carry-chain adder

The next carry is computed through the C.C. (carry-chain) cell as follow:

$$C(i+1) = \begin{cases} C(i) & \text{if } g(i) = 1 \\ g(i) & \text{otherwise} \end{cases} \tag{2.8}$$

Then a carry is generated from $g(i)$ and the carry is propagated from level *i-1* by $p(i)$.Then the sum will be generated through the $mod\ B$ sum cell as follow:

$$S(i) = \big(a(i) + b(i) + C(i)\big) mod\ B \tag{2.9}$$

2.2.2.3 Carry-Lookahead Adder (CLA)

The main concept of this adder is to compute simultaneously several carries. In the extreme, the computation of the carries is done at the same time. Let's consider that $a^{(i)}$and $b^{(i)}$the integers represented by the bit-vector from bit 0 to bit i as follow:

$$a^{(i)} = \sum_{v=0}^{i} a_v 2^v \tag{2.10}$$

and similarly, for $b^{(i)}$, the carry is computed as follow:

$$c_i = 1 \text{ if } (a^{(i-1)} + b^{(i-1)} + c_0 \geq 2^i \tag{2.11}$$

Then, a switching function of $2i+1$ variables will be resulted. This function could be implemented by a two-level network such as: $NAND - NAND$.In case of large i, this implementation is complex due to the large number of gates and inputs. Therefore, the input vector in the CLA is divided into two groups, where the carries are computed simultaneously.

2.2.3 Multiplication

Multiplication of two numbers is usually executed by following the two main steps:

a) Generation of partial products where the partial product is the result of multiplication of the multiplicand with a bit of the multiplier.

b) Partial products accumulation

In this section, we will discuss some techniques to simplify the summation of the partial product, and we will describe the architecture of some exact multipliers.

2.2.3.1 Partial Product Generation

For unsigned number format, the multiplication is done based on the following equation:

$$P = AB = \sum_{i=1}^{n} a_i\, 2^{-i} \sum_{j=1}^{n} b_j\, 2^{-j} \tag{2.12}$$

A partial product array is generated, as shown in Fig. 2.3., where partial products are applied through AND gates. In the case of 2's complement representation as shown in

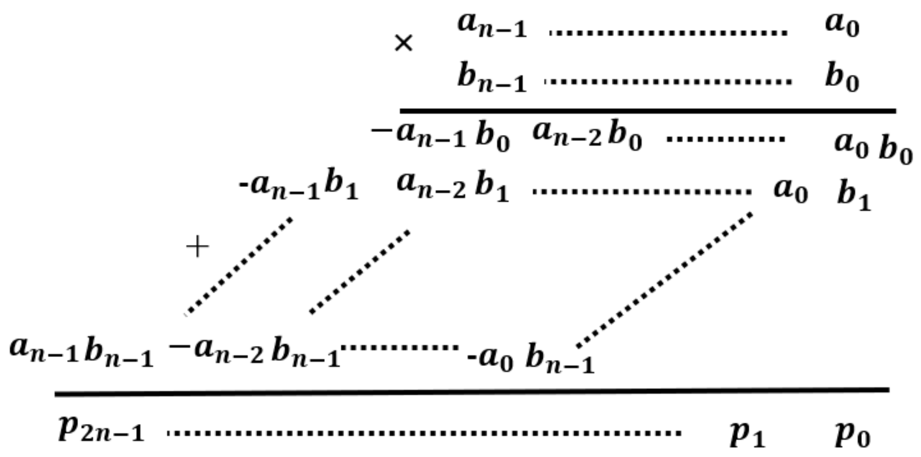

$\times$
a_{n-1} .... a_0
b_{n-1} .... b_0
$-a_{n-1}b_0$ $a_{n-2}b_0$ .... $a_0 b_0$
$-a_{n-1}b_1$ $a_{n-2}b_1$ .... a_0 b_1
$+$
$a_{n-1}b_{n-1}$ $-a_{n-2}b_{n-1}$ .... $-a_0 b_{n-1}$
p_{2n-1} .... p_1 p_0

Fig. 2.3. Partial products for unsigned numbers

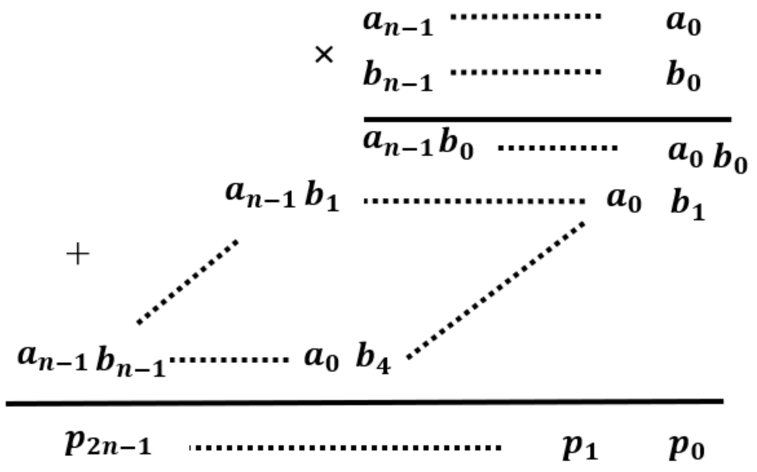


Fig.2.4 Partial products for two's complement numbers

Fig.2.4, the equation is similar to (2.10) except that some bits will have a negative weight. The equation is shown below:

$$P = AB = a_0 b_0 - a_0 \sum_{j=1}^{n} b_j\, 2^{-j} - b_0 \sum_{i=1}^{n} a_i\, 2^{-i} + \sum_{i=1}^{n} \sum_{j=1}^{n} a_i\, b_j 2^{-i-j} \quad (2.13)$$

2.2.3.2 <u>DADDA Multipliers</u>

DADDA multipliers [67] are considered as the remake design of the parallel multipliers presented by Wallace in 1964 [68]. As shown in Fig. 2.5, the multiplier is composed of three stages, wherein the first stage a partial product by N^2AND gates are executed. In the second stage, the height of the partial product matrix is reduced to two, which employs different parallel (m, n) counters. The parallel counter in DADDA multiplier has m inputs providing n outputs. During the compression phase, DADDA multiplier employs at least (3,2) and (2,2) counters at each level, where a (3,2) and (2,2) counters represent respectively a full adder and a half adder. In Fig.2.5, the outputs of the (3,2) counter are

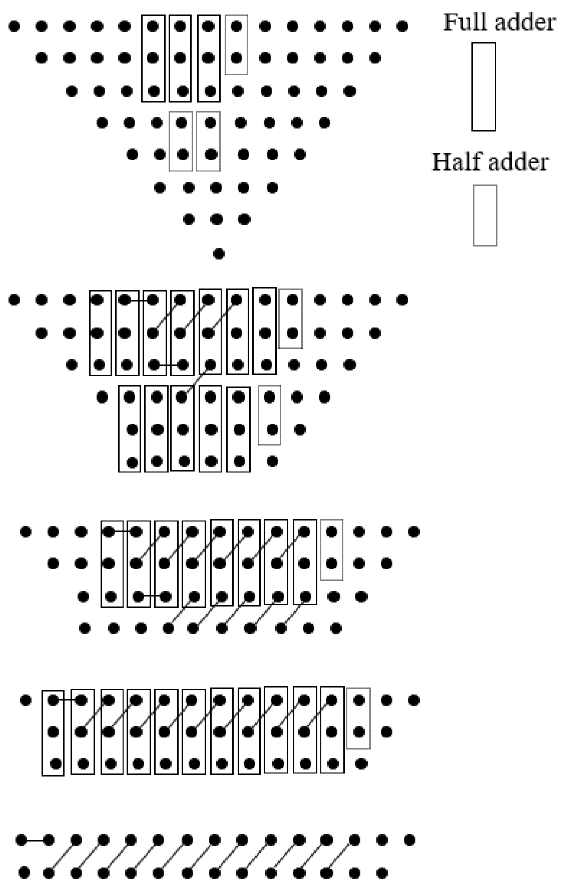


Fig. 2.5. Dot diagram for an 8×8 DADDA multiplier

represented by a diagonal line joining the two squares, while the crossed diagonal line joining the other two squares are the outputs of the (2,2) counter. In order to produce a 16-bit product multiplier, different components are required such as: 64 AND gates, 35 (3,2) counters, 7 (2,2) counters and a 14-bit carry propagation adder. During the compression stage, DADDA multipliers require less counters than Wallace multipliers. Lastly, the final stage of the multiplier uses a carry propagation adder in order to generate the final product.

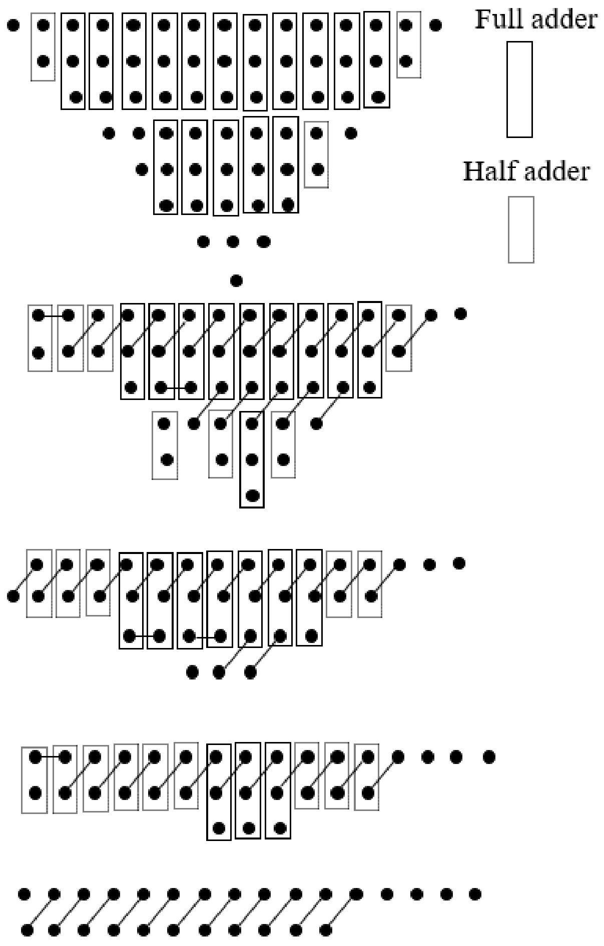


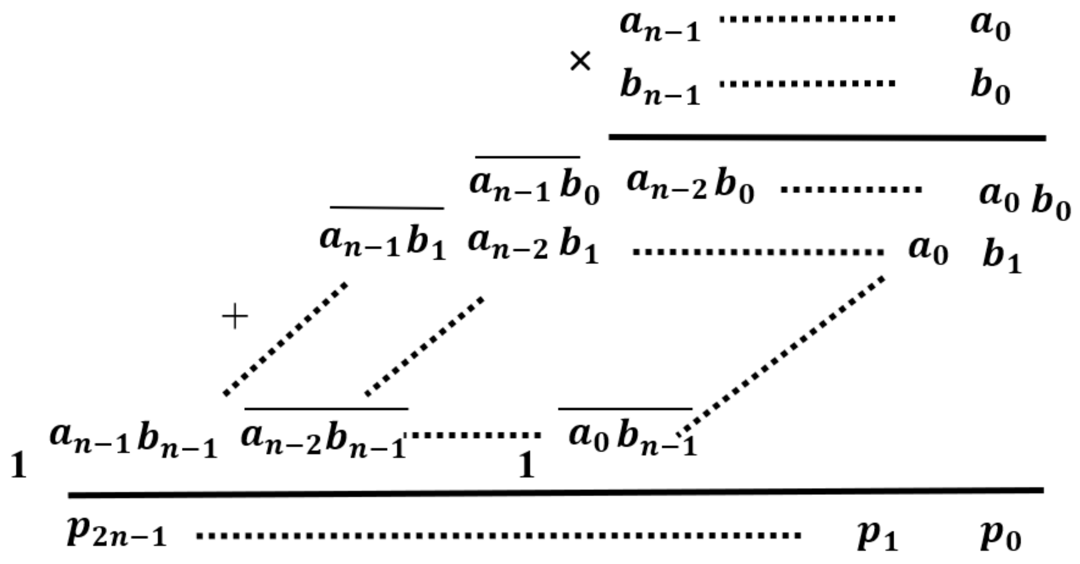
Fig. 2.6. Dot diagram for an 8×8 WALLACE Multiplier

2.2.3.3 WALLACE Multipliers

Similarly to DADDA multiplier, the partial products in WALLACE [68] multiplier are produced through N^2 AND gates as shown in Fig.2.6. Then a set of three rows are grouped together containing N rows of partial products. While each row which is not included in the set of the three rows is transferred to the next phase without applying any change. Then (3,2) and (2,2) counters are applied to columns containing three bits and columns containing two bits, respectively. Nevertheless, each column that contains a single bit will be transferred to the next level without any modification. In WALLACE multipliers, a carry propagating adder is employed for the execution of the final addition whose sum is the product of the final multiplication. During the reduction phase, the WALLACE multiplier has approximately the same numbers of full adders, similarly to the DADDA multiplier. Then a shorter final carry propagating adder is generated after adding half adders to the previous phase. The components required to produce a 16-bit product DADDA multiplier are as follow: 64 AND gates, 1 OR gate, 38 (3,2) counters, 15 (2,2) counters, and a 10-bit carry propagating adder.

2.2.3.4 Baugh-Wooley Multiplier

Baugh-Wooley [69] is a well-known algorithm used to compute the multiplication in many digital signal processing units. It is a signed array multiplier considered in our case as an exact reference multiplier to be compared with the proposed approximate ones [70],



a_{n-1} ……… a_0
$\times$ b_{n-1} ……… b_0
$\overline{a_{n-1} b_0}$ $a_{n-2} b_0$ ……… $a_0 b_0$
$\overline{a_{n-1} b_1}$ $a_{n-2} b_1$ ……… a_0 b_1
$+$
1 $a_{n-1} b_{n-1}$ $\overline{a_{n-2} b_{n-1}}$ ……… 1 $\overline{a_0 b_{n-1}}$
p_{2n-1} ……… p_1 p_0

Fig.2.7. Baugh-Wooley multiplier.

[69]. Baugh-Wooley is designed to compute the multiplication of both signed and unsigned operands using 2's complement number system. During direct multiplication of two 2's complement numbers, the partial products obtained will be signed numbers. Hence sign extension is needed for these partial products to the final product's width to get the accurate answer. This multiplier is very easy to implement since it has a regular architecture as shown in fig.2.7. Moreover, Baugh Wooley multiplication table can be implemented using different full adders such as: carry save array, ripple carry adder or carry select adder. In case of 4×4 multiplication, 3 rows of adders and a final stage adder are needed for the computation, where the partial products are obtained using AND and NAND gates.

2.3 Background on approximate adders and multipliers

In this section, some significant approximate arithmetic circuits (adders and multipliers) are described and presented.

2.3.1 Approximate Adders

In [71], an approximate adder based on the dynamic segmentation with the error compensation technique (DSEC) is proposed. The n-bit adder is divided into smaller sub-adders operating in parallel with fixed carry inputs. This technique reduces 30% power consumption. Authors in [72] described an n-bit Carry Skip Adder (CSA) which is divided into [n/k] blocks. Each block is made of a sub-carry generator and a sub-adder. The power consumption is reduced by 43% when compared to exact adders.

In contrast to SCSA, the speculative carry adder (CSPA) presented in [73] is composed of one sum generator, two internal carry generators, and one carry predictor for each block. The energy efficiency and time delay are improved respectively by 19.03%

and 26.59% with respect to the existing speculative carry-select adder. In [74], the Gracefully-Degrading accuracy-configurable adder (GDA) is presented. Through the control signals, the accuracy of GDA is configured by selecting the approximate of exact carry-in for each sub-adder through a multiplexer. This advanced design achieves a better quality when compared to existing techniques. The consistent Carry Approximate Adder (CCA) based on SCSA, is proposed in [75]: the carry prediction depends on the least significant bit (LSBs) and the MSBs. The time delay and the area are similar to SCSA. Authors in [55] proposed the Lower-Part-OR Adder (LOA), which is based on processing the least significant bits using OR gates. In [76], Approximate Mirror Adders (AMAs) are proposed. The AMAs are implemented in the LSBs of a multiple-bit adder achieving a reduction in power consumption by up to 69% when compared to accurate adders. Authors in [77] proposed three approximate adders (AXAs) based on XOR and XNOR logic gates consuming less power than the exact XOR/XNOR-based adder.

2.3.2 Approximate Multipliers

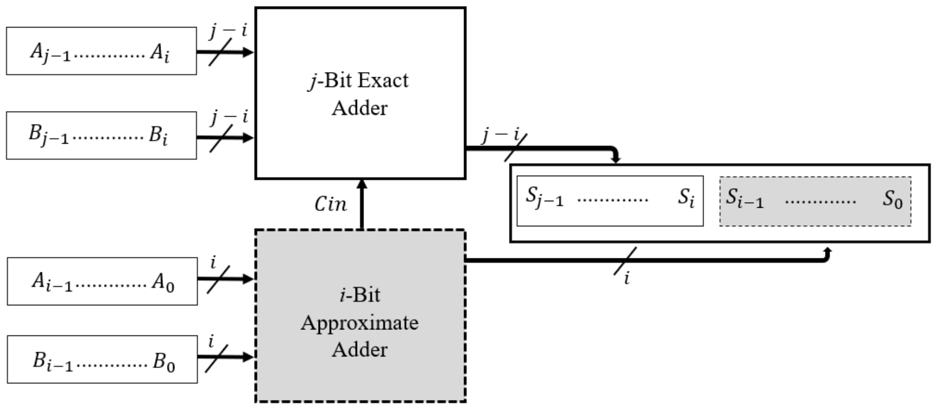
Efficient implementations of approximate multipliers based on different approaches have been recently reported in the literature. Kulkarni et al. [78] proposed an approximate 2×2 multiplier cell, which is employed as a basic block for multiplier architectures having a larger size. An average of 31.8% improves energy efficiency to 45.4% with respect to exact multipliers. Authors in [79] have presented an accuracy-configurable multiplier architecture (ACMA) for error-resilient designs. This architecture is based on a technique called carry-in prediction, employing an efficient precomputation logic, which reduces the latency to around 50% when compared to an accurate multiplier. [80] presented an Approximate Wallace Tree Multiplier (AWTM), which employs a carry-in prediction reducing the power consumption by 42% with respect to the Wallace tree multiplier (WTM). The Partial Product Perforation technique (PPP) presented in [81] is based on neglecting a specific number of partial products, reducing the power consumption and time delay around 50% and 35%, respectively, when compared to an exact design. [82] proposed an approximate (4:2) counter for an approximate 4-bit Wallace multiplier. This inexact multiplier is employed in order to build more massive multipliers having error detection and correction circuits. The power consumption is reduced by 10.7% when compared to the Wallace tree multiplier. Two approximate 4:2 compressors have been proposed in [83], providing efficient reductions in power consumption, hardware resources, and delay with respect to exact designs. Authors in [84] proposed a static segment multiplier (SSM), which takes m segment bits from n-bit operand based on leading 1 bit of the operands. Then, instead of $n \times n$ multiplication, the $m \times m$ multiplication is executed, where ($m<n$). It consumes 58% less energy when compared to a precise multiplier with an average computational error of around 1%. In [70], a new multiplier is proposed, which connects the most significant bit (MSB) to the least significant (LSB), generating an error value of 1. Sekanina in [85] provided an open-source library of approximate adders and multipliers called EvoApprox8b library. Authors in [70] have proposed an efficient multiplier based on the rounding approach. Speed and power consumption have been reduced after omitting the computational part of the multiplication.

2.4 Low Power Approximate Adders and Multipliers

In this section, the relevant state of the art approximate arithmetic circuits, i.e., approximate adders and multipliers, are reported, and novel approximate circuits are proposed.

2.4.1 Approximate Adders Circuits

The full adders consume a high amount of power such as the CLA while other are slow(i.e.: RCA). Thus, approximate adders have been proposed in the literature [77], [55],[86], trading the accuracy for a reduction in power, time delay, and hardware resources. Fig.2.8 represents the general circuit architecture for approximate adder circuits. It divides the addition operation into two blocks: 1) Exact and 2) Approximate. The exact operation deals with the most significant bits of the addition while the approximate one oversees the least significant bits. The error is decreased by using a carry-in signal from approximate to the exact block. The architecture provides the possibility to scale the number of bits in the exact and inexact parts. Consequently, the effect of varying the inexactness represented by the scalability of the number of bits with respect to accuracy could be analyzed. Based on the general architecture, the following subsections introduce the proposed adder with some relevant circuit from state of the art.



A_{j-1} .... A_i
$j-i$
B_{j-1} .... B_i
$j-i$
j-Bit Exact Adder
$j-i$
Cin
S_{j-1} .... S_i
S_{i-1} .... S_0
A_{i-1} .... A_0
i
B_{i-1} .... B_0
i
i-Bit Approximate Adder
i

Fig. 2.8. General hardware architecture for approximate adder circuits

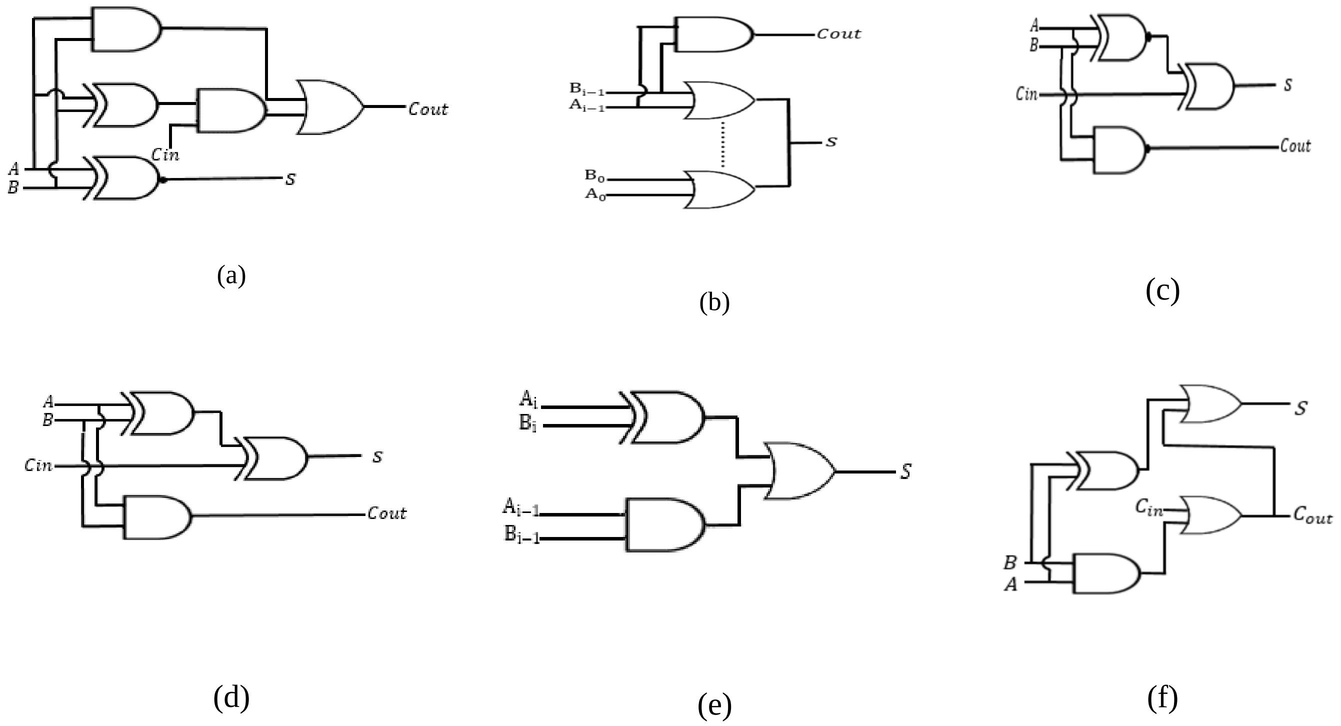


Fig. 2.9. Approximate adder circuits: (a) AXA, (b) Lower-Part-Or-Adder, (c) Approximate NAND-carry out bit, (d) Approximate AND-carry out bit, (e) Input Pre-Processing, (f) AFA adder

2.4.1.1 Approximate XNOR-based Adder (AXA)

Fig.2.9 (a) shows the functionally equivalent remakes design of the original approximate XNOR-based Adder (AXA) [77]. The signal is generated after applying an XNOR logic operation for the two input bits A and B. The sum signal is correct for half of the possible input combinations, C_{out} is exact for all the input combinations, and it is generated according to the following equation:

$$S = (A\ AND\ B)\ OR\ (A\ XOR\ B)\ AND\ C_{in} \tag{2.14}$$

2.4.1.2 Lower-Part-OR-Adder

In the approximate part of the lower part OR adder (LOA) [55] presented by Fig.2.9 (b), OR gates are applied to the respective A and B input bits. An AND gate is applied to the most significant bits of both A and B generating the carry out (C_{out}) signal. In order to decrease the inaccuracy, C_{out} is connected to the upper part of the adder (exact operation) when the most significant bits are equal to one.

2.4.1.3 Approximate NAND-carry out bit

As shown in Fig.2.9 (c) [86], the approximation is applied to the carry-out signal by omitting the two min-terns in the regular expression of Cout as given by (2.14). Then the approximation is expressed by executing a NAND logic operation between A and B. The sum signal is generated at the end by applying the following operation:

$$S = (A\ XNOR\ B) XOR\ C_{in} \tag{2.15}$$

2.4.1.4 Approximate AND-carry out bit

Similar to the approximate NAND-carry out bit [86], this adder adopts an AND gate operation for both inputs A and B generating the approximate C_{out} signal, as shown in Fig.2.9 (d). The signal sum performs the following operation:

$$S = (A\ XOR\ B)\ XOR\ C_{in} \tag{2.16}$$

2.4.1.5 Input Pre-Processing

The Input Pre-Processing approximate adder is presented in Fig. 2.9 (e). It is based on interchanging the bits having the same weights in different addends. The original adder has been proposed in [87]; it generates two signals: sum and error. However, only the sum signal is taken into consideration here. The sum signal resulted is based on the following logic operation:

$$S = (A_i\ XOR\ B_i)\ OR\ (A_{i-1}\ AND\ B_{i-1}) \tag{2.17}$$

2.4.1.6 New Approximate Adder (AFA)

The truth table of AFA adder is illustrated in Table 2.1. Moreover, the equations of the approximate carry bit and the approximate sum bit are shown below:

$$C_{out} = C_{in} + A.B \tag{2.18}$$

$$S = C_{out} + (A \oplus \mathrm{B}) \tag{2.19}$$

Table 2.1. Truth table for AFA Adder

Inputs			**Exact Outputs**		**Approximate Outputs**	
A	**B**	**C**	**C_{out}**	**S**	**C_{out}**	**S**
0	0	0	0	0	0	0
0	0	1	0	1	**1**	1
0	1	0	0	1	0	1
0	1	1	1	0	1	**1**
1	0	0	0	1	0	1
1	0	1	1	0	1	**1**
1	1	0	1	0	1	**1**
1	1	1	1	1	1	1

The proposed approximate adder, called (AFA) [61], is a promising solution to the problem mentioned previously. In the proposed adder, only one case from the eight outputs in the carry bit is erroneous while three errors take place in the sum bit of the adder output. The carry bit is implemented using only two gates OR, AND instead of three logic gates. The sum bit is obtained based on Cout through an OR and XOR operations as shown in Fig.2.9 (f). In this adder, we focused on reducing the erogenous to the carry bit, since the carry is placed in the MSBs. Hence, reducing the inaccuracy of carry bit will decrease the inaccuracy of the output of the approximate adder. Therefore, maintaining the accuracy to the carry bit more than the sum bit will be considered as an essential criterion aiming to assure the best possible performance to the approximate adder .

2.4.2 Approximate Multipliers Architectures

In this section, we present the architecture of our two new approximate multipliers called Approximate Baugh-Wooley (Approx-BW) and Approximate multiplier based on ETA adder (META). The main idea of our multipliers stems from the fact that each multiplier needs an addition operation to perform the multiplication. Thus, the approximation is achieved by using approximate adders. Furthermore, we have generated three versions of the Approx-BW multiplier and three other versions of the META multiplier based on distinct approximate adders. The objective is to select the most effective multiplier in terms of power, latency, LUT utilization and accuracy for our e-skin application after comparing the performance of the proposed approximate multipliers with respect to the previous approximate arithmetic circuits mentioned in state of the art.

2.4.2.1 Approximate META Multiplier

The proposed architecture has been adopted from [70]: it is based on rounding signed and unsigned numbers in the form of 2^n. The main idea is to make use of an approximate adder in place of the exact one in order to reduce power consumption. Before elaborating in the operation of the approximate multiplier, we consider that M_r and N_r are the rounded number of the inputs M and N. The multiplication of the two-input values M and N is written as follows:

$$M \times N = (M_r - M) \times (N_r - N) + M_r \times N + N_r \times M - M_r \times N_r \qquad (2.20)$$

This equation is simplified by eliminating the first part, i.e. $(M_r - M) \times (N_r - N)$, thus the operation is performed using only add/shift operations. Through this approach, the nearest values for M and N are determined in the form of 2^n.When the input (M or N) is

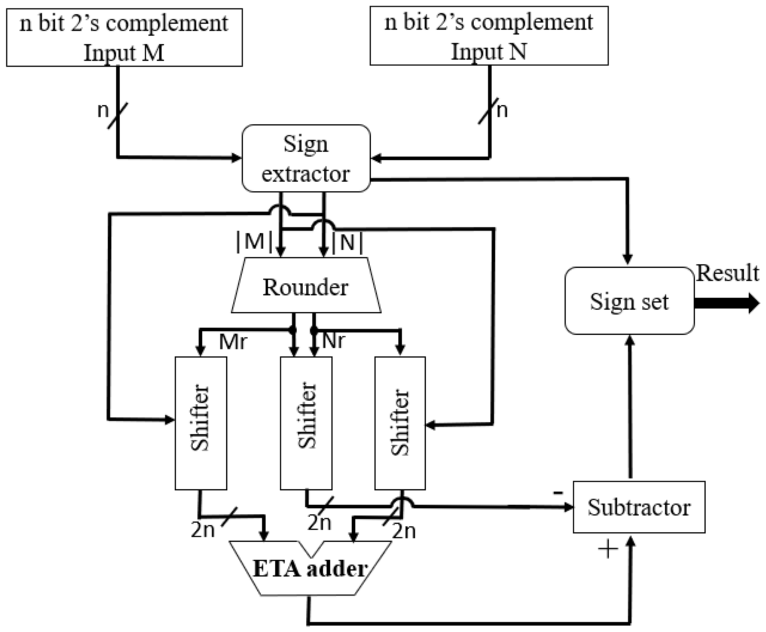


Fig.2.10. Block diagram of approximate META multiplier.

equal to $3 \times 2^{i-2}$ (where I is considered as a positive integer), then the values (M and N) are equal to the two nearest values ($2^{i}$and 2^{i-1}) which are based on the 2^{n} form . Since, both values delivered the same accuracy for the approximate multiplier (except for $i = 2$); then selecting the larger nearest value leads to a smaller hardware implementation. The reason is that the numbers in the form of $3 \times 2^{i-2}$ are considered as do not care when rounding up and down. Only for number three, two is the nearest value in the approximate multiplier in this case.

