INTEGRATED SPIKE DETECTION USING APPROXIMATE METHODS

Ph.D. Thesis presented
for the fulfillment of the Degree of Doctor of Philosophy
in Information Communication Technology for Health
by

EFSTRATIOS ZACHARELOS

Advisor: Prof. Ettore Napoli

March 2023

Index of Contents

List of Acronyms

AC	Approximate Computing
HA	Half Adder
TT	Truth Table
FA	Full Adder
CBAA	Carry Based Approximation Adder
ED	Error Distance
FFT	Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform
HDL	Hardware Description Language
PP	Partial Product
PPM	Partial Product Matrix
SWAC	Single-Weight Approximate Compressor
FET	Field Effect Transistor
ED	Error Distance
RED	Relative Error Distance
ME	Mean Error
NMED	Normalized Mean Error Distance
MRED	Mean Relative Error Distance
NoEB	Number of Effective Bits
ER	Error Rate
TDM	Three-Dimensional Method
SWAC	Single Weight Approximate Compressor
ECAD	Electronic Computer-aided Design
SSIM	Structural Similarity Index Method
PSNR	Peak Signal to Noise Ratio
CNN	Convolutional Neural Network
SD	Standard Deviation
ReLU	Rectified Linear Unit
MNIST	Modified National Institute of Standards and Technology
SVHN	Street View House Number

MSB	Most Significant Bit
LSB	Least Significant Bit
DSP	Digital Signal Processing
ADC	Analog-to-Digital Converter
RMSE	Root Mean Square Error
RMS	Root Mean Square
ARMS	Average Root Mean Square
BMI	Brain-Machine Interface
EEG	Electroencephalography
MEG	Magnetoencephalography
ECoG	Electrocorticography
fMRI	functional Magnetic Resonance Imaging
fNIRS	functional Near Infrared Spectroscopy
PET	Positron Emission Tomography
MEA	Microelectrode Arrays
MAC	Multiply And Accumulate
LNA	Low Noise Amplifier
LFP	Local Field Potential
ADO	Absolute Differential Operator
ASO	Amplitude Slope Operator

Chapter 1

Introduction and Outline

1.1 Motivations

Recent progress in VLSI circuits has allowed the implementation of valuable devices related to healthcare by providing fully integrated solutions to continuously monitor a large variety of data obtained by biosensors. Continuous monitoring and processing of such data in vivo can be achieved by means of integrated and implantable devices, using wireless communication [Car12].

A wide range of neurological and cardiovascular diseases that are not easily tackled with conventional medication techniques, have driven research and industry towards the development of such implantable, yet minimally invasive electronic devices [Baz12]. Implantable biochips could also reduce the need for expensive medical procedures. Millions of patients benefit from such instruments, like cardioverter defibrillators, cochlear implants, gastric and cardiac pacemakers, deep brain, nerve, bone, and spinal cord stimulators, etc.

Long-term implants with in-vivo functionalities, are characterized by certain requirements, like a high-degree of integration, minimally invasive surgery, long-term biocompatibility, as well as security and privacy in data transmission. Other concerns include low energy consumption, small silicon area and weight, and reliable performance. Tasks like real time stimulation, data collection, processing, compression, and transmission, contribute to the overall power budget of the device which should remain as low as possible.

A promising paradigm which refers to a set of methods that relax the constraint of exact equivalence between the specification and implementation of a computing system, has recently emerged to deliver energy efficient designs for cloud computing or embedded and mobile

digital systems [Han13]. Approximate computing (AC) exploits the ability of many systems and applications to tolerate a reasonable loss of quality in the calculated result. By allowing the possibility of inexact outputs, approximate computing can considerably improve power consumption, silicon area requirements, as well as the system's critical delay.

Although deviations from the exact result are generally unwanted, applications such as multimedia, signal processing, machine learning, pattern recognition, and data mining are tolerant to the occurrence of some errors [Liu20]. The error resilience may be attributed to: (a) perpetual limitations, as human for example may not be able to distinguish trivial details in a processed image, (b) redundancy in the processed data, as an algorithm might be able to withstand approximations and still derive the correct result, due to a sufficient flux of input data, (c) noisy inputs that can make rough estimations appropriate.

The binary adder, which is a fundamental arithmetic unit, can serve as a great example for approximate computing applications, as it has already attracted great scientific interest [Gup11], [Seo20], [Seo21]. It is after all, an elementary block of the binary multiplier, which is a core element that contributes significantly to the overall power consumption in microprocessors and signal processing systems.

The half adder (HA) is the most basic combinational logic circuit that can return the sum of two bits, by generating two output bits, the carry, and the sum. As shown in Fig. 1.1.a, one XOR and one AND gate are required for the HA circuit. Unfortunately, XOR gates are known to be slow and bulky and so, avoiding them can be beneficial. A simple way to carefully approximate the HA, while relaxing the hardware requirements is shown in Fig. 1.1.b. The XOR gate is substituted by an OR gate, and the AND gate is discarded. As shown in the truth table (TT), this rather simple configuration results in an error when both inputs are high. In this case, the Error Distance (ED) is 1. The computed output is binary '01', instead of '10' and the result is underestimated by 1. In the three other cases, the inexact circuit returns the correct result. Given the appropriate application, the inexact HA can be an appealing solution, as it offers a significant hardware minimization.

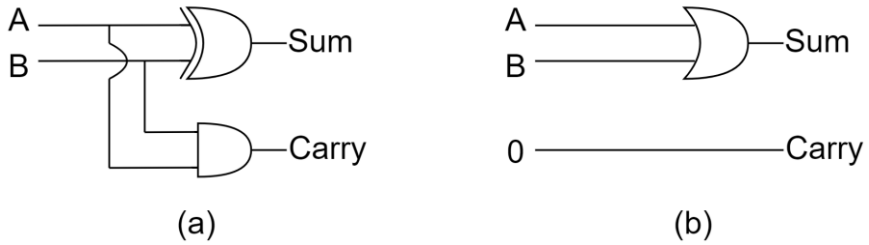

Fig. 1.1: (a) Exact Half Adder. Using one XOR and one AND gate, this circuit is always precise. **(b)** Approximate Half Adder. Using only one OR gate, this circuit returns inexact results when both inputs are high.

Table 1.1: Exact Half Adder Truth Table **(left)** and Approximate Half Adder Truth Table with error distance **(right)**.

A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

A	B	Carry	Sum	ED
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	**0**	**1**	**1**

The Full Adder (FA) is another logic circuit that has been extensively studied in order to derive simplifications. Fig. 1.2.a shows the conventional FA while the Boolean expressions of its outputs are:

$$Sum = A \oplus B \oplus C \quad (1.1)$$

$$Carry = (A \cdot B) + \big((A \oplus B) \cdot C\big) \quad (1.2)$$

The authors in [Ram19] propose a gate level logic modification approach to approximate the FA circuit. The sum term of the conventional full adder is altered to reduce the hardware complexity by proposing a carry-based approximation adder (CBAA) and avoid the critical XOR operation. The conventional Full Adder, and the proposed approximate versions are shown in Fig. 1.2. The TTs and the error distances are summarized in Table 1.2. As shown in Fig. 1.2, in all the approximated versions of the FA, the Sum is the inverse of the corresponding carry, resulting in reduced complexity at gate level with respect to the conventional FA. The Boolean equations for the carries of the approximate FAs are:

$$C1 = (A \cdot B) + C \quad (1.3)$$

$$C2 = (A \cdot B) \oplus C \quad (1.4)$$

$$C3 = (A \cdot B) + (B \cdot C) + (C \cdot A) \quad (1.5)$$

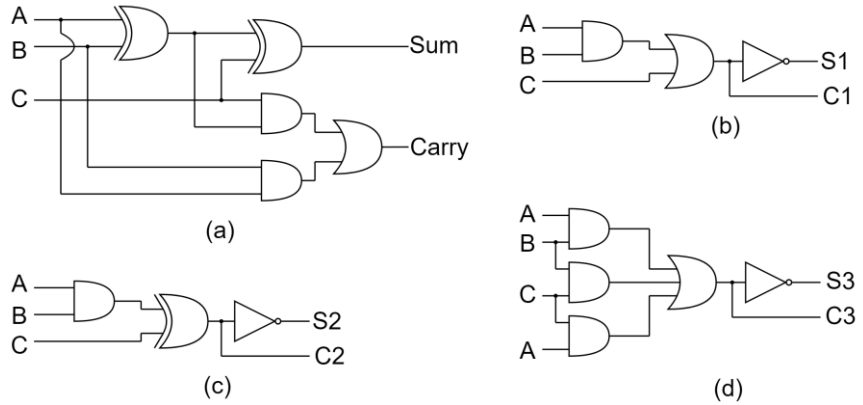

Fig. 1.2: (a) Exact Full Adder. Using two XOR, two AND, and one OR gate, this circuit is always precise. **(b, c, d)** Approximate Full Adder alternatives.

Table 1.2: Exact Full Adder Truth Table **(a)** and Approximate Full Adder Truth Tables with error distance **(b, c, d)**.

			Exact FA (a)		**(b)**			**(c)**			**(d)**		
A	**B**	**C**	**Carry**	**S**	**C1**	**S1**	**ED**	**C2**	**S2**	**ED**	**C3**	**S3**	**ED**
0	0	0	0	0	0	**1**	**1**	0	**1**	**1**	0	**1**	**1**
0	0	1	0	1	**1**	**0**	**1**	**1**	**0**	**1**	0	1	0
0	1	0	0	1	0	1	0	0	1	0	0	1	0
0	1	1	1	0	1	0	0	1	0	0	1	0	0
1	0	0	0	1	0	1	0	0	1	0	0	1	0
1	0	1	1	0	1	0	0	1	0	0	1	0	0
1	1	0	1	0	1	0	0	1	0	0	1	0	0
1	1	1	1	1	1	**0**	**1**	**0**	1	**2**	1	**0**	**1**

Several proposals like the ones already mentioned for the Half and Full Adders, are developed in the literature. The most common circuits targeted for approximation, are adders of an arbitrary number of bits, signed and unsigned binary multipliers, squarers, and dividers. These solutions can be exploited in embedded and mobile devices, where energy, area, and speed constraints are important.

The main research topic of this dissertation is approximate computing and ways to overcome the state of the art, while a health application benefitting from approximate computing is investigated in the final chapter.

1.2 Thesis Outline

The core of this manuscript is divided into the following chapters, each of which ends with a summary section to provide a quick reference to the corresponding main findings and results.

Chapter 2 provides a scientific background including information on the state of the art of approximate computing and a brief overview of the error metrics that are used to measure the error performance of the various approximate circuits.

In **Chapter 3,** a method to reduce the partial products of a binary multiplier, using approximate compressors is described. Different sizes of approximate multipliers are developed and tested against the state of the art.

In **Chapter 4,** various approximate 4×4 multiplier blocks are developed and used recursively to scale up to higher order multipliers. The obtained designs are compared to numerous recursive architectures found in the literature.

In **Chapter 5,** approximations in another fundamental arithmetic operation, are investigated. Approximate recursive squarers are developed and tested against the state of the art.

In **Chapter 6,** a Machine Learning application to detect spikes in brain activity is developed. The obtained network is described and implemented in an integrated circuit to serve as a brain-machine interface.

Finally, **conclusions** are drawn, and **future** goals are mentioned.

1.3 List of Publications

- M. D'Arco, E. Napoli, **E. Zacharelos**, S. Saponara, and A. De Gloria, "Digital Circuit for the Arbitrary Selection of Sample Rate in Digital Storage Oscilloscopes," in *Applications in Electronics Pervading Industry, Environment and Society*, 2020, doi: 10.1007/978-3-030-37277-4_21.

- M. D'Arco, E. Napoli, and **E. Zacharelos**, "Digital Circuit for Seamless Resampling ADC Output Streams," in *Sensors,* vol. 20, no. 6, 2020, doi: https://doi.org/10.3390/s20061619.

- E. Napoli, **E. Zacharelos**, M. D'Arco and A. G. M. Strollo, "Real-Time Downsampling in Digital Storage Oscilloscopes With Multichannel Architectures," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 10, pp. 4142-4155, Oct. 2021, doi: 10.1109/TCSI.2021.3102386.

- M. D'Arco, E. Napoli, **E. Zacharelos**, L. Angrisani, and A. G. M. Strollo, "Enabling Fine Sample Rate Settings in DSOs with Time-Interleaved ADCs," in *Sensors*, vol 22, no. 1, 2022, doi: https://doi.org/10.3390/s22010234.

- I. Nunziata, **E. Zacharelos**, G. Saggese, A. M. G. Strollo and E. Napoli, "Approximate Recursive Multipliers Using Carry Truncation and Error Compensation," *17th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, Villasimius, SU, Italy, pp. 137-140, 2022, doi: 10.1109/PRIME55000.2022.9816787.

- G. Saggese, **E. Zacharelos**, and A. G. M. Strollo, "Low Power Spike Detector for Brain-Silicon Interface using Differential Amplitude Slope Operator," *17th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, Villasimius, SU, Italy, 2022, pp. 301-304, doi: 10.1109/PRIME55000.2022.9816758.

- **E. Zacharelos**, I. Nunziata, G. Saggese, A. G. M. Strollo and E. Napoli, "Approximate Recursive Multipliers Using Low Power Building Blocks," in *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1315-1330, 1 July-Sept. 2022, doi: 10.1109/TETC.2022.3186240.

- **E. Zacharelos**, I. Nunziata, G. Saggese, A. G. M. Strollo and E. Napoli, "Approximate Recursive Multipliers Using Low Power Building Blocks," in *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*, vol. 10, no. 3, pp. 1315-1330, 1 July-Sept. 2022, doi: 10.1109/TETC.2022.3186240.

- **E. Zacharelos**, I. Nunziata, G. Saggese, A. G. M. Strollo and E. Napoli, "Approximate Squaring circuits exploiting Recursive Architectures," in Integration, the VLSI journal, Elsevier, 2023, doi: https://doi.org/10.1016/j.vlsi.2023.02.007.

- **E. Zacharelos**, C. Scognamillo, E. Napoli, and G. Saggese " On-Chip Spike Detection and Classification using Neural Networks and Approximate Computing," 2023. **[Currently Under Revision]**

Chapter 2

Scientific Background

2.1 State of the Art

In the last years, many contributions related to approximate computing have focused on arithmetic operations, such as binary addition, multiplication, and division [Soa19], [Gup13], [Che18]. Binary multipliers constitute a fundamental part of many digital processing systems, such as FFT/IFFT hardware implementation algorithms, and unfortunately are characterized by heavy silicon area, power, and timing requirements [Hor14]. Consequently, nowadays approximate binary multipliers are being studied thoroughly. A comprehensive survey of approximate arithmetic circuits, such as approximate adders, multipliers, and dividers is developed in [Jia20].

Several techniques providing efficient approximate multipliers have been studied in the literature. One such example is the approximate logarithmic multiplier [Liu18], [Kim19], [Lot21]. In this case, approximated versions of the logarithms of the input operands, are added. The result corresponds to the approximated value of the antilogarithm of the sum. These are low-power and high-speed designs, due to the low complexity in their architecture. However, they tend to be less accurate. Another approach is the static segmentation. In this technique, a part of each input operand is given as input to a small multiplier, whose shifted output is the result of the multiplication [Str22]. Static segmentation has been demonstrated to be useful when very low power is needed, and accuracy is not the main issue. In [Yan18] the authors propose an approximate multiplier that can dynamically control accuracy. The circuit can select the length of the carry propagation to effectively satisfy the desired accuracy requirements.

Software-based approaches have been proposed, that merge the approximated multiplier design in the design flow of the circuit. They

automatically generate synthesizable hardware description code (HDL) for approximate arithmetic circuits based on the accuracy requirement of the design [Češ18], [Ull18], [Mra20], [Bal22]. Such techniques can prove useful when the targeted application does not have a uniform input distribution.

The basic binary multiplication process can be divided into three parts: partial product generation, partial product reduction and carry-propagate addition. Approximate computing can be introduced in all these steps. For instance, the first step can be approximated by truncating some of the least significant partial products (PPs) and then possibly employing a compensation strategy [Vah19], [Fru20].

The partial product reduction step is typically the main target for approximations in a binary multiplier. A common approach to reduce the partial product matrix (PPM) relies on the use of approximate compressors. Compressors are logic circuits that aim to minimize the number of operands in the final step, which is the addition of the reduced partial products, using tree-based logarithmic reduction schemes, such as Wallace [Wal64], Dadda [Dad83], or the Three-Dimensional Method (TDM) [Okl96]. The compressors are XOR-rich circuits (thus slow and power hungry), that count the number of ones in the input. The most basic exact compressor is the Full Adder, that reduces three digits into two, maintaining the original information. As shown in the following, many research contributions have focused on the approximation of the PPM compression phase. Approximate compressors developed during this work, are discussed in chapter 3.

In [Kel09] the authors acquire approximate compressors by truncating outputs of some exact compressors, while in [Cil14] and [Guo18], compressors with only 2-bit outputs are proposed. Lossy compression of the rows in the PPM based on bit significance, is investigated in [Qiq17]; the compression exploits approximate, OR-based half adders. In [Esp17] simple OR gates serve as approximate compressors and two designs are proposed. The two designs are obtained using encoded partial products and approximate compressors, delivering different accuracy-electrical performance trade-off. Several solutions employing 3:2 and 4:2 compressors to generate approximated multipliers are presented in [Ans18], [Sab19], [StNa20], [StDe20]. A set of Single-Weight Approximate Compressors (SWACs) is employed in [Esp18], to construct approximate multipliers. Unlike the Full-Adder

that produces a sum and a carry, these designs compress input bits derived from a PPM column, into fewer output bits, maintaining the same initial weight. This allows a significant reduction of circuit complexity since less carry bits are generated and propagated. Maddisetty et al. [Mad19] present the training of a neural network to devise an efficient approximate 4:2 compressor. In [Eda20] two 4:2 compressors are presented; a novel 4:2 architecture, and a modified design by substituting the AND / OR gates with NAND / NOR gates respectively. Although the boolean expression is changed, when the modified version targets multipliers, employing reduction steps in multiples of 2, the difference is nullified. Approximate 4:2 designs implemented in FinFET technology are presented in [Zak20], [Kha21]. In [Pei21] the number of outputs of the approximate 4:2 compressor is innovatively reduced to one; 3 such compressors are proposed, as well as an error-correcting module.

Recursive multipliers are an interesting research area of the approximate computing field that aims to use small elementary approximate multiplier blocks, suitably assembled, to design larger multipliers, [Ans18], [Kul11], [Reh16], [GSK18], [Gil19], [War20], [Yan20], [War21]. The advantage of the recursive building of larger multipliers is that it avoids a dedicated design for every bit width and gains in terms of generality of the proposed approaches. As demonstrated in [Ans18], and in chapter 4 of this dissertation, four $n \times n$ building blocks can be utilized to scale up to a $2n \times 2n$ multiplier. Several authors have used 4×4 approximate multipliers to recursively generate several 8×8 multiplier alternatives. The authors of [Ans18] propose three 4:2 compressors, used to generate two 4×4 multipliers.

Guo et al. [GSK18] propose a 4×4 approximate multiplier module. The corresponding 8×8 multiplier is made up from one 4×4 multiplier featuring OR-based compressors with no carry propagation in the lower part, two of the proposed 4×4 modules in the middle part, and an exact 4×4 multiplier for the most significant part. Differently from the other designs, the four products are summed using an approximate adder.

In [War20] the authors consider the probability distribution of the input operands to propose 4×4 multipliers, consisting of approximate NOR-based half adder and full adder designs. These elementary blocks are exploited to build approximate recursive

multipliers. In [Yan20] a 4×4 approximate multiplier featuring an error detection and correction system, is presented.

Similarly, in [Kul11], [Reh16], [Gil19] and [War21] the authors propose 2×2 approximate sub-multipliers, suitably arranged, to form larger size multipliers. Sixteen 2×2 modules are needed to create an 8×8 multiplier. Kulkarni et al. [Kul11] present a 2×2 inexact multiplier with tunable error characteristics. In [Reh16] the authors provide an exploration of the architectural space and propose their 2×2 module. The 2×2 approximate multiplier presented in [Gil19] has an internal self-healing strategy that does not require coupled modules, while the proposed larger multipliers derived from the 2×2 blocks produce near zero mean error. In [War21] two elementary multipliers are proposed that exhibit double-sided error distribution while the resulting 8×8 design has the advantage of error compensation.

Several attempts to outperform the state of the art in terms of power dissipation, silicon area, and critical delay, are described in the following chapters of this thesis. Approximate arithmetic circuits are developed exploiting novel approximate compressors and recursive architectures. The designs are then described in HDL and simulated, to derive their error performance. Finally, they are synthesized in a FinFET technology provided by Global Foundries. The targeted technology allows a minimum gate length of 14nm, while featuring two layers of metal. The typical DC supply voltage is 0.80 Volts, while the minimum and maximum limits are 0.54V and 0.95V, respectively. The same methodologies (including the technology) are used throughout this work for the proposed and competitive designs found in the literature, to ensure a fair comparison with the state of the art.

2.2 Error Metrics

To quantify the error performance of the different investigated approximate designs, several error metrics have been used throughout this work. These metrics are summarized in the following.

Assuming that A and B are the two n-bit operands in an $n \times n$ binary multiplication operation, with:

$$A = \sum_{i=0}^{n-1} a_i \cdot 2^i \quad , \quad B = \sum_{j=0}^{n-1} b_j \cdot 2^j \tag{2.1}$$

a random partial product p_{ij}, is given by:

$$p_{ij} = a_i b_j \tag{2.2}$$

If the input bits a_i and b_j are uniformly and independently distributed, the probability of each PP being high, is equal to:

$$P(p_{ij}) = 1/4 \tag{2.3}$$

Let Y_{E_k} be the exact result of the multiplication between the two n-bits operands A_k and B_k such that $Y_{E\_k} = A_k B_k$ and let Y_{A_k} be the approximated output returned by the investigated inexact multiplier. The error E_k, of each multiplication is given by:

$$E_k = Y_{E\_k} - Y_{A\_k} \tag{2.4}$$

While the error distance ED_k is defined as:

$$ED_k = \left| Y_{E\_k} - Y_{A\_k} \right| \tag{2.5}$$

And the relative error distance RED_k, as:

$$RED_k = ED_k / Y_{E\_k} \quad \forall \quad Y_{E\_k} \neq 0 \tag{2.6}$$

- The Mean Error, *ME*, is defined as the sum of errors, divided by the total amount of possible inputs 2^{2n}:

$$ME = \left(\sum_{0}^{2^{2n}-1} E_k \right) / 2^{2n} \tag{2.7}$$

- The Normalized Mean Error Distance, *NMED*, is defined as the average value of *ED* divided by the maximum possible value returned by the multiplier, which is: $(2^n\text{-}1)^2$.
- The Mean Relative Error Distance, *MRED*, is given by the average value of *RED*.
- The number of effective bits, *NoEB*, is defined as:

$$NoEB = 2n - log_2\left(1 + \sqrt{E_{ms}}\right) \tag{2.8}$$

  where E_{ms} is the means square error, given by the average value of E^2.
- The error rate, *ER*, is defined as the number of erroneous multiplications (with $E_k \neq 0$) over the total amount of possible inputs 2^{2n}.

2.3 Summary

In this chapter, a quick overview on the state of the art of approximate binary multipliers is given. Two main categories of multipliers are pointed out: those that exploit approximate compressors and those that benefit from recursive topologies. Then, the error metrics used throughout this work to measure the error performance, are presented.

Chapter 3

Approximate Multipliers Based on Approximate Compressors

As mentioned in chapter 2, the first objective of a binary multiplier is to generate the partial product matrix (PPM). Afterwards, the addition of the partial products in each column, takes place. At this stage, compressors can be used to reduce the size of the PPM. An exact compressor produces an output that is the sum of n selected input bits (usually of the same column, i.e., they have the same binary weight). The generated output bits are $m = \lceil log_2 n \rceil$, with different weights.

A common example of exact compressors is the Full-Adder, having an input of three bits and reducing it to two bits: the sum, and the carry, with a double weight. Another useful circuit in the PPM reduction phase, is the Half-Adder, which encodes two input bits into two output bits, exactly like the Full-Adder. While it does not reduce the number of bits in the PPM, many times it proves useful as it rearranges the order of bits, keeping one bit in the column of interest (sum), and generating one for the next column (carry).