Moreover, it should be noted that if one of the inputs (i.e.,: M) is smaller than his rounded value (M_r), while the other input (i.e.,: N) is larger than its rounded value (N_r); thus resulting an approximate result larger than the exact result. On the other side, if the both inputs are larger or smaller than their corresponding rounded values, then the approximate result will be smaller than the exact result.

The advantage of the proposed multiplier exists only for positive inputs since the rounded values of negative inputs are not in the form of 2^{n} in the two's complement representation. Hence, the absolute value of both input and output should be determined and then the operation will be performed on unsigned numbers. In the last stage, the proper sign will be applied to unsigned result. The architecture of the proposed multiplier is presented in Fig. 2.10. The different blocks of the architecture are described as follows:

a) Sign Extractor:

The sign extractor block extracts the sign of the input values and gives as output their absolute value. It detects the sign bit (most significant bit) of the input represented in two's complement format. Then, it reverses the input in case of negative values and keeps it unchanged for the positive ones, as follow:

$$\text{If } M[n-1] = 1$$

$$\text{Then} \tag{2.21}$$

$$M = \overline{M}$$

b) Round/Shift

This block applies rounding to the absolute values by extracting the nearest value for each absolute value. Output values are extracted in the form of 2^n following the rounding process. In order to determine the output of each bit of the rounding block, we use the following equation:

$$M_r[n-1] = \overline{M[n-1]}.M[n-2].M[n-3] + M[n-1].\overline{M[n-2]}$$
$$M_r[n-2] = \left(\overline{M[n-2]}.M[n-3].M[n-4] + M[n-2].\overline{M[n-3]}\right).\overline{M[n-1]}$$
.
$$M_r[p] = \left(\overline{M[p]}.M[p-1].M[p-2] + M[p].\overline{M[p-1]}\right).\prod_{p=p+1}^{n-1}\overline{M[p]}$$
.
$$M_r[3] = \left(\overline{M[3]}.M[2].M[1] + M[3].\overline{M[2]}\right).\prod_{p=4}^{n-1}\overline{M[p]}$$
$$M_r[2] = M[2].\overline{M[1]}.\prod_{p=3}^{n-1}\overline{M[p]}$$
$$M_r[1] = M[1].\prod_{p=2}^{n-1}\overline{M[p]}$$
$$M_r[0] = M[0].\prod_{p=1}^{n-1}\overline{M[p]} \tag{2.22}$$

To summarize, along the process, each rounded bit could be equal to one in the following cases:

- When the two right-side bits of the input bit M[i] is one, M[i] and all the bits on its left side are zero.
- When the right-side bit of M[i] and all its left-side bits are zero, while M[i] is one.

Since the rounded values are represented in the form of 2^n, the products $M_r \times N_r$, $M_r \times N$ and $N_r \times$ M are obtained through a shifter. The products of n bit width are shifted based on $Log_2^{M_r}$ or $Log_2^{N_r}$depending on the operand M or N, respectively. The output bit widths generated from the shifter block are 2n

c) ETA Adder

Exploiting inexact adders instead of exact ones in the multiplier block may significantly reduce the power consumption of the multiplier circuit. In exact adder circuits, the carry

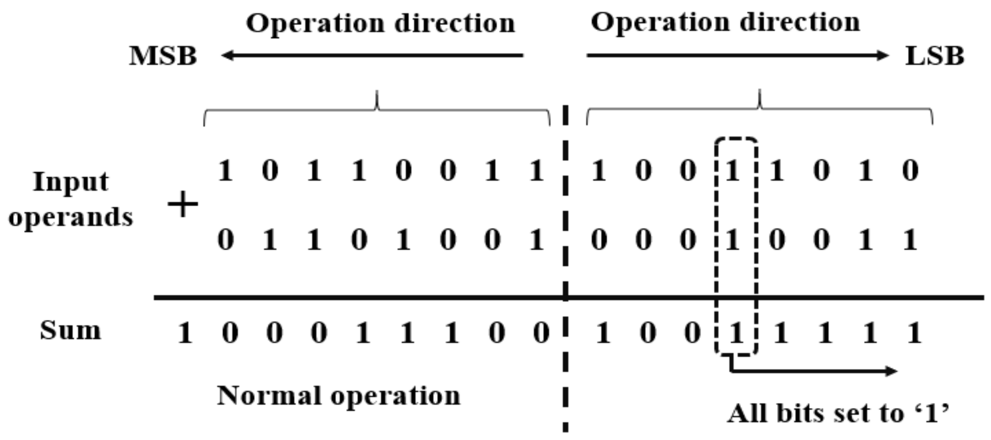


Fig.2.11. Arithmetic addition based on ETA adder

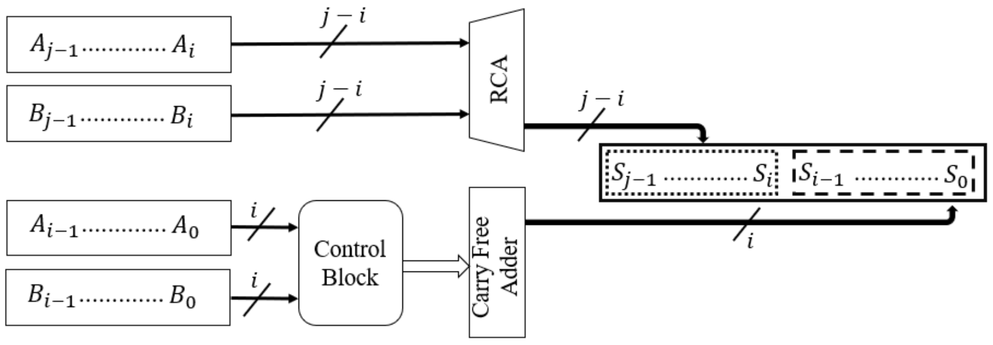

A_{j-1}............ A_i
B_{j-1}............ B_i
A_{i-1}............ A_0
B_{i-1}............ B_0
$j-i$
$j-i$
$j-i$
i
i
i
RCA
Control Block
Carry Free Adder
S_{j-1} .... S_i
S_{i-1} .... S_0

Fig.2.12. Block diagram of ETA adder

propagation chain along the critical path is the leading cause of the delay affecting time latency. By avoiding carry propagation, performance, and power consumption of the adder circuit will be improved at the cost of some accuracy loss. The error-tolerant adder (ETA) could be considered as a solution to the problem mentioned previously [88]. The arithmetic addition based on ETA adder can be illustrated in an example in Fig.2.11. Moreover, Fig. 2.11 shows the functional block diagram of the ETA adder, which is divided into two parts: accurate and inaccurate. The length of each part is specified depending on the requirement of accuracy and power consumption of each application. In our case, the length is divided equally for each part, 8 bits for the accurate part (A_{j-1}…. A_i, B_{j-1}….B_i), and the other 8 bits for the inaccurate one (A_{i-1}…. A_0, B_{i-1}….B_0). The ETA provides inaccurate values in the lower order bits while maintaining the accuracy in the higher-order bits using an exact addition. Each part of this adder is characterized as follows:

- In the accurate part, a standard addition is computed by using any one of the available traditional 1-bit full adders (e.g., ripple carry, carry look-ahead adders). In our ETA, we employed the ripple carry adder.

- The inaccurate part consists of two blocks: control block and carry free addition block. The control block is composed respectively of two operand bits A and B, the control signal from the previous CTRL block, and a CTRL out signal. A and B are added through an AND logic gate, then an OR operation is applied between the result and the previous control signal. On the other hand, the carry free addition consists of an XOR and OR gates and generates a sum bit based on a control signal obtained from the control block. So, in the inaccurate part, from left to right, the values of both input bits are checked: if the bits are 0 or different, a standard addition is computed, whenever the input bits are equal to 1, all the remaining right side bits are set to 1.

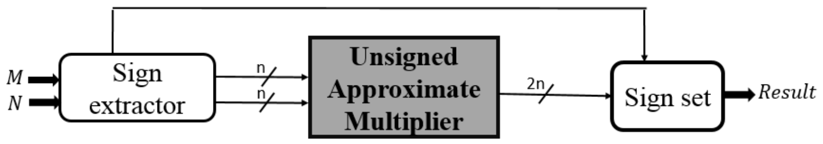
Moreover, the proposed architecture uses an exact subtractor, which generates the difference between two bits adopting the borrow bit of the lower significant stage.

d) Sign Set

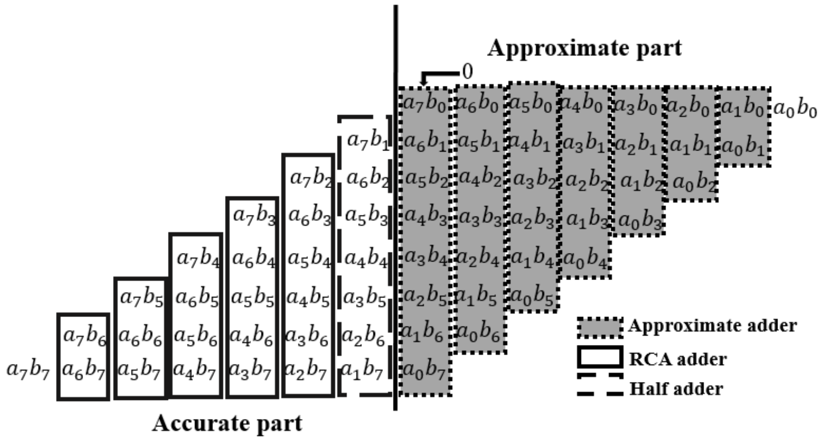
The primary function of the sign set block is to set the sign of the final multiplication result. It reverses the output of the subtractor when the extracted sign (from sign extractor) for the two input values is different.

2.4.2.2 Approximate Baugh-Wooley Multiplier

The approximate proposed multiplier is a signed array multiplier implemented for 8×8 multiplication, as shown in Fig.2.13 (b). The proposed multiplier addresses low power consumption with low accuracy degradation. Through the logic AND gates, the partial product tree is generated from two 2^m bit operands ($m = 3$); in our case, the partial product tree is generated without any approximation. The approximations have been employed in the accumulation phase of the partial products. The proposed architecture divides the



(a)



(b)

Fig.2.13.(a) General block diagram (b) Architecture of approximate Baugh-Wooley multiplier operation into two different groups: accurate and approximated. The architecture is based on Ripple Carry Adder (RCA) in the accurate part and on the proposed adder (AFA) in the approximate part. The dividing strategy adopted to the architecture of the proposed multiplier is dependent on the requirements of the application. Moreover, the MSB part consists of $2n - k$ bits, while the LSB part is composed of k bits. In our case, the input width of the multiplier is $n = 8$ and the imprecision parameter, which is responsible for determining the boundaries of the accurate and the inaccurate parts of the multiplier, is 8.

From right to left, an exact addition is computed in the accurate part. Through a half adder, the partial products are added; the generated carry signal is propagated to the following partial column in the next column. The partial products in the second column are computed through an exact adder. While in the approximate part, the addition of the partial products is done through the proposed approximate adder based on the following equations:

$$\left\{\begin{array}{c} S_7(0) = a_7.b_0 \oplus \; a_6.b_1 + C_{in} \\ S_7(1) = a_5.b_2 \; \oplus S_7(0) \; + C_{out7}(0) \\ \vdots \\ S_7(6) = a_0.b_7 \; \oplus S_7(5) \; + C_{out7}(5) \end{array}\right. \tag{2.23}$$

Where $S_7[0]$ represents the sum bit of the first two partial products for the 2^7 bit position.

And

$$\left\{\begin{array}{c} C_{out7}(0) = (a_7.b_0).(a_6.b_1) + C_{in} \\ \vdots \\ C_{out7}(5) = (a_1.b_6).S_7(4) + C_{out7}(4) \\ C_{out7}(6) = (a_0.b_7).S_7(5) + C_{out7}(5) \end{array}\right. \tag{2.24}$$

$$S_{n-1} = S_{n-1}(0) + \sum_{i=1}^{i=n-2} a_{n-2-i}.b_{i+1} \oplus S_{n-1}(i-1)$$

$$+ \ C_{out(n-1)}(i-1) \tag{2.25}$$

Where

$$\boldsymbol{C_{out(n-1)} = C_{out(n-1)}(0) + \sum_{i=1}^{i=n-2} (a_{n-2-i}.b_{i+1}).\ S_{n-1}(i-1)}$$

$$\boldsymbol{+ C_{out(n-1)}(i-2)}$$

$$S = \cup_{n=1}^{n=8} S_{n-1} \tag{2.26}$$

In the approximate part of the proposed multiplier, an input carry signal with a value set to "$C_{in} = 0$" is added to the first partial product. Then starting from left to right, the process of the operation to obtain the sum bit of the partial products in each of the selected columns of the approximate part of the multiplier is done based on (2.23) and (2.24). The general equation obtained to compute the sum bit of all the partial products for the 2^{n-1} bit position ($n = 8$) can be concluded, as shown in (2.25). The generated $C_{out(n-1)}[i-1]$ will be added to the partial product ($a_6.b_0$). Then for $n = 7$, the sum and the carry out bits at the 2^{n-1} bit position are obtained respectively based on (2.25) and (2.26). Then the sum bits ($S_{n-1}, S_{n-2}, \dots\dots\dots, S_0$) are concatenated in the approximate part, as shown in (2.26).

Table 2.1. Output of Sum and Carry bits For Different Cases

	Partial Products	**Sum bit**	**Carry bit**
1st case	$a_{n-1}.b_0 \| a_{n-2}.b_1 \| \dots \| a_0.b_{n-1} = 1$	$S_{n-1} = 1$	$C_{n-1} = 0$
2nd case	$a_{n-1}.b_0 = a_{n-2}.b_1 = \cdots = a_0.b_{n-1}=0$	$S_{n-1} = 0$	$C_{n-1} = 0$
3rd case	$a_{n-1}.b_0 = a_{n-2}.b_1 = 1$ Or $a_{n-1}.b_0 = a_{n-2}.b_1 = a_{n-3}.b_2 = 1$ Or : $a_{n-1}.b_0 = a_{n-2}.b_1 = \cdots = a_0.b_{n-1}=1$	$S_{n-1} =$ $S_{n-2} =..=$ $S_0 =1$	$C_{n-1} = 1$

As shown in Table 2.1, if only one of the values of the partial products is one or if all the values are equal to zero, then a standard addition is performed. If at least two partial products or more are equal to one, their sum in the selected column and all the remaining right columns are set to one.

Then we have designed three versions of META and Approx-BW multipliers based on the approximate adders presented in section 2.4. The choice was based on (i) the performance that the selected adders provide and (ii) their flexibility to be used in the proposed multiplier architecture. The combinations are as follows:

- The proposed multiplier (approximate Baugh-Wooley) which is based on the AFA adder.

- Mul-LOA and Mul-AXA are based on lower-part-OR and XNOR-based adders, respectively.
- MNAND, MAND, and MIPP multipliers are based respectively on NAND-carry out bit, AND-carry out bit, and Input pre-processing approximate adders.

2.5 Results

This section presents the hardware implementation results of the two proposed approximate multipliers (META and approximate BW) in terms of performance, accuracy, and power consumption.

2.5.1 Hardware Implementation of META Multiplier

In order to evaluate the performance of the META multiplier, four different architectures have been implemented and compared with the exact Baugh-Wooley multiplier. The first architecture signed MRCA (S-MRCA) uses the RCA exact adder as an additional unit, while the second one signed META (S-META) uses an inexact ETA adder as described in section 2.4. In the case where the inputs are always positive, two architectures called unsigned MRCA (U-MRCA) and unsigned META (U-META) have been implemented after removing the sign extractor and sign set blocks from the architecture. The circuits have been implemented in Vivado Design Suite 2017.1 using VHDL Hardware Description Language. The designs have been synthesized using the Xilinx Vivado synthesizer, with Virtex-7 xc7vx485tffg1157-1 as a target device. Based on the implementation results, this section analyzes the performance parameters of the proposed architectures highlighting the computation accuracy and power consumption.

2.5.1.1 Accuracy Evaluation

Some tests have been carried out to assess the accuracy of the multipliers. The variation of the acceptance probability as a function of the minimum acceptable accuracy is analyzed [89]. The inaccuracy of the approximate multipliers is generated after eliminating the term $(M_r - \mathrm{M}) \times (N_r - N)$ from the initial accurate multiplication. Accurate results are obtained only when M_r and N_r are respectively equal to 2^n and 2^m. In this case, both inputs would be equal to 3×2^n and 3×2^m respectively, and the error would be maximum.

Some used terms are explained below:

- Error (E): E = |Re – Ri|, where Re is the exact multiplication result, and Ri is the inexact result obtained by the approximate multiplier simulation.
- Accuracy (ACC): ACC = (1 – E/Re) × 100. To determine how accurate the output of the multiplier is with respect to the exact multiplication. Values range between 0% and 100%.
- Minimum Acceptable Accuracy (MAA): it is considered as the threshold value; to respect the constraints of the system, the obtained accuracy must be higher than this value.
- Probability of acceptance (PA): it is the probability of values with higher accuracy than MAA, which is represented as PA = P (ACC > MAA). Its value ranges from 0 to 1.
- MED and MSE represent, respectively, the mean error distance and the mean square error. The MED is obtained after computing the average error distance (ED),

which is defined as the difference between exact and approximate results. The accuracy metrics are defined as follow:

$$ED = |Approximate\ result - Exact\ result| \tag{2.27}$$

$$MED = \frac{ED}{number\ of\ inputs} \tag{2.28}$$

$$MSE = \frac{\sum_{i=1}^{n} ED^2}{number\ of\ inputs} \tag{2.29}$$

- NMED and MRED represent, respectively, the normalized mean error distance and the mean relative error distance. MRED is the average of the relative error distance defined as:

$$RED = \frac{|\ Approximate\ result - Exact\ result|}{Exact\ result} \tag{2.30}$$

$$MRED = \frac{RED}{number\ of\ inputs} \tag{2.31}$$

Moreover, NMED is computed to evaluate the adders of different sizes defined as follow:

$$NMED = \frac{MED}{M_{max}} \tag{2.32}$$

Where M_{max} is the maximum magnitude of the output of an accurate adder, which is 2n in the case of an n × n adder.

- The passing rate metric is the ratio of exact results over the total number of outputs.

Therefore, the variation of PA with respect to MAA has been analyzed: Fig. 2.14 and Fig.2.15 illustrate a comparison for 8-bit multipliers among U-META, U-MRCA, and Baugh Wooley from one side, respectively, and S-META, S-MRCA and Baugh Wooley from the other side. We randomly selected 10^5 signed inputs for the signed multipliers and

10^5 unsigned inputs for the unsigned ones. Upon simulation, we observe that 89% of the

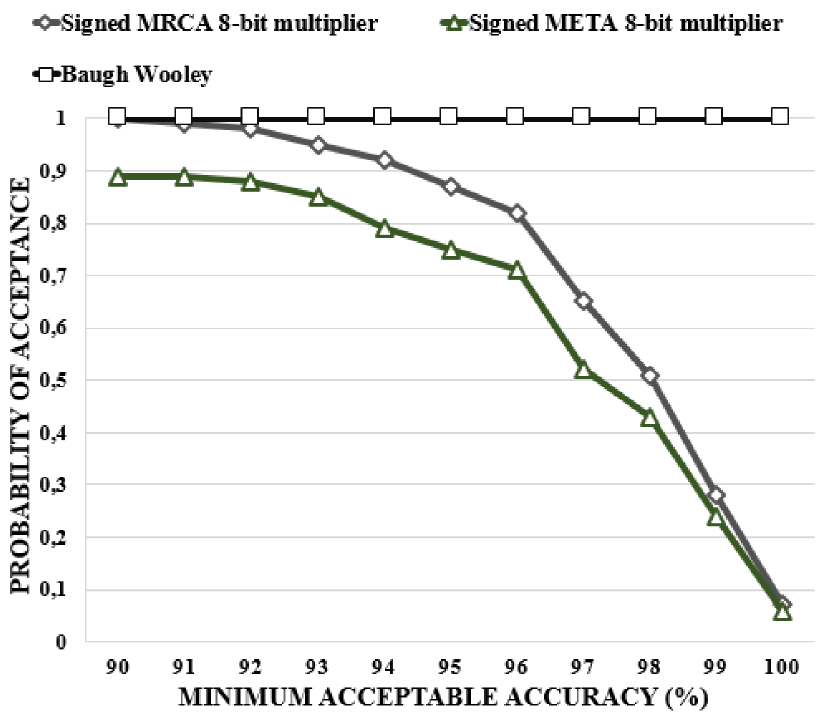


Fig.2.14. Probability of acceptance versus minimum acceptable accuracy for S-META and S-MRCA.

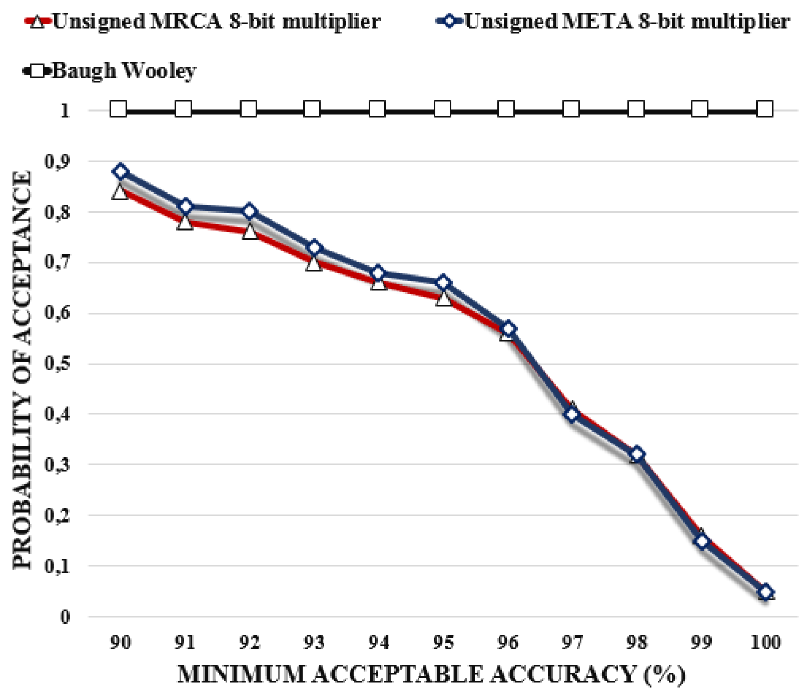


Fig.2.15. Probability of acceptance versus minimum acceptable accuracy for U-META and U-MRCA.

results have higher accuracy than 90% for the S-META multiplier while achieving an accuracy of 94.5% for the S-MRCA multiplier[90]. For the unsigned multipliers, the minimum acceptable accuracy for 88% of the results varies between 90.5% for the U-MRCA and 90% for U-META. We can deduce that the ETA adder affects the accuracy of the approximate multipliers slightly and especially for the unsigned multipliers where the variation is considered negligible. Hence, despite the small accuracy loss (i.e., 4.5% in the worst case) in the proposed architecture, the approximate multipliers still have a reasonable accuracy which could be acceptable for a variety of applications in digital signal processing. Fig. 2.16 represents the error percentage distribution for S-META and S-MRCA. It is pointed out that 43% of the results show an error of less than 2%, and about 31% of the results present an error between 2% and 5%. Hence, less than 14% of the results are characterized by an error of more than 8%, which proves the correctness of the proposed architecture.

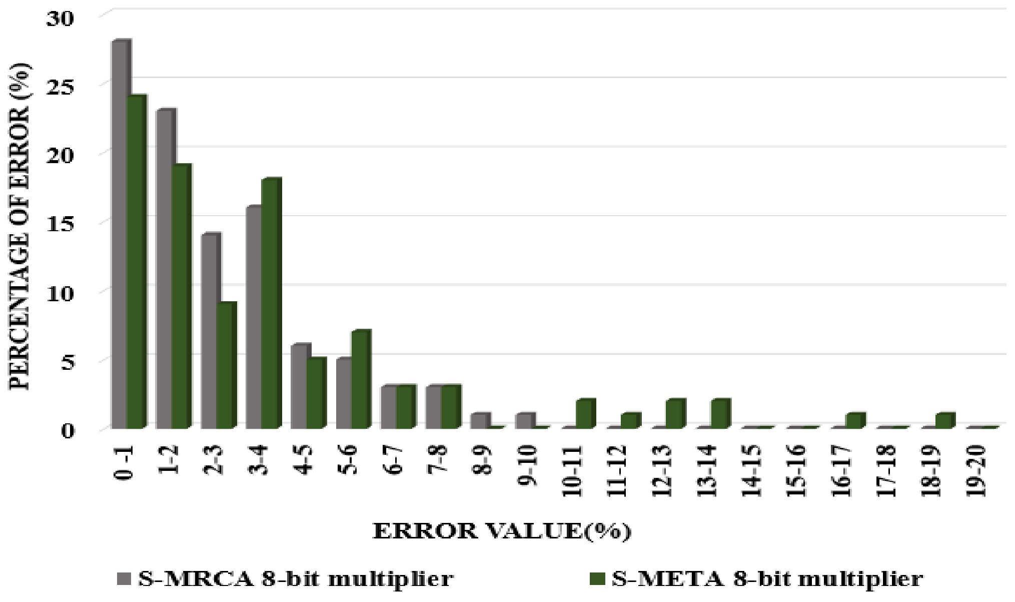


Fig. 2.16. Error percentage distribution for the two approximate multipliers

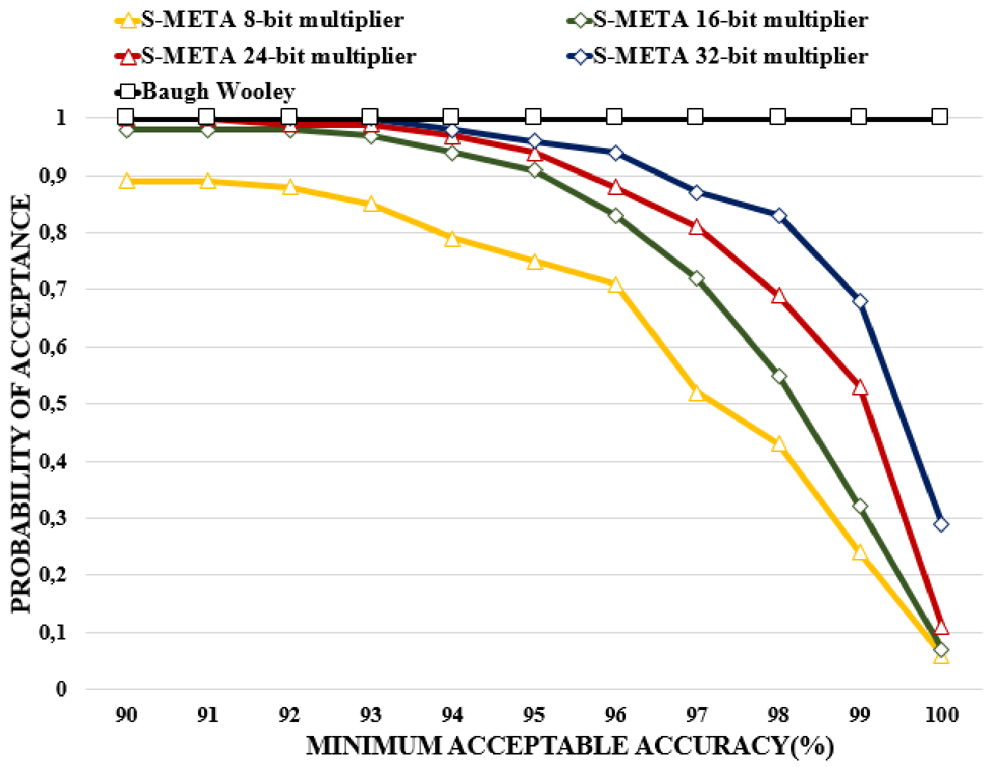


Fig. 2.17. Probability of acceptance of S-META for different bit sizes.

On the other hand, the effect of increasing the number of bits on the accuracy of computation of the proposed multipliers has been analyzed. Fig. 2.17, and Fig. 2.18 present respectively four S-META and U-META multipliers with 8-, 16-, 24-and 32-bit

input bit width. The results show that the accuracy and the probability of acceptances increase as the bit length increases.

2.5.1.2 Performance Evaluation

Table 2.2 reports the power consumption obtained by the simulation of the implemented designs. Using the Vivado tool simulation, a test has been run for 3μs to determine the average dynamic power and the time delay. Results show a sound reduction in power consumption: for the proposed 8-bit approximate multipliers, the power is

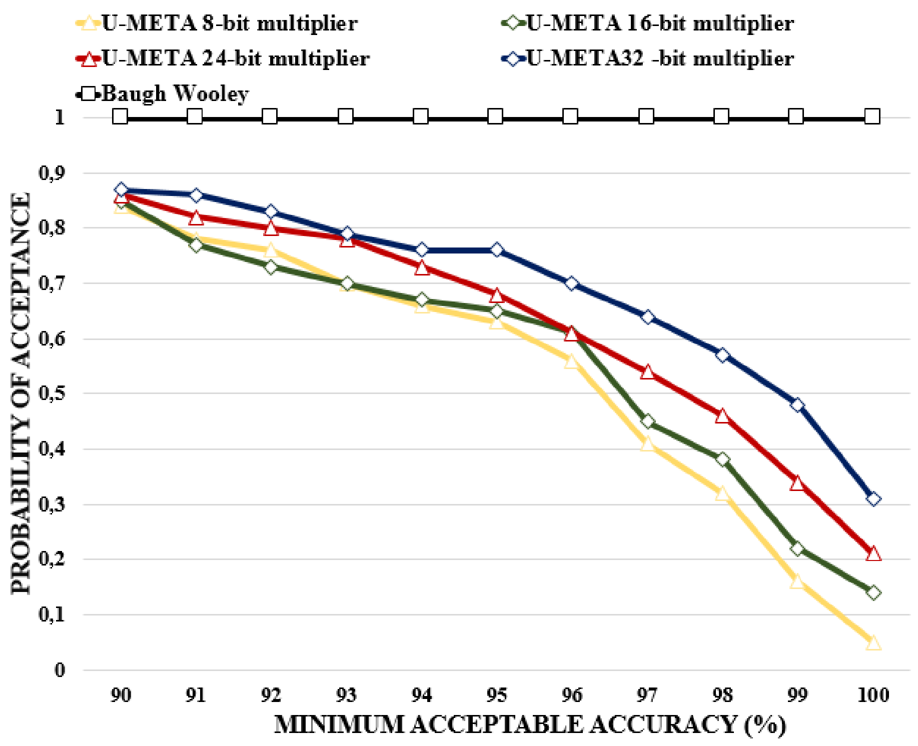


Fig. 2.18. Probability of acceptance of U-META for different bit sizes.

reduced by 56.3% (in the case of U-META) when compared to the exact Baugh-Wooley

Table 2.2. Simulation results of 8-bit multipliers

Multipliers	**Parameters**		
	Power(mW)	**Delay(nS)**	**Utilized LUTs(%)**
Baugh-Wooley	112	6.98	0.03
S-MRCA	69	9.71	0.04
S-META	57	8.4	0.05
U-MRCA	58	7.47	0.05
U-META	49	6.91	0.04

multiplier. Moreover, after omitting the negation sign, the results indicate that the power consumption (delay) for the unsigned approximate multipliers is reduced, respectively, around 15.9%, (23.1%), 14.1%, (17.7%) with respect to the design parameters of the signed approximate multipliers.

On the other hand, the results reveal that by employing an inexact ETA adder, the power consumption and time delay for S-META are reduced respectively of 17.4% and 13.5% when compared to S-MRCA. On the other side, the power and delay of U-META decreased by 15.5% and 7.5%, respectively, with respect to the design parameters of U-MRCA. In contrast, the delay and LUT of the exact multiplier Baugh-Wooley are better than those of the approximate multipliers except for the U-META delay parameter. We conclude that, among the four implemented approximate multipliers, U-META provides the best performance in terms of time delay and power consumption. In order to analyze the

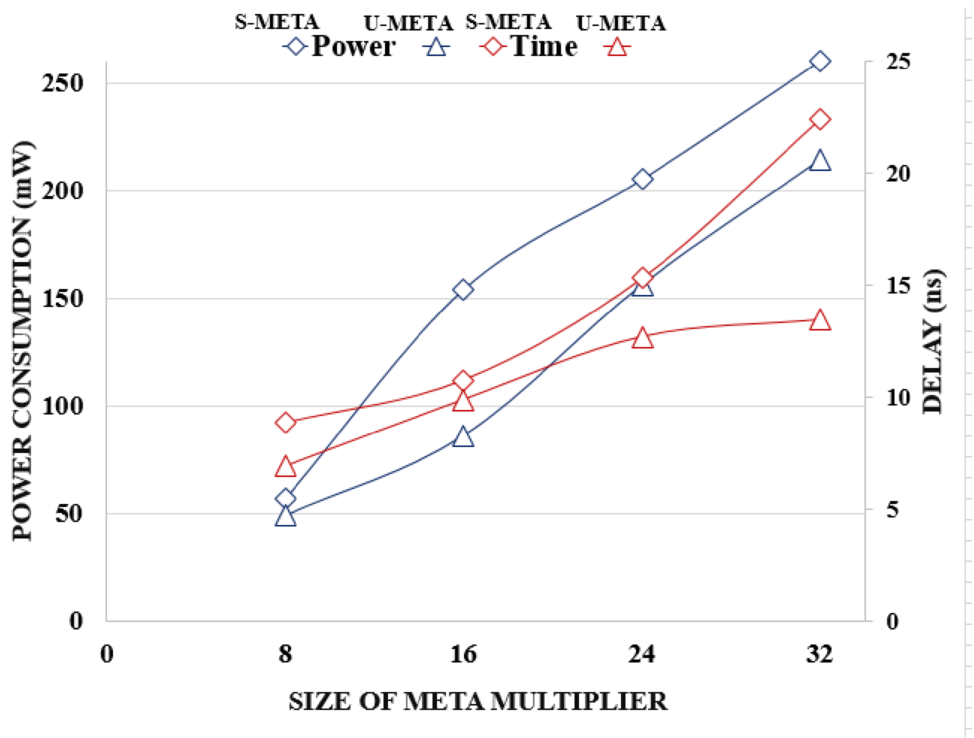


Fig. 2.19. Average power consumption and delay of META multipliers.

performance of both S-META and U-META in terms of power consumption and time delay when increasing the multiplier size, another test has been carried out over 1ms. Fig. 2.19 shows power consumption variation and time delay as a function of multipliers size for S-META and U-META. Power consumption and time delay increase when the size of the signed and unsigned META becomes larger. On the other hand, since power consumption standard deviation is high for a wide range of inputs, we randomly selected five inputs as an example to assess the dynamic power consumption for 8-bit META and MRCA multipliers. Each input has been simulated for a period of 20 ns. The comparison of the obtained results presented in Fig. 2.19 indicates an impressive saving in dynamic power from 9.2% up to 50%. For instance, for the product 1F×7c, the power drops from 82mW to 41mW, while the accuracy of the results remains approximately unchanged.

2.5.2 Hardware Implementation of Approximate Baugh-Wooley Multiplier

In this section, we evaluate the performance of the proposed approximate BW multiplier in terms of accuracy, power consumption, and delay. Moreover, we compare the performance of the Approx-BW with respect to well-known open-source approximate multipliers presented in state of the art.

2.5.2.1 Accuracy Evaluation

MED, MSE, NMED, MRED, and Pass rates results are reported in Table 2.3. Results show that the new Approx-BW multiplier outperforms state of the art multipliers (Shaf[91]and

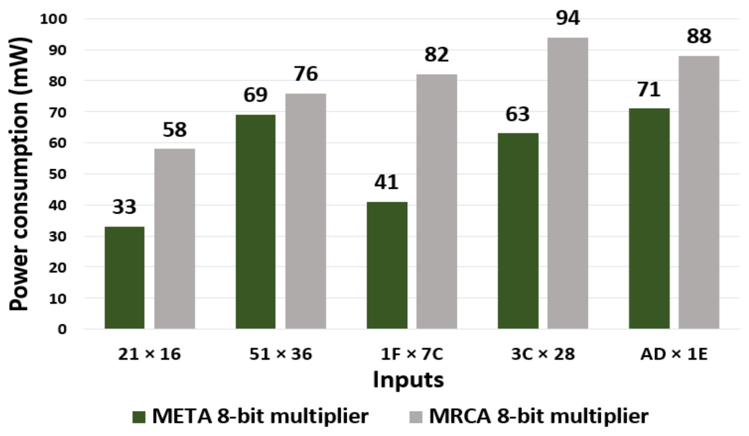


Fig. 2.20. Instantaneous power consumption and delay of META multiplier.

Evo25[85]) in terms of MRED and NMED. Approximate-BW is more accurate than the other generated multipliers, which are based on distinct adders such as (Mul-LOA, Mul-AXA, MAND, MNAND, and MIPP); this could be affected by the fact that the MRED of AFA is the lowest between the approximate adders. In Figure 2.21, we studied the percentage of the outputs as a function of the relative percentage error. It is shown that the accuracy of Approx-BW is within an acceptable range, since more than 70% of its outputs have a REDs smaller than 10%. Approx-BW provides a better performance in terms of accuracy, with respect to the two approximate multipliers proposed in [78] and [85]. [78] and [85] show the lowest accuracy having respectively 42% and 23% of their outputs smaller than 10%. Moreover, the impact of varying the imprecision parameter k (2,4,8 and 16) on the accuracy of the multipliers is evaluated. Fig.2.22 (a), (b), and (c) shows respectively NMED, MRED, pass rates versus the variation of the number of the approximated bits in the approximate multipliers. The results show that the error increases as a function of n. It is shown that Approx-BW has the lowest NMED (0.09%) and MRED (4.9%) when n is equal to 2. While MAND, META, MIPP have the same NMED (0.21%).

Table 2.3. Accuracy metrics for different approximate multipliers designs

Approximate Multipliers	**MED**	**MSE**	**NMED**	**MRED**	**PASS RATES**
APPROX-BW	232.33	9.8E+04	0.35	0.101	5.81%
MUL-LOA	239.4	1.1E+05	0.37	0.109	5.65%
MUL-AXA	168.17	4.6E+04	0.25	0.155	0.17%
MAND	156.15	7.7E+04	0.24	0.102	2.87%
MIPP	200.69	9.9E+04	0.31	0.132	2.68%
ROBA [70]	139.5	7.0E+04	0.21	0.091	2.86%
META [60]	154.15	7.4E+04	0.23	0.09	2.53%
EVO0[85]	109.56	2.3E+04	0.16	0.079	4.13%
EVO [85]	461.17	5.7E+05	0.71	0.22	0.27%
KULKARNI[78]	116.58	4.05E+04	0.18	0.076	16.3%
SHAFIQUE [91]	512.11	8.8E+05	0.51	0.23	3.2%

When increasing n to 16, Mul-AXA shows the highest NMED (4%) and MRED (45%). However, MIPP and Mul-LOA have the best accuracy in terms of NMED (1.09%, 1.29%) and MRED (19.4%, 19.67%) when all the bits of the multipliers are inexact. Fig.22 (c)

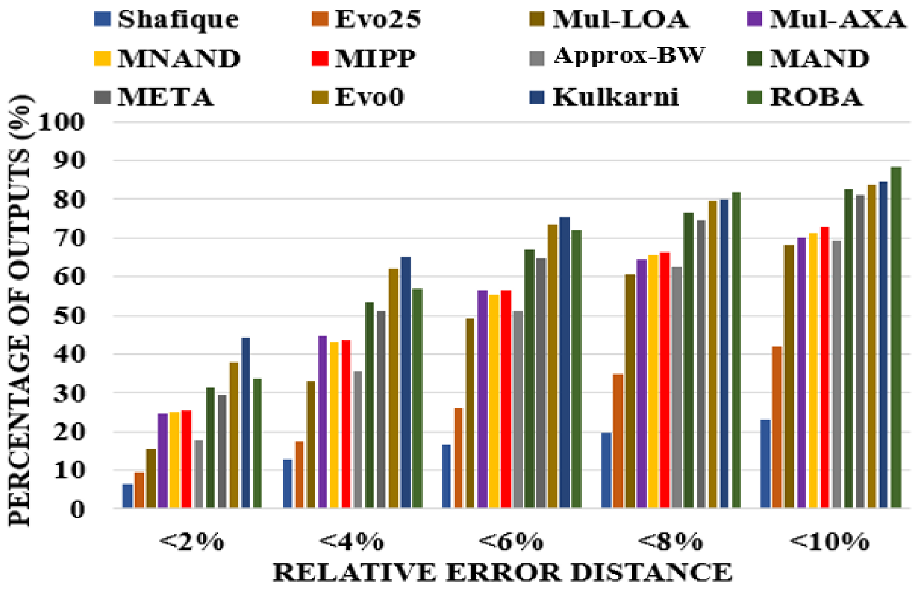


Fig. 2.21. Percentages of outputs versus relative error distance for different inexact multiplier circuits

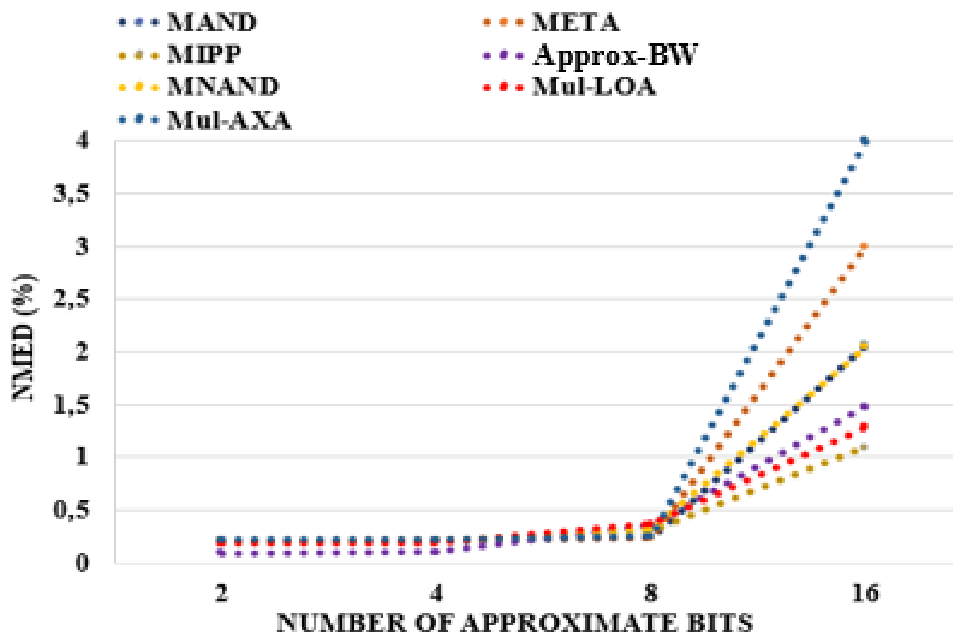
shows that Approx-BW and Mul-LOA have the best passing rates respectively 19.81%, 16.18% when n is 2. While, MAND, META, MIPP and MNAND have the lowest passing rates equal to 2.8% when n is 2. When n increases to 16, the passing rates of Mul-AXA decreases rapidly reaching the lowest value nearly equal to zero. To summarize, among all approximate multipliers, Approx-BW shows the best performance in terms of accuracy when decreasing the number of bits in the approximate part.

2.5.2.2 Performance Evaluation

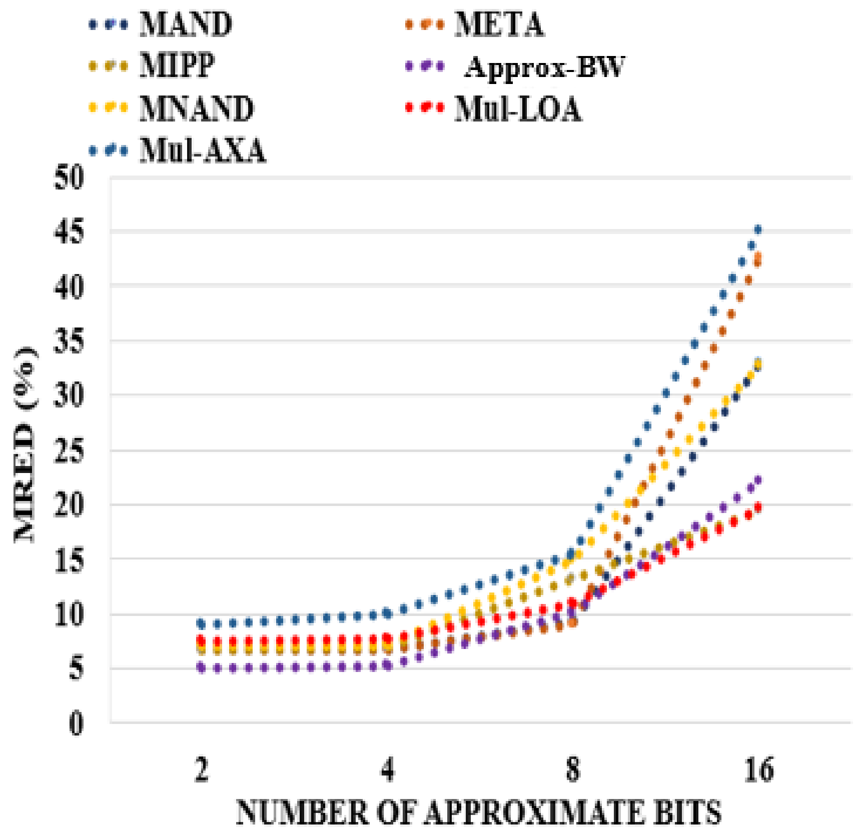
The power consumption and time delay of the implemented multipliers have been determined after running a test for 3 us. Fig.2.23 reports a comprehensive comparison between exact and approximate multipliers by considering the MRE, power consumption, and time delay. Table 2.4 shows the circuit synthesis results for LUT utilization, power-delay product (PDP), and PDP-MRED products. The results show that Approx-BW and Mul-LOA are the most efficient with an MRED around 0.1. The power consumption of

Table 2.4. Area, PDP and PDP-MRED of multipliers designs

Approximate Multipliers	**LUT(%)**	**PDP(nJ)**	**PDP-MRED**
APPROX-BW	0.03	0.13	1.29
MUL-LOA	0.03	0.13	1.38
MUL-AXA	0.03	0.57	8.85
MAND	0.04	0.5	4.53
MIPP	0.04	0.43	5.67
ROBA[70]	0.05	0.67	6.09
META[60]	0.05	0.48	4.31
EVO0[85]	0.03	0.37	2.96
EVO[85]	0.01	0.08	1.72
KULKARNI[78]	0.03	0.41	3.13
SHAFIQUE[91]	0.02	0.32	5.12

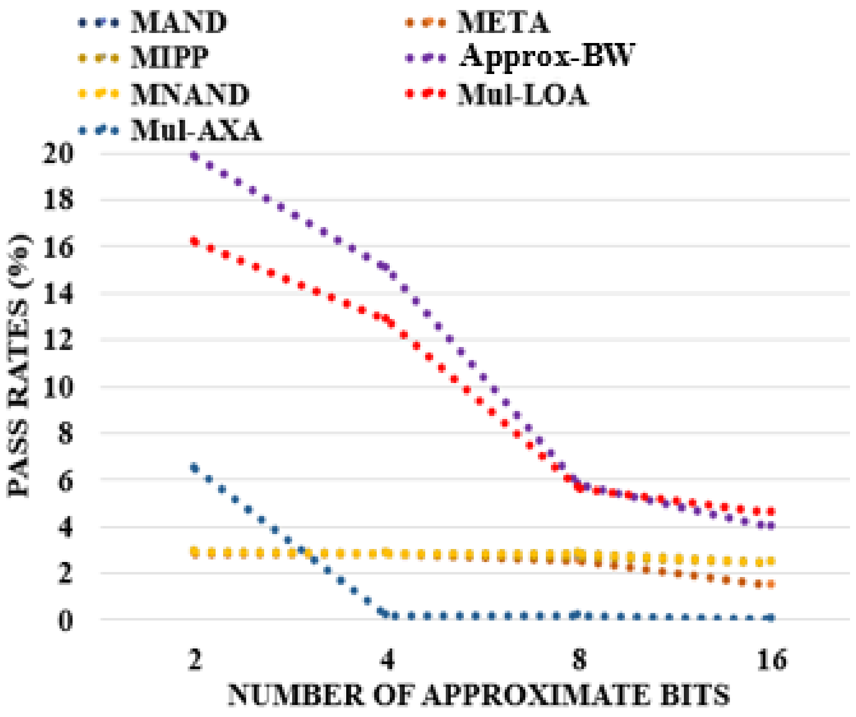


(a)



(b)

Approx-BW and Mul-LOA is reduced by 80.4% when compared to the exact BW



(c)

Fig.2.22 Variation of PASS RATES(c) with respect to the number of approximate bits

Moreover, Approx-BW is 10% more accurate than Mul-LOA, thus outperforming Mul-LOA in terms of accuracy. While META[60] and ROBA [70] show better MRED around 1% but higher power and time delay with respect to Approx-BW. The power consumption of Approx-BW is reduced by more than 68% with respect to META and MRCA. On the other side, MAND and MNAND show the same power, while MRED of MNAND is higher. Mul-AXA shows the highest NMED and power consumption. Among state-of-the-art multipliers, the only circuit that consumes energy

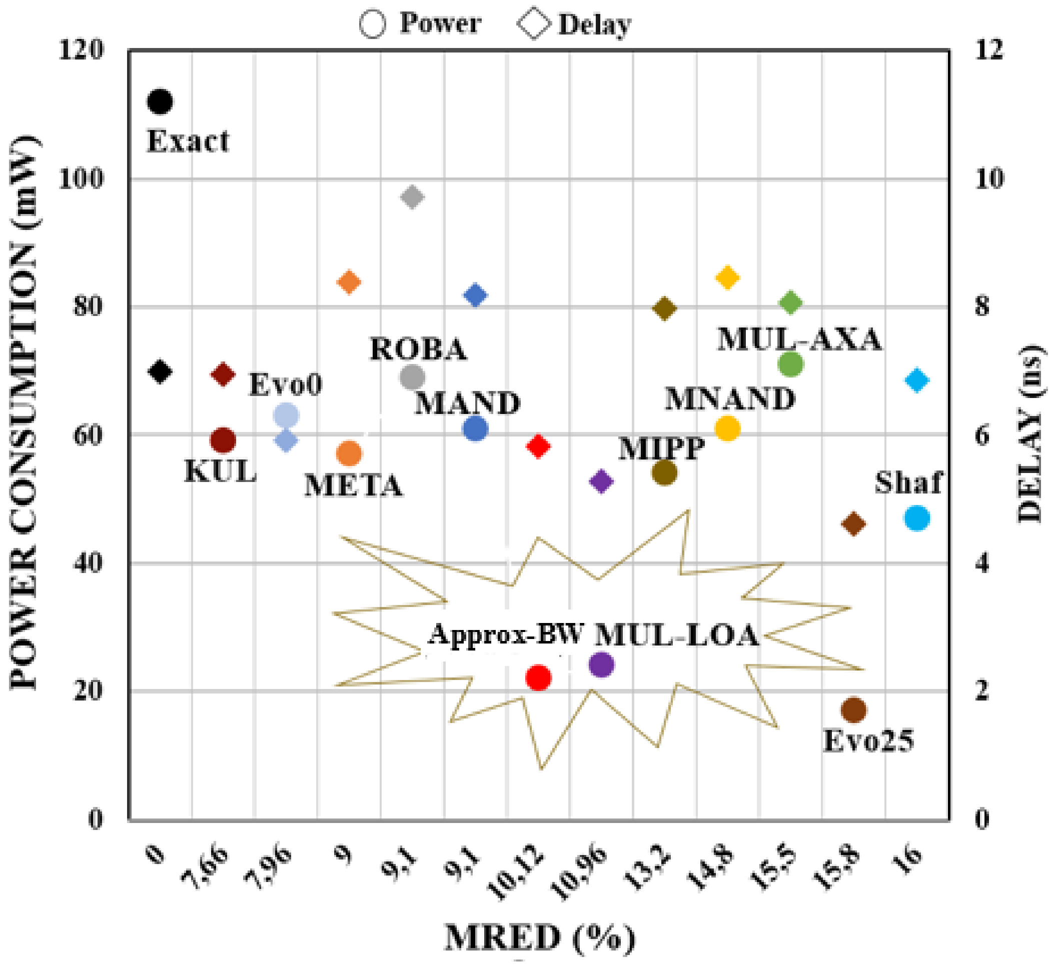


Fig. 2.23. Power consumption, delay and MRE of exact and approximate multiplier designs.

less than Approx-BW is Evo25. However, Evo25 is around 50% less accurate than Approx-BW. Then our design is considered among the most power- and energy-efficient designs with approximately small PDP value. Moreover, we took into consideration the two parameters PDP and MRED to evaluate all the circuits as shown in Table 2.4. Our design shows the lowest PDP-MRED product (1.29). As a conclusion, the proposed multiplier (Approx-BW) achieves the best results in terms of PDP-MRED with a small PDP.

2.6 Conclusion

In this chapter, an FPGA implementation of two new architectures of approximate multipliers called META and Approximate Baugh-Wooley multiplier have been proposed. META multiplier has provided a noticeable improvement in latency and power consumption at the price of a small error (around 5%). Four hardware implementations of the approximate multiplier were compared in terms of power consumption and time delay with respect to the exact Baugh-Wooley: results report up to 56% power saving. On the other hand, META was compared to MRCA. Results revealed that the accuracy of the *META* multiplier slightly decreased (around 5%), while the power consumption and the delay have been reduced respectively by 17.4% and 13.5% when compared to MRCA. By omitting the negation sign from the signed approximate multiplier, results show that power consumptions of U-META and U-MRCA are less than 17.7% and 15.9%, respectively, in comparison to S-META and S-MRCA.

On the other hand, several state-of-the-art adders have been adapted and implemented; and results showing a comparison between different circuits in terms of accuracy and power consumption. Moreover, various inexact multiplier circuits have been implemented based on different inexact adder designs. The main idea was to involve the implemented inexact adders at the place of the exact ones needed to perform the multiplication. However, results showed that the second proposed architecture “Approximate BW” is the most efficient design between the approximate multipliers achieving a relevant reduction in power consumption and time delay around 80.4% and 24% respectively with respect to the exact BW multiplier. While the power consumption of Approximate-BW is reduced by 68.1% with respect to other states of the art solutions

Therefore, the proposed architectures provide a possible tradeoff between accuracy and power saving for improving performance and energy efficiency. In a conclusion, implementation results proved that the achieved power consumption/accuracy tradeoff is satisfactory. In the next chapter, we will study the impact of the proposed approximate multipliers on tactile signal processing in the FIR filter for e-skin application. Then, we will study the use of the proposed approximate arithmetic circuits in the circuits blocks employed in machine learning algorithms for input touch modalities classification.

CHAPTER 3. APPROXIMATE COMPUTING CIRCUITS FOR TACTILE DATA PROCESSING

3.1 Introduction

Restoring the sense of touch to the prosthesis user is a challenging goal emphasized by many research in upper limb prosthetics. To accomplish this, prosthetic devices should incorporate electronic skin (e-skin) and a distributed simulation system [92]. Such a system will have the ability to acquire sensors data, preprocess the signals, and transmit information to the stimulator; consequently, the decoded data is communicated to the prosthetic user through electro-tactile stimulation [57].

The e-skin system should hold autonomous and numerous networked sensors [93]; every sensor hosts multiple input tactile sensors nearly coupled to an embedded electronic system and power supply [59]. Furthermore, the e-skin system is expected to have a long-lifetime while processing tactile data in real-time. Therefore, achieving low energy efficiency is considered as a favorable objective for IoT edge devices.

Approximate computing [94], [95], is considered as a relevant approach providing a tradeoff between accuracy and power savings. Approximate computing has succeeded in improving the energy efficiency for many applications [96],[97],[98]. These applications are inherent resilient since users could not perceive the small errors in the output. In this perspective, the work in this chapter focuses on adopting AC techniques at circuit level for simple tasks (FIR filter) and for complex tasks (CORDIC and SVD) in order to reduce the power consumption of real-time tactile data processing in the e-skin application.

In these perspectives, this chapter aims to investigate the following main points:

1) Could an approximate computing approach be applied to improve the energy efficiency in IoT edge devices for the tactile sensing system in healthcare applications?

2) What is the impact of employing approximate computing techniques on the signals outputs generated from the e-skin application?

In this chapter, we employ the proposed approximate arithmetic circuits presented in Chapter 2 for the tactile sensing system. The objective is to assess the impact of proposed circuits as well as relevant state of the art approximate multipliers on the tactile sensing system. The quality is measured in terms of different metrics, mainly: SNR degradation, MRED, NMED, power consumption, PDP, and time delay. The results presented in this chapter are not all published yet; only some part of the results was published in [61],[62].

3.2 E-skin system

In this section, we will describe the tactile sensing system. The block diagram of the e-skin system is shown in Fig.1, it is mainly composed of four different blocks: 1) A piezoelectric tactile sensor array that receives input stimuli, 2) An electronic interface system in charge of signal conditioning and data acquisition, 3) an embedded electronic system for digital

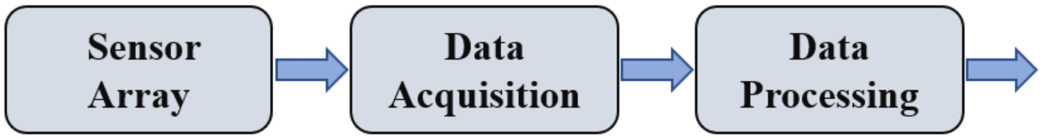


Fig.3.1 Functional diagram of electronic-skin system

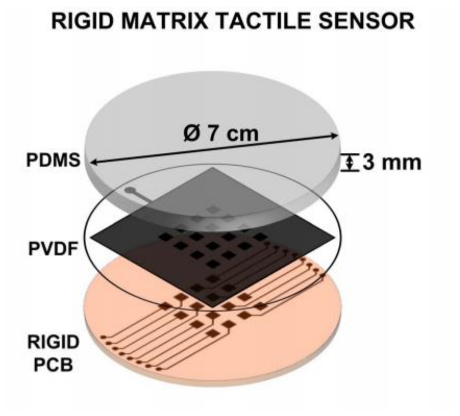


Fig.3.2. Sensor array

signal processing (DSP) to preprocess the tactile signals and 4) data processing block which is responsible in processing the signals generated from the digital signal processing block.

3.2.1 Sensors array

The sensors employed in the system are composed of an array of 4×4 tactile sensors based on PVDF (Polyvinylidene Fluoride) that generate a charge as a response to mechanical stimuli. The main characteristics of the PVDF are i) high sensitivity, ii) wide response range, and ii) sizeable electromechanical transduction bandwidth, i.e.1 Hz up to 1 kHz in tactile application [92]. Figure 3.2 shows the structure of the tactile sensor array, which is composed of three layers: 1) the PDMS protective top layer, 2) the PCB bottom rigid substrate, and 3) a thick PVDF film layer positioned between the two other layers. The PVDF film is provided by 16 square taxels printed by an ad-hoc ink-jet. After contact on the surface of the sensor, stress measured by the PVDF taxels will process sensor data and make decisions [35] autonomously through the PDMS layer.

3.2.2 Interface electronics

As a purely passive device, the sensor needs active interface electronics to amplify and digitize the signals from its various channels. Charge separation in the piezoelectric material of the sensor due to the applied pressure is in the order of 0.1pC to several hundred pC, and hence it requires a low-noise interface with a reasonably high dynamic range. A commercial multichannel current integrator integrated circuit (IC), from Texas Instruments (TI) designed for computed tomography scanners, is suitable to interface PVDF sensors [99]. Key characteristics of this solution for e-skin applications are a high dynamic range, high sampling rate and the availability of large amounts of channels in an integrated form factor. More precisely, we used the TI DDC264 IC with a 64-channel variant and samples rate up to 3.1 kSPS per channel with 20 bits resolution for a full-scale range of 150pC. The IC has a standard SPI interface to provide access to the digital samples, and connection with the processing unit.

3.2.3 Signal and data processing

In the third block of the e-skin system, a low-pass Finite Impulse Response filter (FIR) aims to Filter the tactile signals as described in [6]. Then in the fourth block of the system, Machine Learning algorithms are employed for processing tactile data when classifying touch modalities. Different DSP units such as: Singular Value Decomposition (SVD), Coordinate Rotational Digital Computer Circuits (CORDIC) take a significant part of the embedded ML algorithms. However, the DSP blocks are implemented by using multiplication and addition operations, posing significant challenges on power consumption and time latency since the number of operations to be performed high.

Nevertheless, approximations could be enabled for applications requiring signal and data processing. Therefore, any improvement in the performance of the adders and multipliers will significantly affect the performance of the overall tactile sensing system. In order to understand the behavior of the tactile sensing system and the potential of approximations techniques, we first implement different approximate multipliers on low-pass Finite Impulse Response (FIR) filter in the signal processing stage. Second, we implement approximate adders and multipliers respectively in the CORDIC and SVD blocks in the data processing stage.