An approximate compressor is typically a simpler circuit, liable to producing inexact results. As it will be shown in the following, two types of approximate compressors are presented in this chapter, that generate $m = \lceil n/2 \rceil$ output bits, all of which hold the same weight as the input bits. Therefore, they all are single-weight approximate compressors (SWACs). The first class of compressors is meant to minimize the hardware requirements, while maintaining the introduced error at reasonable levels. On the other hand, the second class of approximate compressors are as accurate as possible. Therefore, if the input bits whose value is high are not exceeding the number of output bits, the exact result will be encoded in the output. Otherwise, the resulting approximate output will be as close to the exact one, as possible. Hence, these compressors are error optimal.

SWACs of both types are presented for various sizes, namely 3:2, 4:2, 5:3, and 6:3. Higher order compressors are also employed, but they are derived as combinations of lower order compressors, as explained in section 3.3. All these designs are combined to form multipliers of different sizes. If suitably combined, they guarantee a great trade-off between outcome precision and electrical performances. Both types of compressors avoid the use of complex XOR gates.

3.1 Hardware-Efficient Approximate Compressors

In order to clarify the process used to produce hardware-efficient (Type-A) single weight approximate compressors, that has been developed in [Esp18], let us start with compressing two partial products into one bit. As shown by eq. 2.2, each PP is generated by multiplying two input bits. Let us assume that the PP p_0 is generated by the multiplication (AND gate) of the input bits x_0 and y_0, while the PP p_1 is generated by the multiplication of the input bits x_1 and y_1.

Table 3.1: Multiplier inputs, corresponding partial products, logical AND and OR of the partial products. In only one out of the 16 cases the logical AND is relevant and can thus be neglected.

x_0	y_0	x_1	y_1	p_0	p_1	$p_0 p_1$	$p_0 + p_1$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	1	0	1
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	0
0	1	1	1	0	1	0	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	1
1	1	0	0	1	0	0	1
1	1	0	1	1	0	0	1
1	1	1	0	1	0	0	1
1	1	1	1	1	1	1	1

As it can be observed in Table 3.1, the addition of two partial products, p_0 and p_1, can be recoded into the sum of the logical AND and the logical OR of the two partial products:

$$S = \sum\{p_0, p_1\} = \sum\{p_0 p_1, p_0 + p_1\} \tag{3.1}$$

Consulting Table 3.1, we can derive the probabilities of each term being high:

$$P(p_0 p_1) = 1/16 \tag{3.2}$$

$$P(p_0 + p_1) = 7/16\,. \tag{3.3}$$

Considering these probabilities, and neglecting the term generated exploiting the AND gate, we get an approximated version of the summing process:

$$S_{APP} = \sum\{p_0 + p_1\}\,. \tag{3.4}$$

The only case in which this approximated process results in an erroneous result, is when all the multiplier inputs in question are high, ($x_0 = y_0 = x_1 = y_1 = 1$), and as such, the two partial products to be compressed are also high, ($p_0 = p_1 = 1$). In this case, the approximated result is encoded in 1 bit and is equal to 1, instead of 2 which is the accurate result.

3.1.1 Approximate Compressor 3:2

An approximate 3:2 compressor reduces 3 partial products of the same column in the PPM, into 2 bits. Applying the same reasoning as in equation 3.1 twice:

$$S = \sum\{p_0, p_1, p_2\} = \sum\{p_0 p_1 p_2, p_0 p_1 + p_2, p_0 + p_1\}\,. \tag{3.5}$$

The first term in equation 3.5 is the logical AND of the three partial products and holds a very low probability of being high:

$$P(p_0 p_1 p_2) = P(p_0) \cdot P(p_1) \cdot P(p_2) = \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{4} = 1/64 \tag{3.6}$$

Therefore, we can neglect this term, thus obtaining:

$$S_{APP} = \sum\{p_0 p_1 + p_2, p_0 + p_1\}\,. \tag{3.7}$$

In table 3.2, the input PPs, the compressor's outputs, the exact and approximate sums, the error, and error probability, for all the possible partial product combinations are reported. We can observe that

Table 3.2: Input partial products, compressor outputs, exact and approximated sum, Error and Error probability for the Type-A 3:2 approximate compressor.

$p_0p_1p_2$	w_0w_1	S	S_{App}	E	P_E
000	00	0	0	0	-
001	10	1	1	0	-
010	01	1	1	0	-
011	11	2	2	0	-
100	01	1	1	0	-
101	11	2	2	0	-
110	11	2	2	0	-
111	11	3	**2**	**1**	**1/64**

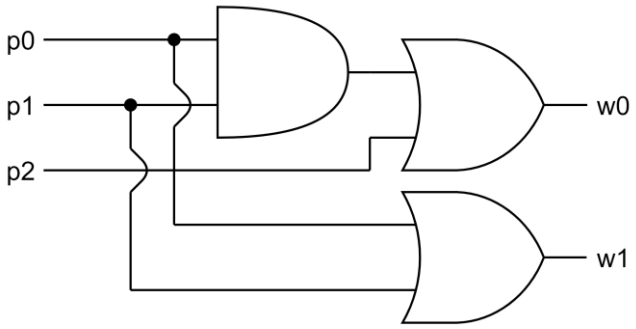

Fig. 3.1: Schematic of the proposed Type-A 3:2 approximate compressor.

the only erroneous case appears for $p_0 = p_1 = p_2 = 1$, where the exact result is 3 and the approximated 2. Hence, the error probability and mean error of the 3:2 approximate compressor respectively, are:

$$P_E = 1/64 \quad (3.8)$$

$$E_{mean} = 1/64 \quad (3.9)$$

The schematic of the proposed 3:2 approximate compressor is shown in Fig. 3.1. It is a quite simple design, employing just AND, OR gates, and avoiding large XOR gates. Furthermore, it provides high accuracy results, as explained earlier, making it an attractive alternative, when compared to similar circuits.

3.1.2 Approximate Compressor 4:2

Applying three times the same recoding technique to four partial products, we can obtain:

$$S = \sum \begin{Bmatrix} (p_0p_1)(p_2+p_3), (p_2p_3)(p_0+p_1), \\ (p_0p_1)+(p_2+p_3), (p_2p_3)+(p_0+p_1) \end{Bmatrix} \quad (3.10)$$

The probability of the first two terms being high is very low, so we can safely neglect them, thus obtaining the proposed 4:2 approximate compressor logic equations:

$$S_{APP} = \sum \{p_0p_1 + p_2 + p_3, p_2p_3 + p_0 + p_1\} \quad (3.11)$$

In table 3.3, we can observe the behavior of the proposed approximate compressor. There are five partial products combinations, out of the sixteen possibilities, that result in an error. In four of these cases, the approximated result is underestimated by one, while in the last, least probable case, it is underestimated by two. So, the error probability and mean error are:

$$P_E = 13/256 \quad (3.12)$$

$$E_{mean} = 14/256 \quad (3.13)$$

Fig. 3.2 shows the schematic of the proposed 4:2 approximate compressor. Again, a simple XOR-free design is enough to create a compressor with low error probability.

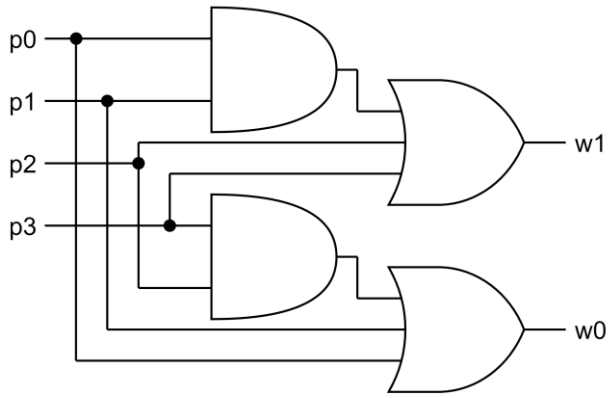

Fig. 3.2: Schematic of the proposed Type-A 4:2 approximate compressor.

Table 3.3: Input partial products, compressor outputs, exact and approximated sum, Error and Error probability for the Type-A 4:2 approximate compressor.

$p_0p_1p_2p_3$	w_0w_1	S	S_{App}	E	P_E
0000	00	0	0	0	-
0001	01	1	1	0	-
0010	01	1	1	0	-
0011	11	2	2	0	-
0100	10	1	1	0	-
0101	11	2	2	0	-
0110	11	2	2	0	-
0111	11	3	**2**	**1**	**3/256**
1000	10	1	1	0	-
1001	11	2	2	0	-
1010	11	2	2	0	-
1011	11	3	**2**	**1**	**3/256**
1100	11	2	2	0	-
1101	11	3	**2**	**1**	**3/256**
1110	11	3	**2**	**1**	**3/256**
1111	11	4	**2**	**2**	**1/256**

3.1.3 Approximate Compressor 5:3

The usual recoding is applied four times accordingly to acquire the 5:3 approximate compressor.

$$S = \sum \begin{Bmatrix} (p_0p_1)(p_2+p_3), (p_0p_1)+(p_2+p_3), \\ (p_2p_3)p_4, (p_2p_3)+p_4, \\ (p_0+p_1) \end{Bmatrix} \tag{3.14}$$

Neglecting the two low probability terms, namely the ones comprised by the AND of three terms, we obtain:

$$S_{APP} = \sum \{p_0p_1 + p_2 + p_3, p_0 + p_1, p_2p_3 + p_4\} \tag{3.15}$$

The behavior of the proposed compressor (Fig. 3.3) is exhibited in table 3.4. Out of the thirty-two cases, there are nine that produce an underestimated result. In the unlikely event that all the partial products are high, the produced result is underestimated by two. Consulting table 3.4, we can obtain:

$$P_E = 43/1024 \tag{3.16}$$

$$E_{mean} = 44/1024 \tag{3.17}$$

Table 3.4: Input partial products, compressor outputs, exact and approximated sum, Error and Error probability for the Type-A 5:3 approximate compressor.

$p_0p_1p_2p_3p_4$	$w_0w_1w_2$	S	S_{App}	E	P_E
00000	000	0	0	0	-
00001	100	1	1	0	-
00010	100	1	1	0	-
00011	101	2	2	0	-
00100	001	1	1	0	-
00101	101	2	2	0	-
00110	101	2	2	0	-
00111	101	3	**2**	**1**	**9/1024**
01000	001	1	1	0	-
01001	101	2	2	0	-
01010	101	2	2	0	-
01011	101	3	**2**	**1**	**9/1024**
01100	011	2	2	0	-
01101	111	3	3	0	-
01110	111	3	3	0	-
01111	111	4	**3**	**1**	**3/1024**
10000	010	1	1	0	-
10001	110	2	2	0	-
10010	110	2	2	0	-
10011	111	3	3	0	-
10100	011	2	2	0	-
10101	111	3	3	0	-
10110	111	3	3	0	-
10111	111	4	**3**	**1**	**3/1024**
11000	011	2	2	0	-
11001	111	3	3	0	-
11010	111	3	3	0	-
11011	111	4	**3**	**1**	**3/1024**
11100	011	3	**2**	**1**	**9/1024**
11101	111	4	**3**	**1**	**3/1024**
11110	111	4	**3**	**1**	**3/1024**
11111	111	5	**3**	**2**	**1/1024**

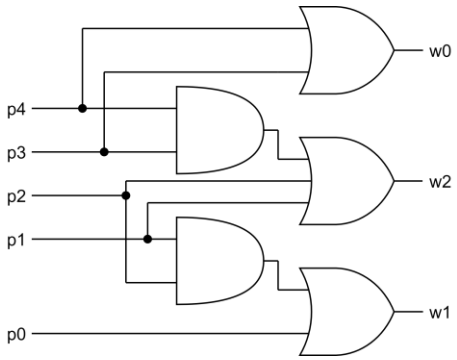

Fig. 3.3: Schematic of the proposed Type-A 5:3 approximate compressor

3.1.4 Approximate Compressor 6:3

For the 6:3 compressor, the recoding is applied six times:

$$S = \sum \begin{Bmatrix} (p_0p_1)(p_2+p_3), (p_0p_1)+(p_2+p_3), \\ (p_2p_3)(p_4+p_5), (p_2p_3)+(p_4+p_5), \\ (p_4p_5)(p_0+p_1), (p_4p_5)+(p_0+p_1) \end{Bmatrix} \quad (3.18)$$

By neglecting all the low probability terms that consist of the AND of three terms, we obtain:

$$S_{APP} = \sum \{p_0p_1 + p_2 + p_3, p_2p_3 + p_4 + p_5, p_4p_5 + p_0 + p_1\} \quad (3.19)$$

Table 3.5 reports only the partial products combinations resulting in an output error. According to this table, the error probability and mean error of the proposed approximate 6:3 compressor is:

$$P_E = 316/4096 \quad (3.20)$$

$$E_{mean} = 336/4096 \quad (3.21)$$

The schematic of the proposed Type-A 6:3 approximate compressor is shown in figure 3.4.

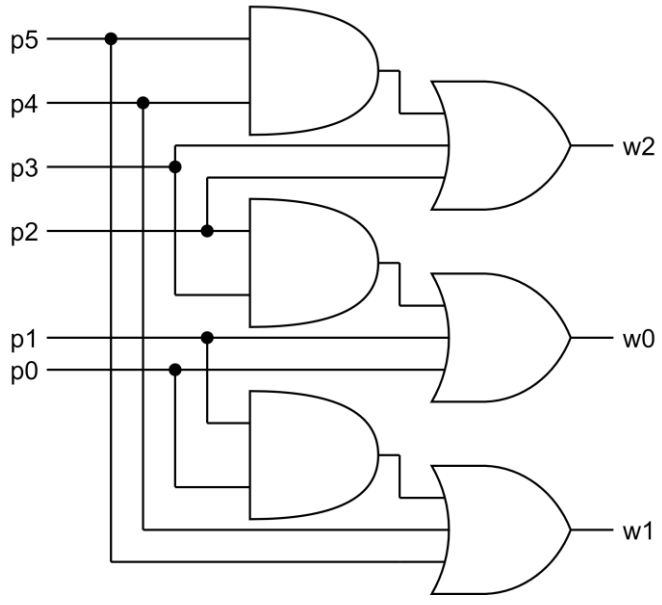

Fig. 3.4: Schematic of the proposed Type-A 6:3 approximate compressor

Table 3.5: Input partial products, compressor outputs, exact and approximated sum, Error and Error probability for the Type-A 6:3 approximate compressor.

$p_0p_1p_2p_3p_4p_5$	$w_0w_1w_2$	S	S_{App}	E	P_E
000111	011	3	2	1	27/4096
001011	011	3	2	1	27/4096
001111	111	4	3	1	9/4096
010111	111	4	3	1	9/4096
011011	111	4	3	1	9/4096
011100	101	3	2	1	27/4096
011101	111	4	3	1	9/4096
011110	111	4	3	1	9/4096
011111	111	5	3	2	3/4096
100111	111	4	3	1	9/4096
101011	111	4	3	1	9/4096
101100	101	3	2	1	27/4096
101101	111	4	3	1	9/4096
101110	111	4	3	1	9/4096
101111	111	5	3	2	3/4096
110001	110	3	2	1	27/4096
110010	110	3	2	1	27/4096
110011	111	4	3	1	9/4096
110101	111	4	3	1	9/4096
110110	111	4	3	1	9/4096
110111	111	5	3	2	3/4096
111001	111	4	3	1	9/4096
111010	111	4	3	1	9/4096
111011	111	5	3	2	3/4096
111100	111	4	3	1	9/4096
111101	111	5	3	2	3/4096
111110	111	5	3	2	3/4096
111111	111	6	3	3	1/4096

3.2 Error-Optimal Approximate Compressors

The novel compressors, shown in this section, are optimal in terms of error. In other words, they produce $m = \lceil n/2 \rceil$ output bits as mentioned earlier, that when added, deliver, if possible, the exact result. In case the exact result cannot be described by the m available output bits, the error is the minimum possible, i.e., the difference between the exact value, and the number of output bits. Unlike the simpler, in terms of hardware, designs shown in section 3.1, these compressors promise the lowest possible error, achieved by smart designs.

Even though the lowest possible error for a compressor of a certain size, is fixed and analytically determined for each input combination, different configurations delivering the same resulting sum, may be proposed. Although indistinguishable in terms of outcome, these circuits have different output combinations and as such, correspond to different circuits, with different electrical performances. Therefore, in order to determine the circuit with the best electrical performances, among the SWACs with optimal error metrics, all circuits need to be synthesized and compared.

The syntheses have been conducted using the ABC synthesis tool, and the freely available standard cell library “mcnc.genlib”. The use of a generic library is intended to provide results that are weakly technology dependent. The correlation between the targeted 14nm FinFET technology, and the ABC synthesis tool, has been determined by synthesizing a good number of compressors in each software and comparing the silicon area values. As it can be seen in Fig. 3.5, the two

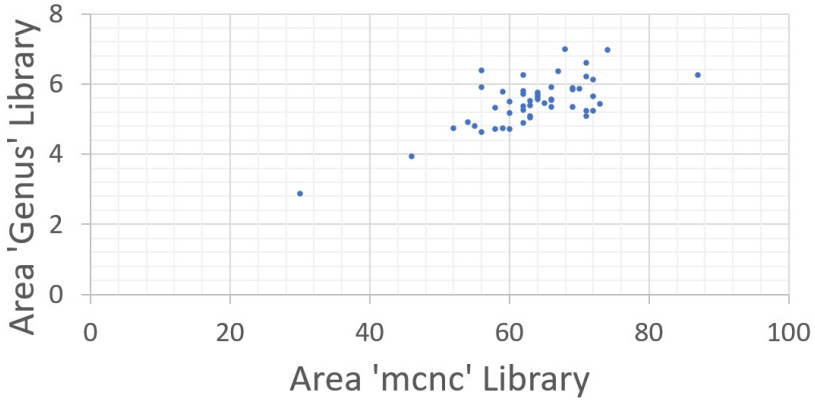

Fig. 3.5: Resulting area correlation of two different synthesizers.

technologies sufficiently follow the same trend, with a degree equal to 70%. Nevertheless, it is not impossible, that two circuits with similar electrical performances, exhibit opposite behaviors when synthesized in different technologies.

3.2.1 Approximate Compressor 3:2

The output information of the Single-Weight Approximate Compressor is encoded into the sum of its output bits. Therefore, according to the cumulative property, changing the order of the output bits while trying to determine the optimal truth table (TT), does not affect the compressor's result. A single "X" is used in table 3.6, to indicate that only one of the two output bits, is high. Notice that as previously mentioned, when possible, the output is correct.

Given the fact that the TT contains three cases that can be expressed in two different ways, i.e., three "X" that can either be "01" or "10", a total of 2^3=8 circuits can be proposed. All these configurations have been synthesized and tested. The best circuit is found to be the same as the one shown in Fig. 3.1.

Table 3.6: Input partial products, compressor outputs, exact and approximated sum, Error and Error probability for the Type-B 3:2 approximate compressor.

$p_0p_1p_2$	General w_0w_1	Proposed w_0w_1	S	S_{App}	E	P_E
000	00	00	0	0	0	-
001	**X**	**01**	1	1	0	-
010	**X**	**01**	1	1	0	-
011	11	11	2	2	0	-
100	**X**	**10**	1	1	0	-
101	11	11	2	2	0	-
110	11	11	2	2	0	-
111	11	11	3	**2**	**1**	**1/64**

3.2.2 Approximate Compressor 4:2

In the 4:2 approximate compressor, there are four cases in which the output needs to be defined as "01" or "10". So, 2^4=16, different circuits can be proposed. Again, the sixteen circuits have been synthesized using both the ABC software and the genus 14nm library,

Table 3.7: Input partial products, compressor outputs, exact and approximated sum, Error and Error probability for the Type-B 4:2 approximate compressor.

$p_0p_1p_2p_3$	General w_0w_1	Proposed w_0w_1	S	S_{App}	E	P_E
0000	00	00	0	0	0	-
0001	**X**	**10**	1	1	0	-
0010	**X**	**10**	1	1	0	-
0011	11	11	2	2	0	-
0100	**X**	**01**	1	1	0	-
0101	11	11	2	2	0	-
0110	11	11	2	2	0	-
0111	11	11	3	**2**	**1**	**3/256**
1000	**X**	**01**	1	1	0	-
1001	11	11	2	2	0	-
1010	11	11	2	2	0	-
1011	11	11	3	**2**	**1**	**3/256**
1100	11	11	2	2	0	-
1101	11	11	3	**2**	**1**	**3/256**
1110	11	11	3	**2**	**1**	**3/256**
1111	11	11	4	**2**	**2**	**1/256**

in order to derive the optimal circuit. The TT of the chosen compressor can be seen in Table 3.7. The resulting TT and circuit are the same as the ones shown in Table 3.3 and Fig. 3.2 respectively.

3.2.3 Approximate Compressor 5:3

The optimal 5:3 approximate compressor has fifteen cases that are not strictly defined in a unique a way. In particular, there are five cases in which the output can be defined in three different ways, “001”, “010”, and “100”, and ten cases, that can also be defined in three possible ways, “011”, “101”, and “110”. A single “X” is used in table 3.8, to indicate that only one of the three output bits is high, while “XX”, is used to indicate that two output bits are high.

Since there are fifteen possible outputs that can be expressed in three different ways, a total of $3^{15} \cong 14 \times 10^6$ different compressors can be proposed. Considering the immense number of possibilities, a random approach has been pursued. A subset of 20.000 random TTs has been synthesized and evaluated. While this subset is quite small with respect to the whole population, it is considered enough to provide a near-optimal, approximate compressor.

Some of the circuits found in the total population share symmetric TTs. Symmetric TTs are considered those that result in their counterpart, just by swapping the input or output bit names. Unfortunately, many electronic computer-aided design (ECAD) tools, including the ABC software, are unstable, thus producing different

Table 3.8: Input partial products, compressor outputs, exact and approximated sum, Error and Error probability for the Type-B 5:3 approximate compressor.

$p_0p_1p_2p_3p_4$	General $w_0w_1w_2$	Proposed $w_0w_1w_2$	S	S_{App}	E	P_E
00000	000	000	0	0	0	-
00001	**X**	**100**	1	1	0	-
00010	**X**	**001**	1	1	0	-
00011	**XX**	**101**	2	2	0	-
00100	**X**	**001**	1	1	0	-
00101	**XX**	**101**	2	2	0	-
00110	**XX**	**011**	2	2	0	-
00111	111	111	3	3	0	-
01000	**X**	**100**	1	1	0	-
01001	**XX**	**110**	2	2	0	-
01010	**XX**	**101**	2	2	0	-
01011	111	111	3	3	0	-
01100	**XX**	**101**	2	2	0	-
01101	111	111	3	3	0	-
01110	111	111	3	3	0	-
01111	111	111	4	**3**	**1**	**3/1024**
10000	**X**	**001**	1	1	0	-
10001	**XX**	**101**	2	2	0	-
10010	**XX**	**011**	2	2	0	-
10011	111	111	3	3	0	-
10100	**XX**	**011**	2	2	0	-
10101	111	111	3	3	0	-
10110	111	111	3	3	0	-
10111	111	111	4	**3**	**1**	**3/1024**
11000	**XX**	**101**	2	2	0	-
11001	111	111	3	3	0	-
11010	111	111	3	3	0	-
11011	111	111	4	**3**	**1**	**3/1024**
11100	111	111	3	3	0	-
11101	111	111	4	**3**	**1**	**3/1024**
11110	111	111	4	**3**	**1**	**3/1024**
11111	111	111	5	**3**	**2**	**1/1024**

circuits when only swapping the inputs of the adder (compressor). However, even though circuits with symmetric TTs differ a bit, their electrical performances tend to fall near. For that reason, the assessed subset does not contain symmetric TTs. After this analysis, the best performing circuit is shown in Fig. 3.6.