3.3 Methodology for approximating the tactile sensing system

Fig.3.3 presents an overview of the adopted methodology for approximating circuits blocks in the e-skin application. The methodology is composed of the main four blocks:

In the first phase, the approximate adders have been implemented and simulated in Vivado Design Suite 2017.1 using VHDL Hardware Description Language. The power consumption and time delay have been reported after synthesizing the designs by using Xilinx Vivado synthesizer, with Virtex-7 xc7vx485tffg1157-1 device after extracting the node activity and exporting it to the form of a SAIF (Switching Activity Interchange Format) files. In order to extract accuracy metrics,10^5 input has been randomly selected based on uniform distribution. In the second phase, the approximate adders have been implemented in the approximate multipliers. The evaluation of the approximate multipliers has been done following the same procedure proposed in the first phase. In order to have a fair comparison with similar state of the art solutions, open-source approximate arithmetic circuits based on relevant previous works[60],[78],[91],[85],[70],have also been implemented and simulated. The performance of the different multipliers is compared,

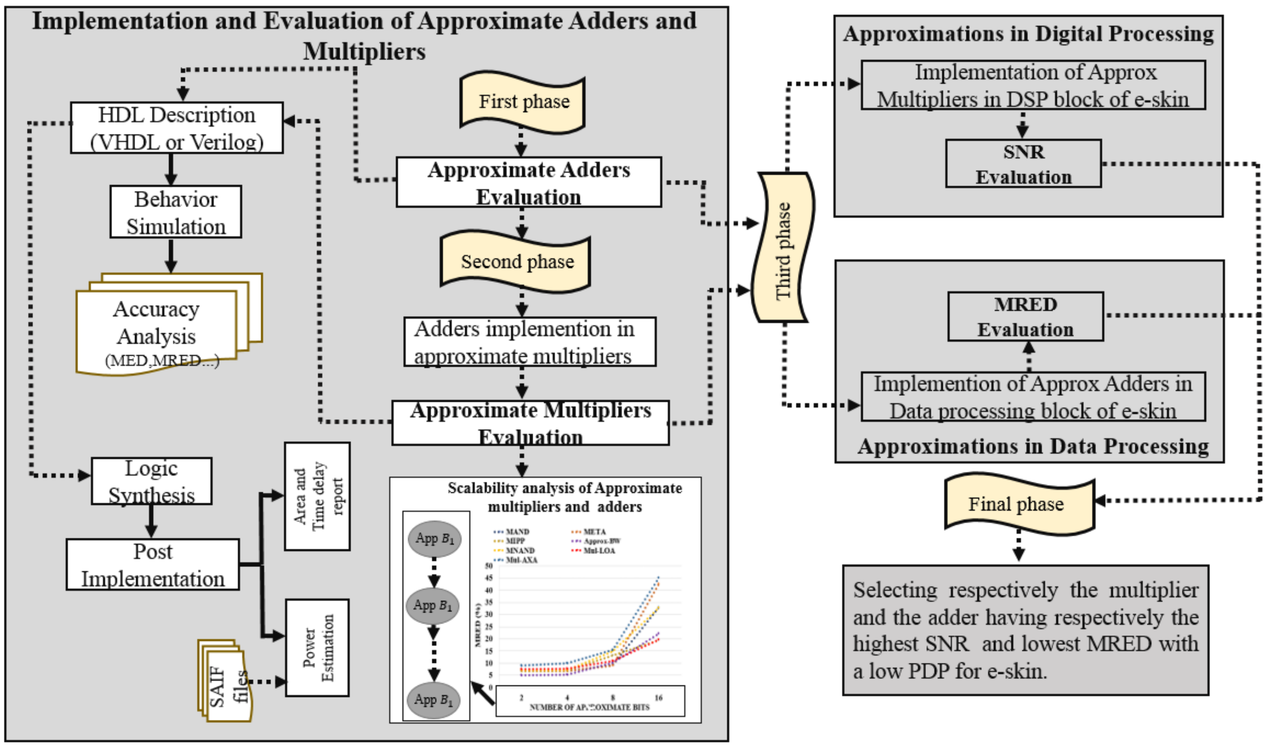

Implementation and Evaluation of Approximate Adders and Multipliers
HDL Description (VHDL or Verilog)
Behavior Simulation
Accuracy Analysis (MED,MRED...)
Logic Synthesis
Post Implementation
Area and Time delay report
Power Estimation
SAIF files
First phase
Approximate Adders Evaluation
Second phase
Adders implemention in approximate multipliers
Approximate Multipliers Evaluation
Scalability analysis of Approximate multipliers and adders
App B_1
NUMBER OF APPROXIMATE BITS
MRED (%)
MAND
MIPP
MNAND
Mul-AXA
META
Approx-BW
Mul-LOA
Third phase
Approximations in Digital Processing
Implementation of Approx Multipliers in DSP block of e-skin
SNR Evaluation
MRED Evaluation
Implemention of Approx Adders in Data processing block of e-skin
Approximations in Data Processing
Final phase
Selecting respectively the multiplier and the adder having respectively the highest SNR and lowest MRED with a low PDP for e-skin.

Fig.3.3 Proposed methodology for selecting the efficient multiplier for tactile sensing system

taking as a reference the exact Baugh-Wooley [35] multiplier. Then, we evaluate the resilience to the error of the approximate blocks by scaling the number of approximated LSBs in order to find the trade-off power-quality. In the third phase, we have implemented the approximate multipliers and adders respectively in the two stages of the tactile sensing system. The quality of the tactile output data is evaluated based on the Signal Noise to Ratio metric (SNR), while the quality of the approximate data processing block is evaluated based on the MRED of the generated data. Finally, the multiplier and the adder having respectively the highest SNR and lowest MRED with a low PDP will be selected for the e-skin application. The first and second phases of the methodology have been already assessed in the second chapter. While in this chapter, we deal with the third and fourth phases highlighting the experimental setup carried out to assess the approximate circuits in the target application.

3.4 Approximations in digital signal processing

3.4.1 Experimental setup description

In this part, we will describe the experimental setup needed to extract the tactile sensing signals. The tactile signals have been obtained from the experimental setup presented in [93]. Fig.3.4 shows the different instruments used in this setup. A frequency of 100 Hz is applied to the shaker through the function generator. The force sensor reads the values of the force applied to the tactile sensor via the LabVIEW tool. On the other hand, an electrical signal is generated when the shaker applies a mechanical stimulus on the surface of the tactile sensor. Then, through the A/D converter, the converted electrical signal is connected to the FPGA, which is responsible for sending the converted data to the MATLAB® tool using a UART to USB interface. Collected data has been normalized in MATLAB® after being extracted for a duration of 5s.

The retrieved tactile signals have been recorded to a text file and have been passed into the FIR filter using the Xilinx Vivado simulator. Then, the output filtered signals have been recorded. Another MATLAB® script has been used for analyzing the filtered signals. The MATLAB® script first computes the Fast Fourier transform (FFT) of the signals then calculates the Signal to Noise ratio (SNR) for the filter, as shown in Fig.3.4 a). The same

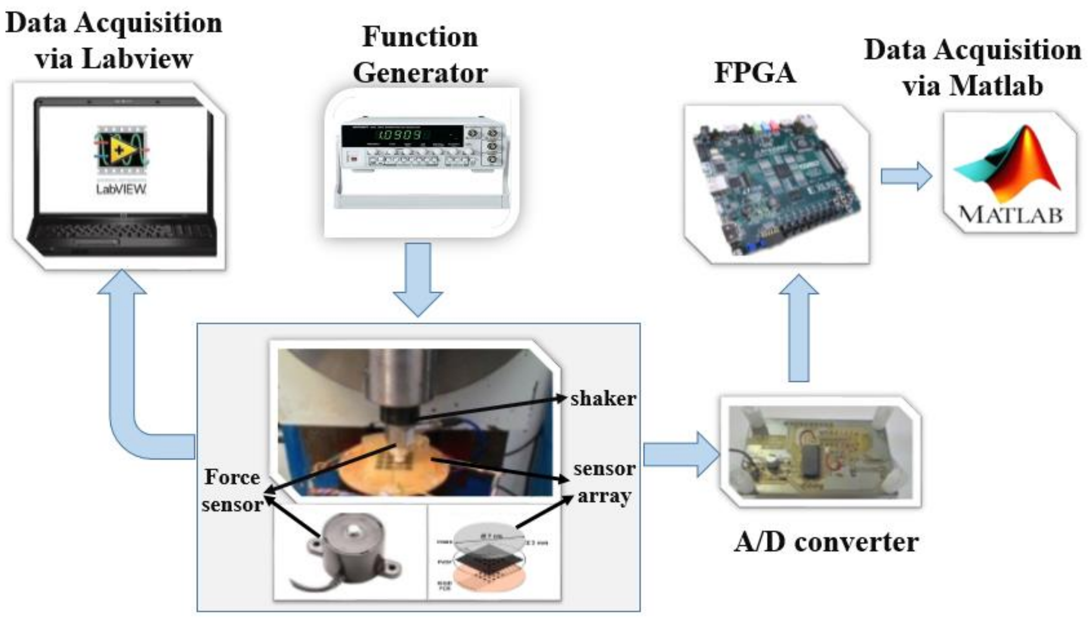


Fig. 3.4. Experimental Setup

procedure has been repeated for eleven different FIR filters based on exact and approximate multipliers presented in[60],[61],[78],[91],[85].

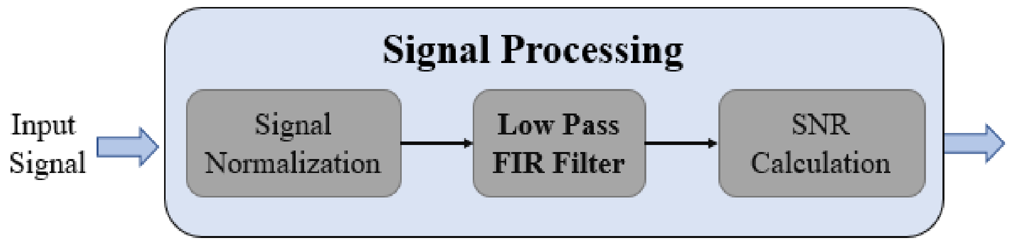
3.4.2 Finite Impulse Response filter structure

A fully-parallel 16- tap low-pass Finite Impulse Response (FIR) filter based on transposed form architecture, as shown in Fig.3.5 b) [100], has been implemented for the tactile signal processing in VHDL language for the Virtex-7 xc7vx485tffg1157-1 FPGA device. The equation of the studied FIR filter is presented as follow:

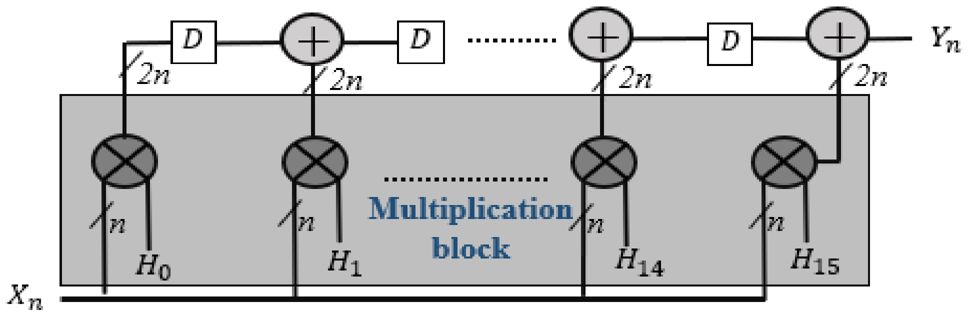
$$y(n) = \sum_{m=0}^{N-1} H(m) \times x(n-m) \tag{3.1}$$

where $H(m)$ are the filter coefficients, $x(n-m)$ is the noisy discrete signal sequence, $y(n)$ is the output filtered signal, and $(N-m)$ is the order of the filter. The coefficients of the filter have been extracted using MATLAB® through the Discrete Fourier Transform (DFT) with a pass-band and stop-band frequencies respectively equal to 775 Hz and 990 Hz (according to the input signals).

Input data has 8 bits 2's complement representation. The registers are put between the adders to increase the throughput of the circuit. The mentioned FIR filter has been selected for the application, since usually dedicated high speed parallel FIR filter with fixed coefficient meet the constraints of the application(real time performance and low power consumption). However, multiplications increase the complexity of the FIR filter, therefore approximate multipliers are adopted in the FIR filter in order to reduce the complexity of the system.



(a)



(b)

Fig. 3.5. a) . Functional block diagram for quality evaluation of FIR filter based approximate multiplier b) Design of 16-tap low pass finite impulse response filter using approximate multipliers.

3.4.3 Filtered output tactile data

The signal to noise ratio metric (SNR) has been employed in order to measure the quality of the filtered tactile signals. SNR of approximate filters has been computed, taking as a reference the exact filter (i.e., based on exact Baugh-Wooley multiplier) to assess the

impact of the approximate multiplier. Fig.3.6 presents a bar plot showing the SNR for the eleven filters. The SNR value in the best case is 23.39 dB.

Kulkarni [78] and ROBA [70] achieve the highest SNR; however, Approx-BW outperforms Kulkarni and ROBA respectively by a factor of 3× and 5× in terms of PDP (power delay product) (see in chapter 2) at the cost of less than 1.39 dB degradation in SNR with respect to the exact multiplier. The degradation is minimal considering the achieved reduction in power and time latency. Fig 3.6 shows that the SNR of the FIR filter based on Approx-BW is better than that of META[60], Shaf [91], and Evo0[85]. Approx-BW competes META[60], Evo0[85], and Shaf [91]by 3.69×, 2.84× and 2.46× in terms of

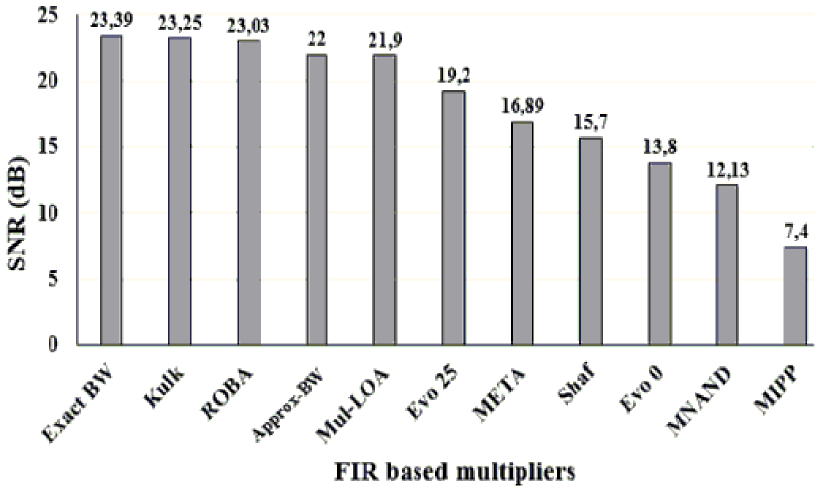


Fig. 3.6. Sorted signal-to-noise ratio for the exact and approximate multipliers.

PDP, respectively, as shown in chapter 2. On the other hand, Mul-AXA (3.06dB) and MAND (3.9dB) reached values far from being accepted for the target application. Such low SNR values indicate the distortion of the tactile signals. We conclude that not all approximate multipliers could be employed in the tactile sensing system.

Moreover, the filtered tactile signals through FIR filters based approximate multipliers are shown in Fig.3.7. It is shown in some cases the signal is wholly degraded, i.e. for MAND, MNAND, Mul-AXA, MIPP. For others, e.g., Shaf [91]and Evo0[85], the tactile signal generated reveals a distortion with respect to the signal generated from the FIR filter based on the exact multiplier. While with Approx-BW, the signal behavior is pretty similar to those generated with Kul [78]and exact BW [69].

To summarize, the FIR filter based on Approx-BW shows the best performance among the other approximate filters, respecting the tradeoff between accuracy and power consumption. Concerning the power consumption, Approx-BW achieves around 80% of power reduction at the cost of only 1.39 dB degradation in SNR with respect to the BW-exact multiplier when applied to FIR filters. Thus, we conclude that approximate computing techniques lead to several advantages when used in the signal processing stage of the tactile sensing system, i.e. reducing the power consumption, time delay, and area with minimal loss in quality.

3.5 Approximations in data processing

In the e-skin application, Machine learning plays an influential role in extracting meaningful information out of the proper amount of sensor data generated. Regrettably, a large number of operations such as multiplications are mainly executed in ML algorithms, which are the most power demanding arithmetic operations. Therefore, implementing approximate computing techniques for the CORDIC and SVD blocks will improve the energy efficiency of embedded ML algorithms adopted for e-skin; since CORDIC and SVD algorithms take a significant part in the real-time ML algorithm for tactile data processing. In this section, we will describe the architecture of the approximate CORDIC and SVD, discussing the improvements obtained in terms of power, latency, and performance after adopting approximate techniques for e-skin.

3.5.1 CORDIC Algorithm

In this section, we provide the circuit architectures and method of implementation of CORDIC for rotation mode[101].

CORDIC is an iterative algorithm which involves a series of shift-add operations for computing a very rich set of functions from the basic set of equations. CORDIC can be either operated in vectoring mode or in rotation mode. In vectoring mode, CORDIC rotates the input vector through whatever angle is necessary to align the resultant vector with the horizontal axis: the result of the vectoring mode operation is the rotation angle and the scaled magnitude of the original vector. In rotation mode, the angle accumulator is initialized with the desired rotation angle (Z). The rotation decision criteria (d_i) at each iteration diminishes the magnitude of the residual angle in the angle accumulator. The decision at each iteration is based on the sign of the residual angle after each step. The iteration equations are given by:

$$X_{i+1} = X_i - Y_i \times d_i \times 2^{-i}$$

$$Y_{i+1} = Y_i - X_i \times d_i \times 2^{-i} \tag{3.2}$$

$$Z_{i+1} = Z_i - d_i \times tan^{-1}(2^{-i})$$

Where:

$$\begin{cases} d_i = +1 \text{ if } Y_i < 0, -1 \text{ otherwise for vectoring mode} \\ \text{and} \\ d_i = -1 \; if \; Z_i < 0, +1 \text{ otherwise for rotation mode} \end{cases} \quad (3.3)$$

Which provides the following result:

- Rotation mode:

$$X_n = A_n[X_0 cosZ_0 - Y_0 sinZ_0]$$

$$Y_n = A_n[Y_0 cosZ_0 - X_0 sinZ_0] \quad (3.4)$$

$$Z_n = 0$$

- Vectoring mode

$$X_n = A_n \sqrt{X_0^2 + Y_0^2}$$

$$Y_n = 0 \quad (3.5)$$

$$Z_n = Z_0 + tan^{-1}(\frac{Y_0}{X_0})$$

3.5.2 CORDIC Circuits

CORDIC design uses a single Shift-Add operation for each component: x, y, and z, as shown in fig.3.7. A MUX (2:1 multiplexer), a shift register, and an adder/subtractor are required for each unit into CORDIC architecture. Before the beginning of CORDIC computation, three inputs X_in, Y_in, and Z_in are provided to the MUX. Then the computation will proceed by using the values stored in X_reg, Y_reg, Z_reg respectively. The micro-rotation angles arctan (2^(-i)) are stored into the ROM, where i is the input of the ROM and varies from 0 to 29 in this case. The FSM is responsible for tracking the shifting distance and enabling the multiplexer signals in order to control the ROM addresses. The FSM has three states $(s_0, s_1\ and\ s_2)$ which depends on three signals as follow: "reset", "start", and "count". Then, three outputs will be controlled such as: "init", "load" and "done" ,indicating the progress of the system during the runtime. If "reset" is set to 1, the FSM is at state s0 and the output "init" is set to 1. At this point, the registers X, Y and Z take their initialization value. For a rotation computation mode, the values are

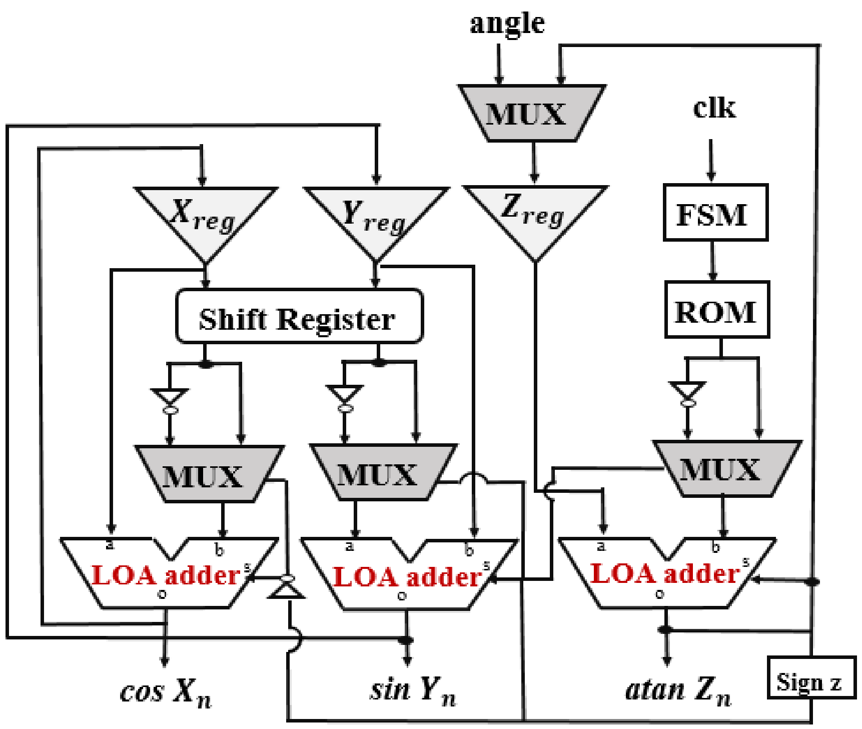


Fig.3.7 Architecture of the CORDIC in rotation mode

as follow: “X =0.6072,” “Y =0,” and “Z=angle” which is the desired angle of rotation. If “reset”and “start”are respectively equal to 0 and 1 , then FSM proceeds to the state s_1, so “load” will be set to 1 at the output. This step indicates that the system calculates the cosine and the sine of the input angle. The state remains at s_1 if the “count” signal stays different then the number of iterations. When “count” reaches the number of iterations, the FSM changes state and switches to s_2 where “done” is assigned to 1 indicating the end of the calculations. Therefore, the system will return to the initial state s_0 waiting for another computation when start will be set to 1 again.

3.5.2.1 Rotation mode

The accumulator angle is initialized to Z_0 which is considered as the desired angle of rotation. Based on (3.2) , the resulted operation is a vector (x', y') rotation of (x, y) by the angle of rotation Z_0. The rotation at each iteration diminishes the magnitude of the residual angle of Z. The sign of Z_{reg} determines the direction of rotations; if Z_{reg}>0 then the two adders components corresponding for X and Y will make an addition while the adder component related to Y operated as subtractor. Otherwise, all the operations will be inverted. Therefore, the direction of rotations is determined by the sign of the Zreg: if it is positive then the two “ADDER” for X and Z components operate as adders, while the one for Y operates as subtractor.

CORDIC in rotation mode could compute simultaneously the sine and cosine functions of the input angle. The derived equations after setting Y component to zero are as follow:

$$X_n = A_n . X_0 cos Z_0 \tag{3.6}$$

$$Y_n = A_n.X_0 sin\, Z_0$$

The rotation produces the unscaled cosine and sine of Z_0 after setting y_0 =0 and x_0=1/A_n where A_n= 0.6073. By adopting this method, the hardware complexity of the circuit is reduced by minimizing the number of required multiplications through the scaling factor 1/A_n.

3.5.3 Approximate CORDIC

As shown in Fig.3.7, CORDIC architecture relies on different adders; the objective is to reduce the power consumption of the CORDIC algorithm by substituting the adders by approximate ones. In this work, we aim to:

1) Select some of the most significant approximate adders presented in the state of the art such as: Approximate XNOR-based Adder (AXA)[77], Approximate NAND-carry out bit[86], Approximate AND-carry out bit[86], Input Pre-Processing[87] ,(LOA)[55] and Error Tolerant Adder (ETA)[61].

2) Evaluate the performance of the approximate adders by studying the accuracy, power consumption, time delay in order to select the optimal adder architecture to be implemented in the CORDIC algorithm.

3) Implement the approximate adders into the CORDIC circuit.

3.5.3.1 <u>Hardware implementation results</u>

The approximate adders have been implemented and simulated in Vivado Design Suite 2017.1 using VHDL Hardware Description Language. The power consumption and time delay have been reported after synthesizing the designs by using the Xilinx Vivado synthesizer, with a Virtex-7 xc7vx485tffg1157-1 device. In order to extract accuracy metrics, 10^5 input has been randomly selected based on uniform distribution. Moreover, the average dynamic power and the time delay have been determined for each adder implementation with a testbench of 3μs.

Among the architectures of different approximate adders described in chapter 2, the LOA and the ETA outperform the other adders having the lowest MED (45.61, 51.99) and NMED (0.07%, 0.08%) respectively. The maximum passing rates (31.18%) belong to the AND-carry out bit adder, as shown in Table 3.1.

Fig.3.8 represents the percentage of the outputs as a function of the relative percentage error distance. Upon simulation results, we observed that from 80.13% (approximate

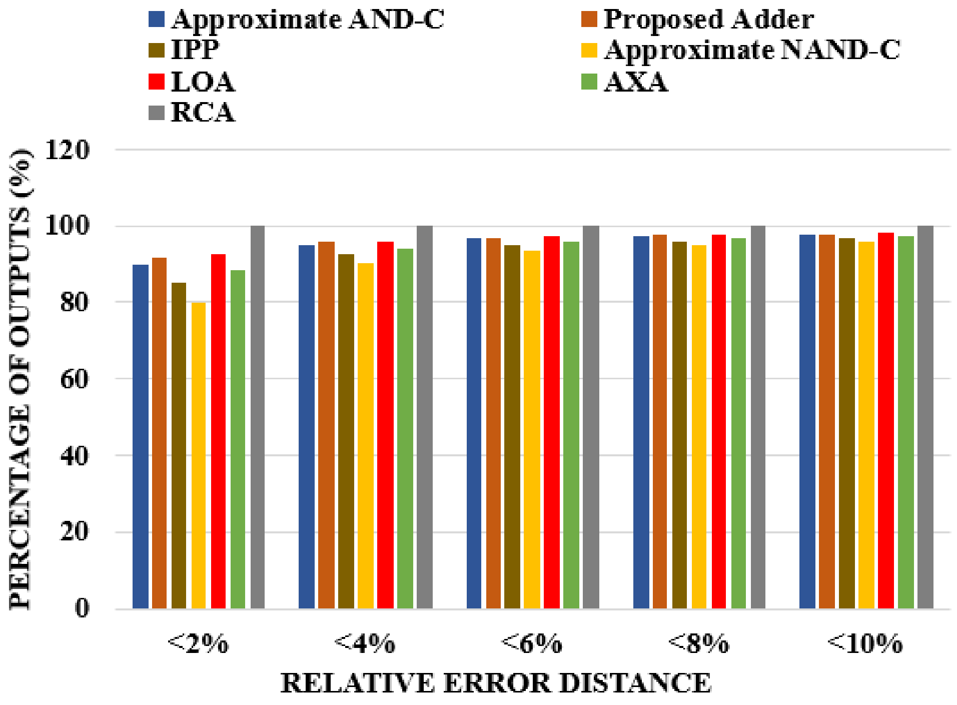


Fig. 3.8. Percentages of outputs versus relative error distance for different inexact adder circuits

Table 3.1. Accuracy Metrics For Different Approximate Adders Designs

Approximate Adders	MED	MSE	NMED	MRED	Pass Rates
ETA	51.99	7.8E+04	0.08	0.015	8.07%
LOA	45.61	4.6E+04	0.07	0.015	8.07%
AXA	75.46	1.1E+04	0.11	0.05	0.01%
NAND-C	75.6	6.6E+04	0.12	0.044	0.02%
AND-C	60.67	3.7E+04	0.09	0.028	31.18%
IPP	89.7	6.8E+04	0.13	0.031	21.77%

NAND-carry out bit) to 92.47% (ETA) of the outputs show a relative error distance of less than 2%. Hence, less than 7% (i.e., Approximate NAND in the worst case) of the outputs are characterized by a relative error distance of more than 6%.

Moreover, the power consumption and time delay of the approximate adders with respect to MRED have been evaluated, as shown in figure 3.10. We notice that ETA and LOA adders (which have the lowest MRED of 1.52%) have also the lowest power consumption (24 mW, 21 mW) since carry propagation has been omitted. While AXA has the worst MRED (4.95%) with higher power consumption (34 mW). We conclude that LOA and ETA are considered as the most optimal approximate adders to be implemented into CORDIC. Therefore, we have implemented ETA and LOA instead of the entirely precise adder (RCA) into the CORDIC algorithm aiming to study the performance of CORDIC in terms of slices, power, and time latency after employing approximate circuits. ETA and LOA in the CORDIC design allow respectively a dynamic power consumption saving up to 13% and 21% with respect to CORDIC-RCA, as shown in Table 3.2. Moreover, we

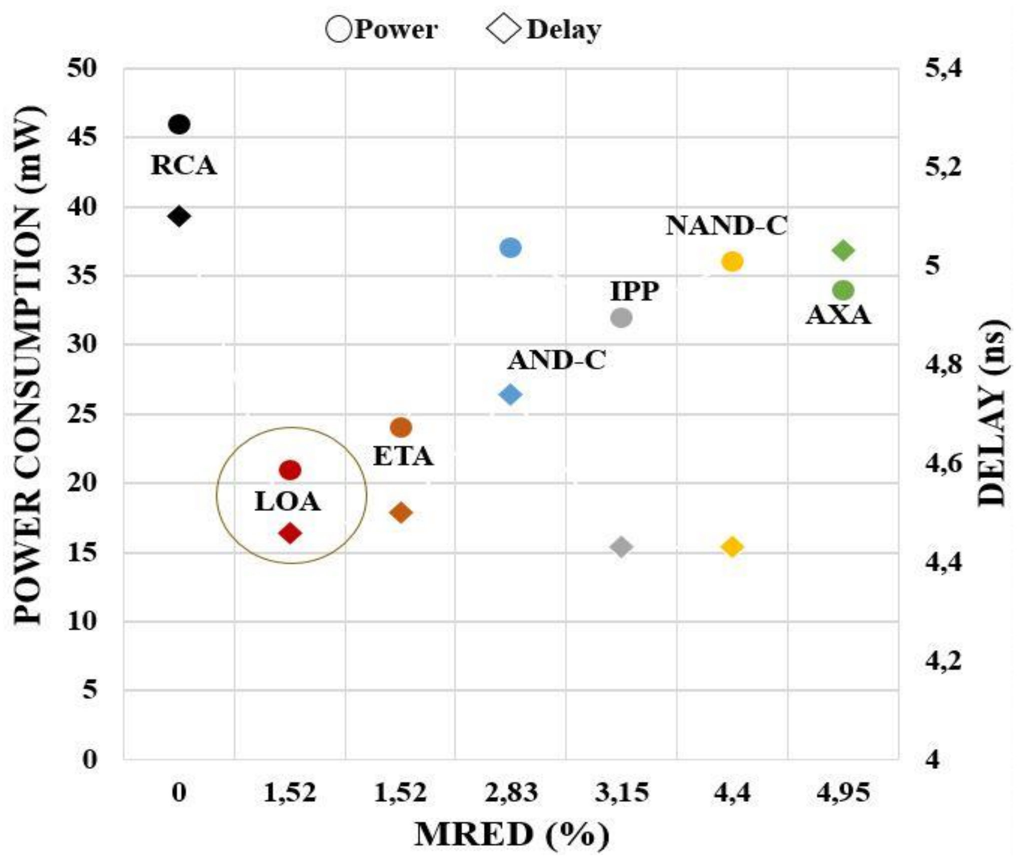


Fig.3.9 Power consumption, delay and MRE of exact and approximate adder designs

notice that the power consumption of CORDIC-LOA will be reduced by up to 11% with respect to CORDIC-ETA, after selecting LOA instead of ETA. We conclude that ETA will decrease the power by 13%, but in the case of selecting LOA, the power will be decreased by up to 21%with respect to[98] and [102].