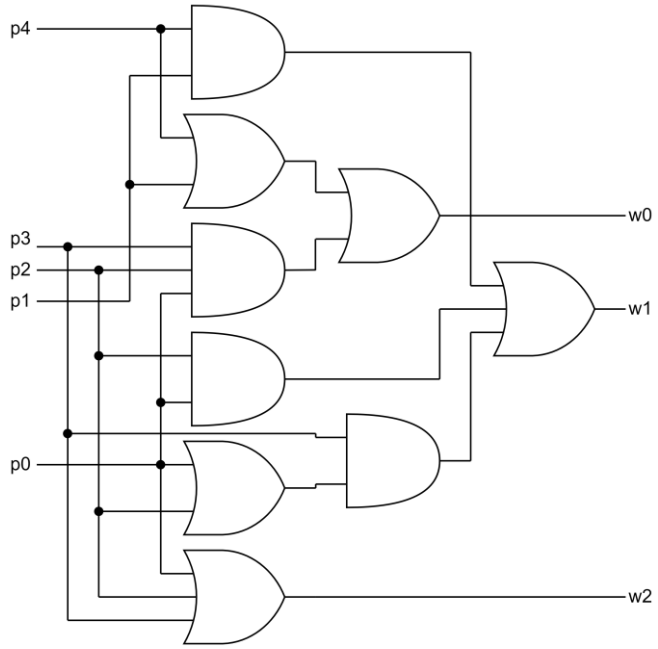

Fig. 3.6: Schematic of the proposed Type-B 5:3 approximate compressor

3.2.4 Approximate Compressor 6:3

In Table 3.9, all erroneous cases and non-uniquely defined outputs are reported. There are twenty-two cases in which S>3, thus resulting in an erroneous approximated result. The error probability of the proposed, optimal approximate compressor is 154/4096, while the error probability of the previous 6:3 compressor, mentioned in section 3.1.4, is equal to 316/4096.

Furthermore, there are six cases, in which the output can be defined as “001”, “010”, or “100”, and fifteen cases, that it can be defined as, “011”, “101”, and “110”. So, there are $3^{21} \cong 10^{10}$ different

configurations. After taking into account symmetries and synthesizing millions of circuits, a high performing SWAC was determined. The preferred configuration is shown in Fig. 3.7.

Table 3.9: Input partial products, compressor outputs, exact and approximated sum, Error and Error probability for the Type-B 5:3 approximate compressor.

$p_0p_1p_2p_3p_4p_5$	General $w_0w_1w_2$	Proposed $w_0w_1w_2$	S	S_{App}	E	P_E
000001	**X**	**001**	1	1	0	-
000010	**X**	**001**	1	1	0	-
000011	**XX**	**101**	2	2	0	-
000100	**X**	**001**	1	1	0	-
000101	**XX**	**101**	2	2	0	-
000110	**XX**	**011**	2	2	0	-
001000	**X**	**001**	1	1	0	-
001001	**XX**	**011**	2	2	0	-
001010	**XX**	**101**	2	2	0	-
001100	**XX**	**101**	2	2	0	-
001111	111	111	4	**3**	**1**	**9/4096**
010000	**X**	**001**	1	1	0	-
010001	**XX**	**101**	2	2	0	-
010010	**XX**	**101**	2	2	0	-
010100	**XX**	**101**	2	2	0	-
010111	111	111	4	**3**	**1**	**9/4096**
011000	**XX**	**101**	2	2	0	-
011011	111	111	4	**3**	**1**	**9/4096**
011101	111	111	4	**3**	**1**	**9/4096**
011110	111	111	4	**3**	**1**	**9/4096**
011111	111	111	5	**3**	**2**	**3/4096**
100000	**X**	**001**	1	1	0	-
100001	**XX**	**101**	2	2	0	-
100010	**XX**	**101**	2	2	0	-
100100	**XX**	**101**	2	2	0	-
100111	111	111	4	**3**	**1**	**9/4096**
101000	**XX**	**101**	2	2	0	-
101011	111	111	4	**3**	**1**	**9/4096**
101101	111	111	4	**3**	**1**	**9/4096**
101110	111	111	4	**3**	**1**	**9/4096**
101111	111	111	5	**3**	**2**	**3/4096**
110000	**XX**	**110**	2	2	0	-
110011	111	111	4	**3**	**1**	**9/4096**
110101	111	111	4	**3**	**1**	**9/4096**
110110	111	111	4	**3**	**1**	**9/4096**
110111	111	111	5	**3**	**2**	**3/4096**
111001	111	111	4	**3**	**1**	**9/4096**
111010	111	111	4	**3**	**1**	**9/4096**
111011	111	111	5	**3**	**2**	**3/4096**
111100	111	111	4	**3**	**1**	**9/4096**
111101	111	111	5	**3**	**2**	**3/4096**
111110	111	111	5	**3**	**2**	**3/4096**
111111	111	111	6	**3**	**3**	**1/4096**

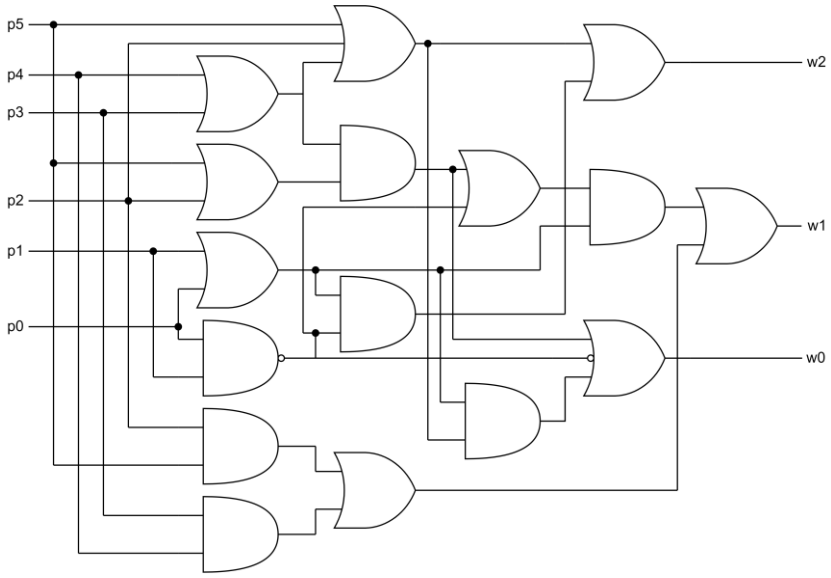

Fig. 3.7: Schematic of the proposed Type-B 6:3 approximate compressor

Table 3.10: Basic components used to create compressors of size 7 to 20.

Desired Compressor	**Building Blocks**				
7:4	4:2	3:2			
8:4	4:2	4:2			
9:5	4:2	5:3			
10:5	4:2	6:3			
11:6	4:2	4:2	3:2		
12:6	4:2	4:2	4:2		
13:7	4:2	4:2	5:3		
14:7	4:2	4:2	6:3		
15:8	4:2	4:2	4:2	3:2	
16:8	4:2	4:2	4:2	4:2	
17:9	4:2	4:2	4:2	5:3	
18:9	4:2	4:2	4:2	6:3	
19:10	4:2	4:2	4:2	4:2	3:2
20:10	4:2	4:2	4:2	4:2	4:2

3.3 Higher Order Approximate Compressors

In order to construct large approximate multipliers, higher order compressors than the ones already proposed, are required. To that end, the proposed compressors are used as building blocks for larger compressors, as summarized in Table 3.10. As it can be observed, the use of 4:2 compressors is generally preferred, since they provide a good trade-off between error and electrical performance. Moreover, it should be noted that, the building blocks might refer to Type-A or Type-B compressors, according to their position in the partial product matrix, as explained in the following section.

3.4 Compressor Allocation Strategy

Let's assume an $n \times n$ multiplier, n being an even number greater or equal to 8. The goal is to utilize Full-Adders, Half-Adders, approximate compressors, or leave PPs unaffected, in order to reduce the PPM, into half its original size, that is: $m = n/2$. As a general, yet not exact rule, approximate compressors are placed only in the least significant half of the PPM. An example of a 12×12 multiplier, detailing the positions of the exact and approximate compressors, is shown in Fig. 3.8, while the provided pseudo-code explains all the steps.

The height of the column i, $h(i)$, is defined as the number of partial products in said column. The height of the same column, in the reduction stage, without accounting for any carries from adders in adjacent columns, is defined as $new\_h(i)$. Starting from the most significant partial product and moving rightwards, the number of Full-Adders and Half-Adders is computed. Finally, information about the remaining PPs is determined, and $new\_h(i)$ is calculated to provide useful information for the next column.

The first m-1 columns are left unaffected. In column m-1, $new\_h(m\text{-}1) = m\text{-}1$ bits. Since the size of the PPM is reduced to a maximum of m, there is space for one more bit, that may be the carry from a Full-Adder placed in the next column. The next column has $h(m) = m$, and three of these bits are fed into the previously mentioned Full-Adder, while the rest are left unaffected. So, in the reduced stage, the

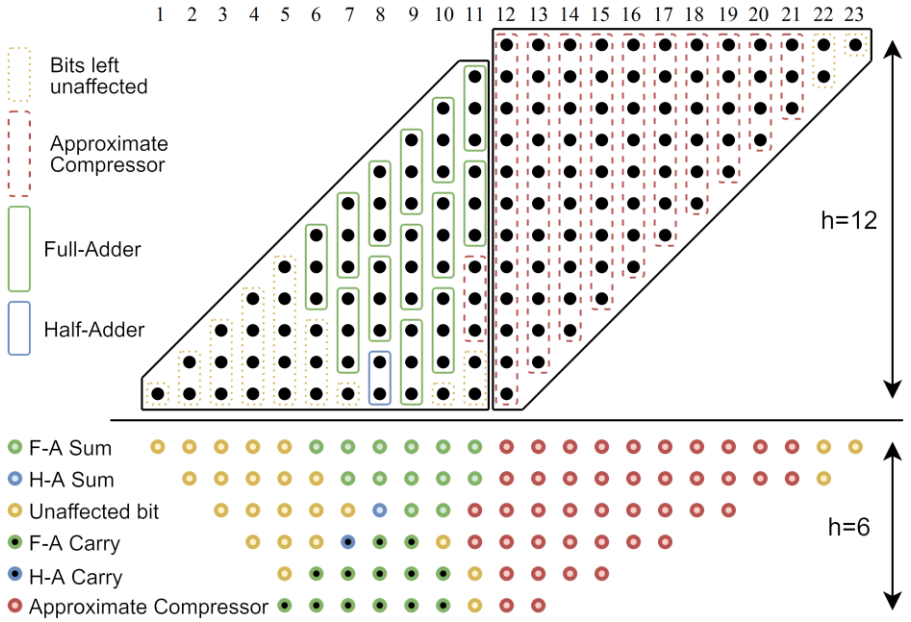

Fig. 3.8: 12x12 Approximate multiplier. Full-Adders, Half-Adders, and Approximate compressors allocation strategy.

same column has *new_h*(*m*) = *m*-2 bits. The remaining 2 bits, needed to fill the reduced matrix, are the carries of 2 Full-Adders, or 1 Full and 1 Half Adders, depending on the size of the next column.

If after placing all the available Full-Adders, PPs still remain, they may be moved unaffected to the reduced stage, provided their number does not exceed the available spots. If the remaining PPs are so many than the reduced stage ends up higher than expected, approximate compressors are employed. The compressors opted in this case, are the error optimal type-B compressors, so that minimum error is introduced to the Most Significant part of the matrix. Moreover, the size of the chosen compressor in this case, *w*:*v*, is as low as possible, allowing as many as possible, *z* bits, to pass unaffected to the reduced stage. Assuming that *k* bits need to be compressed into *l* spots, with $l \leq m$, using the *w*:*v* SWAC, the following two equations with two unknown variables need to be solved to determine the desired configuration:

$$k = z + w \tag{3.22}$$

$$l = z + v = z + \lceil w/2 \rceil \tag{3.23}$$

An example is shown in the 11th column of the PPM of the multiplier shown in Fig. 3.8. This column has *h*(11) = 11 bits and the desired

	Allocation of Exact and Approximate Compressors
1.	**for** i = 1:2n-1 /* left to right */
2.	**if** ($i < n/2 \,\|\|\, i > 2n-3$)
3.	$FA(i) = 0$;
4.	$HA(i) = 0$;
5.	Remaining PPs = unaffected;
6.	$new\_h(i) = h(i)$;
7.	**else if** ($i >= n/2$ && $i < n$)
8.	**if** ($h(i) >= 3*(m-new\_h(i-1))$)
9.	$FA(i) = m-new\_h(i-1)$;
10.	$HA(i) = 0$;
11.	**if** (Remaining PPs <= $m-FA(i)$)
12.	Remaining PPs = unaffected;
13.	$new\_h(i) = h(i)-2*FA(i)$;
14.	**else**
15.	Remaining PPs = unaffected / B-Compressors;
16.	$new\_h(i) = m$;
17.	**end if;**
18.	**else**
19.	$FA(i) = m-new\_h(i-1)-1$;
20.	**if** ($h(i) - 3*FA(i) == 2$)
21.	$HA(i) = 1$;
22.	Remaining PPs = none;
23.	$new\_h(i) = h(i) - 2*FA(i) - 1*HA(i)$;
24.	**else if** ($h(i) - 3*FA(i) == 1$)
25.	$HA(i) = 0$;
26.	Remaining PPs = unaffected;
27.	$new\_h(i) = h(i) - 2*FA(i)$;
28.	**end if;**
29.	**end if;**
30.	**else if** ($i == n$)
31.	$FA(i) = 0$;
32.	$HA(i) = 0$;
33.	Remaining PPs = B-Compressors;
34.	$new\_h(i) = \lceil h(i)/2 \rceil$;
35.	**else if** ($i > n$ && $i <= 2n-3$)
36.	$FA(i) = 0$;
37.	$HA(i) = 0$;
28.	Remaining PPs = A-Compressors;
29.	$new\_h(i) = \lceil h(i)/2 \rceil$;
30.	**end if;**
31.	**end for;**

new_h(11) = $\lceil h(11)/2 \rceil = 6$. The two Full Adders that are used in this column take a total of 6 input bits and generate 2 output bits in the same column. After that, the remaining bits $k = 11 - 6 = 5$ need to be compressed in the remaining spots of the reduced stage, $l = 6 - 2 = 4$. By solving equations 3.22 and 3.23 for $k = 5$ and $l = 4$, we obtain: $w - v = 1$. Taking into account that $v = \lceil w/2 \rceil$, it is derived that a 3:2 SWAC must be used, while $z = 2$ bits will pass unaffected to the reduced stage.

Approximate compressors are employed in the Least Significant part of the matrix, with a size equal to the height of each column, except for the last two columns that remain uncompressed. Optimal error, type-B compressors are chosen for the most significant column, and simpler, type-A designs are placed in the rest of the columns.

3.5 Implementation Results

The multipliers developed in [Esp18] are used as a point of reference, as they showed great results when compared to the state-of-the-art designs. The following comparisons target circuits that employ just one reduction stage and no truncation techniques, namely [Esp18], a close competitor [Qiq17], and the proposed approximate multipliers. The first step towards improving the results presented in [Esp18], was replacing the old compressors with the new ones, presented in section 3.2. This effort resulted in an increase of the number of effective bits, as well as power dissipation. This was expected, since the novel designs are more accurate, at the expense of more hardware resources. Thus, an attempt to approach the "sweet spot" was followed, by placing the new, error-optimal compressors in the more significant PP columns, and the older, less power-hungry designs, in the rest of the columns, as explained in section 3.4.

3.5.1 Electrical Performance

The compressors were used to generate 8×8, 12×12, 16×16, and 20×20 multipliers. All circuits were synthesized targeting a 14nm FinFET technology, using Cadence Genus. Power dissipation is computed by simulating the final netlist to obtain the switching activity

of each node. Silicon area and power dissipation are calculated for these timing constraints, that barely allow the corresponding exact multiplier to have a non-negative slack time. The minimum delay of each circuit is also reported in Table 3.11. Positive percentages indicate improvement with respect to the corresponding one from [Esp18]. From the electrical performance standpoint, most circuits tend to outperform the ones proposed in [Esp18].

Table 3.11: Electrical performance of Exact and Approximate multipliers. All percentages are calculated with respect to [Esp18].

		Exact	[Esp18]	Proposed	[Qiq17] L=2	[Qiq17] L=3	[Qiq17] L=4
8×8	**Min delay (ps)**	220	202	204 -1%	176 13%	153 24%	107 47%
	Area (μm²) delay=220ps	213	125	115 8%	84 33%	48 62%	36 71%
	Power (μW/MHz) delay=220ps	2.79	1.36	1.22 10%	0.77 43%	0.40 71%	0.25 82%
12×12	**Min delay (ps)**	286	261	266 -2%	228 13%	204 22%	178 32%
	Area (μm²) delay=286ps	510	232	232 0%	189 19%	125 46%	96 59%
	Power (μW/MHz) delay=286ps	8.80	1.95	1.89 3%	1.42 27%	0.89 54%	0.56 71%
16×16	**Min delay (ps)**	335	304	304 0%	275 10%	242 20%	217 29%
	Area (μm²) delay=335ps	850	496	485 2%	362 27%	261 47%	187 62%
	Power (μW/MHz) delay=335ps	12.60	3.79	3.79 0%	2.49 34%	1.62 57%	1.00 74%
20×20	**Min delay (ps)**	371	341	339 0.6%	312 9%	280 18%	246 28%
	Area (μm²) delay=371ps	1292	740	753 -2%	560 24%	411 44%	306 59%
	Power (μW/MHz) delay=371ps	18.66	5.89	5.73 3%	3.90 34%	2.63 55%	1.56 74%

Table 3.12: Error performance of Approximate multipliers. All percentages are calculated with respect to [Esp18].

		[Esp18]	Proposed	[Qiq17] L=2	[Qiq17] L=3	[Qiq17] L=4
8×8	**ME**	22.31	22.31 0%	229.38 928%	654.938 2836%	2128 9438%
	NoEB	9.93	10.33 4%	7.03 -29%	5.70 -43%	4.06 -59%
	ER	19.19	16.49 -14%	49.11 156%	65.73 243%	77.57 304%
12×12	**ME**	645	645 0%	15957 2374%	118903 18335%	365819 56616%
	NoEB	13.34	13.87 4%	8.98 -33%	6.23 -53%	4.74 -64%
	ER	34.78	34.78 0%	70.68 103%	87.65 152%	93.22 168%
16×16	**ME**	25429	22837 -10%	1034405 3968%	7640754 29947%	48234449 1895828%
	NoEB	16.31	16.98 4%	10.98 -33%	8.24 -49%	5.72 -65%
	ER	54.19	54.19 0%	84.64 56%	94.82 75%	97.92 81%
20×20	**ME**	866133	824661 -5%	67275029 264460%	667654387 2625463%	1321273162 5195830%
	NoEB	19.31	19.94 3%	12.96 -21%	9.90 -39%	9.25 -43%
	ER	66.29	65.70 -1%	89.08 34%	96.06 45%	98.34 48%

3.5.2 Error Performance

The error performance of the investigated multipliers is shown in Table 3.12. All error metrics, for the 8×8 and 12×12 multipliers are calculated following an exhaustive approach. It should be noted that the mean error and the number of effective bits, for the multipliers proposed in this work or in [Esp18], can be calculated also analytically. Knowing the error and error probability of each erroneous case, of each compressor, placed in each column of the PPM, it is possible to derive both the ME and the NoEB, since each erroneous case contributes individually to the overall error performance of the multiplier.

On the other hand, exploiting each compressor's probability to produce an error in order to calculate the overall Error Rate of the multiplier, is not possible as more than one compressor might contribute to an erroneous case at the same time. Since calculating the ER analytically results in an over-complicated statistical problem, a different approach is needed. For the 16×16 and 20×20 multipliers, the exhaustive approach is not feasible either, due to the immense number of inputs, 2^{32} and 2^{40} accordingly. Thus, an approximate approach was used to calculate the ER of all the 16×16 and 20×20 multipliers, as well as the ME and NoEB of the multipliers proposed in [Qiq17].

As it can be observed in Table 3.12, the proposed designs outperform the reference multipliers in all the error metrics. The circuits introduced in [Qiq17], exhibit a far worse behavior than either design. In Fig. 3.9, the electrical-error performance trade-off is summarized.

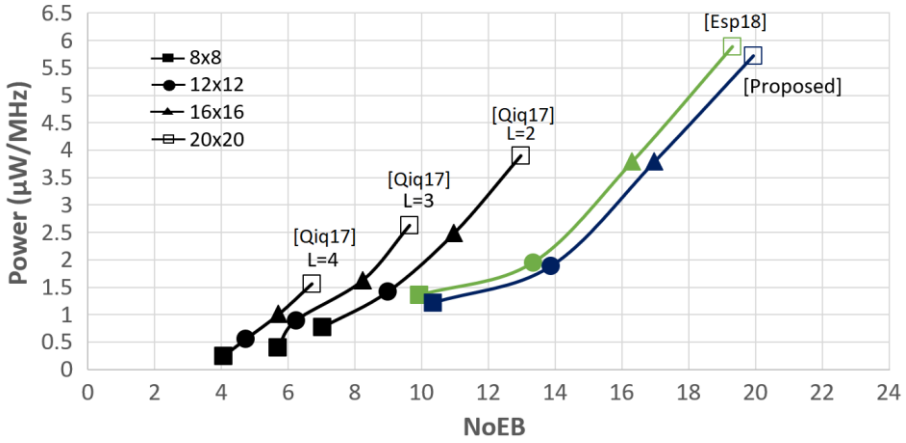

Fig. 3.9: Approximate multipliers compared. Power vs number of effective bits.

3.5.3 Image Smoothing Application

Image smoothing (blurring) is a processing technique, used to reduce the noise and create a less pixelated image. This is typically achieved by substituting each pixel with the weighted average value of all the neighboring pixels. The weight with which each pixel contributes, may be given by a gaussian kernel that moves through the image. Smoothing is an error resilient application, as the human eye is

Table 3.13: Original (Left) and Reformed (Right) Gaussian Kernel

0.095	0.118	0.095
0.118	0.148	0.118
0.095	0.118	0.095

97	121	97
121	151	121
97	121	97

not able to detect trivial details, and it also requires numerous multiplications. Therefore, it is considered an appropriate application to test approximate multipliers.

The two-dimensional gaussian kernel used for image smoothing in this work is a rotationally symmetric, 3×3, lowpass filter, with standard deviation equal to 1.5. The floating-point values of the kernel are multiplied by the constant 2^{n+2} and then rounded to integer values, thus providing the $n \times n$ multipliers, with appropriate input values. The original and adjusted kernels are presented in Table 3.13.

The smoothed "Lena" test images, acquired exploiting each design, are shown in Fig. 3.10. Comparing these images with an image

Table 3.14: SSIM and PSNR values for gaussian smoothing with approximate multipliers compared to exact multipliers. All percentages are calculated with respect to [Esp18].

		[Esp18]	**Proposed**	**[Qiq17] L=2**	**[Qiq17] L=3**	**[Qiq17] L=4**
8×8	**SSIM (%)**	97.94	97.94 0%	97.62 -0.3%	95.04 -3%	89.86 -8.2%
	PSNR	42.91	42.91 0%	39.30 -8.4%	31.11 -27.5%	23.15 -46%
12×12	**SSIM (%)**	97.95	97.95 0%	97.62 -0.3%	93.78 -4.3%	89.86 -8.3%
	PSNR	42.95	42.95 0%	39.29 -8.5%	29.29 -31.8%	23.14 -46.1%
16×16	**SSIM (%)**	97.89	97.92 0%	97.62 -0.3%	93.78 -4.2%	89.87 -8.2%
	PSNR	42.41	42.83 1%	39.29 -7.4%	29.29 -30.9%	23.14 -45.4%
20×20	**SSIM (%)**	97.91	97.93 0%	97.62 -0.3%	93.78 -4.2%	89.87 -8.2%
	PSNR	42.82	42.88 0.1%	39.29 -8.2%	29.29 -31.6%	23.14 -46%

processed by exact multipliers, we obtain the structural similarity index (SSIM) and the peak signal to noise ratio (PSNR), which provide a numerical indication of the performance of the approximate multipliers. The obtained values are reported in Table 3.14. It can be observed that the proposed multipliers, yield significantly better SSIM and PSNR results compared to the multipliers proposed in [Qiq17], and sometimes, slightly better results than the designs proposed in [Esp18].

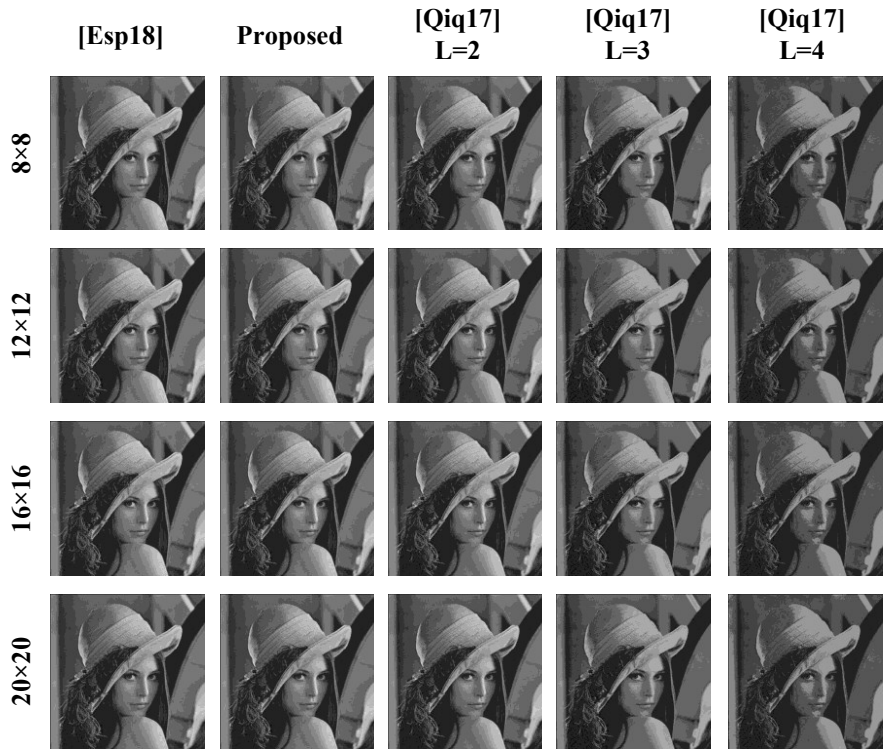

Fig. 3.10: Image Smoothing with Approximate multipliers.

3.6 Summary

In this chapter, the approximate compressors developed in [Esp18] have been presented. The simple, yet effective recoding shown in equation 3.1 is exploited, to generate four Type-A compressors, a

3:2, a 4:2, a 5:3, and a 6:3. These are very simple, XOR-less designs with great error-hardware performance trade off.

The novel type-B approximate compressors of the same sizes have also been presented. They can reduce the size of the partial product matrix columns, while providing error-optimal results. When the high input bits to be compressed are fewer or equal than the number of the compressor outputs, the Type-B approximate compressors guarantee a correct result. Otherwise, the error is as low as possible. Of course, Type-B compressors tend to have greater hardware requirements with respect to Type-A compressors.

The desired error-optimal truth tables resulted in various approximate Type-B compressors. When possible, namely in the cases of the 3:2 and the 4:2 compressors, all possible alternatives have been checked to derive the best performing circuits. For the higher order compressors (the 5:3 and the 6:3), due to the great number of possibilities, a random subset has been selected after discarding circuits with symmetric TTs, to study the electrical performance of the circuits, and derive the best one.

Similarly, the error metrics are calculated analytically using MATLAB when possible, considering each compressor's position, their errors and error probabilities. Otherwise, random methods are used.

The novel Type-B compressors, along with the non-error-optimal Type-A compressors, are used to form approximate multipliers of different sizes, exploiting the allocation strategy presented in section 3.4. The multipliers exhibit competitive behavior when synthesized in a 14 nm FinFET technology and when tested in an error resilient application.

Chapter 4

Approximate Recursive Multipliers

Recursive multipliers use small elementary approximate multiplier blocks, suitably assembled to design larger multipliers, with a typical block size of 2×2, as in [War21] or 4×4, as in [Ans18].

In this work, approximate recursive multipliers based on novel 4×4 multiplier blocks, are investigated. Five approximate 4×4 multipliers, with different error vs hardware trade-off, that are obtained by carry truncation and error compensation, are proposed. The three "T" Multipliers were originally developed in [Nun22], while the two "N" Multipliers", are a more recent, updated work, presented in [Zac22]. These designs, along with an OR-based and an exact 4×4 multiplier, are used to generate 8×8, 16×16, and 32×32, approximate multipliers, following the strategy presented in section 4.2.

The circuits proposed in this work as well as various previously proposed contributions, have been synthesized using a commercial 14nm FinFET standard cell library. The performance of most designs has also been determined in image filtering applications and in the inference step of a pre-trained convolutional neural network.

4.1 4×4 Multipliers

4.1.1 4×4 Exact Multiplier

Let us consider two 4-bit unsigned numbers $a = \sum_{i=0}^{3} a_i 2^i$ and $b = \sum_{j=0}^{3} b_j 2^j$. As already mentioned, the computation of their product, $y = \sum_{k=0}^{7} y_k 2^k$, consists of three steps. Firstly, the partial product matrix is generated using AND gates between all the input bits. There are various techniques to carry out the second and third steps that reduce and sum the entire PPM to obtain the final product, e.g., employing full adders, half adders or 4:2 compressors in Wallace or Dadda configurations. Fig. 4.1 shows the Wallace reduction tree for an exact

4×4 multiplier. Three half adders (dashed rectangles) and five full adders (rectangles) are employed to reduce the PPM. The sum and carry outputs produced by half and full adders are indicated in the figure as *SN_x*, *CNM_x*, where *N* and *M* indicate the origin and destination column, while *x* indicates the reduction stage. After two stages of reduction we obtain the three least-significant bits of the output Y[2]...Y[0] and two 4 bit values that are summed to obtain the most significant bits of the output, Y[7]...Y[3].

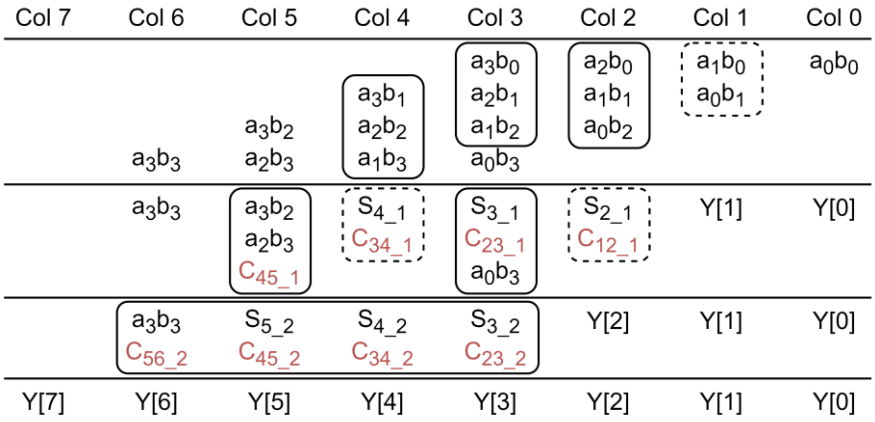

Fig. 4.1: 4x4 Exact Multiplier - **Wallace**.

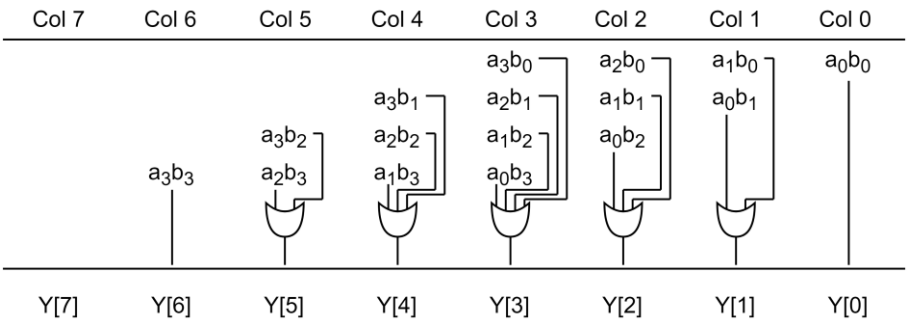

Fig. 4.2: 4x4 Approximate Multiplier - **OR-based**.

4.1.2 4×4 Approximate Multiplier – OR-Based

A simple and fast way to approximate the product of two binary numbers is to use an approximate multiplier with OR compressors. In this case all the partial products in each column of the PPM are fed to OR gates as shown in Fig. 4.2. As it can be observed the most significant bit is always zero. This approximated design is a short of lower bound for circuit complexity, but as shown in section 4.3, it exhibits the worst error performance.

4.1.3 4×4 Approximate Multiplier – T1

As described earlier, the exact multiplier of Fig. 4.1 utilizes a total of eight full / half adders in the two reduction stages, thus generating a total of eight carries in the process. Using the exact circuit as an anchor point, a step-by-step truncation of the carries in the less-significant columns has been performed.

The first step is to truncate the carry C_{12_1}, which is the rightmost carry in the multiplier. Note that by truncating C_{12_1}, also the carry C_{23_2} becomes zero, and can thus be discarded. The following steps are the consecutive truncations of C_{23_1}, C_{34_2}, and C_{34_1}.

Carry truncation results in an underestimation of the result. To balance the mean error out, a compensation method that overestimates

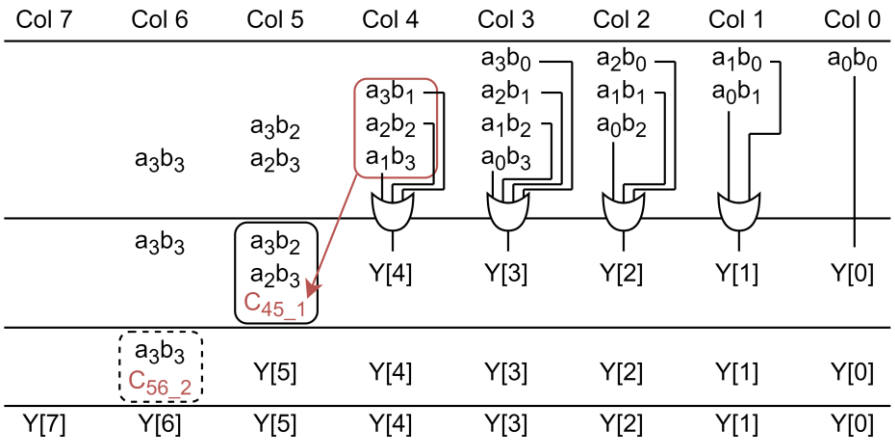

Fig. 4.3: 4x4 Approximate Multiplier – **T1**.

the result, is introduced. To this purpose, the XOR gates that compute the sum of full and half adders are substituted with OR gates, [GSK18], in columns 1, 2, 3, and 4, as shown in Fig. 4.3, thus resulting in a lower absolute mean error, and in a significantly simpler and faster circuit. The red box in Fig. 4.3 shows that even though the sum of the full adder in column 4 is calculated with an OR gate, the carry is still computed as in the exact multiplier with a majority gate and fed to the next column. The resulting circuit, T1 of Fig. 4.3 is composed by four OR gates, a majority gate, a full adder and a half adder.

4.1.4 4×4 Approximate Multiplier – T2

In order to create a faster and less dissipative approximate multiplier than T1, the full adder in column 5 is discarded. In its place, a half adder driven by a_3b_2 and a_2b_3 is introduced, which calculates the sum S_{5_1} and carry, C_{56_1}. The output $Y[5]$ is set to 1 when S_{5_1} or the carry from column 4, C_{45_1}, is high. The outputs $Y[6]$ and $Y[7]$ are calculated with a half-adder driven by a_3b_3 and C_{56_1}. The sum is given by $Y[6] = a_3b_3 \oplus C_{56\_1}$, while the carry is $Y[7] = a_3b_3 \cdot C_{56\_1}$. The expression for $Y[7]$ can be simplified, given that $C_{56\_1} = a_3b_2 \cdot a_2b_3$. One easily obtains:

$$Y[7] = a_3b_3 \cdot a_3b_2 \cdot a_2b_3 = a_3b_2 \cdot a_2b_3 = C_{56\_1} \qquad (4.1)$$

The final circuit is shown in Fig. 4.4 and will be called T2 in the following. It uses five OR gates, a majority gate, one half-adder and one 2-inputs XOR.

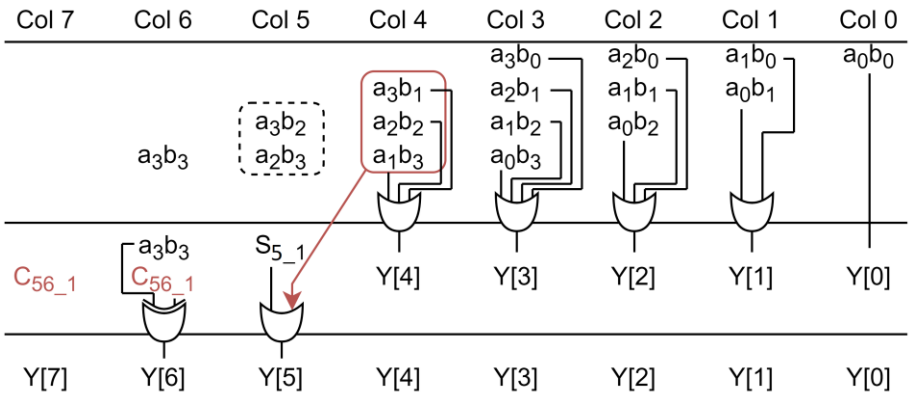

Fig. 4.4: 4x4 Approximate Multiplier – **T2**.

4.1.5 4×4 Approximate Multiplier – T3

An even more aggressive approach consists in discarding the carry from column 4. In this case, *Y*[5] is sum of the half adder in column 5, while *Y*[6] and *Y*[7] are computed as in T2. The resulting circuit is named T3 and shown in Fig. 4.5. This simple approximate multiplier uses only one half-adder, one XOR and four OR gates.

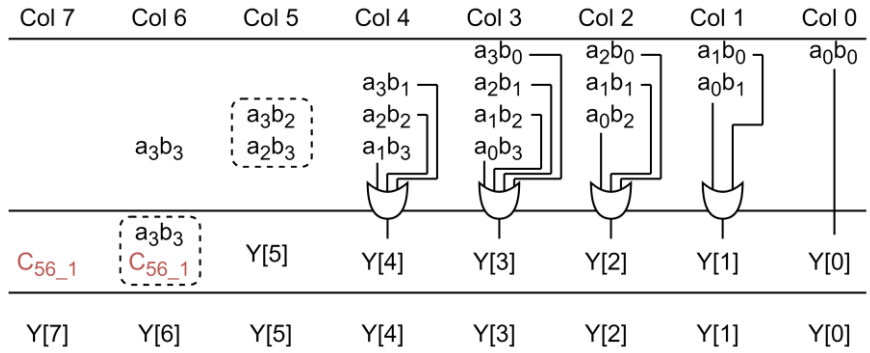

Fig. 4.5: 4x4 Approximate Multiplier – **T3**.

4.1.6 4×4 Approximate Multiplier – N1

In the circuit shown in Fig. 4.2, the sums of the partial products are approximated using OR gates. As a more accurate base point, we can assume an approximate multiplier that uses OR gates to sum the lower half of the matrix of partial products, and full or half adders for the higher part, as shown in Fig. 4.6. Note that the approximated multiplier in Fig. 4.6 requires three compression stages, while the OR based in Fig. 4.2 obtains the result with a fast single stage.

The design in Fig. 4.6 contains three XOR gates that are known to be bulky and slow, and thus, an attempt to simplify it has been made. The first step is to substitute the XOR gate in column 4 with a simpler OR gate:

$$Y[4] = a_3b_1 + a_2b_2 + a_1b_3 \tag{4.2}$$

The next step is the manipulation of the carry of the same Full Adder:

$$C_{45\_1} = a_3b_1 \cdot a_2b_2 + a_1b_3 \cdot a_2b_2 + a_3b_1 \cdot a_1b_3 \tag{4.3}$$

Let us simplify the expression by neglecting the last term:

$$C^*_{45\_1} = a_2b_2 \cdot (a_1b_3 + a_3b_1) \tag{4.4}$$

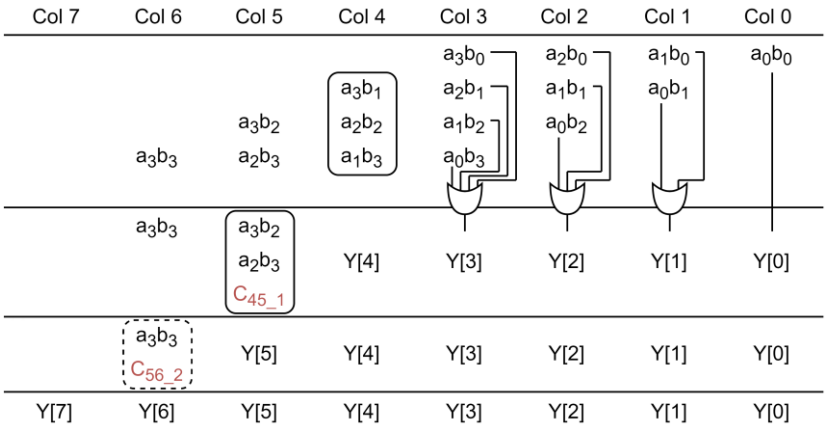

Fig. 4.6: 4x4 Approximate Multiplier – Half OR-based. The starting architecture for the proposed N1 design.

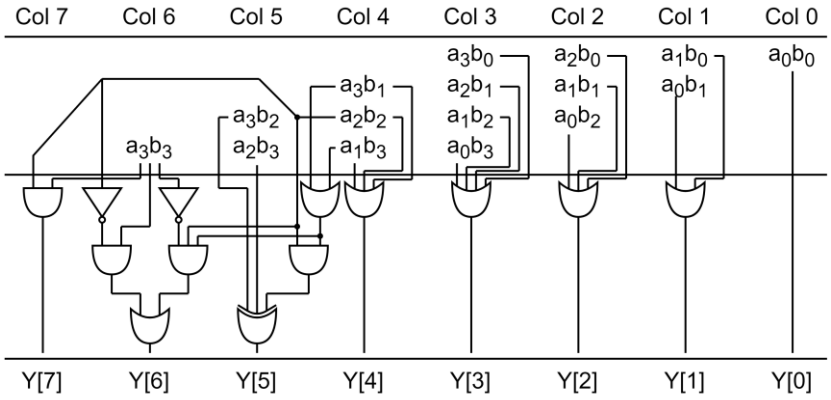

Fig. 4.7: 4x4 Approximate Multiplier – **N1**.

A customized Full Adder is employed in column 5 to add the three terms. The sum is exact and uses a XOR gate:

$$Y[5] = a_3b_2 \oplus a_2b_3 \oplus C^*_{45\_1} \tag{4.5}$$

The carry can be significantly simplified:

$$C_{56\_2} = a_3b_2 \cdot a_2b_3 + a_3b_2 \cdot C^*_{45\_1} + a_2b_3 \cdot C^*_{45\_1} =$$
$$= a_2b_2 \cdot (a_3b_3 + a_3b_3a_1 + a_3b_3b_1 + a_3b_1 + a_1b_3) \tag{4.6}$$

By neglecting the terms that are a product of three literals (they have a lower probability of being '1') we get:

$$C^*_{56\_2} = a_2b_2 \cdot (a_3b_3 + a_3b_1 + a_1b_3) \tag{4.7}$$

The two terms in column 6 are fed into a customized half adder. The sum is the XOR of the two inputs:

$$Y[6] = a_3b_3 \oplus C^*_{56\_2} = a_3b_3 \cdot \overline{C^*_{56\_2}} + \overline{a_3b_3} \cdot C^*_{56\_2} \Rightarrow$$

$$Y[6] \cong a_3b_3 \cdot \overline{a_2b_2} + \overline{a_3b_3} \cdot a_2b_2 \cdot (a_3b_1 + a_1b_3) \quad (4.8)$$

Finally, the carry of the Half Adder is approximated as:

$$Y[7] = C^*_{56\_2} \cdot a_3b_3 \cong a_2b_2 \cdot a_3b_3 \quad (4.9)$$

The resulting design is named N1 and is shown in Fig. 4.7. N1 uses three stages to reach the result and uses six OR gates, four AND gates, and one XOR gate. Compared to the exact Wallace 4×4 multiplier, it shows a vast improvement in terms of both power and speed. In fact, in the exact design the third stage consists of cascaded half and full adders, resulting in three sub-stages, all of them containing at least one XOR gate. Namely, 28 AND gates, 8 OR gates and 12 XOR gates are used in the exact design. Obviously, the proposed design provides an inexact result. The error characteristics of the proposed blocks are discussed in section 4.3.

4.1.7 4×4 Approximate Multiplier – N2

Let us now start from a less accurate circuit, which is T3. In this circuit, shown in Fig. 4.5, all the terms from Y[0] to Y[4] are computed as the output of OR gates, while the remaining bits are computed without approximations. As shown in Fig. 4.5, two half adders are needed together with the OR gates to complete the design of the multiplier. The proposed architecture takes the circuit in Fig. 4.5 as a starting point for further simplification.

The first step is to substitute the XOR gate of the half adder in column 5, with an OR gate:

$$Y[5] = a_3b_2 + a_2b_3 \quad (4.10)$$

The carry of the same half adder is:

$$C_{56\_1} = a_3b_2 \cdot a_2b_3 \quad (4.11)$$

The sum of the last half adder is:

$$S_6 = a_3b_3 \oplus C_{56\_1} = a_3b_3 \cdot \overline{C_{56\_1}} + \overline{a_3b_3} \cdot C_{56\_1} =$$

$$= a_3b_3 \cdot \overline{a_3b_2 \cdot a_2b_3} + \overline{a_3b_3} \cdot a_3b_2 \cdot a_2b_3 \quad (4.12)$$

By neglecting the second term:

$$S^*_6 = a_3b_3 \cdot \overline{a_3b_2a_2b_3} = a_3b_3 \cdot \left(\overline{a_3b_3} + \overline{a_2b_2}\right) \quad (4.13)$$

With a final approximation:

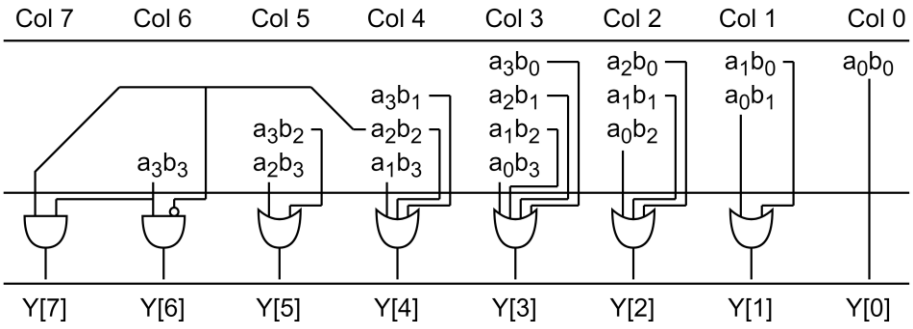

Fig. 4.8: 4x4 Approximate Multiplier – **N2**.

$$Y[6] = a_3 b_3 \cdot \overline{a_2 b_2} \tag{4.14}$$

The carry of the last Half Adder is:

$$Y[7] = C_{56\_1} \cdot a_3 b_3 = a_3 b_2 \cdot a_2 b_3 \cdot a_3 b_3 = \\ = a_3 b_3 \cdot a_2 b_2 \tag{4.15}$$

The resulting design is named N2 and shown in Fig. 4.8. This rather simple design has only two additional AND gates with respect to the OR-based design shown in Fig. 4.2. However, the performances of the proposed design are considerably better as will be discussed in section 4.3, making this design useful for higher order multipliers.

4.2 8×8 Approximate Recursive Architectures

State-of-the-art designs found in the literature have been used for comparison: [Str22], [Yan18], [Fru20], [Ans18], [Kul11], [Reh16], [GSK18], [Gil19], [War20], and [War21]. These are mostly approximate recursive proposals, but contributions from other fields are also considered. In [Str22], [Yan18], [Fru20], and [Gil19] no explicit 4×4 designs are proposed.

As mentioned in section 2.1, scaling up to a $2n \times 2n$ multiplier can be achieved by exploiting four $n \times n$ multipliers. The same technique can be used recursively to design even larger multipliers. For instance, four suitably placed 2×2 multipliers form a 4×4 multiplier, while sixteen 2×2 multipliers can be used to generate an 8×8 design. Note that the building blocks do not need to be the same, and different ones can be used, to obtain different electrical performance-accuracy trade-offs. As a rule of thumb, if uniform distribution is expected for the input operands, exact or high precision modules should occupy the most

significant portion of the design. Moving towards the least significant part, modules that are less accurate, but also less demanding in terms of resources, might be used.

Table 4.1: 8×8 Approximate Multiplier Compositions. Exact Sub-Products Addition.