Table 3.2. Full adder versus approximate adders based CORDIC circuits

Word length 32 bits	**CORDIC-RCA**	**Reference[101]**	**CORDIC-ETA[98]**	**CORDIC-LOA**
Registers	135	135	134	130
Power (mW)	92	85	80	72
latency (ns)	153	110	130	127

3.5.4 Singular Value Decomposition Algorithm

In algebra, the Singular Value Decomposition (SVD) is considered as a matrix factorization method employed to analyse the structure and properties of a particular matrix. Moreover, the SVD generates the least square solution computation of a system since the SVD could produce a complete orthogonal decomposition. The definition of the SVD is explained in details below:

A real matrix A of size m×n could be represented as a product of three matrices as below:

$$A = USV^T \tag{3.7}$$

Where the matrices are defined as follow:

- Orthogonal matrix U(m×m): $U^{-1} = U^T$. The generated vectors $(u_1, u_2, \dots, u_m)$ from the U columns are considered as an orthonormal base for R^mspace and u vectors are called "left singular vectors."
- Orthogonal matrix V(n×n): $V^{-1} = V^T$. The generated vectors $(v_1, v_2, \dots, v_m)$ from the V columns are considered as an orthonormal base for R^mspace and v vectors are called "right singular vectors."
- Matrix S(m×n). The main diagonal contains the singular values such as:

  $$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0 \tag{3.8}$$

  These obtained values are expressed in decreasing order while the other values are set to zeros. Therefore, if p=rank(A), then p≤min(m,n) and the singular values greater than zero are as follow: $\sigma_1, \ldots, \sigma_n$. Then, the following equation is generated:

$$S=\begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \sigma_n & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix}=\begin{bmatrix} S_n & 0 \\ 0 & 0 \end{bmatrix} \tag{3.9}$$

Where

$$S_n = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_p \end{bmatrix} \tag{3.10}$$

Based on (4.17), the SVD product could be expressed as follow:

$$A = [\, u_1\ u_2 \ldots u_m] \begin{bmatrix} S_n & 0 \\ 0 & 0 \end{bmatrix} [\, v_1\ v_2\ \ldots\ v_m]^T = [\, U_n | U_0] \begin{bmatrix} S_n & 0 \\ 0 & 0 \end{bmatrix} [\, V_n | V_0]^T = U_n S_n V_n^T$$

Where U_0 and V_0 are considered as the left and right eigenvalues, respectively, with respect to the null eigenvalues. However, if n is considered as the matrix rank then the first p singular values will be different from zero. In this case, the first n U and V column vectors which are considered as the submatrices (U_n , V_n), will have a role in the matrix decomposition as shown below:

$$U = [\, u_1\ u_2 \ldots u_n |\ u_{n+1}\ u_{n+2} \ldots u_m\,]; \tag{3.11}$$

$$V = [\, v_1\ v_2 \ldots v_n |\ v_{n+1}\ v_{n+2} \ldots v_p\,]; \tag{3.12}$$

In case U_n and V_n are written as their respective column vectors, then the SVD product will be written as below:

$$A = U_n S_n V_n^T = \sum_{i=1}^{n} U_n u_i v_i^T = \sum_{i=1}^{n} \sigma_i E_i \tag{3.13}$$

Where each eigenvalue is multiplied by the matrix $E_i = u_i v_i^T$, having a size of (m×n).

Different procedures exist for determining the SVD of a given matrix A. One of them which is referred to the matrix $A^T A$ is presented below:

- The first n vectors v_i are expressed as the $A^T A$ eigenvectors for the corresponding $\sigma_i^2 \neq 0$ eigenvalues. While the remaining $v_{n+1}\ v_{n+2} \dots v_m$ vectors are computed through the orthogonalization process.
- The first n vectors $u_i$are generated based on the following property:

$$Av_i = USV^T v_i = USe_i = U\sigma_i e_i = \sigma_i u_i \tag{3.14}$$

Where $e_i = [0 \dots 010 \dots 0]^T$and one is positioned at the ith row. Also, the remaining (m-p) vectors will be computed through the orthogonalization process.

On the other hand, another method could be employed to compute the SVD, called “dual one.” It is responsible for building the u_i vectors starting from $A^T A$ and v_i.

To summarize, these methods described previously could be applied only for matrices having a small size and for a small condition number. However, the accuracy loss will be significant after increasing the size of the matrix. Therefore, a more interesting algorithm proposed by Carl Gustav Jacob in 1846 will solve this issue mentioned previously. This algorithm is called the “ Jacobi algorithm.”

3.5.4.1 One-Sided Jacobi Algorithm

One-sided Jacobi is an algorithm that consists of applying a sequence of rotations to an initial matrix (i.e., matrix A). Then the diagonal matrix S could be reached. Eventually, Jacobi generates a sequence as follow $(\mathrm{A}_1, \mathrm{A}_2, \dots . \mathrm{A}_\mathrm{n})$, which usually converge to a diagonal matrix, having the eigenvalues on the diagonal. The transformation is applied to matrix A through the following formula:

$$\mathrm{A}_{\mathrm{m}+1} = J(\mathrm{m}, \mathrm{n}, \alpha)^\mathrm{T} \mathrm{A}_\mathrm{m} J(\mathrm{m}, \mathrm{n}, \alpha)^\mathrm{T} \qquad (3.15)$$

Where $J(\mathrm{m}, \mathrm{n}, \alpha)$ is called a Jacobi rotation, which is equal to the identity matrix I with four additional elements on the intersection of rows m and columns n. Then, the Jacobi is computed for each 2×2 sub-matrix in order to annihilate the off-diagonal elements of the matrix A. The rotation matrix J will be constructed such that the w elements will be annihilated, by following the equation below:

$$\begin{bmatrix} \hat{x} & 0 \\ 0 & \hat{y} \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}^T \begin{bmatrix} \mathrm{x} & \mathrm{w} \\ \mathrm{w} & \mathrm{y} \end{bmatrix} \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \qquad (3.16)$$

Where $\hat{x}$ and $\hat{y}$ are the diagonal elements of the 2×2 matrix related respectively to the following two elements x and y after applying the rotation to the corresponding angle. Indeed, through the one-sided Jacobi algorithm, high accuracy is achieved, followed by a fast convergence. In fact, five to ten iteration is required by the algorithm in order to achieve the convergence; thus, the time and the number of resources to be employed will be reduced in this case.

3.5.5 SVD Circuits

In the following section, we will describe the hardware implementation of the main blocks of the SVD: matrix symmetrization, phase solve and Pre-, Post-rotations.

The matrix input of the SVD must be symmetrized through the matrix symmetrization function in order to obtain a square matrix. It consists of multiplying the input matrix by its transpose as follow:

$$\mathrm{U_n} = A^T A \tag{3.17}$$

Then, the resulted symmetric matrix will be stored in the memory, as shown in fig.3.11. After the symmetrization phase, the computation will start with the four elements of the pair (a, b), which have been stored in the memory. Then, the CORDIC IP from Vivado

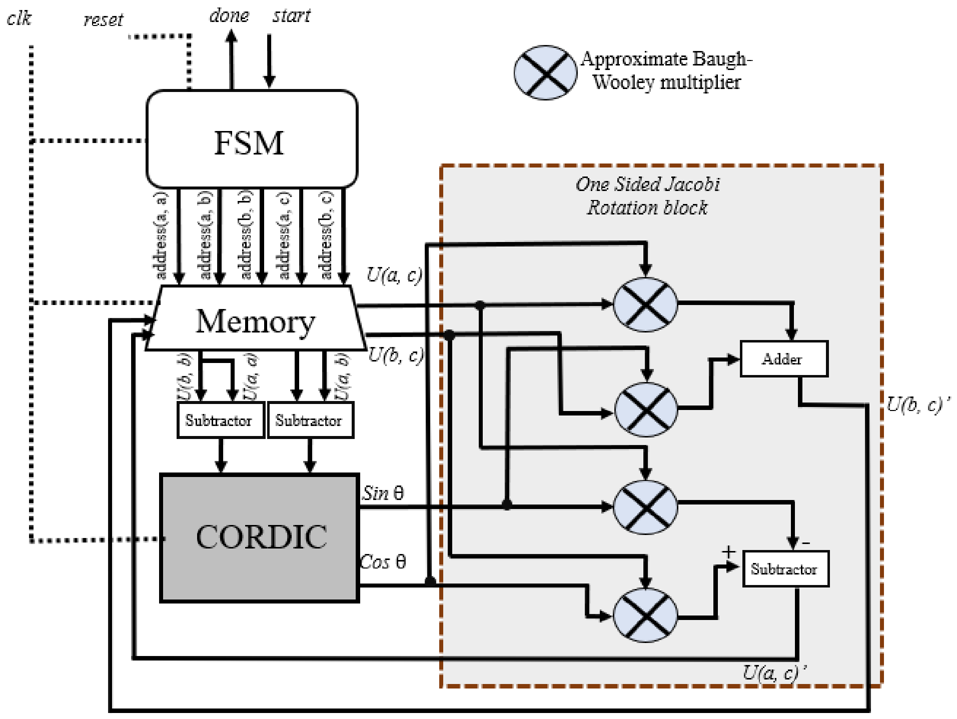


Fig.3.10 Approximate SVD block diagram

2017.1 1 is employed in order to compute the angle of rotation in vectoring mode, while the Sine and Cosine are computed in the rotational mode. Then, the pre and post-rotations blocks are responsible for rotating the rows of U and the columns of U and V, respectively, using the one-sided Jacobi block. These rotations are based on the sine and the cosine functions, which have been calculated by the previous phase solver block. However, a reliable stopping criterion must be defined in order to ensure that the algorithm will arrive at the convergence while reducing time and power operations.

3.5.6 Approximate SVD

3.5.6.1 <u>Accuracy Analysis</u>

As shown in Fig.3.10, four multipliers are needed for the pre-rotation and post-rotation blocks, respectively. The number of operations generated from the multiplication takes a high amount of power. Therefore, the focus in our work is reducing the power consumption of the SVD by implementing approximate multipliers instead of the exact multiplier. As shown in fig.3.11, the approximate Baugh-Wooley multiplier is implemented in the architecture of the SVD. Only adders and subtractors are kept correct. The scalability of the approximate multiplier has been assessed into the SVD by approximating eight from

Table 3.3 Percentage relative error of Approximate SVD

Eigen-values	**Relative Error (%)**					
	SVD-approx8	**SVD-approx12**	**SVD-approx16**	**SVD-approx20**	**SVD-approx24**	**SVD-approx28**
S1	0	0	0.018	0.45	4.89	26.24
S2	0	0	0.097	3.33	36.57	93.77
S3	0	0.054	0.27	8.27	72.32	99.89
S4	0	0.31	0.62	10.14	73.79	99.68
S5	0	0	100	0	100	0

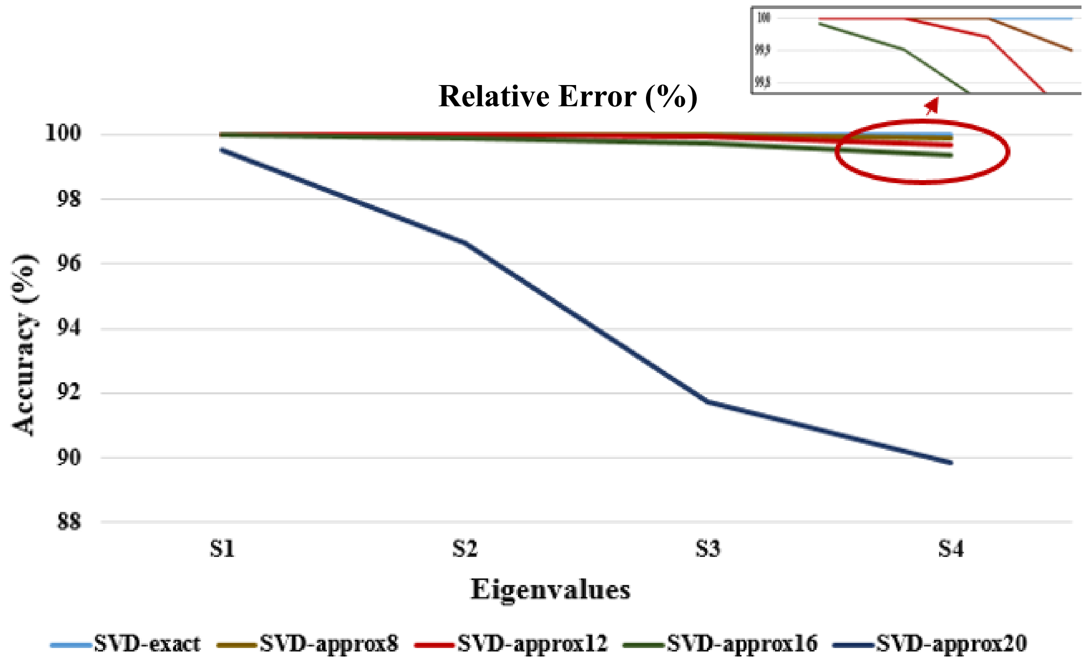


Fig.3.11. Accuracy of the eigenvalues of the approximate SVD for an input matrix of size (5×5).

the Least Significant bits (LSB), then increasing the number of approximated bits reaching 20 approximated LSB's. Experiments have shown that for an input matrix of size 5×5, the accuracy of the eigenvalues of the SVD (S1, S2, S3, and S4) remain higher than 99% after approximating 16 LSB's as shown in Fig. 3.11. While the accuracy of the eigenvalues decreases from 99% to 90% after approximating 20 LSB's (which is still considered acceptable for our application). Moreover, the relative error has been evaluated for all the eigenvalues for different configurations of approximated LSB's as shown in Table3.3. The eigenvalues of the SVD-approx24 and SVD-approx28 reach a relative error of around 36% to 99%, thus concluding that the number of approximated LSB's in approximate BW should not exceed 20 bits. Moreover, the performance of the approximate SVD has been studied for an input matrix of size 8×8, resulting in eight eigenvalues from the SVD.

Table 3.4: Relative error of the eigenvalues resulted from the approximate SVD for an input matrix of size (8 ×8).

Eigen-values	**Relative Error (%)**				
	SVD-approx8	**SVD-approx12**	**SVD-approx16**	**SVD-approx20**	**SVD-approx28**
S1	0	0	0.01	0.28	4.3
S2	0	0.10	0.21	4.58	33.37
S3	0	0	0.49	5.93	55.98
S4	0	0.13	9.27	7.82	83.58
S5	0	3.41	0.35	16.12	98.08
S6	0	3.36	7.22	27.1	98.41
S7	0	1.68	27.71	18.79	49.87

Fig.3.12 and Table 3.4 represent, respectively, the accuracy and the relative error of the eigenvalues generated from the SVD. As shown in Table 3.4, when increasing the number of approximated LSBs till 20, the relative error of the eigenvalues (S1, S2, S3, S4, S5, S6, and S7) will not exceed 27% just for two eigenvalues (i.e., S6 and S7); while most of the

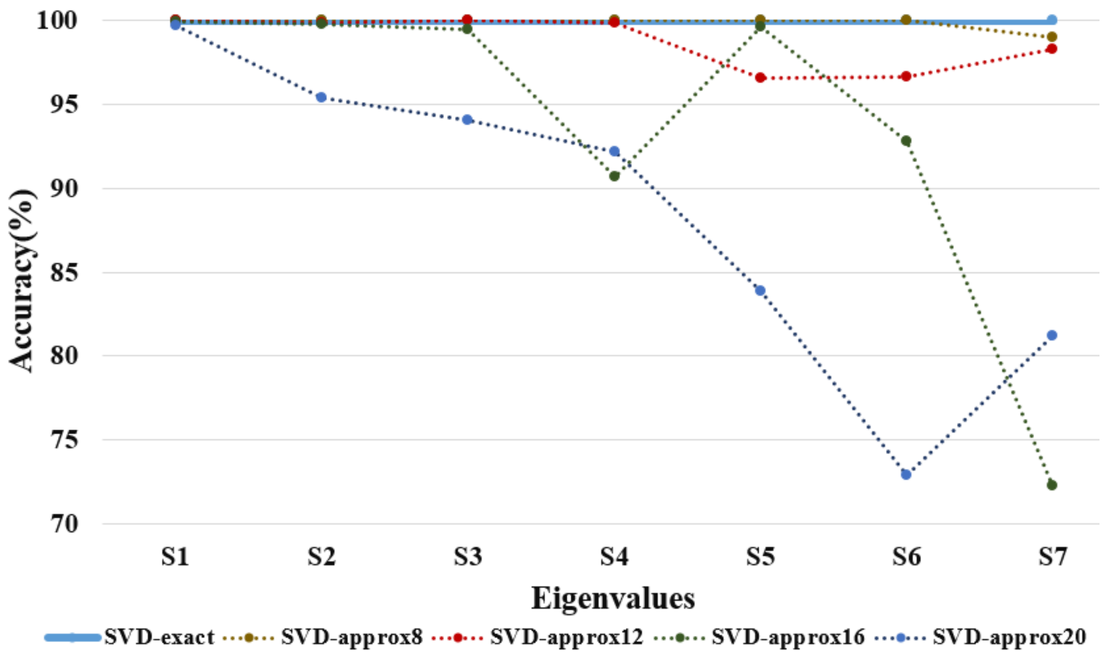


Fig.3.12. Accuracy of the eigenvalues of the approximate SVD for an input matrix of size (8×8).

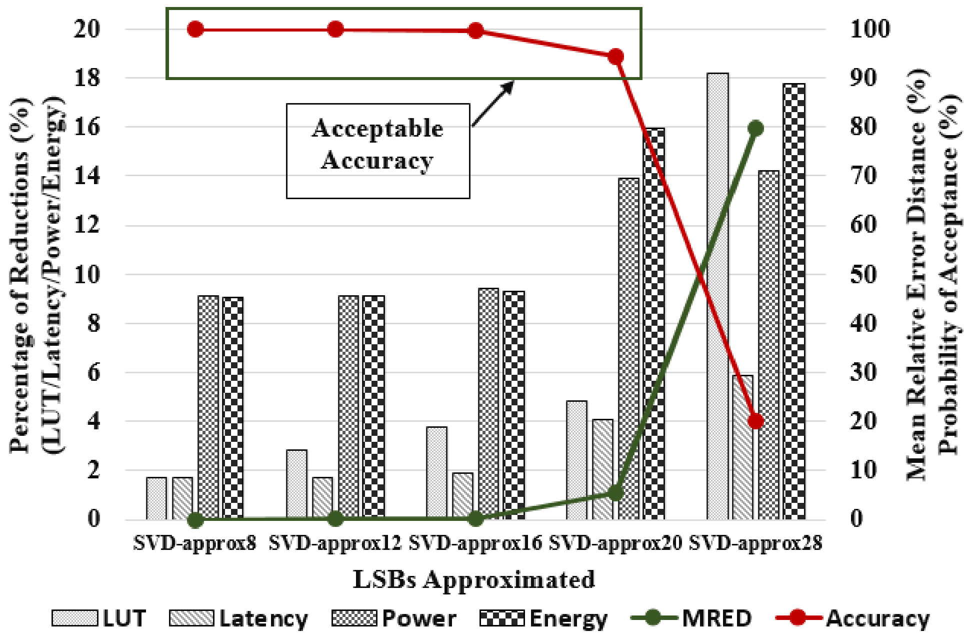
eigenvalues have a relative error less than 10%. However, when approximating 24 LSB's of the approximate BW multiplier, the relative error of the eigenvalues will remain high, exceeding 50%.

3.5.6.2 Power/Energy Consumption, LUT, Latency Analysis

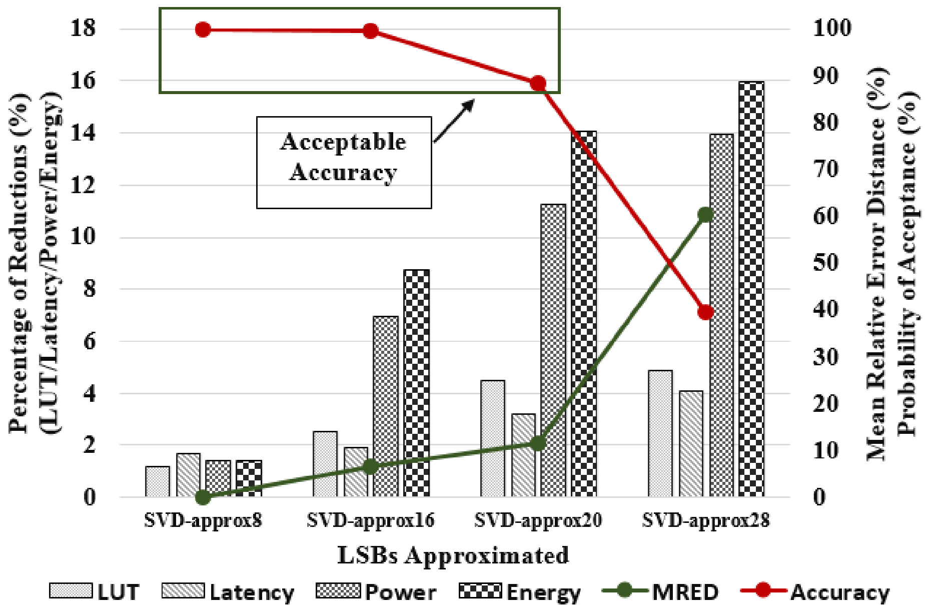
In this subsection, we analyse in detail the percentage of reduction of the power/energy consumption, latency, and LUT utilization of the approximate SVD. Fig. 3.13 illustrates, respectively the results of the experiment for two input matrices of size respectively equal to (5×5) and (8×8). The x-axis represents the numbers of output LSBs approximated in the SVD. While, y-axis represents the percentage of reduction of power/Energy consumption, LUT, and Latency with respect to the accurate SVD based on Exact-BW. Moreover, the y-axis denotes the probability of acceptance and the MRED of the resulted eigenvalues; indicating the range of acceptable accuracy. Based on these experiments, we notice that:

- The Power/energy consumption, LUT utilization, latency will be reduced when increasing the number of approximated LSBs. For example, the power and the energy consumption have been reduced respectively by up to 14% and 16%, as shown in Fig.3.13, after approximating 20 LSBs.
- The MRED of the resulted eigenvalues dramatically decreases after approximating more than 20 LSBs of the SVD, as illustrated in Fig. 3.13. For example, the MRED in Fig 3.14 has been increased drastically to 80% after approximating 28 LSBs, which is considered out of the range of accuracy acceptance.

Therefore, we conclude that it is possible to approximate up to 20LSBs of the SVD, achieving an energy reduction of 16%.



(a)



(b)

Fig. 3.13. Performance and Error resilience analysis of the Singular Value Decomposition for an input matrix of size (5×5) (a) and (8×8) (b)

We conclude that approximate computing techniques are considered as a promising approach to be employed for the data processing stage of the tactile sensing system, aiming to improve the energy efficiency of the overall e-skin application.

3.6 Conclusion

This chapter assesses the impact of approximate computing techniques on the tactile sensing system, which is composed of two main blocks (signal and data processing blocks). Therefore, we implement the proposed approximate arithmetic circuits and some relevant state of the art approximate circuits in the tactile sensing system: aiming to understand the behavior of the target application when enabling approximations techniques. The quality is measured in terms of different metrics, mainly: SNR degradation, MRED, NMED, power consumption, PDP, and time delay. Thus, based on the methodology employed for the approximate tactile sensing system, we implemented different approximate multipliers on low-pass Finite Impulse Response (FIR) filter in the first stage from one side and approximate adders and approximate multipliers respectively in the CORDIC and SVD blocks in the second stage from another side. Results prove that the FIR filter based on the proposed Approx-BW outperforms state of the art solutions, respecting the tradeoff between accuracy and power consumption. Concerning the power consumption, Approx-BW achieves around 80% of power reduction at the cost of only 1.39 dB degradation in SNR with respect to exact and another relevant state of the art multipliers when applied to FIR filters. Moreover, we improve the power consumption of embedded machine learning after implementing approximate arithmetic circuits into the Coordinate Rotational Digital Computer (CORDIC) and the Singular Value Decomposition (SVD) circuits, which take a significant part of the real-time ML algorithm for tactile data processing. The power consumption of CORDIC and SVD has been reduced respectively by 21% and19% at the cost of less than 5% accuracy loss after scaling the number of the approximate bits.

This study demonstrates the feasibility of the proposed approach based on applying approximate circuits in the signal and data processing blocks of the e-skin system. Thus, we conclude that approximate computing techniques lead to several advantages when used for a tactile sensing system, i.e. reducing the power consumption, time delay, and area with minimal loss in quality.

CHAPTER 4. MACHINE LEARNING ALGORITHMS FOR TENSORIAL TACTILE DATA PROCESSING

4.1 Introduction

For the aim, to improve the interaction between humans and robots, different technologies have been developed for tactile systems. Therefore, several processing methods have been suggested in order to extract meaningful information from sensor data generated from a sensitive skin[103],[99]. Interpreting sensor data has been proved through different pattern recognition methods, which is considered as a challenging task in e-skin applications (e.g., classification of shapes and patterns [104],[105]). Hence, Machine Learning (ML) provides an efficient solution for tactile sensing systems. In[106], the authors employed the k-nearest neighbor (K-NN) for the haptic interface, where five touch modalities are recognized. In [107], eight different touch modalities are recognized through a " LogicBoost" implementation. Then, authors in [108] have suggested a neural network's model called " modified counter propagation," which is designed for discriminating between ten touch modalities when recognizing tactile data. Flag in [109] has recognized three gestures after employing a fur supported by a touch sensor. Authors in [110] exploited the Support Vector Machines (SVM) to recognized affectionate behaviors. In[111], a K-NN is implemented for the biometric system.

In this work, two main aspects are addressed to interpret tactile data through ML algorithms, as described in [4]. The first aspect is dedicated to mapping the variation of stimuli time extracted from a two-dimensional sensor array. Therefore, a tensor-based on tactile signals morphology is suggested. Then, feature extraction is needed to map the signal in the form of tensors into multi-dimensional vectors, while the second aspect deals in recognizing the specifications of the tactile sensing system.

Moreover, implementing supervised learning tools will be considered as a challenging task to be employed for the tactile sensing system. The critical step is to attain a reliable generalization, which predicts the data excluded from the training set correctly. Therefore, the ML framework designed for the e-skin application can handle the problem of learning under noisy signals.

In this chapter, we will describe the framework of ML algorithms employed for the tactile sensing system, applying the classification for images as a case study and on touch modalities for e-skin application. The aim of this chapter is to validate the effectiveness of the SVM based tensor kernel algorithm for touch modalities classification before exploiting the embedded implementation of the ML on the low power platform, as it will be described in chapter 5.

4.2 Tactile data based on the tensorial approach

The electronic skin, which has a dimension of 2D, is considered as the main component of the tactile sensing system. Usually, the sensors in the tactile systems lie into typical grids at specific positions forming a network into the skin based on piezoelectric or capacitive transducers. Each cell in the skin area is composed of several sensor cells responsible for processing the data generated from the multiple sensors. Moreover, another existing technique which is called “imaging technique” can process the generated information after

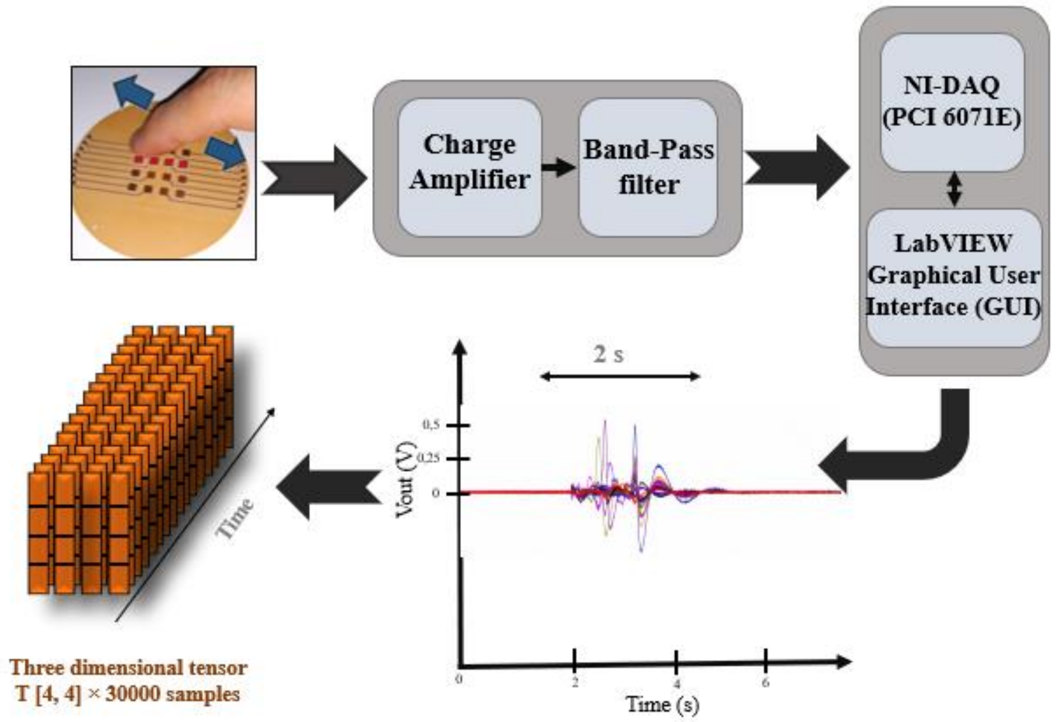


Fig.4.1. Schematic of tactile acquisition system

performing some measurements on the electrodes located at the edge of the conductance sheet.

Hence, the collected sensors data could be organized into the form of tensor, where the tactile images represented in the form of 2D are structured in the time domain, as shown in Fig. 4.1. Thus, it is estimated that the described framework based on pattern recognition will handle the challenging task by exploiting the tensorial structure in our application.

4.3 ML approaches for touch recognition

The two main reasons for choosing the ML approach are as follows. Firstly, ML techniques can predict and make decisions on unknown input samples. When dealing with sensor data, attaining an input-output relationship is considered severe. Then, the input-output function will be modelled by ML through the “learning from examples” approach. Secondly, through the presented framework in[112], the learning machine will be extended from the kernel method to the tensor-learning model.

4.3.1 Pattern recognition based on kernel methods

Generally, the decision function could be identified through classification methods after minimizing the error of the classification accuracy in the problem domain. For this aim, an optimization supporting the trade-off between the regularized term and the empirical risk is required for the training methods. The decision function f resides to the reproducing Kernel Hilbert Space (RHKS), which takes advantage of the “Kernel Trick.” The computation takes place based on the following kernel function $K(x_i, x_j)$ which includes the product of x_i and x_j Patterns. Then, the decision-making flow will be mapped into another space where linearity could be applied.

In this subsection, we will describe the well-known learning paradigms, which are: Support vector machines (SVMs) and Kernel-based Extreme Learning Machines (K-ELM).

4.3.1.1 Support Vector Machines

The Support Vector Machine (SVM) is a supervised learning model, designed and proposed in 1995[113]for classification and regression purposes. SVM aims to classify each dataset in different categories through a model built from a set of training examples. Usually, the classification is executed after finding the best separation hyperplane, which divides the two specific classes with a maximum gap, as shown in Fig. 4.2.