8×8 Design		$a_H b_H$	$a_L b_H$	$a_H b_L$	$a_L b_L$
Proposed	**T8-1**	T1	T1	T1	T3
	T8-2	T2	T2	T2	T3
	T8-3	Exact	T1	T1	T1
	T8-4	Exact	T2	T2	T2
	T8-5	Exact	Exact	Exact	T1
	T8-6	Exact	Exact	Exact	T2
Proposed	**N8-5**	Exact	Exact	Exact	N1
	N8-6	Exact	Exact	Exact	N2
[Ans18]	**M8-1**	M1	M1	M1	M1
	M8-2	M2	M2	M2	M2
	M8-3	Exact	M1	M1	M1
	M8-4	Exact	M2	M2	M2
	M8-5	Exact	Exact	Exact	M1
	M8-6	Exact	Exact	Exact	M2
[Kul11]	**Kul8**	Kul4	Kul4	Kul4	Kul4
[Reh16]	**Reh8**	Reh4	Reh4	Reh4	Reh4
[War20]	**Ax8_1**	Exact	Exact	Exact	MxA
	Ax8_2	Exact	Exact	LxA	MxA
	Ax8_3	Exact	LxA	LxA	MxA
[War21]	**AxRM1**	Exact	Exact	Exact	mul2b4
	AxRM2	Exact	Exact	mul2b4	mul2b4
	AxRM3	Exact	mul2a4	mul2b4	mul2b4

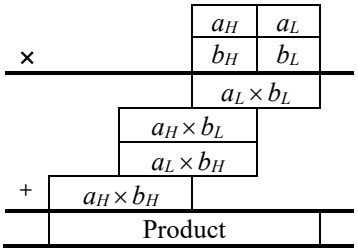

Fig. 4.9: Recursive Multiplier using four building blocks.

Consider two 8-bit unsigned numbers $a = \sum_{i=0}^{7} a_i 2^i$ and $b = \sum_{j=0}^{7} b_j 2^j$. In order to exploit recursive 4×4 multipliers to calculate the product $y = \sum_{k=0}^{15} y_k 2^k$, each number is divided into two 4-bit parts: $a_L = \sum_{i=0}^{3} a_i 2^i$, $a_H = \sum_{i=4}^{7} a_i 2^i$, $b_L = \sum_{i=0}^{3} b_i 2^i$ and $b_H = \sum_{i=4}^{7} b_i 2^i$ and the multiplications $a_L b_L$, $a_H b_L$, $a_L b_H$, and $a_H b_H$ are performed exploiting the corresponding blocks. Finally, the four sub-products need to be added. As shown in Fig. 4.9, the four sub-products are added employing an exact adder.

Table 4.1 shows the circuits considered for comparison that apply this design methodology, the corresponding 4×4 building blocks, and how they are used to build larger multipliers. Note that the 4×4 approximate modules used in [War21], namely mul2a4 and mul2b4, are also recursive multipliers made up by 2×2 blocks.

Table 4.1 also shows the composition of six 8×8 multipliers proposed in this work, namely T8-1 to T8-6, N8-5, and N8-6. They use the proposed T1, T2, T3, N1, and N2 blocks and show competitive results or even overcome the state-of-the-art.

Table 4.2: 8×8 Approximate Multiplier Compositions. Approximate Sub-Products Addition.

8×8 Design		$a_H b_H$	$a_L b_H$	$a_H b_L$	$a_L b_L$
Proposed	**N8-L1**	Exact	N1	N1	OR-based
	N8-L2	Exact	N2	N2	OR-based
[GSK18]	**LOAM**	Exact	guo4	guo4	OR-based

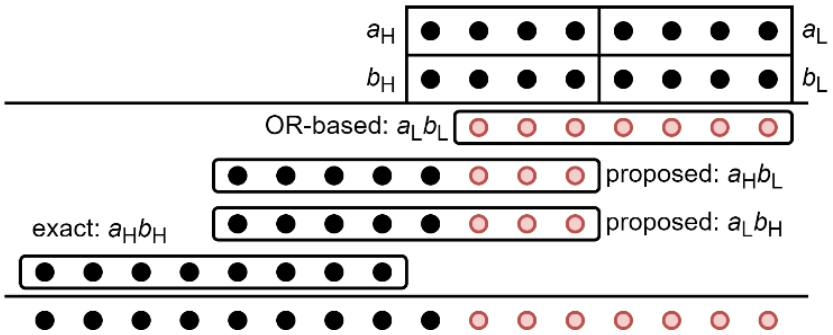

Fig. 4.10: Proposed 8x8 Approximate Multiplier Architecture. Red bits are added with OR gates, black bits with exact adders.

An alternative way to add the sub-products is proposed in [GSK18] and used also in this work. The utilized building blocks and their positions are shown in Table 4.2. Differently from Fig. 4.9, the final product is not the exact addition of the four sub-products, but an approximated version of it. As it can be seen in Fig. 4.10, the seven least significant columns of the sub-products are marked with red color, indicating that they are summed using an approximate adder that uses one OR gate in every column. However, the nine most significant columns are added with an exact adder. Note that the first sub-product has only seven output bits as shown in Fig. 4.2, for the OR-based 4×4 multiplier.

4.3 Performances

The proposed and reference circuits are all synthesized in a 14nm FinFET technology, using Cadence Genus and imposing proper timing constraints. Power dissipation is computed by simulating the final netlist with random inputs, to obtain the switching activity of each node. The input vector array is identical for all designs with the same input bit width. In the following tables "Min delay" refers to the strictest timing constraint, at which each circuit can be synthesized with non-negative slack and provides information regarding the maximum working speed of each design.

Area, power, and delay are compared against the results of the corresponding (4×4, 8×8, 16×16, or 32×32) exact multiplier. The exact design is obtained by describing the circuit in HDL (Verilog) with the multiplication operator and letting the synthesizer choose the near-optimal topology for the given constraint. Therefore, the electrical performances are sometimes slightly worse than those presented in the literature that compare with a fixed exact design.

Error performance is obtained by an exhaustive simulation, for both 4×4 and 8×8 multiplier designs. For 16×16 and 32×32 designs the error performances are computed using a random set of uniformly distributed test vectors. The number of test vectors is 10^5 and 10^6 for 16- and 32-bit multipliers, respectively.

4.3.1 4×4 Approximate Multipliers

The electrical and error performances of the considered 4×4 approximate multipliers are summarized in Table 4.3. To ensure a fair comparison between the circuits, avoid biased optimizations by the synthesizing tool, and emphasize the low power performance of the structures, the circuits have been synthesized with the timing constraint of 250ps to obtain the area and power metrics. The circuits are simulated applying a uniformly distributed random set of $2 \cdot 10^4$ test vectors to gather the switching activity. The total power reported in the table is computed for a clock frequency of 1GHz. It is worth noting that the circuits proposed in this work are for general purpose applications thus a uniform distribution of the input is considered. However, automated designs [Češ18], [Ull18], [Mra20], [Bal22] or dedicated circuits previously presented in the literature, could provide better performances for a specific distribution of the input vectors.

Table 4.3: Performances of 4×4 Approximate Multipliers.

4×4 Design		Area * [μm^2]	Power Reduction [%]	Min Delay* [ps]	Error Rate [%]	NMED ($\times 10^{-2}$)	MRED ($\times 10^{-2}$)	NoEB
	Exact	17.27	-	115	-	-	-	8
	OR-Based	3.90	69.56	19	37.11	3.60	8.78	3.74
Proposed	**T1**	6.41	57.44	45	35.93	1.48	5.09	5.12
	T2	5.58	64.62	30	35.94	2.26	6.94	4.53
	T3	4.72	66.73	24	35.94	2.41	7.51	4.22
Proposed	**N1**	5.42	63.91	42	35.94	1.76	5.57	4.83
	N2	4.54	68.44	19	37.71	2.44	7.24	4.42
[Ans18]	**M1**	8.85	44.44	51	35.94	1.75	6.08	4.88
	M2	6.01	61.90	24	35.94	2.76	7.87	4.20
[Kul11]	**Kul4**	11.87	21.27	92	19.14	1.39	2.97	4.61
[Reh16]	**Reh4**	16.69	12.06	105	46.48	2.08	15.90	4.71
[GSK18]	**guo4**	9.57	29.03	72	28.52	1.89	4.57	4.46
[War20]	**MxA**	6.36	62.09	30	53.91	6.99	22.80	3.16
	LxA	6.61	59.64	35	53.91	9.81	27.20	2.65
[War21]	**mul2a**	9.77	22.81	95	64.45	3.72	29.98	4.15
	mul2b	10.79	27.13	81	75.00	7.46	51.38	3.36

**Area and power are reported for the circuits synthesized with a timing constraint of 250ps. Min Delay is the minimum timing at which the circuit can be synthesized with a non-negative slack.*

As it can be observed in Table 4.3, the proposed circuits are very small and come second only to the OR-based design, with the exception of T1, which is still one of the smallest designs. The same can be stated also for power dissipation, with N2 having an unquestionable advantage. When it comes to speed, N2 is the fastest design, T3 and M2 from [Ans18] hold the second position, and T2 comes third with MxA from [War20]. The proposed multipliers exhibit competitive NMED, MRED and NoEB with respect to the state-of-the-art. The relative reduction in power dissipation with respect to the exact design vs NoEB is shown in Fig. 4.11. The proposed design N2 dissipates 18% less power than the least energy-hungry architectures up to date, M2 and MxA proposed in [Ans18] and [War20] respectively, while still providing a smaller approximation error.

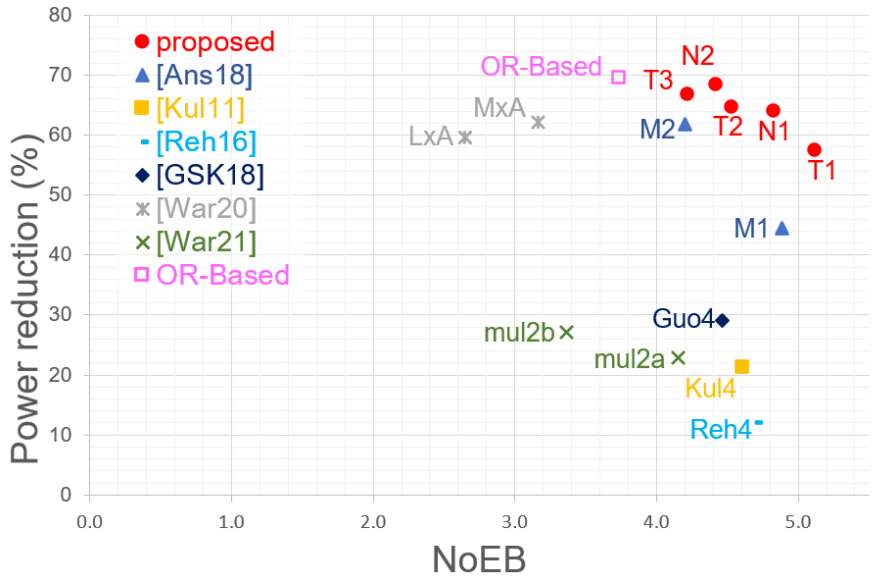

Fig. 4.11: Power reduction of the considered 4×4 Approximate Multipliers with respect to the exact one vs Number of Effective Bits. The proposed circuits have lower power for the same NoEB. The exact design would have NoEB=8 and zero power reduction.

4.3.2 8×8 Approximate Multipliers

The results of the 8×8 approximate multipliers are shown in Table 4.4. Recursive designs are reported in Table 4.4.a, while selected approximate designs following different methodologies, are shown in in Table 4.4.b. Power reduction against number of effective bits for all designs is displayed in Fig. 4.12. Non-filled shapes in the figure correspond to non-recursive designs.

All circuits are synthesized for a 1000ps timing constraint and simulated with same set of $2 \cdot 10^4$ uniformly distributed random vectors. The total power reported in the table is computed for a clock frequency equal to 1GHz.

The designs presented in [Str22] employ a smaller, segmented multiplier. Specifically, instead of an 8-bit multiplier, a 4-bit multiplier with or without error correction respectively, is used. The product is then shifted accordingly. In this simple circuit, hardware resources and power consumption are kept to significantly low levels, while the error metrics are still competitive.

Note that the entries of [Yan18] and [Fru20] exhibit identical electrical performances respectively, since they refer to the same circuits with different settings (both designs allow for configurable accuracy). While the range of chosen accuracy in [Yan18] is limited and the innate flexibility results in increased area requirements, the circuit is very fast, overcoming all the investigated contributions except for T8-2. As it can be observed in Table 4.4, the minimum accuracy of this design, is still greater than that of the design M8-2 proposed in [Ans18], while power reductions are similar.

The circuit presented in [Fru20] offers dynamic truncation at runtime, by enabling or disabling AND gates that form specific partial products. “DT0” refers to the case where all the AND gates are enabled, resulting in an exact multiplier. However, the additional hardware resources result in a greater power consumption with respect to the exact design (hence the negative power reduction). “DT8” refers to the maximum possible truncation where a 43.62% power reduction is achieved. The numbers in the names indicate the level of truncation.

The authors in [Ans18], offer a number of circuits covering a wide range of accuracy. Designs M8-5 and M8-6 are the most precise ones, using one approximate and three exact 4-bit multipliers. While

the synthesized circuits are slightly slower than the exact multiplier, they offer some power reduction at a relatively small expense in accuracy.

Designs Ax8_1 and AxRM1 presented in [War20] and [War21] respectively, employ three exact and one approximate module. While these are the most accurate designs presented in the respective papers, they are still less accurate than M8-6 and M8-5 of [Ans18], and even less accurate than the proposed T8-5, T8-6, N8-5, and N8-6. At the same time, the circuits are quite large, and slower than the exact multiplier. This behavior follows the pattern presented in Fig. 4.11, for the 4×4 building blocks. For the less accurate designs, Ax8_3 with one accurate module, manages to surpass M8-1 that uses no accurate modules, both in accuracy and in power reduction. However, it is slightly larger and slower.

An interesting architecture is proposed in [GSK18]. It uses one exact multiplier, two custom modules, and an OR-based 4×4 approximate multiplier for the least significant part. This relatively small design, in terms of accuracy performs similarly to the proposed design N8-L1, as well as to M8-3 and M8-4. It achieves a significant power reduction with respect to M8-3 and M8-4 but N8-L1 leads. Among circuits with a similar power reduction percentage, M8-2 and Yang_7'b1, it exhibits a far more accurate behavior.

As it can be seen in Table 4.4.a, among the recursive topologies, the proposed circuits T8-2, N8 L1 and N8-L2 occupy the smallest area and achieve the biggest reduction in power consumption. Moreover, they are among the fastest circuits. At the same time, they exhibit competitive behavior in terms of accuracy. As it can be observed in Fig. 4.12, even though there are more precise circuits in the literature, the proposed designs provide a certain level of accuracy at a very low cost.

On the other hand, proposals T8-5, T8-6, N8-5, and N8-6, are very accurate circuits, exploiting three exact, and one proposed 4×4 multipliers. They offer a very high number of effective bits, matched only by the designs, M8-5 and M8-6 [Ans18]. However, exploiting the proposed 4×4 building blocks, T8-5, T8-6, N8-5, and N8-6, achieve a greater power reduction, as it can be observed in Fig. 4.12 and Table 4.4.

Table 4.4.a: Performances of 8×8 Approximate Recursive Multipliers.

8×8 Design		Area* [μm^2]	Power Reduction [%]	Min Delay* [ps]	Error Rate [%]	NMED	MRED	NoEB
	Exact	81.45	-	220	-	-	-	16
Proposed	**T8-1**	47.28	34.06	178	72.59	1.4×10^{-2}	5.2×10^{-2}	5.30
	T8-2	44.31	41.20	**160**	72.60	2.2×10^{-2}	7.3×10^{-2}	4.58
	T8-3	60.59	20.87	210	65.70	1.5×10^{-3}	1.3×10^{-2}	8.79
	T8-4	57.23	26.61	184	65.88	2.4×10^{-3}	1.9×10^{-2}	8.13
	T8-5	71.57	9.02	225	**35.94**	$\mathbf{5.1\times10^{-5}}$	$\mathbf{1.0\times10^{-3}}$	**13.12**
	T8-6	69.97	13.49	225	**35.94**	7.7×10^{-5}	1.5×10^{-3}	12.53
Proposed	**N8-L1**	45.86	44.38	181	72.56	2.6×10^{-3}	2.4×10^{-2}	8.01
	N8-L2	**43.79**	**45.96**	174	72.58	2.9×10^{-3}	2.9×10^{-2}	7.64
	N8-5	70.68	13.14	229	65.63	6.1×10^{-5}	1.2×10^{-3}	12.83
	N8-6	70.02	14.21	222	65.80	8.5×10^{-5}	1.6×10^{-3}	12.42
[Ans18]	**M8-1**	57.43	26.18	195	**35.94**	1.7×10^{-2}	6.1×10^{-2}	5.03
	M8-2	47.56	37.08	165	**35.94**	2.8×10^{-2}	8.4×10^{-2}	4.22
	M8-3	64.46	17.48	205	46.73	1.8×10^{-3}	1.6×10^{-3}	8.51
	M8-4	57.52	26.09	186	81.44	3.1×10^{-3}	2.2×10^{-2}	7.63
	M8-5	72.38	7.10	225	74.76	6.1×10^{-5}	1.3×10^{-3}	12.88
	M8-6	70.94	12.73	226	46.73	9.6×10^{-5}	1.8×10^{-3}	12.20
[Kul11]	**Kul8**	63.29	13.99	204	42.65	1.4×10^{-2}	3.3×10^{-2}	4.69
[Reh16]	**Reh8**	84.27	2.81	240	53.91	2.1×10^{-2}	1.5×10^{-1}	4.76
[GSK18]	**LOAM**	50.78	35.00	209	70.46	2.0×10^{-3}	1.8×10^{-2}	8.21
[Gil19]	**ISH1**	66.22	11.29	203	83.88	2.3×10^{-2}	4.8×10^{-2}	3.80
	ISH2	76.84	4.76	216	75.00	1.2×10^{-2}	2.8×10^{-2}	4.85
[War20]	**Ax8_1**	71.20	12.48	225	89.36	2.4×10^{-4}	4.7×10^{-3}	11.16
	Ax8_2	68.40	19.84	215	96.17	5.5×10^{-3}	3.9×10^{-2}	6.69
	Ax8_3	60.46	27.61	199	97.85	1.1×10^{-2}	7.4×10^{-2}	6.00
[War21]	**AxRM1**	71.03	7.35	229	97.85	2.6×10^{-4}	7.7×10^{-3}	11.36
	AxRM2	65.20	13.67	205	80.02	4.3×10^{-3}	1.5×10^{-1}	7.38
	AxRM3	62.80	15.35	209	36.16	5.2×10^{-3}	2.1×10^{-1}	7.20

**Area and power are reported for the circuits synthesized with a timing constraint of 1000ps. Min Delay is the minimum timing at which the circuit can be synthesized with a non-negative slack.*

Table 4.4.b: Performances of 8×8 Approximate Non-Recursive Multipliers.

8×8 Design		Area* [µm²]	Power Reduction [%]	Min Delay* [ps]	Error Rate [%]	NMED	MRED	NoEB
[Str22]	SSM_m4	28.42	69.16	165	97.85	2.7×10^{-2}	1.5×10^{-1}	4.91
	SSM_m4_u3	36.11	56.96	194	97.85	9.0×10^{-3}	6.5×10^{-2}	6.37
[Yan18]	Yang_7'b0	59.70	36.04	161	80.02	1.6×10^{-2}	9.1×10^{-2}	5.29
	Yang_7'b1	59.70	24.18	161	36.16	2.5×10^{-3}	8.5×10^{-3}	6.93
[Fru20]	DT0	93.47	-5.87	249	0.00	0	0	16
	DT2	93.47	-3.88	249	50.00	1.9×10^{-5}	7.7×10^{-4}	14.45
	DT4	93.47	4.55	249	81.25	1.9×10^{-4}	5.6×10^{-3}	11.93
	DT8	93.47	43.62	249	98.05	6.9×10^{-3}	9.8×10^{-2}	6.99

**Area and power are reported for the circuits synthesized with a timing constraint of 1000ps. Min Delay is the minimum timing at which the circuit can be synthesized with a non-negative slack.*

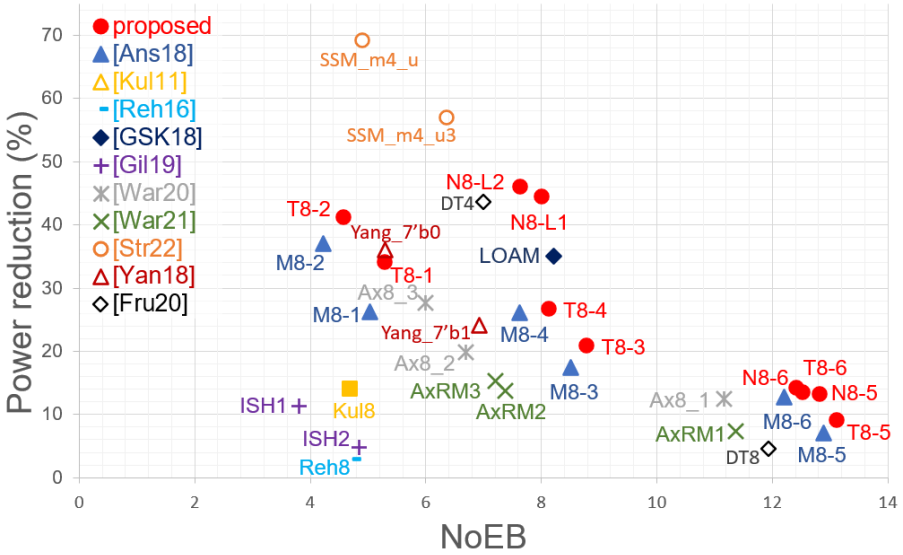

Fig. 4.12: Power reduction of the considered 8×8 Approximate Multipliers with respect to the exact one vs Number of Effective Bits. The proposed circuits have lower power for the same NoEB. The exact design would have NoEB=16 and zero power reduction.

4.3.3 16×16 Approximate Multipliers

The 8×8 designs, and the methodologies described above, can be used to scale up to 16×16 multipliers. As already shown in section 4.2, two different approaches are used to generate selected 16×16 designs. Table 4.5 summarizes the architectures of the considered designs. The circuits following the most straightforward approach (exact sub-product addition) are presented at the top part of table 4.5, while the ones using the technique presented in [GSK18] (approximate sub-product addition), are at the bottom part.

The performances of 16×16 approximate recursive multipliers are shown in Table 4.6. All 16×16 designs have been synthesized under the same timing constraint: 1000ps. Furthermore, they have been simulated with the same set of 10^5 uniformly distributed random vectors, with an input switching frequency equal to 1GHz.

Table 4.5: 16×16 Approximate Multiplier Compositions. Exact Sub-Products Addition.

16×16 Design		$a_H b_H$	$a_L b_H$	$a_H b_L$	$a_L b_L$
Proposed	**N16-5**	N8-5	N8-5	N8-5	N8-5
	N16-6	N8-6	N8-6	N8-6	N8-6
[Ans18]	**M16-1**	M8-1	M8-1	M8-1	M8-1
	M16-2	M8-2	M8-2	M8-2	M8-2
	M16-3	M8-3	M8-3	M8-3	M8-3
	M16-4	M8-4	M8-4	M8-4	M8-4
	M16-5	M8-5	M8-5	M8-5	M8-5
	M16-6	M8-6	M8-6	M8-6	M8-6
[Kul11]	**Kul16**	Kul8	Kul8	Kul8	Kul8
[Reh16]	**Reh16**	Reh8	Reh8	Reh8	Reh8
[War20]	**Ax16_1**	Ax8_1	Ax8_1	Ax8_1	Ax8_1
	Ax16_2	Ax8_2	Ax8_2	Ax8_2	Ax8_2
	Ax16_3	Ax8_3	Ax8_3	Ax8_3	Ax8_3
[War21]	**AxRM16_1**	AxRM1	AxRM1	AxRM1	AxRM1
	AxRM16_2	AxRM2	AxRM2	AxRM2	AxRM2
	AxRM16_3	AxRM3	AxRM3	AxRM3	AxRM3
Proposed	**N16-L1**	Exact	N8-L1	N8-L1	OR-Based
	N16-L2	Exact	N8-L2	N8-L2	OR-Based
[GSK18]	**LOAM16**	Exact	LOAM	LOAM	OR-Based

The architecture proposed in [GSK18], results in designs that significantly outperform other contributions. It should be noted that the most straightforward approach from an algorithmic point of view, followed by the circuits at the top part of Table 4.5, does not result in optimal configurations. In fact, each 16×16 multiplier is composed by four identical approximate 8×8 multipliers, that in turn may be composed by some exact 4×4 multipliers. On the other hand, [GSK18] employs a single exact 8×8 multiplier placed in the most significant part, thus making an important impact on accuracy, despite the approximate final addition.

Moreover, the non-recursive exact and OR-based 8×8 multipliers in the most and least significant parts respectively, as well as the approximation in the final addition, allow this architecture to exploit minimal hardware resources. Therefore, the three designs that follow this approach are the smallest, fastest, and least-power hungry. The circuit proposed in [GSK18], manages to outperform N16-L1 and N16-L2 in accuracy, while N16-L2 slightly overcomes LOAM16 in terms of power reduction. N16-L2 also occupies the smallest area. Among the strictly recursive designs, N16-5 and N16-6 achieve a higher power reduction than circuits with similar or even lower accuracy.

4.3.4 32×32 Approximate Multipliers

The performances of the proposed 32×32 approximate multipliers are shown in Table 4.7. The circuits have been synthesized under a timing constraint of 1000ps and simulated with 10^6 uniformly distributed random vectors, and an input switching frequency equal to 1GHz.

Table 4.6: Performances of 16×16 Approximate Recursive Multipliers.

16×16 Design		Area* [μm²]	Power Reduction [%]	Min Delay* [ps]	Error Rate [%]	NMED	MRED	NoEB
	Exact	356.30	-	335	-	-	-	32
Proposed	**N16-L1**	226.35	66.38	355	99.31	2.7×10^{-5}	7.4×10^{-4}	14.91
	N16-L2	**224.88**	**66.70**	346	99.34	3.3×10^{-5}	8.8×10^{-4}	14.61
	N16-5	377.04	25.49	407	72.57	6.1×10^{-5}	1.1×10^{-3}	12.99
	N16-6	357.09	32.26	405	73.67	8.5×10^{-5}	1.5×10^{-3}	12.54
[Ans18]	**M16-1**	305.02	36.76	359	96.67	1.6×10^{-2}	6.1×10^{-2}	5.03
	M16-2	266.55	43.26	328	96.68	2.8×10^{-2}	8.4×10^{-2}	4.21
	M16-3	359.28	32.26	367	94.61	1.8×10^{-3}	1.5×10^{-2}	8.52
	M16-4	310.16	37.71	346	94.68	3.1×10^{-3}	2.2×10^{-2}	7.63
	M16-5	393.73	22.99	410	**72.55**	6.1×10^{-5}	1.2×10^{-3}	13.05
	M16-6	368.58	26.92	403	72.57	9.6×10^{-5}	1.7×10^{-3}	12.30
[Kul11]	**Kul16**	351.65	23.93	391	81.05	1.4×10^{-2}	3.3×10^{-2}	4.69
[Reh16]	**Reh16**	428.64	24.44	412	98.34	2.1×10^{-2}	1.4×10^{-1}	4.75
[GSK18]	**LOAM16**	238.04	66.45	**297**	99.25	$\mathbf{1.4\times10^{-5}}$	$\mathbf{4.9\times10^{-4}}$	**15.76**
[War20]	**Ax16_1**	378.60	26.70	404	88.6	2.4×10^{-4}	4.4×10^{-3}	11.22
	Ax16_2	352.17	32.21	393	95.18	5.5×10^{-3}	3.9×10^{-2}	6.69
	Ax16_3	319.13	40.91	357	99.16	1.1×10^{-2}	7.3×10^{-2}	5.99
[War21]	**AxRM16_1**	395.49	19.19	401	97.85	2.6×10^{-4}	1.2×10^{-2}	11.42
	AxRM16_2	363.25	22.10	390	99.52	4.3×10^{-3}	2.2×10^{-1}	7.39
	AxRM16_3	350.54	23.52	391	99.97	5.2×10^{-3}	3.4×10^{-1}	7.20

Table 4.7: Performances of 32×32 Approximate Recursive Multipliers.

32×32 Design		Area* [μm²]	Power Reduction [%]	Min Delay* [ps]	Error Rate [%]	NMED	MRED	NoEB
	Exact	1477.6	-	448	-	-	-	64
Proposed	**N32-L1**	1074.1	40.42	553	100.00	9.6×10^{-10}	4.9×10^{-8}	29.78
	N32-L2	1073.1	36.80	540	100.00	1.1×10^{-9}	5.5×10^{-8}	29.53
	N32-5	1933.6	-24.14	590	92.66	6.1×10^{-5}	1.1×10^{-3}	13.01
	N32-6	1932.2	-26.20	590	92.97	8.4×10^{-5}	1.5×10^{-3}	12.54

**Area and power are reported for the circuits synthesized with a timing constraint of 1000ps. Min Delay is the minimum timing at which the circuit can be synthesized with a non-negative slack.*

4.4 Applications

Image processing is one of the most considered error resilient applications and many papers test the proposed circuits in this scenario. In this work, two image processing applications are considered: image blurring and image sharpening. The applications provide a more in depth understanding of the applicability range of the proposed designs. Additionally, the considered 8×8 designs are tested in an image classification application using a convolutional neural network (CNN).

4.4.1 Image Smoothing

In image processing, low pass filtering (image smoothing), effectively removes the high spatial frequency noise. As mentioned in section 3.5, the low pass filter exploits a moving kernel that processes one pixel at a time and modifies it considering the pixels in proximity.

The kernel considered for the smoothing is again the two dimensional, rotationally symmetric, 3×3 Gaussian low-pass filter, with a standard deviation equal to 1.5, as in [Esp18]. The floating-point numbers of the kernel are multiplied by 2^{10} and then rounded. In this way, the kernel's values are appropriate for the considered 8-bit input multipliers. The original and modified kernels are shown in Table 4.8.

Image processing, exploiting the investigated multipliers, has been performed aiming to blur a test image. The obtained images are shown in Fig. 4.13. The same processing has been also performed with exact multipliers to provide an effective comparison for all designs. The structural similarity index (SSIM) and the peak signal to noise ratio (PSNR) provide a numerical indication of each multiplier's performance in image smoothing.

The results are shown in Table 4.9. The recursive designs are in the top part of the table, while the non-recursive ones occupy the bottom part. The proposed circuits N8-5 and N8-6 share the best results with

Table 4.8: Gaussian Kernels with SD=1.5.

Original			Modified		
0.095	0.118	0.095	97	121	97
0.118	0.148	0.118	121	151	121
0.095	0.118	0.095	97	121	97

the designs M8-5 and M8-6 proposed in [Ans18] and Ax8_1 proposed in [War20]. N8-L1 and N8-L2 follow close behind but still show a competitive behavior while achieving the greatest power reduction, as shown in Fig. 4.12.

Table 4.9: Performances of 8×8 Approximate Multipliers in Image Smoothing.

8×8 Design		SSIM [%]	PSNR [dB]
Proposed	**N8-L1**	97.85	41.7
	N8-L2	97.66	39. 5
	N8-5	97.98	43.0
	N8-6	97.98	43.0
Proposed	**T8_1**	25.3	0.910
	T8_2	24.3	0.911
	T8_3	42.6	0.979
	T8_4	40.8	0.977
	T8_5	43.0	0.980
	T8_6	43.0	0.980
[Ans18]	**M8-1**	90.86	28.8
	M8-2	90.18	23.9
	M8-3	97.93	42.2
	M8-4	97.72	40.7
	M8-5	97.98	43.0
	M8-6	97.98	43.0
[Kul11]	**Kul8**	97.81	41.0
[Reh16]	**Reh8**	78.88	18.5
[GSK18]	**LOAM**	97.90	42.4
[Gil19]	**ISH1**	97.38	39.8
	ISH2	97.88	42.0
[War20]	**Ax8_1**	97.96	43.0
	Ax8_2	97.85	39.2
	Ax8_3	97.25	35.6
[War21]	**AxRM1**	97.97	43.0
	AxRM2	97.90	41.5
	AxRM3	97.85	41.2
[Str22]	**SSM_m4**	94.39	26.8
	SSM_m4_u3	96.41	38.9
[Yan18]	**Yang_7'b0**	93.44	29.1
	Yang_7'b1	97.44	38.9
[Fru20]	**DT2**	97.67	42.31
	DT4	97.67	42.31
	DT8	97.37	35.61

Fig. 4.13.a: Gaussian smoothing of images obtained with different multipliers. The circuits proposed in this work are highlighted in bold.

Fig. 4.13.b: Gaussian smoothing of images obtained with different multipliers.

4.4.2 Image Sharpening

Sharpening or high pass filtering aims to make fine details more distinct and remove the blurring of a digital image, by enhancing transitions in the spatial intensity of the image. High frequencies are boosted while low frequencies are reduced. It should be noted that over-sharpening might result in unwanted halo artifacts.

The image sharpening process is similar to the smoothing process, but it uses a different kernel for the convolution. The authors in [Yan18], [Vah19], [Guo18], [Kha21] performed image sharpening exploiting the following 5×5 kernel:

$$Mask = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{4.16}$$

Table 4.10: Performances of 8×8 Approximate Multipliers in Image Sharpening

8×8 Design		SSIM [%]	PSNR [dB]
Proposed	**N8-L1**	99.47	38.2
	N8-L2	99.41	37.4
	N8-5	99.92	56.6
	N8-6	99.88	53.9
[Ans18]	**M8-1**	99.77	48.9
	M8-2	99.58	40.3
	M8-3	99.77	48.9
	M8-4	99.58	40.3
	M8-5	99.96	60.7
	M8-6	99.88	54.0
[Kul11]	**Kul8**	99.97	56.9
[Reh16]	**Reh8**	79.17	22.2
[GSK18]	**LOAM**	99.53	44.0
[Gil19]	**ISH1**	99.97	56.6
	ISH2	99.96	52.9
[War20]	**Ax8_1**	99.85	53.9
	Ax8_2	97.67	22.7
	Ax8_3	97.67	22.7
[War21]	**AxRM1**	99.59	49.3
	AxRM2	98.32	26.4
	AxRM3	80.78	28.3
[Str22]	**SSM_m4**	93.19	18.5
	SSM_m4_u3	96.52	31.2
[Yan18]	**Yang_7'b0**	94.37	29.2
	Yang_7'b1	99.92	51.3
[Fru20]	**DT2**	99.95	59.3
	DT4	99.86	49.3
	DT8	96.51	23.8

The output pixels of the sharpened image are given by:

$$Y(i,j) = 2 \cdot X(i,j) + \\ -\frac{1}{273}\sum_{m=-2}^{2}\sum_{n=-2}^{2}[X(i+m,j+n) \times Mask(m+3,n+3)] \quad (4.17)$$

In (4.17), $X(i,j)$ denotes a pixel from the input image, while $Y(i,j)$ from the sharpened output.

The considered approximate multipliers, as well as an exact multiplier have been used to sharpen an RGB test image. The results are demonstrated in Fig. 4.14. SSIM and PSNR with respect to the sharpened image by exact multipliers are reported in Table 4.10. All proposed circuits have a high similarity ratio with the reference image. Even though there are better performing multipliers for this application, the proposed circuits exhibit reasonable behavior for such low-power designs.

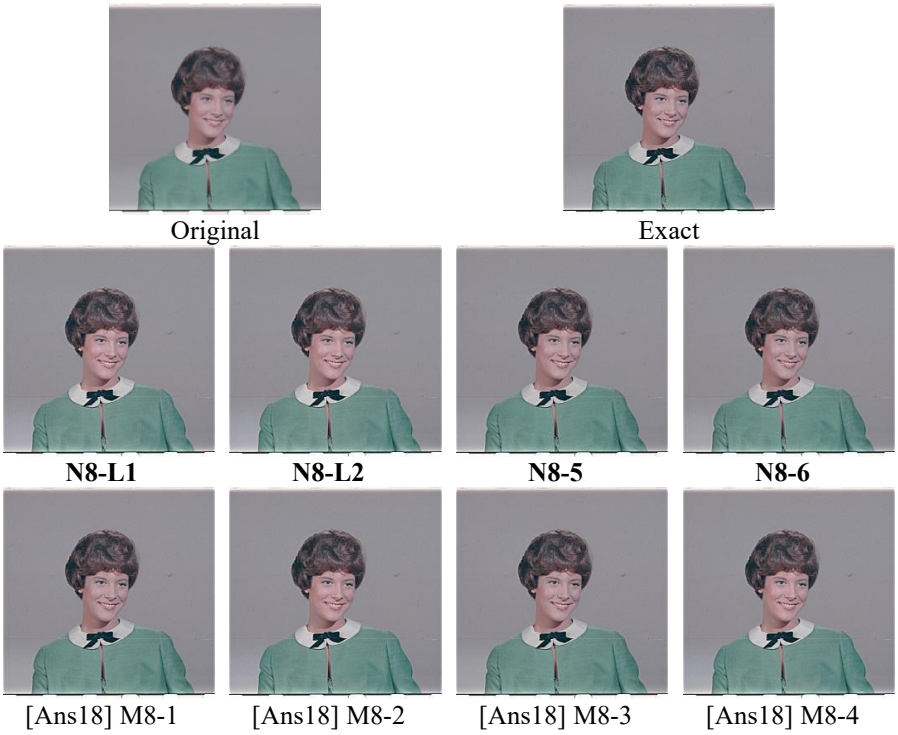

Fig. 4.14.a: Image sharpening obtained with different multipliers. The circuits proposed in this work are highlighted in bold.

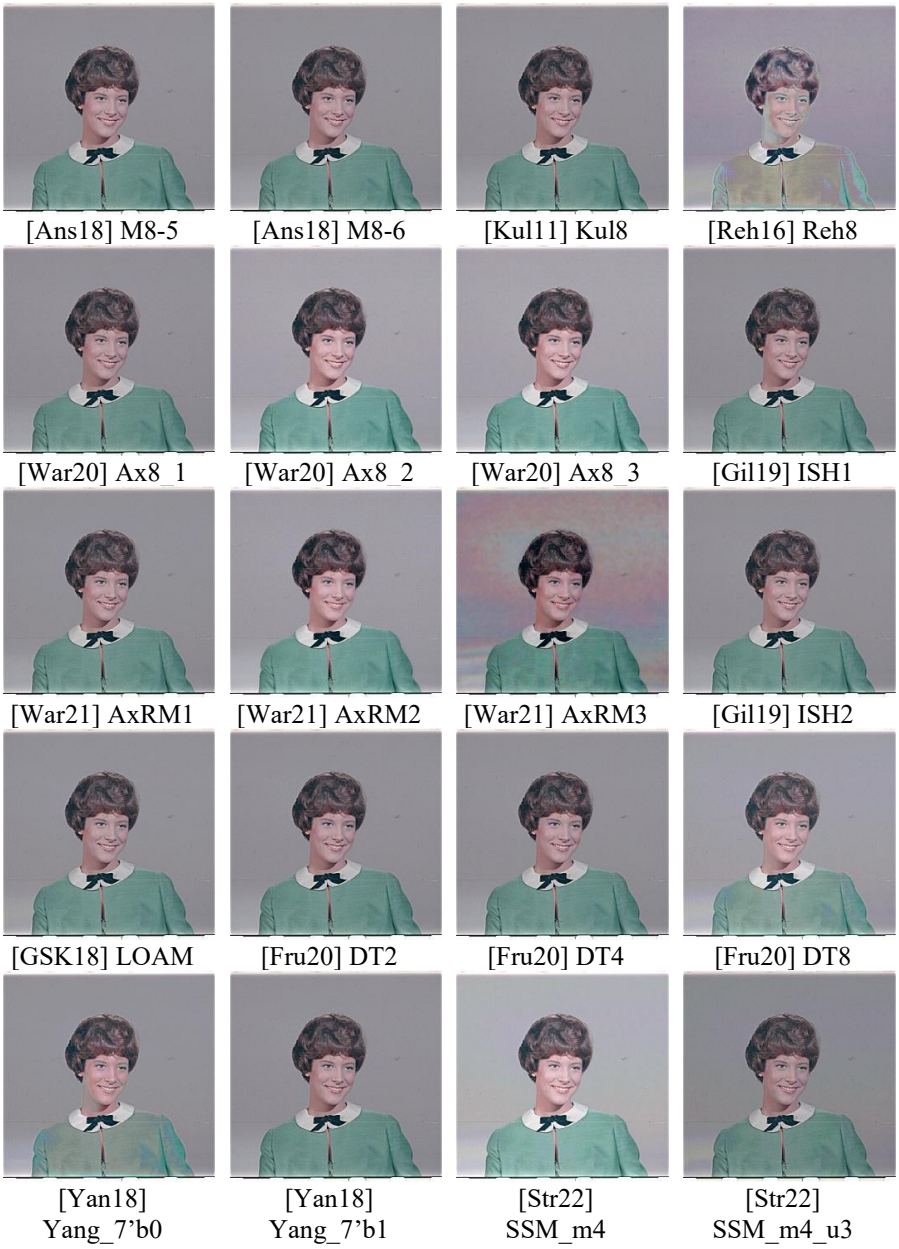

Fig. 4.14.b: Image sharpening obtained with different multipliers.

4.4.3 Image Classification

Convolutional Neural Networks (CNNs) play an increasingly important role in machine learning, particularly for image recognition, object identification and speech recognition tasks. CNNs are error-tolerant and require a huge number of multiplications, therefore they are ideal candidates for using approximate multipliers [Ans20].

Image recognition experiments have been performed with the investigated approximate multipliers, using a simple CNN composed by 9 layers, not counting the input one. The CNN includes two convolutional layers, each one followed by batch normalization and Rectified Linear Unit (ReLU) layers, a max pooling layer, a fully connected layer and a final softmax layer. Two datasets have been considered: MNIST and SVHN. The former is a dataset of handwritten digits containing 70,000 28x28-pixel, greyscale images split into 60,000 training images and 10,000 testing images [LeC10]. The Street View House Number (SVHN) dataset contains 100,000 32x32 RGB images of house numbers obtained from Google Street View, divided in 73,257 training and 26,032 test images [Net11]. In this work, SVHN images have been converted into greyscale as the color has no significance in classification [Ans20].

The training of the CNNs has been performed in MATLAB, by using floating-point arithmetic. After training, quantization of the convolutional and fully connected layers, requiring the vast majority of calculations, has been performed, to allow the testing of the approximate multipliers. We use test images to exercise the network and collect the dynamic ranges of the inputs of convolutional and fully connected layers. These inputs are positive values, due to the ReLU layers, and are easily quantized as 8-bit unsigned numbers that can directly feed the multipliers. The weights in the convolutional and fully connected layers of the network, on the other hand, are learnt during training and are signed numbers. Therefore, following [Ahm22], after quantization converted the weights have been converted in sign-magnitude representation to perform multiplications using the investigated unsigned approximate multipliers.

Classification results are reported in Table 4.11. Column “Acc. loss” refers to the reduction in classification accuracy (in percentage) compared to the floating-point multiplier.

Table 4.11: Image Classification results using 8×8 Approximate Multipliers

8×8 Design		MNIST		SVHN	
		Accuracy	Acc. loss	Accuracy	Acc. loss
	Floating Point	99.04%	-	87.18%	-
	8bit Exact	99.01%	0.03%	87.25%	-0.07%
Proposed	**N8-L1**	98.40%	0.64%	76.71%	10.48%
	N8-L2	97.43%	1.61%	70.82%	16.36%
	N8-5	99.01%	0.03%	87.01%	0.17%
	N8-6	99.00%	0.04%	86.88%	0.30%
[Ans18]	**M8-1**	98.46%	0.58%	50.93%	36.25%
	M8-2	98.48%	0.56%	40.44%	46.74%
	M8-3	98.92%	0.12%	83.54%	3.65%
	M8-4	98.06%	0.98%	75.93%	11.25%
	M8-5	99.00%	0.04%	87.08%	0.10%
	M8-6	99.01%	0.03%	86.88%	0.30%
[Kul11]	**Kul8**	89.62%	9.42%	77.56%	9.62%
[Reh16]	**Reh8**	77.12%	21.92%	24.58%	62.60%
[GSK18]	**LOAM**	99.02%	0.02%	83.66%	3.52%
[Gil19]	**ISH1**	68.75%	30.29%	77.67%	9.52%
	ISH2	96.40%	2.64%	73.99%	13.20%
[War20]	**Ax8_1**	99.02%	0.02%	86.90%	0.28%
	Ax8_2	68.47%	30.57%	22.12%	65.06%
	Ax8_3	58.32%	40.72%	24.17%	63.01%
[War21]	**AxRM1**	99.00%	0.04%	86.56%	0.62%
	AxRM2	96.98%	2.06%	56.20%	30.98%
	AxRM3	49.50%	49.54%	20.77%	66.41%
[Str22]	**SSM_m4**	17.65%	81.39%	18.20%	68.98%
	SSM_m4_u3	96.75%	2.29%	53.63%	33.55%
[Yan18]	**Yang_7'b0**	75.09%	23.95%	36.65%	50.53%
	Yang_7'b1	98.68%	0.36%	83.84%	3.35%
[Fru20]	**DT2**	99.02%	0.02%	87.17%	0.01%
	DT4	98.97%	0.07%	86.26%	0.92%
	DT8	72.17%	26.87%	25.32%	61.86%

For the MNIST dataset, the considered CNN in floating-point implementation shows a remarkable accuracy of more than 99%. The accuracy remains almost unchanged by using exact 8-bit multiplier after network quantization. The majority of investigated approximate multipliers perform well with this simple dataset, with some exception (Yang_7'b0, Ax8_2, Ax8_3, SSM_m4, Kul8, Reh8, AxRM3, ISH1).

The proposed N8_L1 gives very good results, showing a mere 0.64% reduction in accuracy, with more than 44% power saving.

For the SVHN dataset the CNN accuracy is about 87%. In this case, network quantization yields a slight accuracy improvement, a phenomenon already observed in literature [Ans20].

Several approximate multipliers yield a large accuracy reduction in this more demanding application. The multipliers giving an accuracy drop lower than 0.5% are: proposed N8_5 and N8_6, DT2 of [Fru20], M8_5 and M8_6 of [Ans18], Ax8_1 of [War20] and AxRM1 [War21]. Among these, the proposed N8_6 gives the best power reduction of more than 14%. Design Yang_7'b1 of [Yan18] also performs well, with a reduction in accuracy of 3.3% and a power saving of more than 24%.

4.5 Summary

In this chapter, five low-energy 4×4 approximate multipliers have been presented. They are obtained by simplifying the sum and carry expressions of the partial product matrix adders while avoiding the bulky and slow XOR gates as much as possible. The proposed designs exhibit a very good tradeoff between power reduction and precision.

The proposed designs, an exact multiplier, and a multiplier that performs the PPM addition by means of OR gates, are used recursively scale up to 8×8, 16×16 and 32×32 approximate multipliers. Two different methodologies are utilized to generate the higher order multipliers. The first one, uses four n×n modules to perform the following multiplications: $a_L b_L$, $a_H b_L$, $a_L b_H$, and $a_H b_H$. After that, an exact adder is used to add the shifted sub-products according to Fig 4.9 and obtain the final product of the $2n$-bit numbers, a and b. The second methodology uses four n×n building blocks as well and employs an exact multiplier for the most significant bits (MSBs) and an OR-based approximate multiplier for the least significant bits (LSBs). An approximate adder is then used to add the four sub-products, as shown in Fig. 4.10.

The second methodology is generally less accurate, especially for low order multipliers, due to the low accuracy 4×4 module used for the LSBs and the approximation in the addition process. However, it

uses considerably less resources than the strictly recursive architecture, and still manages to generate useful circuits that fall on the Pareto front. For higher order multipliers, the exact module used for the MSBs makes up for the loss of accuracy, with respect to the strictly recursive methodology, that might handle the MSBs with exact and approximate sub-multipliers.

Ten 8×8 designs have been proposed that cover a wide range of accuracy. Each 8×8 approximate multiplier consists of exact, proposed, and/or OR-based, 4×4 designs. By exploiting the low power proposed circuits, T1, T2, T3, N1, and N2, the 8×8 approximate multipliers developed in this work, achieve great power reduction while still exhibiting competitive error performance. Selected higher order designs are also reported in this chapter.