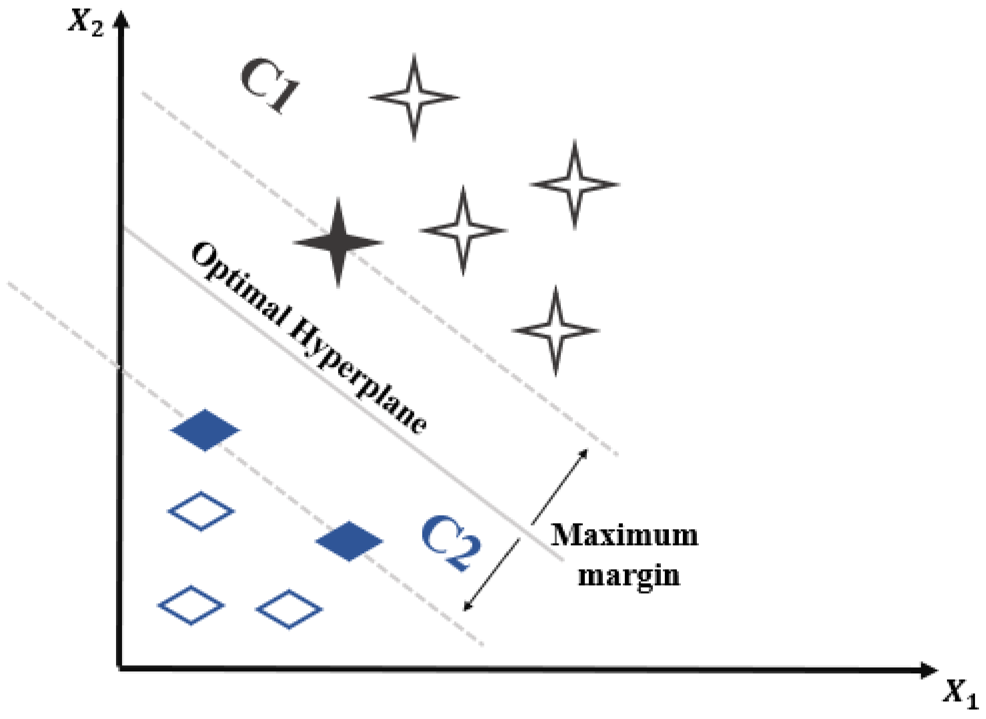


Fig.4.2. SVM linear case

The classification consists of determining a function that is defined as a plane Π in the linear case, where C1 and C2 are divided into the data space [114]. Two different labels are marked to the two classes as follows: $y = +1$ for C1 and $y = -1$ for C2.

The separation area between the C1 and C2 points is characterized after defining π_a and π_b as two hyperplanes having the same normal w. Then, the hyperplane separation $\pi\,(\mathrm{w}, \mathrm{b})$ is considered as the halfway separation from the two support hyperplanes. Therefore, the two following inequalities could be written below:

$$\begin{cases} w^T x_i + b \geq +1 \;\; x_i \,\epsilon\, C1(y_i = +1) \\ w^T x_i + b \leq -1 \; x_i \,\epsilon\, C2(y_i = -1) \end{cases} \tag{4.1}$$

Based on (4.1), the following property could be derived:

$$y_i(w^T x_i + b) \geq 1 \ \forall x_i \in C1 \, UC2 \qquad (4.2)$$

When targeting the linear case, the separator should have the following characteristics:

- Maximizing the distance ($d = 2/\|w\|$) between the two hyperplanes π_a andπ_b, or minimizing$\|w\|$.
- The two hyperplanes should be able to surpass through $x_a \in C1$ and $x_b \in C2$, respectively.

Then, the parameter b could be determined as follow:

$$\begin{cases} w^T x_a + b = +1 \\ w^T x_b + b = -1 \end{cases} \Longrightarrow \ w^T(x_a + x_b) + 2b = 0 \Longrightarrow \ b = -1/2 w^T(x_a + x_b)$$

Thus, the primal problem could be presented based on the following equation:

$$min_{w,b}\left(\frac{1}{2}\|w\|^2\right) \qquad (4.3)$$

With

$$y_i(w^T x_i + b) \geq 1 \ \forall x_i \in C1UC2(\text{i=1,....,m}) \qquad (4.4)$$

Then, the following equation could be written as below:

$$max_\alpha \left[\Gamma_D(\alpha) = -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{m}\alpha_i\right] \qquad (4.5)$$

$$\begin{cases} \sum_{i=1}^{m}\alpha_i y_i = 0 \\ \alpha_i \geq 0 \end{cases} \qquad (4.6)$$

After obtaining the α vector, the separator could be obtained as follow:

Since $w = \sum_{i=1}^{m} \alpha_i y_i x_i$ then

$$w^T x = \sum_{i=1}^{m} \alpha_i y_i (x_i, x) \tag{4.7}$$

(4.7) is considered as the product between the current input with the known data (m). Then, the term b could be generated as follow:

$$b + w^T x_i - y_i = 0 \tag{4.8}$$

$$b = y_i - w^T x_i \tag{4.9}$$

Therefore, the classification function will be generated as below:

$$Class = sign(f(x) = w^T x + b) \tag{4.10}$$

However, (4.10) is issued based on the assumption that the classes are classified linearly. While this assumption could not be accurate, then the two classes C1 and C2 could not be able to be separated linearly, which means that the hyperplane could not be employed to separate the two classes.

Therefore, Vapnik, Boser, and Guyon have proposed an algorithm to solve this problem mentioned previously by creating a nonlinear classifier using the kernel trick, which will be presented in section 4.3.2.

4.3.1.2 *Kernel-based Extreme Learning Machines*

The ELM model is employed for feedforward neural networks and kernel-based methods. In the training phase, the K-ELM[115] is based on the following equation:

$$(K + \mu I)\beta = y \tag{4.11}$$

Where μ is the parameter for regularization and K is the kernel matrix, including the following elements which are defined below:

$$K(a_i, a_j) = h(a_i).h(a_j) \tag{4.12}$$

Where $h_1(a) = G(b_l, c_l, a)$. In this equation, the two parameters b_l and c_l are randomly extracted through a probabilistic continuous distribution.

4.3.2 Exploiting kernel functions for tensorial approach

When dealing with ML for a tactile sensing system, a feature extraction step is required for converting the signals generated in the tensorial form to multidimensional vectors. This fact could lead to some information loss in the structure of the signal generated initially. Moreover, authors in[112],[116]have demonstrated that signals could be characterized adequately when dealing with a tensorial form. Also, several benefits have resulted when using learning methods that assess the information embedded in the tensorial form. Therefore, the two kernel methods presented in the previous section could be protracted to the tensor-learning approach. In the following subsections, we will describe the required steps to get a general kernel entry $K(i, j)$ based on the introduced framework in[4].

4.3.2.1 <u>Tensor Unfolding:</u>

The system requires an input of three-dimensional tensors, where the first two dimensions are the data received from the sensor array, and the third dimension is considered as the time sample. During the data processing phase, the main goal is to rearrange the received

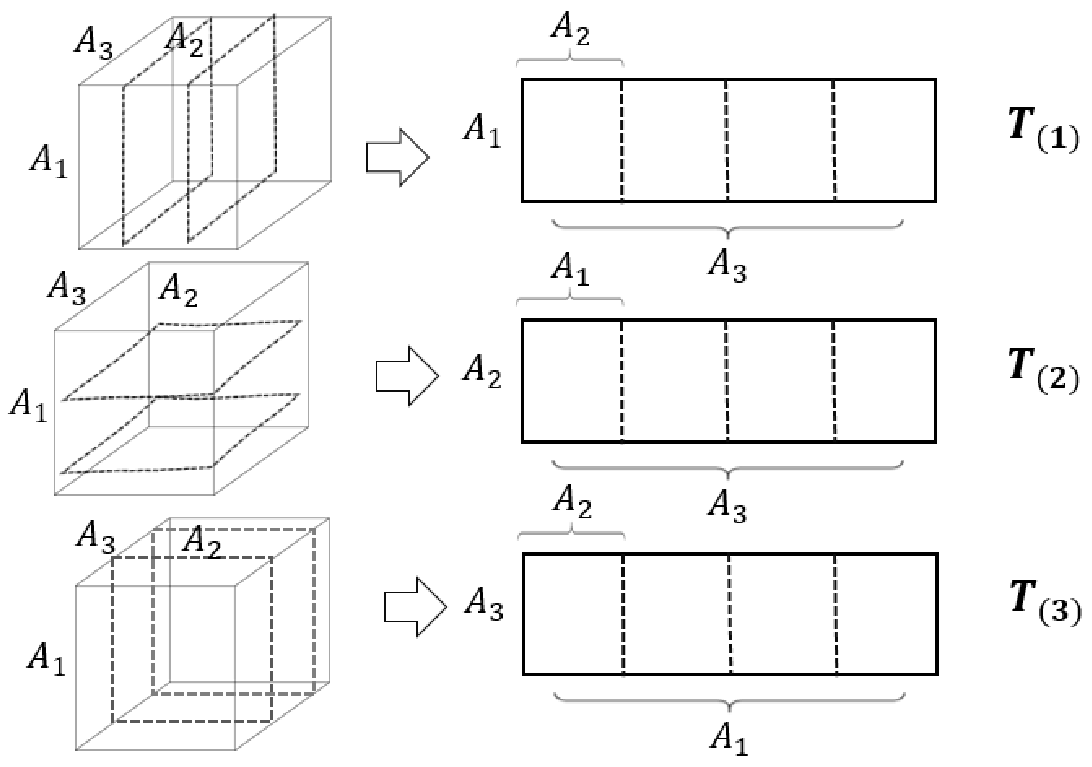


Fig.4.3. Tensor unfolding

data in two-dimensional matrices without losing any information. This process is called "tensor unfolding," where the results are represented in the form of matrix representation unfolded into three matrices. The first matrix contains the column information, i.e., all the column vectors are stacked one after the other, while the second matrix contains the rows information (i.e., all rows are stacked one after the other) and the final matrix represents the information brought along the third tensor dimension. The definition of *tensor unfolding* is explained as follow:

- For an N^{th} Order tensor $T\epsilon\ C^{A_1 \times A_2 \times \ldots\ldots A_n}$, the unfolded matrix $T_{(n)}\ \epsilon\ C^{(A_n \times A_{(n+1)} \times \ldots\ldots A_{(n-1)})}$ contains the element $t_{a1a2\ldots.aN}$ at the row position number a_n and for a column number equal to: $(a_{n+1}-1)a_{n+2}a_{n+3}\ldots.a_n a_1 a_2 \ldots.a_{n-1}+(a_{n+2}-1)a_{n+3}a_{n+4}\ldots a_n a_1 a_2 \ldots a_{n-1}+\cdots a_{n-1}$

Thus three unfolded matrices result from the third-order tensor are as follow: *T(1)*ϵ $C^{A_1 \times (A_2 A_3)}$, *T(2)*ϵ $C^{A_2 \times (A_1 A_3)}$, *and T(3)*ϵ $C^{A_3 \times (A_1 A_2)}$. An example of an unfolded tensor is shown in Fig. 4.4.

4.3.2.2 Symmetrization

After obtaining the unfolded matrices, the symmetrization phase aims to symmetrize the matrices by performing a matrix multiplication as follow:

$$T_{sym} = T^{Trans} T \tag{4.13}$$

Where T is the unfolded matrix. In other terms, the symmetric matrix is equal to the transpose of the square matrix.

4.3.2.3 Kernel Trick

The main difference when employing the kernel trick is the replacement of each dot product in the SVM algorithm by a non-linear kernel function, as shown in Fig. 4.4. By applying this trick, the problem will be mapped in a transformed feature space. In this way, the problem will remain linear again, and then the regular linear classifier could be employed, as described in the previous section. In the state of the art, different possible kernels have been presented, but in our work, the suggested trick in [4] will be employed, which is a Gaussian kernel function designed for tensor-based models. The two tensors (T_i, T_j) are processed through the following kernel equation:

$$K(T_i, T_j) = \prod_{n=1}^{N} K^n(T_i, T_j) \tag{4.24}$$

Where K^n is the computed kernel factor for the requested tensor. Generally, the kernel factor is presented below:

$$K^n(T_i, T_j) = \exp\left(\frac{-1}{2\sigma^2} ||V_{T_i(n)} V_{T_i(n)}^T \text{-} V_{T_j(n)} V_{T_j(n)}^T||_F^2\right) \tag{4.25}$$

Where $||_F^2$ is the Frobenius norm, σ is the Gaussian kernel width, $V_{T(n)}$ is the SVD's matrix eigenvectors. The previous equations could be rewritten as follow:

$$K^n(T_i, T_j) = \exp\left(\frac{-1}{2\sigma^2} (I_m - trace(Z^T Z))\right) \tag{4.26}$$

Where $Z = V_{T_i(n)}^T V_{T_j(n)}$ and I_m contains the columns of $V_{T(n)}$.

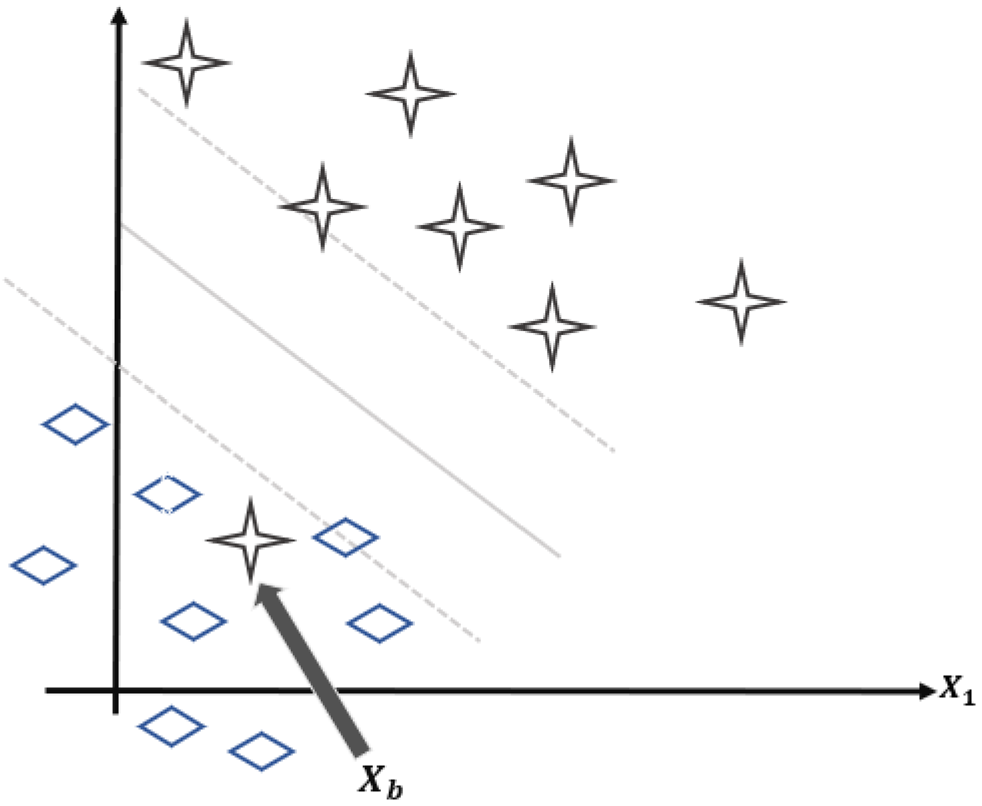


Fig.4.4. Non linear SVM

4.4 Experimental setup

4.4.1 The model choice for touch recognition

Interpreting the touch modality problem is divided into two tasks:

1) Defining an optimal description for the input signal generated from the sensor lies in feature space as follow:

$$\beta(S) \rightarrow \gamma \tag{4.27}$$

Where S is the 3rd order tensor characterizing the sensor outputs.

2) Involving practical learning for the decision function, responsible for mapping the tensor space into a set containing several categories of tactile stimuli:

$$\S\colon \gamma \rightarrow Ƒ \tag{4.28}$$

Where Ƒ contains a finite number of stimuli, and § entails a task of multi-class classification.

Indeed, a pre-processing phase is required to characterize the tactile data as proposed in [4]. Thus, β should delights $\mathbb{R}^{l(1)} \times \mathbb{R}^{l(2)}\, \mathbb{R}^{l(3)}$, where l(1), l(2) ,and l(3) are the pattern quantities. The function § is modeled through a dataset X which includes N_p patterns, where the data tensor γ is included in each pattern with a category label y ϵ *(-1,1).* However, the results in[117] show that the performances achieved by the SVM based tensor are persistent when classifying a problem of three classes. The experimental results prove that the tensorial kernel function is beneficial when targeting a classification problem that concedes a multidimensional representation. Therefore, a general training framework

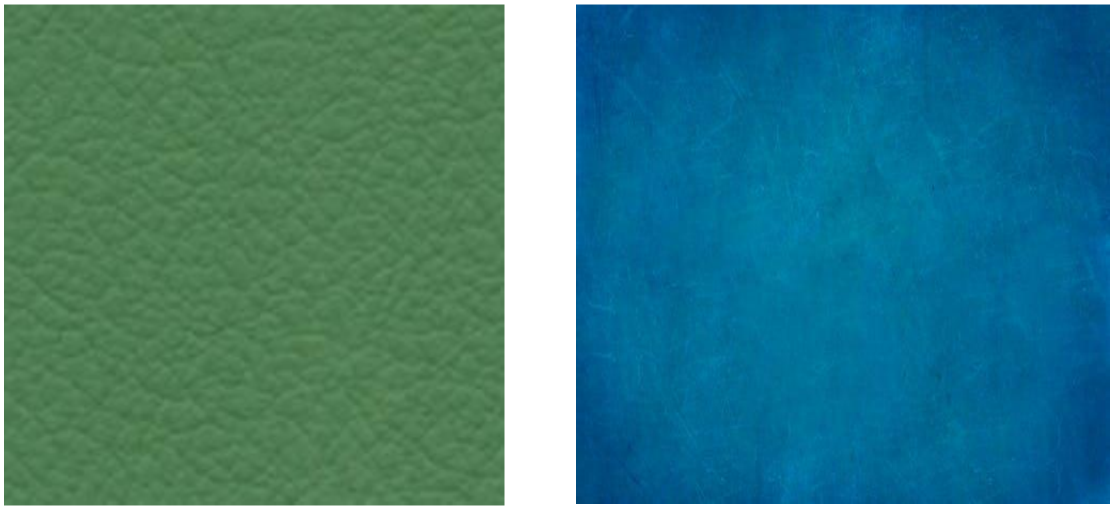


Fig.4.5. Sample images

is afforded after choosing the model since it analyses the capability of the trained ML after evaluating the ability of system generalization, achieving highly accurate results when classifying patterns out of the training set.

Moreover, estimating the error is a difficult task due to the noisy signals and the limited training set. Since the training data is dependent on the nature of the interpreted touch modality, for example, the applied touch modality could generate different stimuli with variable pressure amounts. Therefore, by adopting the Maximal Discrepancy framework (MD)[118], the difficulty of the learning machine could be evaluated by letting the machine learn noise.

In our work, the SVM model is adopted from the LIBSVM tool, which is known as open-source software for the support vector machine. LIBSVM library contains the following packages: the leading directory for implementing training and testing algorithms, a tool to check the data type, a sub-directory including binary files and interface with software such as (Windows and Matlab).

4.4.2 Dataset

In our work, the SVM based tensor kernel algorithm is written in C language, built, and run under Ubuntu 16.04. Then, the tensorial algorithm has been tested on images as a case study, before applying it to tactile data.

4.4.2.1 <u>RGB image</u>

The objective is to classify tensorial data extracted from the sensor array. Nevertheless, before applying the classification for touch modality recognition, a test for the algorithm implementation has been done by classifying two uniform color images, as shown in Fig.4.5. The two images selected below have, respectively two predominant color spectrum (green and blue). An image is characterized by three color channels (red, green, and blue),

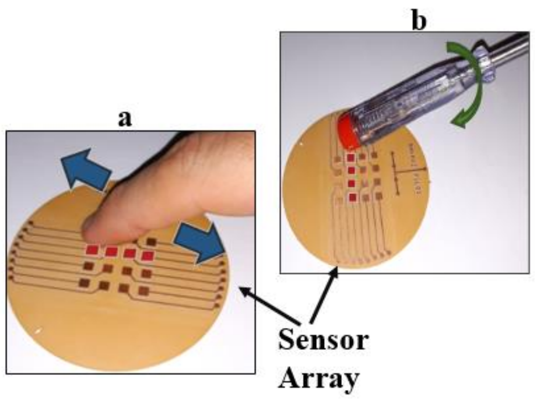


Fig. 4.6. Touch modalities. (a) Finger sliding; (b) washer rolling.

where a matrix of values (1 to 255) is included in each from the three channels. Then the image could be represented into the form of a tensor with a dimension of $n \times m \times 3$.

4.4.2.2 Tactile data

As described in [4], the data collection process has been done after asking 70 participants to touch the tactile sensor array using two predetermined possible stimuli: sliding the finger and rolling a washer, as shown in Fig. 4.6. Every participant was asked to complete one action on the tactile sensor array by moving horizontally over a random line. Moreover, the participants were supposed to complete every single touch within a time window of 10 seconds. Overall, the total number of resulted patterns is equal to 140 patterns (70 participants, two gestures, one pattern for each gesture).

4.4.3 Data pre-processing

In every single experiment, the collected patterns were expressed by a 3-dimensional tensor (4 × 4 sensor array and time acquisition). The third component of the tensor was determined by a time window of 10 seconds at the sample rate of (3k samples per second). In our case, the original size of the input tensor is T (4×4×30000).

Therefore, as mentioned in [4], a pre-processing scheme is suggested which aims to remapping the original tensor by reducing the dimensionality of the third component T. The time window is defined after evaluating the amount of energy provided from the sensors as proposed in[4]. This procedure will not affect the accuracy of the algorithm, as only a limited portion of the 30000 elements carries meaningful information about the tactile stimulus, where the signal of interest lies within a limited time window of 2 s. Then

Table 4.1. Results of model selection and accuracy obtained for image classification

	Truncation			**Sigma**	**Accuracy (%)**	
	α_x	α_y	α_z	*(σ)*	*Blue image*	*Green image*
Run 1	15	15	10	1	75	77.5
Run 2	12	12	4	1	80	85
Run 3	10	10	3	2	85	82.5
Run 4	12	12	4	2	78	80
Run 5	8	8	5	1	78.5	85
Run 6	***10***	***10***	***3***	***1***	***98***	***97***

by applying the pre-processing method [4] , the high amount of data contained in the original tensor T (4 ×4×30000) will be reduced to a tensor t (4 ×4×20).

4.4.4 Validating SVM algorithm

Before the prediction phase, the algorithm needs to be tested through LIBSVM in Windows. Therefore, the following procedures are required:

1) Reading the kernel training matrix through the "*SVM-train –t 4*" command. Then, a .model file, including the model's information, is generated. For the prediction phase, the following command "*svm-predict[test][model][output]*" is required.

2) After reading the data, the model will be tested through the following command " model=svmtrain(label, training,[-t4])"

Therefore, we validate our SVM algorithm for image classification as a case study and touch modalities recognition for the e-skin application.

Table 4.2. Results of model selection and accuracy obtained for touch modalities classification

	Truncation			**Sigma**	**Accuracy (%)**	
	α_x	α_y	α_z	(σ)	*Sliding*	*Rolling*
Run 1	5	5	2	4	75	70
Run 2	5	5	2	1	80	70
Run 3	10	10	3	1	75	72.5
Run 4	4	4	16	1	70	75
Run 5	4	4	5	1	67.5	75
Run 6	4	4	2	1	87.5	80

4.4.4.1 Image classification

The data set is built after cropping several images from the initial one, and then the dataset has been split into a test set (30%) and training set (70%). Then, two different classes (+1 and -1) are generated, since the classification problem is a bi-class classification. In our case, the size of the tensor extracted from each sample image is 15×15×3. As a training set, four hundred samples are cropped, two hundred for each class. While 85 samples have been chosen for the test set, satisfying the 70%-30% training testing ratio. Different parameters have been chosen for the SVM classifier as shown in Table 4.1, in order to select the best parameters that achieve the optimal performance in terms of accuracy when classifying the images. α_x , α_y and α_z represent the number of truncated columns that have been kept, while σ is the width of the Gaussian kernel. We conclude that by setting σ=1 and ($\alpha_x = \alpha_y = 10, \alpha_z = 3$), the average accuracy achieved is 97.5%.

4.4.4.2 Touch modalities classification

To evaluate the generalization performance of ML algorithms, the dataset obtained in section 4.4.2.2 has been split into training and test sets, with respectively 100 and 40 patterns (i.e., approximately 71% and 29% of the dataset). Moreover, it is noticeable that the generalization ability of the ML algorithm could be estimated with respect to unseen inputs; since no typical participant exists between the training set and the test set. In our case, after selecting different parameters for truncation and sigma as shown in Table 4.2, we conclude that with the following setting σ=1 and ($\alpha_x = \alpha_y = 4, \alpha_z = 2$), we obtain the highest accuracy having an average of 83.75%.

4.4.5 Prediction phase

After validating the program, the prediction program achieved is presented in Fig.4.7 as follow:

1) Reading the inputs (tensor and parameters)
    - The dimension of the tensor
    - Total number of tensors to be classified
    - The parameter sigma () for kernel computation.
2) Reading the memory (Matrices of eigenvectors and SVM model)
    - The truncated columns of the eigenvectors matrices employed to generate the SVM model.
    - The SVM model file that contains all the necessary information.
3) Executing the algorithm:

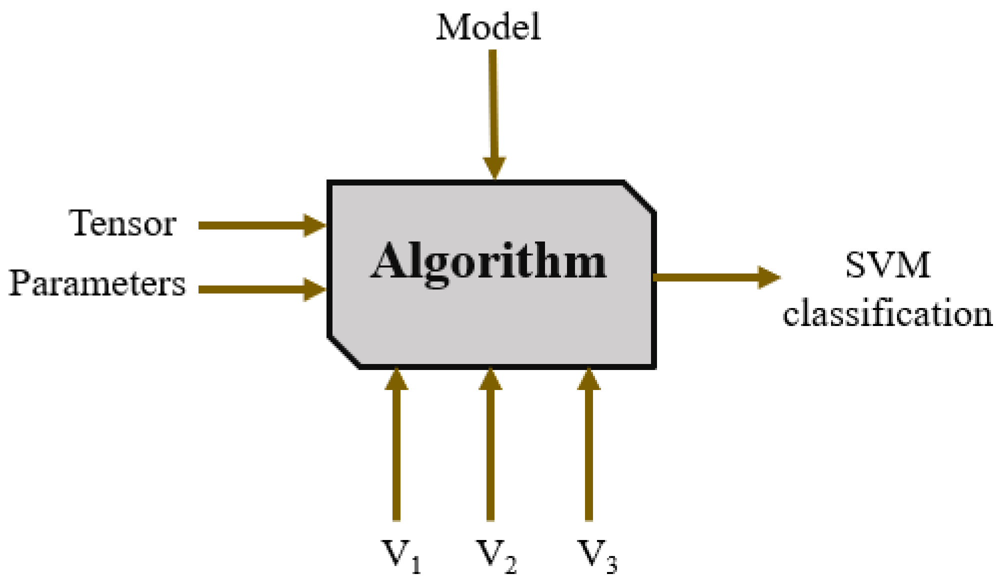


Fig.4.7. Schematic of the algorithm box

- Unfolding the tensor
- Symmetrizing the unfolded matrices
- Decomposing the symmetrized matrices through SVD.
- Building the kernel matrix
- Classifying the tensor

4.5 Conclusion

In this chapter, we described the framework of ML-based pattern recognition for sensing systems dealing with multidimensional tensor. The main two reasons behind using this approach are the following: 1) the tactile data generated from the sensor could be characterized only though the tensorial approach, and 2) learning algorithms are considered an effective method to deal with sophisticated mechanisms. Then, the steps required to build the kernel entry $K(i,j)$ based on the SVM classification are described in detail.

Moreover, the tensor-based classification algorithm is written in C language, where it has been tested on images as a case study, before applying it to tactile data. After adjusting the parameters of the model, an accuracy of 97% and 83.5% has been achieved respectively for images and touch modalities classification. Therefore, after validating the effectiveness of the tensorial SVM algorithm at the software level, in the next chapter, we will describe and present the hardware implementation of the SVM based tensor kernel approach on an Ultra-Low-Power Platform for the aim to reach an embedded low power implementation for wearable devices.

CHAPTER 5. ENERGY EFFICIENT SYSTEM FOR TOUCH MODALITIES CLASSIFICATION

5.1 Introduction

As described in chapter 4, smart e-skin is expected to process close to the sensor, raw data to extract specific, and structured information. To be effective, such smart systems should be able to process sensor data and make decisions [35] autonomously. Such a goal poses numerous challenges, as the amount of data generated by e-skin to be processed is relevant, and pattern recognition involves computationally demanding methods. Machine Learning (ML) paradigms provide a powerful tool to solve the classification problems in complex application domains, where no explicit mathematical model is available, and only raw data provide information about the observed phenomenon [117]. However, the computational complexity of embedding machine learning algorithms for e-skin is a severe obstacle toward their implementation of resource-constrained low-power embedded systems [10].

Recently, many implementations of embedded ML algorithms have been proposed in the literature using hardware [119],[120], and software implementation [121]. For e-skin applications, Field Programmable Gate Arrays (FPGAs) have been adopted[122] , [123] to achieve real-time functionality of machine learning[10]. Regrettably, affording machine intelligence with typical machine learning algorithms is still a challenge in battery-powered wearable devices due to algorithms complexity and large datasets[124]. For example, microcontrollers, such as the ARM-Cortex-M Family, suitable for battery-operated devices, are offering a low-power solution in the range of mW, but they are limited in computational power that is typically in the range of hundreds of MOPS, not enough for e-skin ML [10].

To improve energy efficiency and increase the overall computational availability of low power processing, many efforts have been made to design new processors matching the required hardware size with low power consumption. Among others, two different approaches have been adopted recently to improve the performance of ultra-low-power processor [125]: 1) exploiting parallel architectures for near-threshold operation based on multi-core clusters [99], and 2) exploiting low power hardware accelerators coupled with programmable parallel processors [99]. Near-threshold computing is a novel approach used to reduce power consumption and improve energy efficiency. Examples of near-threshold ultra-low-power microcontrollers have been shown in[93], [4]. To further improve energy efficiency, some microcontrollers have been proposed embedding custom hardware accelerators [88]. However, the flexibility of such approaches is limited, affecting the cost and scale economy.

In this chapter, we exploit a state-of-the-art multi-core platform to improve the energy efficiency for embedded machine learning systems for touch modalities classification. The main focus in this chapter is the implementation of tactile data decoding on a parallel ultra-low-power platform (PULP) embedding a high-efficient processor called Mr. Wolf. The low-power touch modality classification is designed for battery-powered applications such as wearable electronics and to interface with tactile sensors. The classification based on

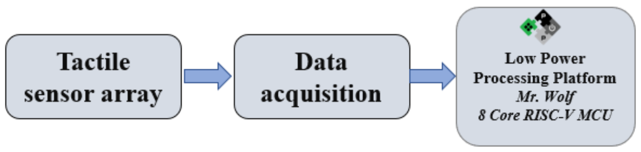


Fig.5.1. Block Diagram including the tactile sensor array, the sensor's interface and the processing platform.