The proposed 8×8 circuits are tested in image processing and image classification using a convolutional neural network. It is demonstrated that these designs can be fruitfully used to save power without sacrificing the result in typical error resilient applications.

Chapter 5

Approximate Recursive Squarers

Many operations in digital signal processing require the square of a signal [Gil18], [She02], [Lan06], [Man20], [Ans22], [Gar10], [Sha15], and [Pet14], for polynomial evaluation [Noe89], computation of Euclidean distances with the Viterbi [Vit67] or other algorithms, signal demodulation [Xu16], [Chi13], [Ans20], [Avr14], vector quantization [Sol89], etc.

Despite the fact that the squaring operation can be regarded as a special multiplication case, it is often preferable to develop independent squaring circuits to exploit possible architectural symmetries. A dedicated design can exploit the inherent symmetries by folding the partial product matrix, thus reducing the required resources significantly. The folded PPM of a squared 4-bit number $A = \sum_{i=0}^{3} a_i 2^i$ is shown in Fig. 5.1.

Various approximate squaring circuits have been proposed in literature. Some of the most prominent techniques are presented briefly in the following. The authors in [Kol98] propose a scheme that improves the critical delay by encoding the partial products along the

				a_3	a_2	a_1	a_0
				x a_3	a_2	a_1	a_0
				a_3a_0	a_2a_0	a_1a_0	a_0a_0
			a_3a_1	a_2a_2	a_1a_2	a_0a_1	
		a_3a_2	a_2a_2	a_1a_2	a_0a_2		
	a_3a_3	a_2a_3	a_1a_3	a_0a_3			
	a_3	a_3a_1	a_3a_0	a_2a_0	a_1	0	a_0
	a_3a_2		a_2a_1		a_1a_0		
			a_2				
y[7]	y[6]	y[5]	y[4]	y[3]	y[2]	y[1]	y[0]

Fig. 5.1: Partial product matrix folding for a 4-bit squared number.

diagonal, in the rest of the folded matrix. Boolean simplifications are proposed in [Bui14] and [Das16] to reduce the folded PPM. A divide and conquer method to produce optimal blocks has been presented in [Yoo97]. The authors in [Car01] and [Str03] investigate the combination of Booth-encoding and folding techniques to significantly reduce the computational burden. Cellular logic arrays to compute the squaring function are developed in [Dea69], [Sha91].

Numerous digital signal processing (DSP) applications handle data acquired from noisy analog-to-digital converters (ADCs). The innate error in such cases, along with the strict power, area, and timing requirements, make approximate squaring an appealing solution. In [Gil18] the authors propose an approximate squarer, comprised by a pair of mirrored modules to average out the error. In [She02] and [Lan06] simple Boolean expressions for the approximate computation of the squaring function, for any bit-width, are presented. In [Man20] the authors exploit approximate partial product generators and accumulators to develop three radix-4 Booth squarers. In [Avr14] and [Ans22] approximate logarithmic squaring circuits are proposed. The first employs a compensation block to minimize the average error while the latter uses double sided error distribution. In [Sha15] the authors present a general model for array-based approximate arithmetic computing and an error compensation unit. A class of truncated squarers with smart rounding for error compensation is presented in [Pet14].

In this work, novel approximate binary squarers, obtained by recursively exploiting 4-bit approximate multipliers and smaller size squarers are proposed. The final designs cover a wide range of computing precision, providing the user with multiple choices of different cost vs. accuracy trade-offs. The proposed circuits, as well as competitive designs, are synthesized targeting a 14nm FinFET technology to determine the electrical characteristics.

5.1 Recursive Squaring Methodology

Let's assume we want to calculate the square of an unsigned 8-bit number: $A = \sum_{i=0}^{7} a_i 2^i$. We divide A into two parts, the lower part: $A_L = \sum_{i=0}^{3} a_i 2^i$, and the higher part: $A_H = \sum_{i=4}^{7} a_i 2^i$, where $A = A_H + A_L$. The square of A can be calculated as:

$$Y = A^2 = (A_H + A_L)^2 = A_H^2 + 2A_L A_H + A_L^2 \qquad (5.1)$$

From equation 5.1, it is easily derived that an 8-bit squarer can be implemented exploiting two 4-bit squarers and a 4×4 multiplier. As shown in Fig. 5.2, the sub-product $2A_L A_H$ can be implemented by one multiplier with A_H and A_L as inputs, and an output shifted by 5 bits, instead of 4. This technique manages to offer a certain flexibility to approximate multipliers, at a small hardware cost, since every operation in (5.1), including the additions, can be suitably approximated.

A detailed description of the 4-bit approximate squarers exploited in this work, is reported in the following. Exact and approximate 4-bit multipliers are also used as equation 5.1 suggests. The 4×4 approximate multipliers used to generate higher-order squaring circuits are N1 and N2, that are discussed in chapter 4.

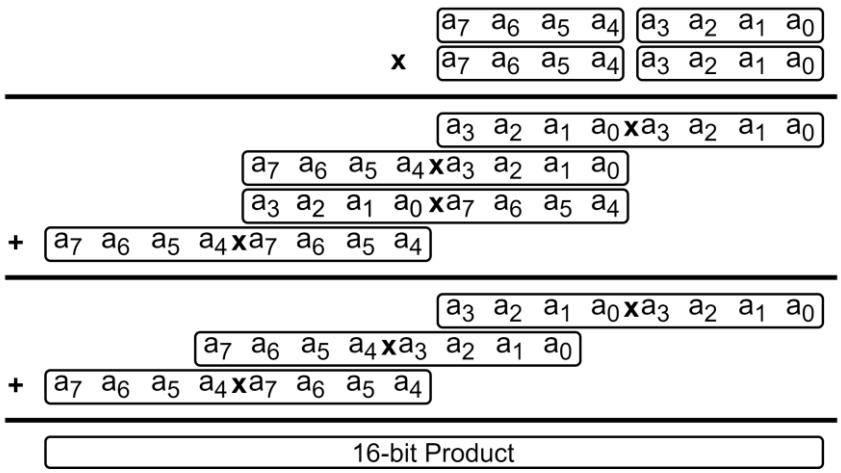

Fig. 5.2: Recursive squaring $A^2 = (A_H + A_L)^2$.

5.2 4-Bit Squarers

5.2.1 4-Bit Exact Squarer

The squaring of A_H and A_L can be implemented with an exact or an approximate squarer, depending on the desired precision. The 8-bit output of the exact 4 bit squarer, after logic minimization, is given by the Boolean equations reported below:

$$\begin{aligned} Y[7] &= a_3 a_2 \\ Y[6] &= a_3 \overline{a_2} + a_3 a_1 \\ Y[5] &= \overline{a_3} a_2 a_1 + a_3 \overline{a_2} a_1 + a_3 a_2 a_0 \\ Y[4] &= \overline{a_3} a_2 a_0 + a_3 \overline{a_2} a_0 + a_2 \overline{a_1} \overline{a_0} \\ Y[3] &= \overline{a_2} a_1 a_0 + a_2 \overline{a_1} a_0 \\ Y[2] &= a_1 \overline{a_0} \\ Y[1] &= 0 \\ Y[0] &= a_0 \end{aligned} \tag{5.2}$$

By neglecting specific terms from the exact 4-bit squarer, while trying to account for error compensation, two approximate squarers named S1 and S2, have been developed. The proposed approximate squarers offer double sided error distribution with beneficial effects on the precision of the results.

5.2.2 4-Bit Approximate Squarer S1

To obtain S1, let us observe column 2 of the folded partial product matrix in Fig. 5.1, where the two terms a_1, $a_1 a_0$ must be added. Table 5.1 shows the sum and carry bits obtained by adding these terms. As it can be seen, the addition can be simplified as $Y_1[2] \approx a_1$. This simplified expression results in a single error case, underestimating the exact result for $a_1 = 1$, $a_0 = 1$ while not providing any carry for the next column of the PPM. Therefore (from Fig. 5.1) we immediately obtain $Y_1[3] = a_2 a_0$.

Table 5.1: Approximate addition for $Y_1[2]$.

a_1	a_0	$a_1 a_0$	Carry	Sum	Sapp
0	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	1	1
1	1	1	1	0	**1**

Table 5.2: Karnaugh Maps for $Y_1[4]$ (left) and c_5 (right). The values in bold red are inverted in the *S*1 approximate squarer.

a_1a_0 \ a_3a_2	00	01	11	10
00	0	1	1	0
01	0	1	**0**	1
11	0	**0**	1	1
10	0	0	0	0

a_1a_0 \ a_3a_2	00	01	11	10
00	0	0	0	0
01	0	0	**1**	0
11	0	1	1	0
10	0	1	1	0

The terms to be added in column 4 are: a_3a_0, a_2a_1, and a_2. Table 5.2 reports the Karnaugh map for the sum and carry obtained by summing these terms. The red terms marked in bold, are inverted, to obtain a simpler expression. The sum $Y_1[4]$ can be approximated as: $Y_1[4] \approx a_2a_0 + a_3a_0 + a_2\overline{a_1}$ with two errors in the Karnaugh map, that result in an over-estimation of the result. The carry entering in column 5 can also be approximated as: $c_5 = a_2a_1$, with a single error in the Karnaugh map, under-estimating the exact result.

The terms to be added in column 5 are c_5 and the partial product a_3a_1. Thus, we can obtain: $Y_1[5] = \overline{a_3}a_2a_1 + a_3\overline{a_2}a_1$. For $Y_1[6]$ and $Y_1[7]$ we use the exact equations, shown in equation 5.2.

The resulting expressions are summarized below:

$$\begin{aligned} Y_1[7] &= a_3a_2 \\ Y_1[6] &= a_3\overline{a_2} + a_3a_1 \\ Y_1[5] &= \overline{a_3}a_2a_1 + a_3\overline{a_2}a_1 \\ Y_1[4] &= a_2a_0 + a_3a_0 + a_2\overline{a_1} \\ Y_1[3] &= a_2a_0 \\ Y_1[2] &= a_1 \\ Y_1[1] &= 0 \\ Y_1[0] &= a_0 \end{aligned} \tag{5.3}$$

5.2.3 4-Bit Approximate Squarer S2

To achieve additional hardware minimization while sacrificing accuracy in the process, the calculation of signal $Y_1[4]$ is further simplified. Following [GSK18], an OR gate, whose hardware impact is very small, is used to sum the partial products in column 4. The resulting Boolean expression is: $Y_2[4] \approx a_2 + a_3a_0$. As shown in Table 5.3, on the left, this approximation overestimates the exact result. To mitigate this error, the carry generated in column 4 is underestimated, which as shown in Table 5.3, is approximated as: $c_5 \approx a_3a_2a_1$.

Table 5.3: Karnaugh Maps for $Y_2[4]$ (left) and c_5 (right). The values in bold red are inverted in the *S*2 approximate squarer.

a_1a_0 \ a_3a_2	00	01	11	10
00	0	1	1	0
01	0	1	**0**	1
11	0	**0**	1	1
10	0	**0**	**0**	0

a_1a_0 \ a_3a_2	00	01	11	10
00	0	0	0	0
01	0	0	**1**	0
11	0	**1**	1	0
10	0	**1**	1	0

In this way, the output in column 5 is obtained as: $Y_2[5] = (a_3a_1) \oplus (a_3a_2a_1) = a_3\overline{a_2}a_1$. Moreover, a carry $c_6 = a_3a_2a_1$ is also generated in column 5. The XOR of c_6 with the partial products in column 6 gives: $Y_2[6] = a_3\overline{a_2} + a_3a_2a_1$. For $Y_2[7]$, the exact equation, given in (5.2) is used.

The Boolean equations for *S*2 are summarized below:

$$\begin{aligned} Y_2[7] &= a_3a_2 \\ Y_2[6] &= a_3\overline{a_2} + a_3a_2a_1 \\ Y_2[5] &= a_3\overline{a_2}a_1 \\ Y_2[4] &= a_2 + a_3a_0 \\ Y_2[3] &= a_2a_0 \\ Y_2[2] &= a_1 \\ Y_2[1] &= 0 \\ Y_2[0] &= a_0 \end{aligned} \tag{5.4}$$

5.2.4 4-Bit Modules summary

The performance improvement obtained by N1, N2, S1 and S2 when compared to the corresponding exact designs is summarized in Table 5.4. The ER, NMED, MRED, and NoEB, are also reported.

Table 5.4: Electrical and error performances of the approximate 4-bit Multipliers and Squarers with respect to the exact designs.

4-Bit Design		Reduction in Area [%]	Reduction in Power [%]	Reduction in Min Delay [%]	Error Rate [%]	NMED ($\times 10^{-2}$)	MRED ($\times 10^{-2}$)	NoEB
Mult.	**Exact**	-	-	-	-	-	-	8.00
	N1	68.62	63.91	63.48	35.94	1.76	5.57	4.83
	N2	73.71	68.44	83.48	37.71	2.44	7.24	4.42
Squar.	**Exact**	-	-	-	-	-	-	8.00
	S1	27.75	23.30	20.37	31.25	1.11	5.57	5.35
	S2	45.65	35.08	40.74	43.75	2.22	10.16	4.72

5.3 Proposed Approximate 8-Bit Squarers

5.3.1 Configurations

As already described, a binary squaring circuit can be implemented by exploiting modular building blocks as shown in (5.1). The desired approximation may be introduced by means of approximate components, and/or using approximate adders. Assuming a uniform input distribution, as is appropriate for general purpose applications, the utilized modules should be selected following a decreasing order of accuracy while moving towards the least significant bit of the result. In this way, high-precision components may be used to compute the most significant terms, while less demanding circuits in terms of hardware resources, will compute the least significant term(s).

Aiming to effectively populate the pareto front of the design space many possible configurations have been considered. The best performing configurations are those summarized in Table 5.5. As shown in the last three columns, the instantiated squarers and multipliers may be the exact ones or one of the previously presented approximate circuits.

The nomenclature of the designs is 'S8_Num_Add' where 'Num' is an increasing number and 'Add' indicates the type of final adder that has been implemented, when the exact one is not chosen. Designs "S8_x" use an exact carry save adder to add the three terms produced by the sub-modules. "S8_x_OR" exploit OR gates to approximately add the three terms [GSK18], resulting in a significant

Table 5.5: Configurations of proposed 8-bit Approximate Squarers.

8-Bit Design	Adder	a_H^2	$a_H a_L$	a_L^2
S8_1	Exact	Exact	Exact	*S*1
S8_2	Exact	Exact	Exact	*S*2
S8_3	Exact	Exact	*N*1	*S*2
S8_4	Exact	Exact	*N*2	*S*2
S8_1_MIX	Exact/OR	Exact	Exact	*S*2
S8_2_MIX	Exact/OR	Exact	*N*1	*S*2
S8_3_MIX	Exact/OR	Exact	*N*2	*S*2
S8_1_OR	OR	Exact	*N*2	*S*2
S8_2_OR	OR	*S*1	*N*2	*S*2

hardware usage minimization, as shown in Table 5.6. A fine compromise between the two previous architectures, is achieved by designs "S8_x_MIX". In this case, the two most significant terms that correspond to a_H^2 and $a_H a_L$, are added using an exact adder, while the least significant term, is summed approximately exploiting OR gates.

5.3.2 Performances

The proposed designs, as well as competitive squaring circuits found in the literature, have been synthesized, targeting a 14nm FinFET technology, using Cadence Genus. The power dissipation is derived by simulating the final netlist with a random set of inputs, that effectively trigger the switching activity of each node. The input vector array and the timing constraints are identical for all designs, thus ensuring a fair comparison between different designs. Specifically, the set of inputs consists of 10^5 uniformly distributed 8-bit numbers, and the synthesizing timing constraint, corresponds to 1ns. Moreover, the area requirements of each design are also extracted under the same timing constraint. "Min Delay" on the other hand, refers to the tightest timing constraint under which each design can be synthesized with a non-negative slack time. Thus "Min Delay" represents the maximum working speed of each investigated circuit.

In order to put the obtained area, power, and delay values for the considered designs into perspective, the exact 8-bit squarer is included in the comparison. The exact design is obtained by describing the circuit in HDL (Verilog), by means of the multiplication operator:

$$y = a * a \tag{5.5}$$

In this way, the synthesizer picks the optimal topology for the selected timing constraint.

The error performance of the considered designs is gathered by performing exhaustive simulations, i.e., considering all 256 possible inputs.

The electrical and error performance of the approximate and exact 8-bit squarers is summarized in Table 5.6. It should be noted that the total power reported in this table is computed for a clock frequency equal to 1GHz. Fig. 5.6 shows the normalized mean error distance in a logarithmic scale, with respect to power consumption. The optimal point in this graph is located at the bottom-left corner, where NMED

and power consumption are minimal. The number of effective bits with respect to power consumption is shown in Fig. 5.7. In this graph, the optimal spot is located at the bottom-right corner, where great accuracy and low power consumption are achieved.

As shown in Fig. 5.6 and 5.7 the proposed circuits, along with [Pet14], form the Pareto front. By covering a wide range of precision and hardware cost, the proposed architectures can be extremely useful in approximate computing applications.

The proposed circuits, as well as [Pet14], are the only available for the lower error range (NoEB larger than 7 or NMED lower than 5×10^{-3}). When the error is comparable with the state of the art (S8_2_OR), the closest competitor (approx_1 from [Lan06]) dissipates about 30% more power.

Table 5.6: Performances of 8-bit Approximate Squarers. Area and Power are obtained for a timing constraint equal to 1ns and a simulation data rate equal to 1G test vectors per second. Error Performance is obtained by exhaustive simulations.

8-Bit Design		Area [μm²]	Power [μW]	Min Delay [ps]	Error Rate [%]	NMED	MRED ($\times10^{-2}$)	NoEB
	Exact	36.57	52.83	201	-	-	-	16.00
Proposed	**S8_1**	31.02	45.15	215	31.3	3.8×10^{-5}	0.4	13.35
	S8_2	31.56	43.99	234	43.8	7.7×10^{-5}	0.7	12.72
	S8_3	23.96	35.57	148	53.1	2.0×10^{-3}	1.3	7.99
	S8_4	21.52	33.03	128	53.9	2.8×10^{-3}	1.5	7.53
	S8_1_MIX	26.97	40.05	186	53.1	4.7×10^{-4}	1.3	10.11
	S8_2_MIX	18.22	28.18	144	58.2	2.4×10^{-3}	2.0	7.78
	S8_3_MIX	16.26	26.98	125	58.6	3.2×10^{-3}	2.2	7.39
	S8_1_OR	10.30	17.52	66	70.3	8.5×10^{-3}	4.8	6.06
	S8_2_OR	8.89	15.42	57	74.2	1.9×10^{-2}	8.7	4.89
[Avr14]	**BB**	12.84	15.80	79	96.5	4.7×10^{-2}	11.0	3.65
	BB1ERC	41.36	50.62	188	85.5	6.6×10^{-3}	1.4	6.14
[She02]	**fast_comp**	8.42	20.40	61	59.4	1.7×10^{-2}	3.8	4.87
[Lan06]	**approx_1**	11.57	20.34	66	81.3	1.9×10^{-2}	5.4	5.16
	approx_2	11.82	21.16	68	75.4	1.3×10^{-2}	3.2	5.55
[Ans22]	**LESF**	35.95	54.47	249	98.4	9.2×10^{-3}	3.7	6.11
	LESF_t	33.95	55.15	241	98.4	1.7×10^{-2}	3.0	3.52
[Pet14]	**Gar**	23.53	37.93	213	93.75	1.3×10^{-3}	8.4	9.30

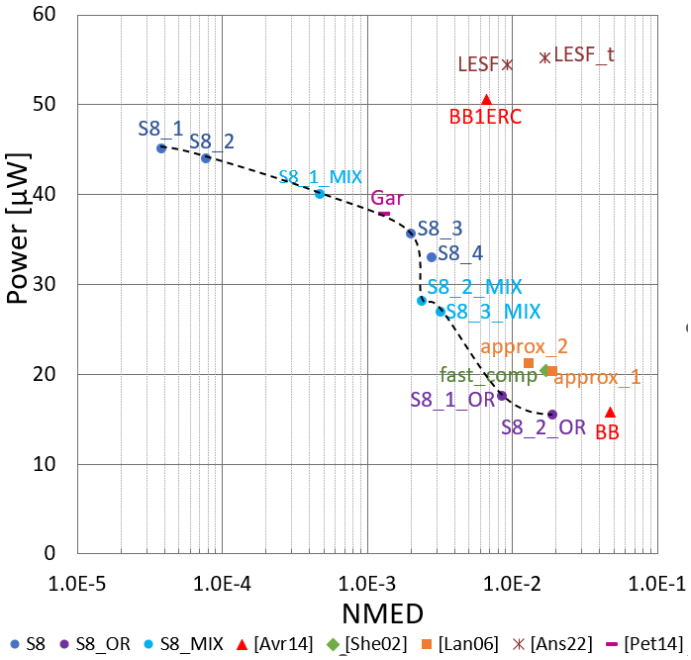

Fig 5.6: NMED vs Power for 8-bit squarers.

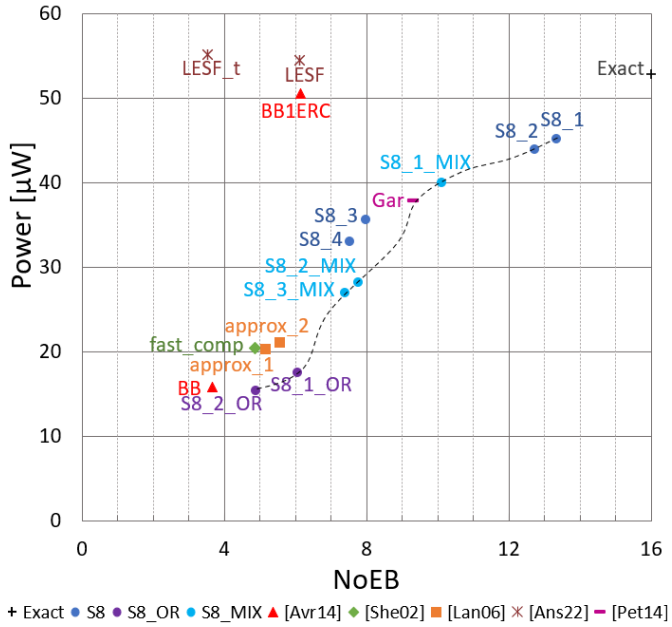

Fig 5.7: NoEB vs Power for 8-bit squarers.

5.4 Applications

To better evaluate the examined designs, two signal processing applications have been considered. The merit of each squaring circuit has been obtained by quantifying the degradation of the output signals, after approximate squaring has taken place.

5.4.1 AM Demodulation

One application that requires a lot of squaring operations is the square law detector [Avr14], [Ans22]. In the field of telecommunications, it is well known that a signal needs to be modulated to carry information over a band-pass channel. Amplitude modulation (AM) is a simple modulation technique, in which the carrier amplitude varies according to the value of the signal to be transmitted. Let the carrier be:

$$c(t) = A_c cos(2\pi f_c t) \tag{5.6}$$

with A_c and f_c being the amplitude and the frequency of the carrier, respectively. Let us indicate as $m(t)$ the signal to be transmitted, with $|m(t)|<1$. The modulated signal is given by:

$$s_m(t) = \big(1 + m(t)\big)\, A_c cos(2\pi f_c t) \tag{5.7}$$

At the receiver side, the modulated signal $s_m(t)$ can be demodulated by the square-law detector, to recover the message signal. The first step is the squaring of the received signal:

$$s_m^2(t) = {A_c}^2 \big(1 + m(t)\big)^2 \frac{1 + cos(4\pi f_m t)}{2} \tag{5.8}$$

Then a low-pass filter suppresses the high frequency term. Finally, the message signal, $m(t)$, is extracted by applying the square root and by dropping the DC terms.

In the following experiments, a 1kHz carrier signal has been used. As $m(t)$, a 50Hz square and a 50Hz triangular wave have been used. The low-pass filter has been implemented with the lowpass MATLAB function, with a Finite Impulse Response (FIR) filter, of order 48, and a pass-band frequency of 150 Hz.

All the investigated approximate squarers have been used in the demodulation experiments. Table 5.7 reports the root mean square error (RMSE) of the demodulated square and triangular signals, for all the considered designs. In Fig. 5.8 and 5.9, the y-axis reports the power

dissipation of the considered squarer circuits. The proposed squarers perform very well in both applications providing a range of possibilities to trade power versus accuracy of the results. The most accurate proposals have very low RMSE values. So, for the triangular wave case shown in Fig. 5.8, they are not represented in scale for visualization purposes. The circuits considered as a refence do not overcome the performances of the proposed squarers except for [Pet14] that, performs well for the triangular waveform but less well for the square waveform. Please note that [Pet14] belongs to the category of truncated circuits that have a large mean relative error (see Table 5.6) and tend to exhibit a somewhat inconsistent behaviour in several applications.