Support Vector Machine (SVM), which is considered as an effective method for the e-skin application [57], runs directly on PULP classifying two touch modalities (finger sliding and washer rolling) as shown in Fig. 5.1. Experimental results show that the target application runs 3.7 times faster than an ARM Cortex M4 by consuming less than 28 mW on Mr. Wolf, outperforming ARM Cortex M4 by 15 times in terms of energy efficiency. The results in this chapter were published in [36].

5.2 PULP processing unit

Mr. Wolf is a state-of-the-art microprocessor[64], featuring an energy-efficient I/O subsystem , an area optimized CPU for control purposes, an energy-efficient parallel ultra-low power cluster build upon 8 RISC-V cores, composed of SoC and cluster domains as shown in Fig. 5.2. The main characteristics of Mr. Wolf are summarized in Table 5.1.

5.2.1 SoC Domain

Number of cores	8 cores
Chip area	10 mm^2
Technology node	CMOS 40nm LP
L2 memory	512 kB
L1 memory	64 kB
Core voltage	0.8V-1.1V
Frequency range	32kHz-450MHz
Power range	72µW-153mW

Table 5.1. Characteristics of Mr. Wolf

The SoC domain, as shown in Fig. 5.3 is based on the MCU controller, having two power-efficient pipeline stages of the RISC-V processor. A full set of peripherals is contained in the SoC, such as I2C, I2S and Quad SPI, GPIOs, four channels for the PWM interface, and a JTAG interface. The multi-channel I/O DMA are responsible for transferring the data from/to the peripherals. The µDMA of Mr. Wolf is connected to all the IP peripherals, while two specified 32-bit ports are connected to the L2 memory, satisfying the needs of the parallelization consuming power of 2mW with a frequency of 57 MHz. Therefore, the transfer efficiency will be maximized, where the need for large buffers to be connected to the peripherals will not be significant. Cores in the computing cluster have been enhanced with dedicated Digital Signal Processing (DSP) extensions to tackle DSP intensive operations typical of IoT applications.

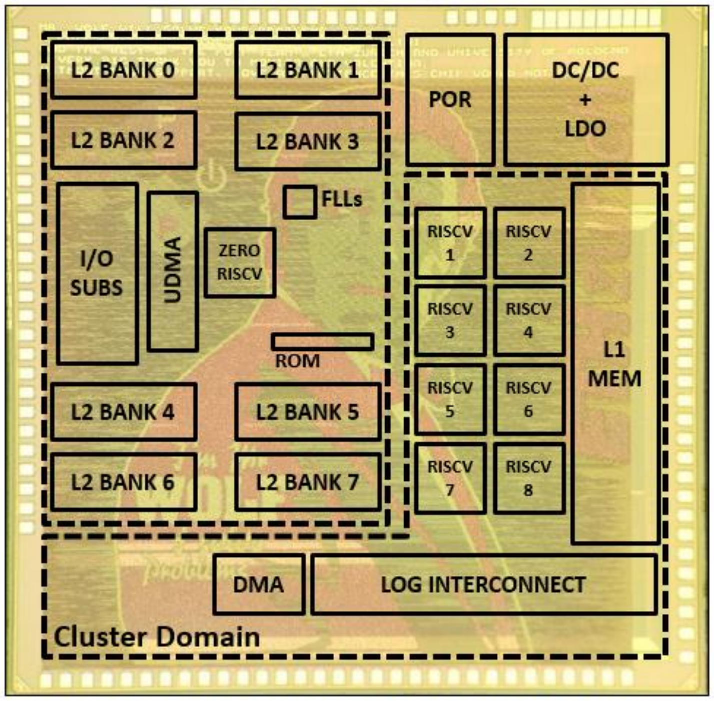


Fig.5.2. High Level Mr. Wolf architecture

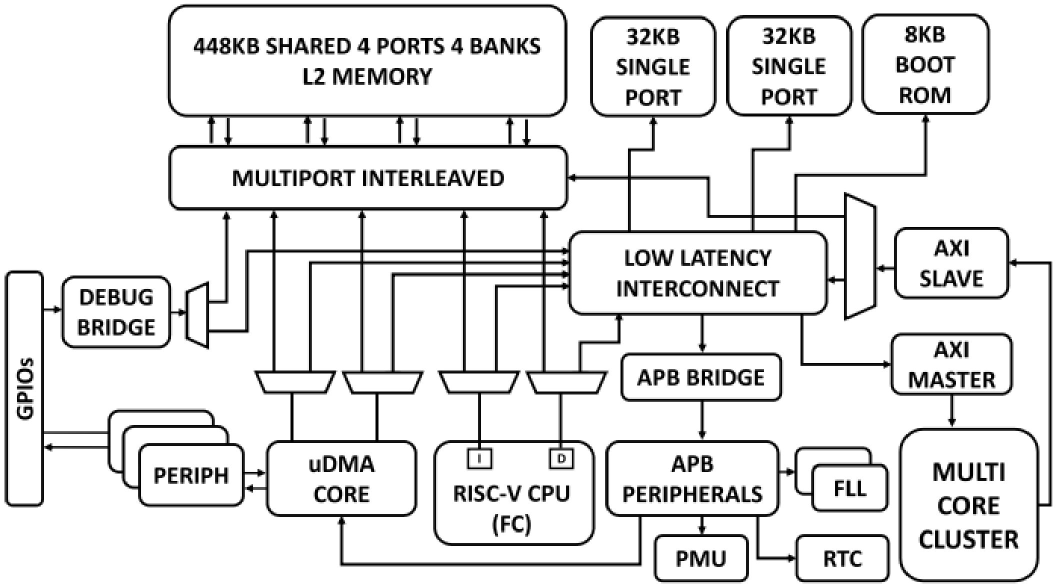


Fig.5.3. Mr. Wolf SoC block diagram

The size of the L2 memory on the SoC is 512 kB, containing a ROM which stores the primary boot-code. The L2 memory is composed as follows: four 112 Kbytes, which allow reducing the conflicts when parallelizing, two banks of 32 Kbytes used by the Fabric Controller (FC) without getting any banking conflicts. The organization of the memory increases the bandwidth of the system memory by 4×, in a way that satisfies the needs of the master resources. Data is moved to/from the main system memory (L2) via a dedicated DMA engine capable of handling complex traffic patterns (ex. 2D transfers) with a very low programming overhead. The main memory has been organized in multiple banks and shared through a logarithmic interconnect among all the resources to allow concurrent operation by control CPU, I/O subsystem, and processing cluster. Moreover, the hierarchy of Mr. Wolf's memory is characterized by a single namespace where every single core could access all the memory locations.

5.2.2 Cluster Domain

For the cluster domain, a specific frequency and voltage values are adjusted when running the applications on the FC, as shown in fig. 5.4. The 8 RISC-V cores on the cluster are supported by the RVC32IM instruction set, including other specific instructions for energy-efficient digital signal processing applications.

The cores share L1 memory through a low latency logarithmic interconnect with as low as one cycle access under no contention. The cluster has two shared floating-point units (FPUs) with add/sub support and one shared div/SQRT FPU allowing efficient floating-point operation at the low area and power cost. The shared FPU reduces the area by 4× while decreasing the performance by 10%. Then, a multi-banked L1 memory is connected

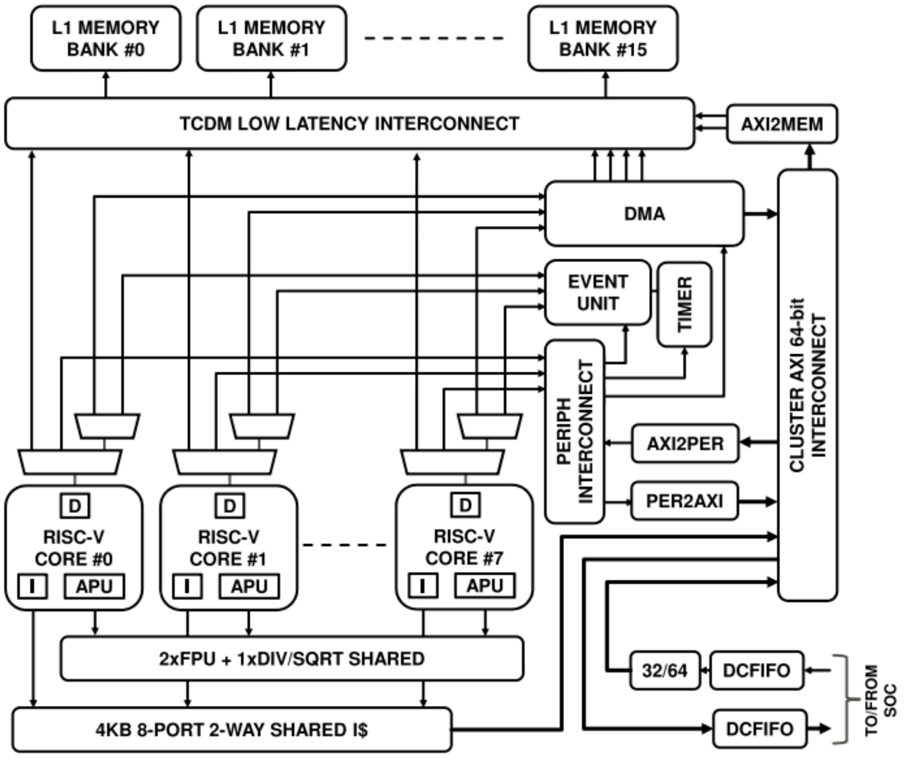


Fig.5.4. Mr. Wolf cluster block diagram

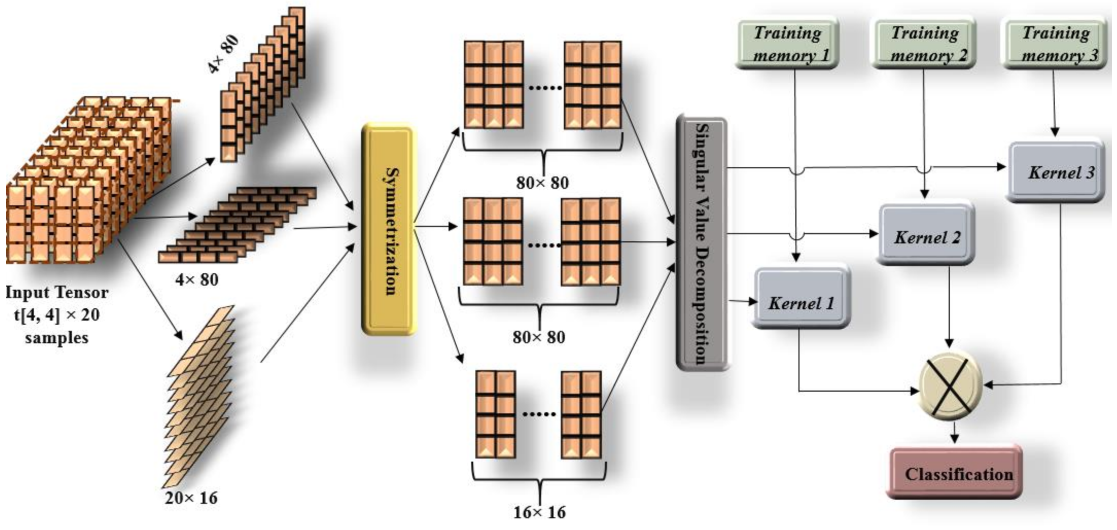


Fig. 5.6. Online computation of the SVM based tensor kernel algorithm

to the cluster, which targets parallel programming models (i.e., OpenMP). With single access, the L1 memory can maintain the request of all the memory in parallel. The parallelism is enabled through the dedicated hardware block (HW Sync), resulting in an energy-efficient parallel workload. In the cluster, the HW sync is responsible for controlling the clock gating of each core.

5.3 SVM based tensor kernel algorithm on the PULP Architecture

5.3.1 Inference Implementation

In this section, we present the online implementation of the SVM based tensor kernel approach on Mr. Wolf, as shown in fig. 5.6.

5.3.1.1 Tensor unfolding

The first step for the online classification is unfolding the input tensor t (4×4×20) into three different matrices by rearranging all the rows and columns one after the other. Thus, three unfolded matrices A (4×80), B (4×80), and C (20×16) have resulted.

5.3.1.2 Symmetrization

The obtained matrices are symmetrized through a simple matrix multiplication, as shown below:

$$A_{sym} = A^T A \tag{5.1}$$

Where the unfolded matrix A is multiplied by its transpose A^T. Hence, three square matrices (A (80×80), B (80×80), and C (16×16)) are obtained.

5.3.1.3 Singular Value Decomposition

The SVD blocks compute the singular value decomposition of the three symmetrized matrices based on the one-sided Jacobi algorithm. Thus, the symmetrized matrices are transformed into the product of three matrices.

Where the orthogonal matrices U and V^T contain respectively the eigenvectors of AA^T and A^TA. While the singular values of A arranged in descending order ($\sigma 0, \ldots\ldots\ldots, \sigma n\text{-}1$) are stored into the diagonal matrix S.

5.3.1.4 Kernel Computation

The kernel factor adopted in computing the distances between the data from the input touch modality and the memorized data obtained from the training phase as referred to in [15] is defined as follow:

$$k^n(x, y) = \exp\left(-\frac{1}{2\sigma^2}\left(I_n - trace(Z^T Z)\right)\right) \tag{5.2}$$

$$Z = V_x^T V_y \tag{5.3}$$

Where I_n is the identity matrix, V_x contains the singular vectors of the symmetric matrix while $V_y$is composed of the singular vectors resulted from the tensor in the training phase.

While the kernel function is the product of the three kernel factors as shown in the equation below:

$$K(x, y) = \prod_1^n k^n(x, y) \tag{5.4}$$

5.3.1.5 Classification

The equation of the SVM classification is represented by:

$$\hat{y} = f_{SVM}(x_j) = \sum_{i=1}^{N_p} \beta_i \, K(x_i, x_j) + b \tag{5.5}$$

Where $x_j, \hat{y}$ and β_i represent respectively, the input, the predicted category and the weights obtained during the training phase.

The SVM model has been built using the LIBSVM tool, which is a library for Support Vector Machine that has been widely used in many areas. A. model file containing all the information will be generated after applying the training. Nevertheless, on PULP, the SVM algorithm from LIBSVM could not be implemented as it is since Mr. Wolf is not able to open and read files. Thus, some modifications must be implemented to the SVM c code, which is summarized as follow:

a) Studying the SVM c code implementation and removing the unused functions such as:
   - Saving the model (svm_save_model)
   - Reading the model (read_model_header)
   - Loading the model (svm_load_model)
   - Freeing the model (svm_free_model)
b) Adding all the parameters generated from the model directly into the algorithm, such as:
   - The number of support vectors
   - The coefficients of the support vectors
   - The parameter rho
   - The two labels (1 and -1).
c) Doing some other modifications to the code in order to be compatible with the recent changes such as:
   - Allocating a memory space for the coefficients of the support vectors.
   - Allocating a memory space for the kernel value.

5.3.2 Fixed-Point Implementation

Usually, a double or single-precision floating-point representation is employed in CPU's or when using Matlab, in order to reach a high level of accuracy when executing the SVM based tensor kernel algorithm. However, it is well known that floating-point data needs a significant amount of computational efforts when embedding an ML algorithm on the processor; since the two main factors (power consumption and energy efficiency) must be taken into consideration when designing the application. Eventually, floating-point operations could be computed through specific hardware, where the representation in most of the floating-point processors follow the IEEE 754 data type[126]. For example, the numbers in a 32-bit word are represented as follows: one bit for the sign, eight bits for the exponent while the mantissa is represented by 23 bits. Therefore, the numbers could be represented in a fixed-point format. The Qm.n format is commonly used in most of the representation types, where m and n represent respectively the integer and the fractional word length part of the desired number, such as $-32768 \leq Q16.16 \leq 32768$, weight of LSB 2^{-16}.

In our algorithm, the SVD is considered as the most critical part during the fixed-point implementation; since the dynamic range of the algorithm is high when moving data from small range to high range. Unfortunately, a numeric overflow could result when using fixed-point representation. For this algorithm, the lib fix math library has been used. Libfixmath is considered as a common fixed-point maths library platform[127]. All the math.h functions are implemented in the fixed-point format in this library. However, the Q16.16 format has been chosen after analyzing the dynamics of the variables when executing the algorithm. The precision resulted when implementing a Q16.16 format is reported in the table below. Thus, all the math.h functions in the code have been replaced with the libfixmath library.

5.3.3 Parallelization and Memory Management Strategy

This section describes the parallelization and the proposed memory management strategy adopted for the algorithm on PULP. Implementing a memory management strategy is considered as a significant task to be applied in Ultra-Low Power embedded SoC's having

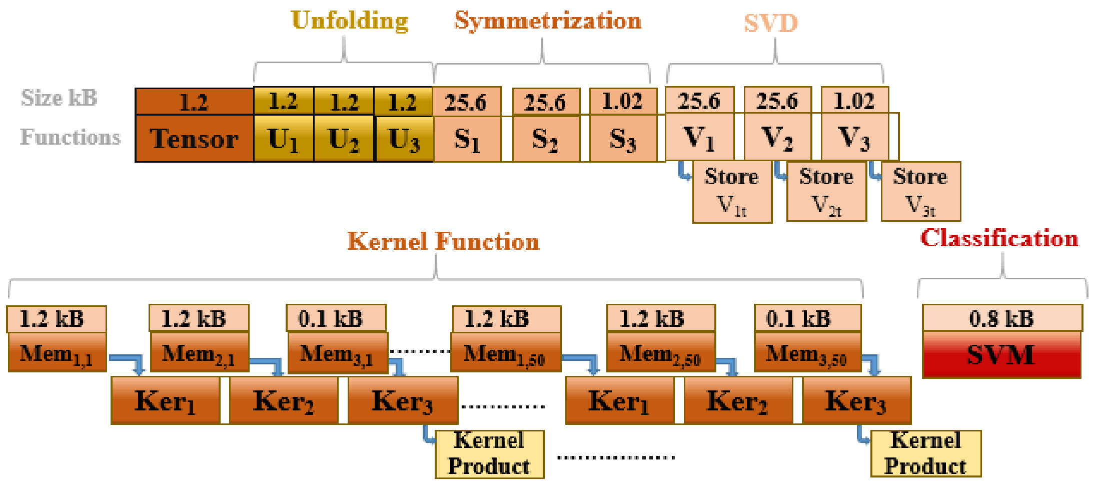


Fig. 5.7. L2 memory allocation sequence.

no caches. Thus, in our work, we have proposed a memory management strategy since 50 trained tensors are required for each class in the SVM model. Table 5.3 reports the memory needed for the algorithm in order to store 50 trained tensors for each class. Thus, a memory management strategy is required in L2 memory preventing fragmentation problems. The allocation memory sequence is reported in Fig.5.7 (from left to right) for the entire algorithm execution. At first, before the actual computation loop starts, all the trained tensors computed in the offline phase are stored in the L2 memory. Then, the input tensor is unfolded into three different matrices and stored in L2. After that, memory space for S1 is allocated in the symmetrization and de-allocate U1 after symmetrizing the unfolded

Data	**Number of Elements**	**Size [Kbits]**
Input tensor	4×4×20	1.28
Training Data SVD(A)	80×4×100	128
Training Data SVD(B)	80×4×100	128
Training Data SVD(C)	16×2×100	12.8

Table 5.2. Algorithm memory

matrix. Sequentially, the same process is adopted for S2 and S3. Then, three SVD blocks that take the highest amount of operations are required, as shown in Table 5.4. Therefore, each SVD block is executed in parallel on each core for reducing time latency. The three SVS's blocks are parallelized on the three cores of Mr. Wolf. Then, each matrix is decomposed through the SVD by adopting the same allocate/de-allocate process mentioned previously. The resulted eigenvectors V1t, V2t, V3t, are stored in L2 and then passed to the SVM Kernel phase. Due to the size of the trained tensors which cannot be stored entirely in L1, as shown in Table 5.3; then each block from the trained tensors stored in L2 needs to be moved to the kernel function phase in order to compute the kernel product with respect to the resulted eigenvectors. The data is transferred from L2 to L1 memory through

	T (m,n,t)	T(4,4,16)	T(4,4,20)	T(4,4,40)	T(4,8,20)	T(4,8,40)	T(8,8,20)
Functions	**Size (Kbits)**	**Size (Kbits)**	**Size (Kbits)**	**Size (Kbits)**	**Size (Kbits)**	**Size (Kbits)**	**Size (Kbits)**
Input Tensor	4×m×n×t	1.024	1.28	2.56	2.56	5.12	5.12
Symmetrization	4×m×n×t^2	16.3	25.6	102.4	51.2	204.8	102.4
Training Data SVD(A)	16×m×t×x	1.024x	1.28x	2.56x	1.28x	2.56x	2.56x
Training Data SVD (B)	16×n×t×x	1.024x	1.28x	2.56x	2.56x	5.12x	2.56x
Training Data SVD (C)	8×m×n×x	0.128x	0.128x	0.128x	0.256x	0.256x	0.512x
Truncated Eigenvectors SVD (A)	16×m×t	1.024x	1.28	2.56	1.28	2.56	2.56
Truncated Eigenvectors SVD (B)	16×n×t	1.024x	1.28	2.56	2.56	5.12	2.56
Truncated Eigenvectors SVD (C)	8×m×n	0.128x	0.128	0.128	0.256	0.256	0.512
Library fixed point 16.16	32.7	32.7	32.7	32.7	32.7	32.7	32.7
Number of trained tensors for each class (x_1, x_2)		**99**	**80**	**49**	**55**	**39**	**45**

Table 5.3. Algorithm memory for different size of tensors

the double buffering policy using the DMA. Finally, all the kernel products obtained will be classified through the SVM.

On the other hand, a study has been done on the memory level in order to find the maximum possible number of trained tensors which could be stored in L2 on-chip memory when varying the size of the input tensor from T (4,4,16) till T (8,8,20) as reported in Table 5.4. Moreover, a generalized equation is generated which computes the maximum number of trained tensors able to be stored for any size of tensor for different configurations of (m,n,t), represented as follow:

$$x = \frac{415.3\times10^3 - 4(mnt - mnt^2 - 4mt - 4nt - 2mn)}{8(2mt + 2nt + mn)} \tag{5.6}$$

Where

$$x_1 = x_2 = \frac{x}{2} \tag{5.7}$$

Therefore, the maximum number of trained tensors which could be stored is not more than 45 tensors for each class, when increasing the size of the input tensor to T (8,8,20).

5.4 Experimental Results and Performance Assessment

This section presents the experimental evaluation of the proposed platform considering two primary metrics: i) the execution time of the algorithm and ii) the capability in performing the required computations within a low power envelope. Moreover, we present a comparison with an ARM Cortex-M4F microcontroller.

5.4.1 Performance

Functions	**Wolf 1 core**		**Wolf 2 cores**		**Wolf 3 cores**		**Number of Operations [OPS]**
	Cyc(M)	**Sp(×)**	Cyc(M)	**Sp(×)**	Cyc(M)	**Sp(×)**	
Sym+Kernel+Class	89.8	2.25	89.8	2.25	89.8	2.25	1.26×10^{7}
SVDs	1522	2.26	923	3.59	276	3.7	5.3×10^{8}
Total	1616	**2.26**	1020	**3.62**	366.69	**3.73x**	$\mathbf{5.45\times10^{8}}$

Table 5.4. Performance of the tensorial kernel online computation on Wolf platform. Cyc, T ,Sp stand for cycles,Time and speed-up

The execution cycles of each block of the entire algorithm are presented in Table 5.5. We evaluate the execution cycles for Mr. Wolf using a single core, 2 cores, and 3 cores, to evaluate the benefits of the parallelization. To take advantage of the 3-cores cluster, two parallelization schemes have been adopted in this paper, i) executing the two SVDs block in parallel on two different cores on the Wolf SoC and ii) parallelizing the three SVDs on 3 cores. The two parallelization schemes have been implemented on Mr. Wolf. In addition, in the column (Sp (×)), the table reports the speedup compared to STM32F40, Arm Cortex M4F microcontroller, running the classification in around 12s at 168 MHz.

Results show that a 2.26 × speed-up is achieved by migrating from Arm Cortex M4F to the Wolf SoC using 1 Core. Moreover, Table 5.4 highlights that the SVD blocks require a higher number of operations in the algorithm. Thus, our algorithm benefits from the parallel cores available on Mr. Wolf to reduce the execution time. Table 5.4 shows that a 3.62× speed-up can be achieved after executing SVD (A) and SVD (B) blocks on two different cores in parallel. However, in the case of implementing the three SVD on the 3-cores cluster, the implementation will gain a speed-up of 3.73× with respect to STM32F40.

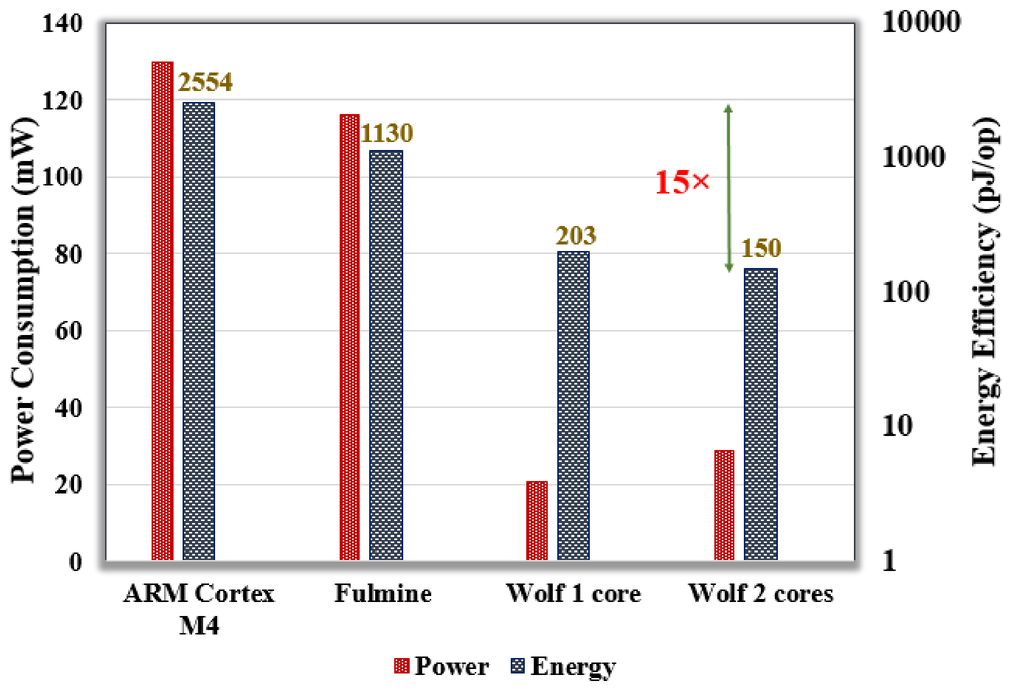


Fig. 5.8. Power consumption and energy efficiency comparison of the online computation algorithm on the ARM Cortex M4 and Wolf .

5.4.2 Power Consumption

Fig.5.8 shows the power consumption and the energy efficiency of the tensorial SVM algorithm implementation on the Wolf platform compared to the Arm Cortex M4F and Fulmine platform[36]. In our experiment, the PULP platform runs at 300 MHz. The single-core Wolf platform shows a 5.6× power reduction with respect to the Arm Cortex M4F. The power consumption is reduced due to the technology gap between the two platforms (i.e., 90 nm vs. 40 nm) and also due to the optimized implementation strategy adopted to the cluster's architecture for energy-efficient operation[36].

Moreover, the energy efficiency of Wolf (1core) has been improved respectively by 12.5× and 5.5× when compared to Arm Cortex M4F and a previous version of PULP, called Fulmine [59] using a single core. On the other hand, as shown in Fig. 5.7 that a significant

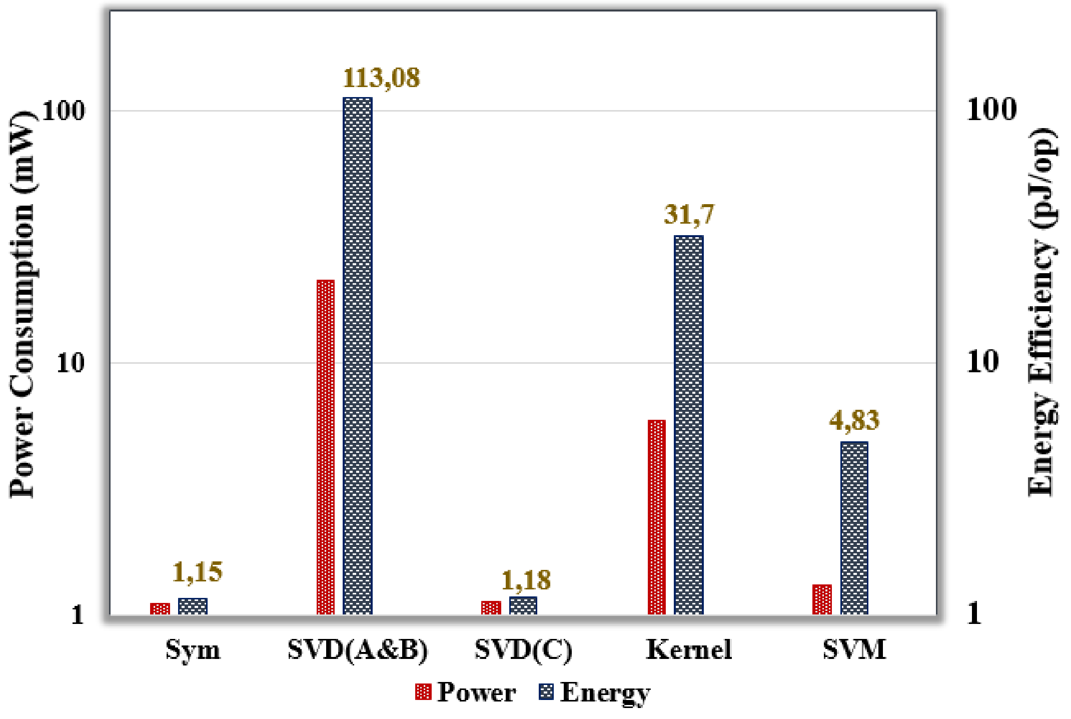


Fig. 5.9. Power consumption and energy efficiency of each function of the online computation algorithm on Mr. Wolf

energy boost can be achieved after using the 2 cores of the cluster, leading to 1.4× energy efficiency reduction with respect to the single-core Wolf. It should be noted that after parallelizing the algorithm on 2 cores only, the energy efficiency has been reduced by 15× when compared to STM32F40.

Moreover, in Fig. 5.9, we measure the power of the execution of each function of the online computation of tensorial SVM on Mr. Wolf. The power consumption is measured by powering Mr. Wolf SoC at the core voltage of 1V with an operating frequency of 300 MHz. As shown in Fig. 5.9, the two SVDs (A&B) have the peak power consumption of 21 mW with an energy efficiency of 113 pJ/op.

5.5 Conclusion

In this chapter, we presented the implementation of tactile data sensing on a novel low power parallel platform embedding a high-efficient processor called Mr. Wolf. We demonstrated that the algorithm on the proposed platform outperforms ARM Cortex M4F (STM32F40) and by 15 times in terms of energy efficiency, without exceeding the power envelope of a 28mW. Future work will consist of using approximate computing techniques at the algorithmic level [128] to improve energy efficiency [61], [60].

CHAPTER 6. CONCLUSION AND FUTURE EXTENSIONS

6.1 Conclusion

This thesis focused on the implementation of energy-efficient techniques for embedded machine learning algorithms in smart sensing systems.

Apart from a few examples in the state of the art [97], still not significant results have been achieved concerning the improvement of power consumption in embedded ML algorithms employed for sensing systems [57]. The embedded computing unit in sensing systems is responsible of extracting meaningful information, usually through ML algorithms methods. However, deploying ML in embedded devices poses several challenges in terms of hardware resources and energy consumption. In particular, the energy-efficient techniques proposed in this work are applied to a case study such as the “electronic-skin” (e-skin) application. The e-skin system includes 1) structural materials, 2) signal processing, 3) data acquisition and 4) data processing. An essential task of the e-skin system is to process the signal and the tactile data aiming to mimic human skin. More precisely, ML algorithms based on the tensor kernel approach are applied to classify different touch modalities. Nevertheless, the exploited ML algorithms for the system are complex and need too many resources. As shown by [10], the demands of the embedded ML algorithms in the e-skin are not satisfied since the estimated energy/power required for the application is not feasible(i.e., 100 pJ/op), time latency (i.e., around 7 s) and the computational load is of about 1.2 GOPS.

Attempting to improve the energy efficiency of the embedded ML algorithms, two mains approaches are discussed and investigated ,i.e., Approximate Computing and Parallel Computing Platforms.

The energy-efficiency of computing systems can be improved through approximation computing techniques at circuit and algorithmic levels. These techniques will lead to an interesting reduction in power consumption. This thesis focused on the design and implementation of two approximate multipliers in ML algorithms for tactile data processing. The first multiplier is based on the rounding approach called the “META” multiplier [60], while the second one is called the “Approximate Baugh-Wooley(BW)” multiplier[61].Furthermore, we designed three versions of the Approx-BW multiplier and three versions of the META multiplier based on approximate adders. We showed that the proposed approximate arithmetic circuits could achieve a relevant reduction in power consumption and time delay around 80.4% and 24%, respectively, with respect to the exact BW multiplier.

We evaluated the behavior of the e-skin application when implementing the proposed approximate arithmetic circuits in the system. In this regard, we implemented the approximate multipliers on the low-pass Finite Impulse Response (FIR) filter in the signal processing block of the system. The FIR filter, based on (Approx-BW), outperforms state of the art solutions while respecting the tradeoff between accuracy and power consumption, with an SNR degradation of 1.39dB. Then, ACTs are applied to the data processing block to improve the performance of the Coordinate Rotational Digital Computer (CORDIC) and the Singular Value Decomposition (SVD) circuits used for embedded ML algorithms in the sensing system. We showed benefits of up to 21% and 19% in power reduction at the cost of less than 5% accuracy loss for CORDIC and SVD circuits when scaling the number of approximated bits.

On the other hand, we explored another approach (i.e., Parallel computing platforms), aiming to improve the energy efficiency of the system. We implemented the ML algorithm based tensor kernel approach on a RISC-V parallel ultra-low-power platform (PULP) after validating the effectiveness of the ML algorithm at software level. We performed the on-board classification of different touch modalities on a PULP processor called Mr. Wolf" demonstrating the promising use of on-board classification for smart sensing systems. We presented a comparison with the popular low power ARM Cortex-M4F microcontroller employed, usually for battery-operated devices. We proved that the algorithm on the proposed platform runs 3.7 times faster than ARM Cortex M4F (STM32F40), consuming 28 mW. The platform has allowed to improve the energy efficiency by 15× than the classification done on the STM32F40, consuming 81mJ per classification and 150pJ per operation.

6.2 Potential research extensions

The ideas put forward in this thesis could be extended as follow.

The implementations of the approximate arithmetic circuits into the Coordinate Rotation Digital Computer and the Singular Value Decomposition algorithms represented a successful investigation for embedded machine learning algorithms employed for tactile data processing. However, one less explored idea is the implementation of the approximate techniques into the overall embedded machine learning algorithm for classifying different touch modalities, aiming to reduce the complexity of the algorithm. Moreover, including new approximate arithmetic circuits in the computation of the SVD and CORDIC, such as approximate dividers and square root, could also be considered as a potential addition to this work.

On the other hand, since the first implementation of the embedded machine learning algorithms for touch modalities classification on an ultra low power platform has received a significant improvement in terms of energy efficiency. Therefore, these results motivate the parallel implementation of the algorithm on eight cores of Wolf platform, boosting more the overall energy efficiency to achieve a real-time low power implementation. Alternatively, approximate computing techniques at the algorithmic level could also be applied to the embedded machine learning algorithm on the PULP platform. Finally, if these methods succeed in achieving better results, the possible future work could be designing an integrated circuit specific for smart sensing systems.

APPENDIX

PULP Set Up

In order to set up the PULP software development kit for Mr. Wolf, board, as shown in Fig. 5.5, the installation of a Linux Operative System is mandatory since the SDK is designed to work in a Linux environment. Two OS such as Ubuntu and CentOS are possible, but for this work, Ubuntu16.04 has been selected. The chosen OS is directly installed on a virtual machine created with VirtualBox. Then it is possible to start by downloading all the procedures aiming to set up the PULP SDK for Mr. Wolf after creating the new Linux machine, as mentioned previously.

At first, ETH permission is required in order to use the PULP SDK for Mr. Wolf. An SSH key must be created and uploaded to the Github account on the ETH server in order to download and update all the development material. Then, the SSH key is loaded into the ssh-agent as follow:

```
ssh-add ~/.ssh/<your key>
```

Before starting using the board, Python and Python3 must be installed with all the necessary packages related to Python as follow:

```
sudo apt-get install python python3
sudo apt install git python3-pip gawk texinfo libgmp-dev \
 libmpfr-dev libmpc-dev swig3.0 libjpeg-dev \
 lsb-core doxygen python-sphinx sox \
sudo pip3 install artifactory twisted prettytable \
 sqlalchemy pyelftools openpyxl xlsxwriter
```

Then the git repository from the Github account, containing all the files for the SDK version must be cloned :

```
git clone https://github.com/pulp-platform/pulp-sdk.git -b <sdk branch or tag>
```

After, we have to source the config files for the desired pulp chip (**Wolfe.sh**) and platform (**platform-board.sh**) in order to use the development board :

```
source configs/<the desired chip>.sh
source configs/<the desired platform>.sh
```

The password for the artifactory server must be configured by following the guide on https://iis-git.ee.ethz.ch/pulp-sw/pulp-sdk-internal. This page is only accessible with a valid Gitlab account for the IIS Gitlab server and if the account was added to the PULP group.

The system dependencies should be configured, so the dependencies for the SDK will be downloaded from the artifactory server by running the following command.

```
make deps
```

If this command fails in running, then it means that there were no precompiled binaries found for the Linux distribution installed. So the binaries must be used for a different distribution. The Ubuntu_16 binaries were verified to be working with Ubuntu 17.10. The following command should be run to force the distribution:

```
plpbuild --p sdk deps --stdout --distrib=Ubuntu_16
```

6.2.3 Compilation

Making sure that GCC version 5.4.1 has been using by running :

```
gcc -v
```

Then installing GCC 5.4.1, if another version is running by applying the following command:

```
sudo apt-get install gcc-5 g++-5
```

Then the default GCC version is changed by executing:

```
sudo update-alternatives --config gcc

sudo update-alternatives --config g++
```

Finally, the SDK will be compiled by running:

```
make all
```

Then the **source.sh** environment settings file should be created. This file needs to be sourced whenever a development is done for PULP in a new shell-session. Otherwise, building running will not work as intended using **make**.

```
make env
```

6.2.4 Setup of JTAG-USB- programmer for the board

In the next steps, the environment that should be able to use the development board via the ARM-USB-Tiny-H cable from Olimex must be set up as follow:

```
make env
```

Then, searching for an entry like:

```
Bus 002 Device 003: ID 15ba:002a Olimex Ltd. ARM-USB-TINY-H JTAG interface
```

Then a new udev rule should be created to set up the USB programmer by creating the file (/etc/udev/rules.d/10-ftdi.rules), by running the following command:

```
sudo touch /etc/udev/rules.d/10-ftdi.rules
```

The file must be opened with sudo rights, so it could be possible to write to it:

```
sudo nano /etc/udev/rules.d/10-ftdi.rules
```

(002a) and the username by the username of Linux’s user.

```
ATTR{idVendor}=="15ba", ATTR{idProduct}=="<4 chars>", MODE="0666", GROUP="<\ username>"
```

After that, the udev rules must be reloaded:

```
sudo udevadm control --reload-rules && udevadm trigger
```

Then, the sourceme.sh file should be opened in the installation directory with the editor, and the following line will be added to the beginning of the file:

```
export OLIMEX_PID=0x<4 chars>
```

To the end, the output should be like this:

```
export PULP_CURRENT_CONFIG=honey@user_config_file=...

export PULP_CURRENT_CONFIG_ARGS=platform=board

export OLIMEX_PID=0x002a
```

BIBLIOGRAPHY

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys*. 2016.

[2] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proceedings - 2013 18th IEEE European Test Symposium, ETS 2013*, 2013.

[3] A. Ibrahim, P. Gastaldo, H. Chible, and M. Valle, "Real-time digital signal processing based on FPGAs for electronic skin implementation," *Sensors (Switzerland)*, 2017.

[4] P. Gastaldo, L. Pinna, L. Seminara, M. Valle, and R. Zunino, "A tensor-based approach to touch modality classification by using machine learning," *Rob. Auton. Syst.*, 2015.

[5] R. Seva, P. Metku, K. K. Kim, Y. Bin Kim, and M. Choi, "Approximate stochastic computing (ASC) for image processing applications," in *ISOCC 2016 - International SoC Design Conference: Smart SoC for Intelligent Things*, 2016.

[6] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "ApproxANN: An approximate computing framework for artificial neural network," in *Proceedings -Design, Automation and Test in Europe, DATE*, 2015.

[7] D. Mohapatra, G. Karakonstantis, and K. Roy, "Significance driven computation: A voltage-scalable, variation-aware, quality-tuning motion estimator," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2009.

[8] B. Murmann, D. Bankman, E. Chai, D. Miyashita, and L. Yang, "Mixed-signal circuits for embedded machine-learning applications," in *Conference Record - Asilomar Conference on Signals, Systems and Computers*, 2016.

[9] V. Sze, "Designing Hardware for Machine Learning: The Important Role Played by Circuit Designers," *IEEE Solid-State Circuits Mag.*, 2017.

[10] A. Ibrahim and M. Valle, "Real-Time embedded machine learning for tensorial tactile data processing," *IEEE Trans. Circuits Syst. I Regul. Pap.*, 2018.

[11] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, 2010.

[12] M. L. Kringelbach, N. Jenkinson, S. L. F. Owen, and T. Z. Aziz, "Translational

principles of deep brain stimulation," *Nature Reviews Neuroscience*. 2007.

[13] L. Clifton, D. A. Clifton, M. A. F. Pimentel, P. J. Watkinson, and L. Tarassenko, "Gaussian process regression in vital-sign early warning systems," in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS,* 2012.

[14] L. Cunial *et al.*, "Parallelized Convolutions for Embedded Ultra Low Power Deep Learning SoC," in *IEEE 4th International Forum on Research and Technologies for Society and Industry, RTSI 2018 - Proceedings*, 2018.

[15] Z. Du, A. Lingamneni, Y. Chen, K. V. Palem, O. Temam, and C. Wu, "Leveraging the Error Resilience of Neural Networks for Designing Highly Energy Efficient Accelerators," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, 2015.

[16] O. Krestinskaya, T. Ibrayev, and A. P. James, "Hierarchical Temporal Memory Features with Memristor Logic Circuits for Pattern Recognition," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, 2018.

[17] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE J. Solid-State Circuits*, 2017.

[18] A. Ranjan, A. Raha, V. Raghunathan, and A. Raghunathan, "Approximate memory compression for energy-efficiency," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2017.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[20] M. Ring, U. Jensen, P. Kugler, and B. Eskofier, "Software-based performance and complexity analysis for the design of embedded classification systems," in *Proceedings - International Conference on Pattern Recognition*, 2012.

[21] M. Nadeski, "Mark Nadeski Embedded Processing Texas Instruments Bringing machine learning to embedded systems."

[22] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," pp. 1–13, 2016.

[23] P. Tang, H. Wang, and S. Kwong, "G-MS2F: GoogLeNet based multi-stage feature fusion of deep CNN for scene recognition," *Neurocomputing*, 2017.

[24] S. Heller *et al.*, "Hardware Implementation of a Performance and Energy-optimized Convolutional Neural Network for Seizure Detection," *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBS*, vol. 2018-July, pp. 2268–2271, 2018.

[25] D. Shin, J. Lee, J. Lee, J. Lee, and H. J. Yoo, “An energy-efficient deep learning processor with heterogeneous multi-core architecture for convolutional neural networks and recurrent neural networks,” in *Proceedings for 2017 IEEE Symposium on Low-Power and High-Speed Chips, COOL Chips 2017*, 2017.

[26] Y. H. Chen, J. Emer, and V. Sze, “Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks,” in *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, 2016.

[27] R. Kułaga and M. Gorgoń, “FPGA Implementation of Decision Trees and Tree Ensembles for Character Recognition in Vivado Hls,” *Image Process. Commun.*, 2015.

[28] S. Afifi, H. GholamHosseini, and R. Sinha, “A low-cost FPGA-based SVM classifier for melanoma detection,” in *IECBES 2016 - IEEE-EMBS Conference on Biomedical Engineering and Sciences*, 2016.

[29] Y. Pu, J. Peng, L. Huang, and J. Chen, “An efficient KNN algorithm implemented on FPGA based heterogeneous computing system using OpenCL,” in *Proceedings - 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2015*, 2015.

[30] L. Cavigelli and L. Benini, “Origami: A 803-GOp/s/W Convolutional Network Accelerator,” *IEEE Trans. Circuits Syst. Video Technol.*, 2017.

[31] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, “Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI,” *Dig. Tech. Pap. - IEEE Int. Solid-State Circuits Conf.*, vol. 60, pp. 246–247, 2017.

[32] H. A. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” *Reports Pract. Oncol. Radiother.*, 2009.

[33] D. Kim, J. Ahn, and S. Yoo, “A novel zero weight/activation-aware hardware architecture of convolutional neural network,” *Proc. 2017 Des. Autom. Test Eur. DATE 2017*, pp. 1462–1467, 2017.

[34] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, “A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones,” *IEEE Internet Things J.*, 2019.

[35] S. Benatti, F. Montagna, V. Kartsch, A. Rahimi, D. Rossi, and L. Benini, “Online Learning and Classification of EMG-Based Gestures on a Parallel Ultra-Low Power Platform Using Hyperdimensional Computing,” *IEEE Trans. Biomed. Circuits Syst.*, 2019.

[36] M. Osta *et al.*, "An energy efficient system for touch modality classification in electronic skin applications," in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2019, vol. 2019-May.

[37] M. Shafique *et al.*, "Adaptive and Energy-Efficient Architectures for Machine Learning: Challenges, Opportunities, and Research Roadmap," in *Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, 2017.

[38] J. Kung, D. Kim, and S. Mukhopadhyay, "A power-aware digital feedforward neural network platform with backpropagation driven approximate synapses," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2015.

[39] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "AxNN: Energy-efficient neuromorphic systems using approximate computing," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2015.

[40] C. Y. Chen, J. Choi, K. Gopalakrishnan, V. Srinivasan, and S. Venkataramani, "Exploiting approximate computing for deep learning acceleration," in *Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition, DATE 2018*, 2018.

[41] S. Nakagawa, H. Osawa, K. Asakura, N. Obara, Y. Tsuchiya, and M. Narita, "Web application technologies for integration of remote operation, camera image and voice communication into a cloud-based robotics platform," in *10th France-Japan Congress, 8th Europe-Asia Congress on Mecatronics, MECATRONICS 2014*, 2014.

[42] M. Dridi, M. A. Hajjaji, B. Bouallegue, and A. Mtibaa, "Cryptography of medical images based on a combination between chaotic and neural network," *IET Image Process.*, 2016.

[43] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient ConvNets through approximate computing," in *2016 IEEE Winter Conference on Applications of Computer Vision, WACV 2016*, 2016.

[44] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *33rd International Conference on Machine Learning, ICML 2016*, 2016.

[45] P. Judd *et al.*, "Reduced-Precision Strategies for Bounded Memory in Deep Neural Nets," pp. 1–12, 2015.

[46] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *MICRO 2013 - Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013.

[47] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar,

"Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," in *Proceedings - Design Automation Conference*, 2010.

[48] F. Sampaio, M. Shafique, B. Zatt, S. Bampi, and J. Henkel, "Approximation-aware Multi-Level Cells STT-RAM cache architecture," in *2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, CASES 2015*, 2015.

[49] A. Ranjan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan, "Approximate storage for energy efficient spintronic memories," in *Proceedings - Design Automation Conference*, 2015.

[50] J. He and J. Callenes-Sloan, "Poster: A software-defned hybrid cache with reduced energy," in *Middleware 2017 - Proceedings of the 2017 Middleware Posters and Demos 2017: Proceedings of the Posters and Demos Session of the 18th International Middleware Conference*, 2017.

[51] M. S. Razlighi, M. Imani, F. Koushanfar, and T. Rosing, "LookNN: Neural network with no multiplication," *Proc. 2017 Des. Autom. Test Eur. DATE 2017*, pp. 1775–1780, 2017.

[52] G. Venkatesh, E. Nurvitadhi, and D. Marr, "Accelerating Deep Convolutional Networks using low-precision and sparsity," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2017.

[53] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *32nd International Conference on Machine Learning, ICML 2015*, 2015.

[54] S. S. Sarwar *et al.*, "Energy efficient neural computing: A study of cross-layer approximations," *IEEE J. Emerg. Sel. Top. Circuits Syst.*, 2018.

[55] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," *IEEE Trans. Circuits Syst. I Regul. Pap.*, 2010.

[56] T. J. Yang, Y. H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017.

[57] H. Fares *et al.*, "Distributed sensing and stimulation systems for sense of touch restoration in prosthetics," in *Proceedings - 2017 1st New Generation of CAS, NGCAS 2017*, 2017.

[58] M. Magno, A. Ibrahim, A. Pullini, M. Valle, and L. Benini, "Energy efficient system for tactile data decoding using an ultra-low power parallel platform," in *Proceedings - 2017 1st New Generation of CAS, NGCAS 2017*, 2017.

[59] M. Magno, A. Ibrahim, A. Pullini, M. Valle, and L. Benini, “An energy efficient E-skin embedded system for real-time tactile data decoding,” *J. Low Power Electron.*, 2018.

[60] M. Osta, A. Ibrahim, L. Seminara, H. Chible, and M. Valle, “Low power approximate multipliers for energy efficient data processing,” *J. Low Power Electron.*, vol. 14, no. 1, 2018.

[61] M. Osta, A. Ibrahim, H. Chible, and M. Valle, “Inexact Arithmetic Circuits for Energy Efficient IoT Sensors Data Processing,” in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2018, vol. 2018-May.

[62] M. Osta, A. Ibrahim, and M. Valle, “FPGA Implementation of Approximate CORDIC Circuits for Energy Efficient Applications,” pp. 1–2.

[63] A. Ibrahim, M. Valle, L. Noli, and H. Chible, “FPGA implementation of fixed point CORDIC-SVD for E-skin systems,” in *2015 11th Conference on Ph.D. Research in Microelectronics and Electronics, PRIME 2015*, 2015.

[64] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, “Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing,” *IEEE J. Solid-State Circuits*, 2019.

[65] J. Han, “Introduction to approximate computing,” in *Proceedings of the IEEE VLSI Test Symposium*, 2016.

[66] Uma, R., et al. "Area, delay and power comparison of adder topologies." International Journal of VLSI Design & Communication Systems 3.1 (2012): 153.

[67] Srinivas, N., and Y. Rajasree Rao. "DESIGN AND ANALYSIS OF ENHANCED DADDA MULTIPLIER USING 5: 2 COMPRESSORS." International Journal of Pure and Applied Mathematics 118.19 (2018): 3021-3033.

[68] C. S. Wallace, “A Suggestion for a Fast Multiplier,” *IEEE Trans. Electron. Comput.*, 1964.

[69] V. S. Muley, A. Tom, and T. Vigneswaran, “Design of Baugh Wooley and Wallace tree multiplier using two phase clocked adibatic static CMOS logic,” in *2015 International Conference on Industrial Instrumentation and Control, ICIC 2015*, 2015.

[70] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, “RoBA Multiplier: A Rounding-Based Approximate Multiplier for High-Speed yet Energy-Efficient Digital Signal Processing,” *IEEE Trans. Very Large Scale Integr. Syst.*, 2017.

[71] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, “Design of voltage-

scalable meta-functions for approximate computing," in *Proceedings -Design, Automation and Test in Europe, DATE*, 2011.

[72] Y. Kim, Y. Zhang, and P. Li, "Energy Efficient Approximate Arithmetic for Error Resilient Neuromorphic Computing," *IEEE Trans. Very Large Scale Integr. Syst.*, 2015.

[73] I. C. Lin, Y. M. Yang, and C. C. Lin, "High-Performance Low-Power Carry Speculative Addition with Variable Latency," *IEEE Trans. Very Large Scale Integr. Syst.*, 2015.

[74] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, 2013.

[75] L. Li and H. Zhou, "On error modeling and analysis of approximate adders," in *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, 2015.

[76] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, 2013.

[77] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate XOR/XNOR-based adders for inexact computing," *Proc. IEEE Conf. Nanotechnol.*, pp. 690–693, 2013.

[78] P. Kulkarni, P. Gupta, and M. D. Ercegovac, "Trading accuracy for power in a multiplier architecture," in *Journal of Low Power Electronics*, 2011.

[79] K. Bhardwaj and P. S. Mane, "ACMA: Accuracy-configurable multiplier architecture for error-resilient System-on-Chip," in *2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip, ReCoSoC 2013*, 2013.

[80] K. Bhardwaj, P. S. Mane, and J. Henkel, "Power- and area-efficient approximate Wallace tree multiplier for error-resilient systems," in *Proceedings - International Symposium on Quality Electronic Design, ISQED*, 2014.

[81] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi, "Design-Efficient Approximate Multiplication Circuits Through Partial Product Perforation," *IEEE Trans. Very Large Scale Integr. Syst.*, 2016.

[82] C. H. Lin and I. C. Lin, "High accuracy approximate multiplier with error correction," in *2013 IEEE 31st International Conference on Computer Design, ICCD 2013*, 2013.

[83] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of

approximate compressors for multiplication," *IEEE Trans. Comput.*, 2015.

[84] 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 Trans. Very Large Scale Integr. Syst.*, 2015.

[85] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "EvoApproxSb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proceedings of the 2017 Design, Automation and Test in Europe, DATE 2017*, 2017.

[86] C. I. Allen, D. Langley, and J. C. Lyke, "Inexact computing with approximate adder application," in *National Aerospace and Electronics Conference, Proceedings of the IEEE*, 2015.

[87] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Proceedings -Design, Automation and Test in Europe, DATE*, 2014.

[88] A. Ibrahim, M. Valle, L. Noli, and H. Chible, "Singular value decomposition FPGA implementation for tactile data processing," in *Conference Proceedings - 13th IEEE International NEW Circuits and Systems Conference, NEWCAS 2015*, 2015.

[89] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE Trans. Very Large Scale Integr. Syst.*, 2010.

[90] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *2010 IEEE International Conference of Electron Devices and Solid-State Circuits, EDSSC 2010*, 2010.

[91] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited: Cross-layer approximate computing: From logic to architectures," in *Proceedings - Design Automation Conference*, 2016.

[92] M. Franceschi, L. Seminara, S. Dosen, M. Strbac, M. Valle, and D. Farina, "A System for Electrotactile Feedback Using Electronic Skin and Flexible Matrix Electrodes: Experimental Evaluation," *IEEE Trans. Haptics*, 2017.

[93] L. Pinna, A. Ibrahim, and M. Valle, "Interface Electronics for Tactile Sensors Based on Piezoelectric Polymers," *IEEE Sens. J.*, 2017.

[94] B. Shao and P. Li, "Array-Based Approximate Arithmetic Computing: A General Model and Applications to Multiplier and Squarer Design," *IEEE Trans. Circuits Syst. I Regul. Pap.*, 2015.

[95] Y. Zhou, Z. Chen, J. Lin, and Z. Wang, "A High-Speed Successive-Cancellation Decoder for Polar Codes Using Approximate Computing," *IEEE Trans. Circuits*

Syst. II Express Briefs, 2019.

[96] A. Ibrahim and M. Valle, “Approximate computing techniques for low power implementation of reconfigurable coordinate rotation digital computer circuits,” *J. Low Power Electron.*, 2017.

[97] H. Sun, Z. Cheng, A. M. Gharehbaghi, S. Kimura, and M. Fujita, “Approximate DCT Design for Video Encoding Based on Novel Truncation Scheme,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, 2019.

[98] L. Chen, J. Han, W. Liu, P. Montuschi, and F. Lombardi, “Design, evaluation and application of approximate high-radix dividers,” *IEEE Trans. Multi-Scale Comput. Syst.*, 2018.

[99] L. Seminara *et al.*, “Piezoelectric polymer transducer arrays for flexible tactile sensors,” *IEEE Sens. J.*, 2013.

[100] V. S. Rosa, F. F. Daitx, E. Costa, and S. Bampi, “Design flow for the generation of optimized FIR filters,” in *2009 16th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2009*, 2009.

[101] M. D. Ercegovac and T. Lang, “CORDIC Algorithm and Implementations,” in *Digital Arithmetic*, 2004.

[102] B. Khurshid and R. N. Mir, “Power efficient implementation of bit-parallel unrolled CORDIC structures for FPGA platforms,” in *2015 International Conference on VLSI Systems, Architecture, Technology and Applications, VLSI-SATA 2015*, 2015.

[103] R. S. Dahiya, G. Metta, M. Valle, and G. Sandini, “Tactile sensing-from humans to humanoids,” *IEEE Trans. Robot.*, 2010.

[104] S. Decherchi, P. Gastaldo, R. S. Dahiya, M. Valle, and R. Zunino, “Tactile-data classification of contact materials using computational intelligence,” *IEEE Trans. Robot.*, 2011.

[105] N. Jamali and C. Sammut, “Majority voting: Material classification by tactile sensing using surface texture,” *IEEE Trans. Robot.*, 2011.

[106] F. Naya, J. Yamato, and K. Shinozawa, “Recognizing human touching behaviors using a haptic interface for a pet-robot,” in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 1999.

[107] D. Silvera-Tawil, D. Rye, and M. Velonaki, “Interpretation of Social Touch on an Artificial Arm Covered with an EIT-based Sensitive Skin,” *Int. J. Soc. Robot.*, 2014.

[108] H. Iwata and S. Sugano, “Human-robot-contact-state identification based on tactile recognition,” *IEEE Trans. Ind. Electron.*, 2005.

[109] A. Flagg, D. Tam, K. MacLean, and R. Flagg, "Conductive fur sensing for a gesture-aware furry robot," in *Haptics Symposium 2012, HAPTICS 2012 - Proceedings,* 2012.

[110] M. D. Cooney, S. Nishio, and H. Ishiguro, "Recognizing affection for a touch-based interaction with a humanoid robot," in *IEEE International Conference on Intelligent Robots and Systems,* 2012.

[111] N. Sae-Bae, N. Memon, and K. Isbister, "Investigating multi-touch gestures as a novel biometric modality," in *2012 IEEE 5th International Conference on Biometrics: Theory, Applications and Systems, BTAS 2012,* 2012.

[112] M. Signoretto, L. De Lathauwer, and J. A. K. Suykens, "A kernel-based framework to tensorial data analysis," *Neural Networks*, 2011.

[113] C. Cortes and V. Vapnik, "Support-Vector Networks," *Mach. Learn.*, 1995.

[114] V. Vapnik, *Statistical learning theory. 1998*. 1998.

[115] G. Bin Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst. Man, Cybern. Part B Cybern.,* 2012.

[116] Q. Zhao, G. Zhou, T. Adali, L. Zhang, and A. Cichocki, "Kernelization of tensor-based models for multiway data analysis: Processing of multidimensional structured data," *IEEE Signal Process. Mag.*, 2013.

[117] P. Gastaldo, L. Pinna, L. Seminara, M. Valle, and R. Zunino, "A tensor-based pattern-recognition framework for the interpretation of touch modality in artificial skin systems," *IEEE Sens. J.*, 2014.

[118] S. Decherchi, P. Gastaldo, J. Redi, and R. Zunino, "Maximal-Discrepancy bounds for Regularized Classifiers," in *Proceedings of the International Joint Conference on Neural Networks*, 2009.

[119] E. Ragusa, C. Gianoglio, P. Gastaldo, and R. Zunino, "A digital implementation of extreme learning machines for resource-constrained devices," *IEEE Trans. Circuits Syst. II Express Briefs*, 2018.

[120] A. Rubaai and P. Young, "Hardware/software implementation of fuzzy-neural-network self-learning control methods for brushless DC motor drives," *IEEE Trans. Ind. Appl.*, vol. 52, no. 1, pp. 414–424, 2016.

[121] D. Tong, Y. R. Qu, and V. K. Prasanna, "Accelerating Decision Tree Based Traffic Classification on FPGA and Multicore Platforms," *IEEE Trans. Parallel Distrib. Syst.*, 2017.

[122] M. Magno, M. Pritz, P. Mayer, and L. Benini, "DeepEmote: Towards multi-layer

neural networks in a low power wearable multi-sensors bracelet," in *Proceedings - 2017 7th International Workshop on Advances in Sensors and Interfaces, IWASI 2017*, 2017.

[123] Z. Wang, Y. Liu, Y. Sun, Y. Li, D. Zhang, and H. Yang, "An energy-efficient heterogeneous dual-core processor for Internet of Things," in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2015.

[124] F. Conti, D. Palossi, R. Andri, M. Magno, and L. Benini, "Accelerated Visual Context Classification on a Low-Power Smartwatch," *IEEE Trans. Human-Machine Syst.*, 2017.

[125] A. Pullini, D. Rossi, I. Loi, A. Di Mauro, and L. Benini, "Mr. Wolf: A 1 GFLOP/s Energy-Proportional Parallel Ultra Low Power SoC for IOT Edge Processing," in *ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference*, 2018.

[126] IEEE, *(754) IEEE Standard for Floating-Point Arithmetic*. 2008.

[127] GNU, "Gnu MPFR," *Library (Lond).*, 2010.

[128] A. Ibrahim, M. Osta, M. Alameh, M. Saleh, H. Chible, and M. Valle, "Approximate Computing Methods for Embedded Machine Learning," in *2018 25th IEEE International Conference on Electronics Circuits and Systems, ICECS 2018*, 2019.