Table 5.7: Demodulation with 8-bit Approximate Squarers.

8-Bit Design		**Power [µW]**	**RMSE**	
			Square [%]	**Triangular [%]**
	Exact	52.83	-	-
Proposed	**S8_1**	45.15	0.00	0.20
	S8_2	43.99	0.00	0.23
	S8_3	35.57	0.91	3.63
	S8_4	33.03	0.91	8.87
	S8_1_MIX	40.05	0.00	1.40
	S8_2_MIX	28.18	1.07	4.81
	S8_3_MIX	26.98	1.07	9.51
	S8_1_OR	17.52	3.16	14.97
	S8_2_OR	15.42	6.65	7.61
[Avr14]	**BB**	15.80	42.00	44.42
	BB1ERC	50.62	5.71	9.07
[She02]	**fast_comp**	20.40	8.00	18.73
[Lan06]	**approx_1**	20.34	9.80	25.93
	approx_2	21.16	6.80	17.29
[Ans22]	**LESF**	54.47	26.08	13.18
	LESF_t	55.15	45.96	18.87
[Pet14]	**Gar**	37.93	0.49	0.17

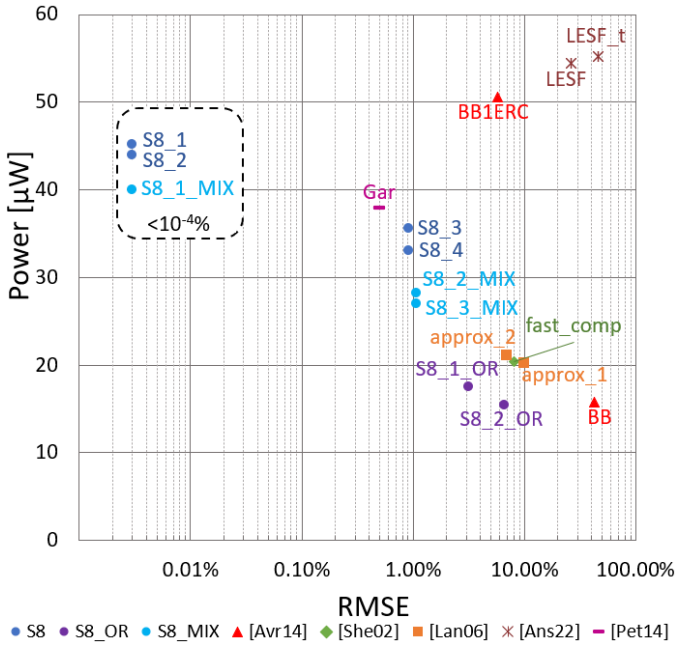

Fig 5.8: RMSE vs Power for a square-wave signal.

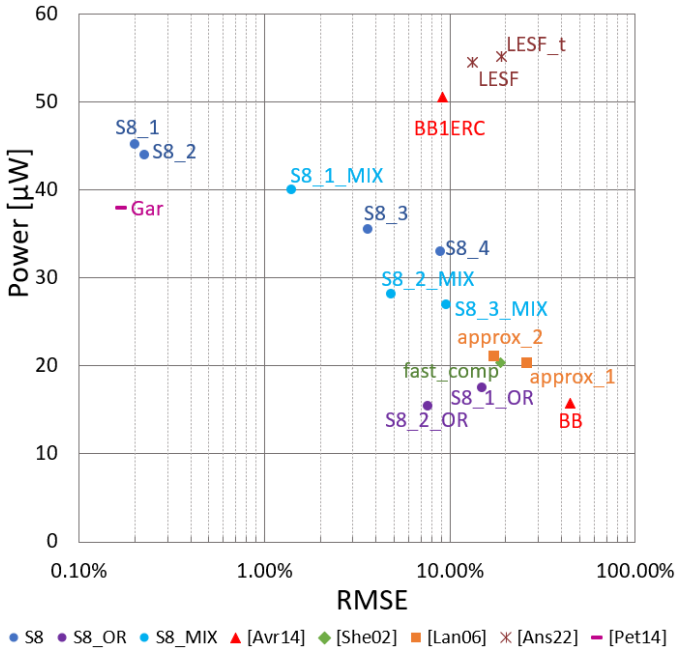

Fig 5.9: RMSE vs Power for a triangular-wave signal.

5.4.2 Image Energy

Image energy may be useful in applications like segmentation, image fusion, and classification [Avr14]. These applications (or other applications that require a quality distance measurement), often require the calculation of the Root Mean Square (RMS) (or of the energy) of the image.

Assuming a gray-scale image with a total number of N pixels, the RMS value is calculated as:

$$RMS = \frac{1}{N}\sqrt{\sum_{i,j} p^2(i,j)} \tag{5.9}$$

where $p(i,j)$ is the value of the pixel i,j. Obviously, a great number of squaring operations (as many as the image pixels) is required to obtain the RMS value.

In the following experiment, a hundred images of different file formats and dimensions, obtained from the image database DEMOS of MATLAB have been considered for the evaluation of the RMS. Each squaring circuit, including the exact squarer, has been used to calculate the RMS of all images, following (5.9). After that, the Average Root Mean Square (ARMS) has been calculated as the average of all the RMS values computed by each squarer:

$$ARMS = \frac{1}{M}\sum_{k=1}^{M} RMS \tag{10}$$

where M is the number of the considered images.

Table 5.8 shows the ARMS value obtained by the various squarers, and the error percentage compared to the exact squarer. The power consumption of each squarer is also reported in this table. It is easy to see that the proposed circuits offer the ARMS values closest to the one obtained by the exact squarer. The two designs that exhibit comparable accuracy, BB1ERC [Avr14] and LESF [Ans22], dissipate significantly more power, while [Pet14] is placed in between S8_1MIX and S8_3 both in terms of power and accuracy.

Table 5.8: ARMS for 8-bit Approximate Squarers.

8-Bit Design		Power [µW]	ARMS	
			Absolute	[%]
	Exact	52.83	117.91	-
Proposed	**S8_1**	45.15	117.90	0.01
	S8_2	43.99	117.89	0.02
	S8_3	35.57	117.57	0.29
	S8_4	33.03	117.38	0.45
	S8_1_MIX	40.05	117.77	0.12
	S8_2_MIX	28.18	117.41	0.42
	S8_3_MIX	26.98	117.22	0.59
	S8_1_OR	17.52	116.15	1.49
	S8_2_OR	15.42	115.11	2.37
[Avr14]	**BB**	15.80	110.25	6.50
	BB1ERC	50.62	116.84	0.91
[She02]	**fast_comp**	20.40	115.03	2.44
[Lan06]	**approx_1**	20.34	114.36	3.01
	approx_2	21.16	115.57	1.98
[Ans22]	**LESF**	54.47	117.27	0.54
	LESF_t	55.15	113.63	3.63
[Pet14]	**Gar**	37.93	118.10	0.16

5.5 Summary

In this work, several 8-bit approximate squarers, made up by exact or approximate novel 4-bit squarers and multipliers, have been proposed. The two novel 4-bit approximate squaring circuits are obtained by simplifying the expressions in the folded partial product matrix, which is shown in Fig. 5.1, while the 4-bit multipliers are developed in chapter 4. These elementary building blocks (4-bit squarers and multipliers) can be used recursively to scale up to $4n$-bit squarers, with $n \in \mathbb{N}$. Simple approximate accumulators are also considered to add the obtained sub-products.

The circuits developed in this paper, as well as state-of-the-art designs gathered from the literature, have been synthesized using a commercial 14nm FinFET standard cell library. Syntheses and error analyses demonstrate that the proposed 8-bit approximate squarers cover a wide range of precision and power requirements, thus providing plenty of options to the user. At the same time, the proposed architectures effectively populate the Pareto front, outperforming the state-of-the-art in terms of power vs. precision, as shown in figures 5.6 and 5.7.

Compared to the exact 8-bit squarer, the least dissipative proposed design, S8_2_OR, reduces silicon area by 76%, power consumption by 71%, and critical delay by 72%. The same circuit dissipates 2.4% less power than the least dissipative design found in literature, while providing 34% more accurate results. As another example, one of the proposed architectures, S8_3_MIX, allows to obtain 49% power reduction and 38% speed increase (compared to the exact squarer) with the low mean relative error of 2.2%.

The considered designs have been tested in signal demodulation, and RMS calculation applications. The results confirm that the proposed circuits are able to fruitfully populate the design space, as demonstrated in tables 5.7 and 5.8 and in figures 5.8 and 5.9.

Chapter 6

Spike Detection in Brainwaves

6.1 Introduction

6.1.1 General Background

Research on Brain-Machine Interfaces (BMIs) has progressed at a notable speed since the realization that devices directly controlled by ensembles of cortical neurons are viable [Cha99]. After that milestone, BMI related research has seen unyielding growth [Leb06].

There are several ways to observe the neural activity [Rap21] and some of them include electroencephalography (EEG), magnetoencephalography (MEG), electrocorticography (ECoG), functional magnetic resonance imaging (fMRI), functional near infrared spectroscopy (fNIRS), positron emission tomography (PET), and others [Sah21]. Implantable approaches to electrically contact the cortical tissue, like ECoG and penetrating microelectrode arrays (MEAs), generally yield better results with respect to non-invasive techniques, such as electrodes attached on the scalp, that have been shown to represent the activity of the surface layer of the brain. That is due to a significantly higher signal-to-noise ratio. Also, even though it is possible to observe neural activity exploiting different means of recording, like electrical, optical, and magnetic [Fra19], electrical recording is the most used.

Low-power and low-noise implantable BMIs for the observation of neural activity through an array of multiple channels, has become possible with the progress in integrated circuit and microsystem technologies. Furthermore, substantial research using animal models has provided important insight for matters like electrode types, target brain area, electronic architecture, and processing strategies.

It should be noted, that effectively monitoring brain activity with integrated implantable BMIs is still a very challenging problem. Scalability and portability, electrode stability and yield, and information transfer rate are main issues to be addressed.

The brain's electrical charge is maintained by billions of neurons. Neurons are electrically charged by membrane transport proteins that pump ions across their membranes. Ions of similar charge repel each other, and when many ions are pushed out of many neurons at the same time, they can push their neighbors in a wave, which is known as volume conduction. When the wave of ions reaches the electrodes on the scalp, they can push or pull electrons on the metal in the electrodes. Since metal conducts the push and pull of electrons easily, the difference in push or pull voltages between any two electrodes can be measured by a voltmeter. Recording the voltage fluctuations resulting from neural oscillations (brain waves), helps us determine changes in brain activity that might be useful in diagnosing brain disorders, especially epilepsy, or other seizure disorders. These readings reflect the summation of the synchronous activity of millions of neurons that have similar spatial orientation.

Progress in integrated circuits and microsystem technologies has made implanting thousands of intracortical electrodes possible, allowing researchers to investigate bigger neural ensembles. Of course, transmitting massive data wirelessly for external post-processing poses unrealistic bandwidth and power requirements [Shaer15], [Sag22], [Sagge22]. Spike detection algorithms can mitigate the problem by alleviating the need to stream the whole raw signal. Instead, the raw signal is processed on-line, the spikes are detected and possibly sorted into different categories, and only that information is transmitted off-brain [Shaer15]. After that, verification and statistical post-processing can be performed without heavy electrical constraints [Sag21].

6.1.2 Literature

An analog front-end for spike detection and sorting systems is proposed in [Bar14]. A Low Noise Amplifier (LNA) is employed to amplify the signals of the electrodes, from the sub-mV range to 10s of mVs. A bandpass filter is then used to: a) reject the local field potential (LFP) (high pass) and b) avoid aliasing (low pass). Finally, an Analog

to Digital Converter (ADC) provides the system with a digital signal. Once the spikes exceed the LFP and background noise, they can be detected using an amplitude threshold [Sag22], [Sagge22], [Bar14], [Riz09], [Nav14], [Nie16], [Reh19]. In [Sag22] and [Sagge22], an adaptive threshold is utilized, after two cascaded energy operators for spike enhancement, specifically the Absolute Differential Operator (ADO) and the Amplitude Slope Operator (ASO). A novel spike extraction method, called spike detection by differences, is proposed in [Tam21]. The power consumption and resources of this technique are independent from the total channels count.

Different proximity and orientation with respect to detecting electrodes, make neurons exhibit different spiking profiles. Suitable feature extraction can be used to perform spike sorting. One of the most common techniques followed in literature is template matching [Bar14], [Riz09], [Nav14], [Nie16], [Fre16], which consists in measuring the distance between a spike and a template as a similarity index. The authors in [Cam19] propose an iterative Bayesian approach to separate the LFP from the spiking activity, using prior information about the power spectral density of the LFP. Other methods frequently used for spike sorting are Principal Component Analysis, First and Second Derivative Features Extraction [Bar14], K-means, and Superparamagnetic Clustering. During the last years, machine learning approaches have also been considered for spike detection and classification [Reh19], [Rác20], [Reh21].

Table 6.1: Literature Review.

	Spike Detection	**Spike Sorting**	**Implementation**
[Sag22]	Absolute Differential Operator Amplitude Slope Operator Adaptive thresholding	**NO sorting**	ASIC 28nm
[Sagge22]			
[Bar14]	Non-linear Energy Operator Absolute Value Thresholding Amplitude Threshold	Template Matching Principal Component Analysis 1st/2nd Derivative Features	ASIC 0.18 µm (front-end)
[Riz09]	Amplitude Threshold	Template Matching	Chip / FPGA
[Nav14]	Amplitude Threshold	Template Matching	Chip / Software
[Nie16]	Amplitude Threshold	Template Matching	Software
[Reh19]	Amplitude Threshold	**NO sorting**	Software
[Tam21]	LFP filtering and "SDD"	**NO sorting**	Software
[Fre16]	Both detection and sorting with Template Matching		Software
[Cam19]	HF thresholding	Iterative Bayesian approach	Software
[Rác20]	Both detection and sorting wit Convolutional Neural Network		Software
[Reh21]	Both detection and sorting with K-means		Software

Some of the proposed systems in the literature provide real-time functionalities as they can detect and/or sort spikes at run time. Moreover, authors have implemented their contributions in ASIC technology, FPGAs, or developed software algorithms. Table 6.1 summarizes most of the aforementioned information.

6.1.3 Objective - Trivial Problem?

In this work, machine learning approaches are used to detect and classify as Type A, B or C the spiking activity in simulated recordings of brain activity. The trained network is described in Verilog and synthesized in a 14nm FinFET technology. The innate error factor in machine learning and the vast number of required multiplications allow the fruitful introduction of approximate multiplying circuits [Str22], [Zac22], [Esp18], to achieve significant reductions in silicon area and power dissipation. Such approximate designs are frequently used in embedded devices and error tolerant applications, aiming to improve the electrical performances by reasonably allowing imprecise results.

In order to verify that using machine learning methods to detect spikes in the synthesized data is not an overcomplicated solution for a trivial problem, an alternative way initially investigated. Using MATLAB, a "smart threshold" that adjusts itself according to the considered recording, and thus, does not work on run-time, has been developed. Allowing this approach the advantage of calibration according to a specific recording, is meant to ensure that even better results can be obtained using NNs.

In Fig. 6.1, a positive and a negative threshold have been chosen in a way that ensures no false negatives for the considered recording. In other words, all spikes (marked in red) are outside the area limited by the two thresholds and can be easily detected. However, as shown in Fig. 6.1, sometimes, also normal brain activity (marked in black) overcomes the thresholds, resulting in false positives. The accuracy achieved by this method (ensuring no false negatives) is obtained by:

$$Accuracy = 1 - \left(\frac{Number\ of\ False\ Positives}{Number\ of\ Total\ Windows}\right) \qquad 6.1$$

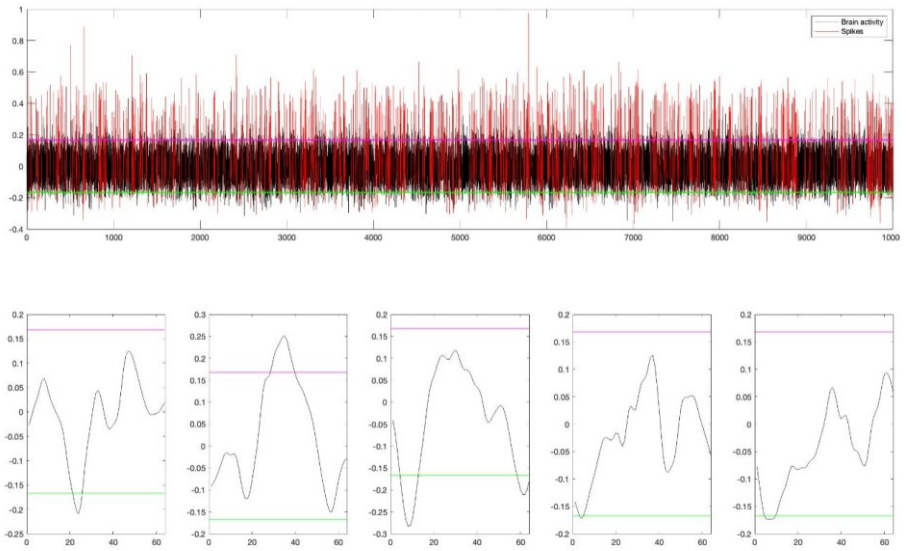

Fig. 6.1: Thresholds ensuring no False Negatives.

Another attempt to utilize thresholds has been made, in a way that returns no false positives, as shown in Fig. 6.2. In this way, the normal brain activity (black signal) never overcomes the thresholds, but certain spikes will remain undetected below the thresholds. The accuracy achieved by this method is obtained by:

$$Accuracy = 1 - \left(\frac{Number\ of\ False\ Negatives}{Number\ of\ Total\ Windows}\right) \qquad 6.2$$

Depending on the level of noise embedded in the considered dataset, both methods may offer accuracies ranging from 50% to almost 100%. The high accuracy values when there is no noise are anticipated, as the spikes are clearly distinguished from normal brain activity. However, a machine learning approach can be used to produce reliable results also in more realistic scenarios, and even when the spikes are buried beneath the noise. And in any case, the classification problem remains unsolved when thresholds are employed, as the various types of spikes do not differ in amplitude. As already mentioned, there are various methods in the literature for spike detection, like template matching [Jia22], [Hao21] or adaptive thresholds with differential amplitude slope operators [Sag22].

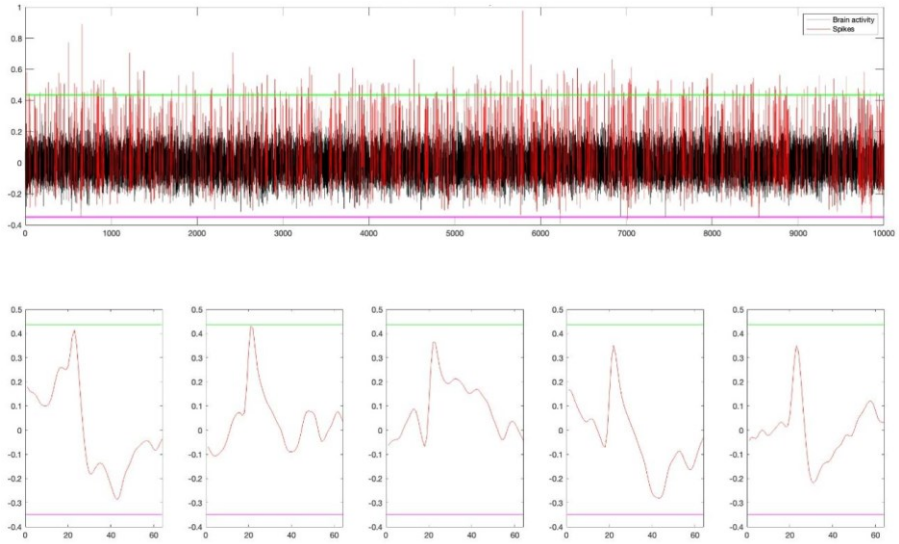

Fig. 6.2: Thresholds ensuring no False Negatives.

6.2 Dataset - NeuroCube

NeuroCube, presented in [Cam13], is a freely available and commonly used [Sag22], [Sag21], [Bar14], [Nav14], [Fre16], [Cam19], method to generate realistic simulations of extra-cellular recordings. The simulations are obtained by superimposing the activity of neurons randomly placed in a cube of brain tissue recorded by a finite-size electrode. The obtained recordings have varying types and amount of noise and consist of labelled data that provide information relating to abnormal brain activity (from here on spikes). NeuroCube has been used in this work, including the analysis performed in section 6.1.3.

The simulated data correspond to continuous samples. In the original dataset, obtained by Neurocube, each 64 continuous samples constitute a “window”. Windows containing a spike, are assigned information about the existence and type of spike. Specifically, this information is assigned to the first, out of the 64 samples that contain a spike. Using Neurocube, various datasets can be generated, containing simulated brain activity with different levels of noise. The spikes shown in Fig. 6.3 are taken from a dataset with low-level noise, and amplitude thresholding may be a viable option in those high-SNR waves.

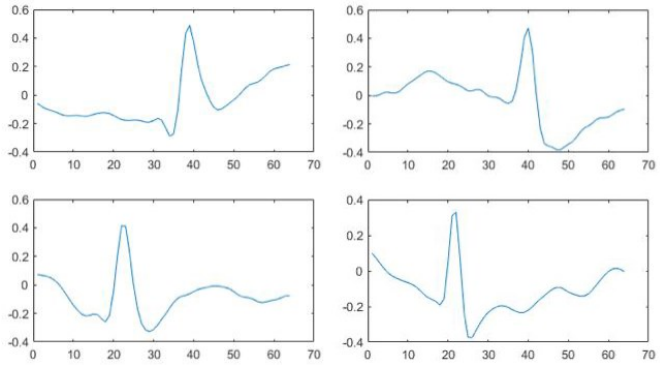

Fig. 6.3: Low-Noise Spikes in synthesized brain activity.

However, in more realistic recordings, such as the ones used during training, spikes often require a less trivial solution to be detected. The used sub-dataset contains approximately 1.5 million samples and is characterized by a quite low SNR, ensuring a more realistic behavior for the trained Neural Network.

The energy of the spikes is contained in the middle of the corresponding windows. So, bearing in mind that the final goal is a low power integrated network, smaller windows were considered in this work, namely windows of 32 samples. Thus, as shown in Fig. 6.4, the aim is to create a NN with 32 input neurons to receive the incoming window, and 4 output neurons. The first one turns “high” to indicate that there is no spike. In the event of a spike however, one of the remaining three neurons turns high to indicate the type of spike. As mentioned in paragraph 6.1.3, NeuroCube defines three different types of spikes, and the windows are classified accordingly.

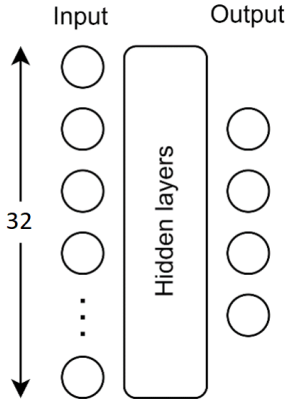

Fig. 6.4: Spike Detection Architecture.

Since the energy of the spikes is concentrated in the middle samples, apart from opting for 32-sampled windows instead of 64-sampled, shifting the spikes left and right has been used for data augmentation. Specifically, as shown in Figure 6.5, six preceding, and six succeeding windows have been considered as spikes, thus generating (and adding to the dataset) 2×6 more versions of the same spike to help train the networks.

On the other hand, spikes that are almost out of the processing window, should be classified as normal brain activity. To ensure the correct operation of the network, spikes that are almost out of the processing window, "half-spikes", were introduced to the training dataset labelled as "non-spikes". Inserting these confusing windows in the training set, made the investigated networks exhibit a more robust behavior.

Throughout the experimental phase, the chosen training datasets have been processed accordingly to ensure a good balance. Namely, the training data is made up by 25% normal brain activity, 25% spikes of "Type-A", 25% spikes of "Type-B", and 25% spikes of "Type-C". Furthermore, the modified datasets have been divided into two sets. 75% of the windows were used for training and 25% for validation.

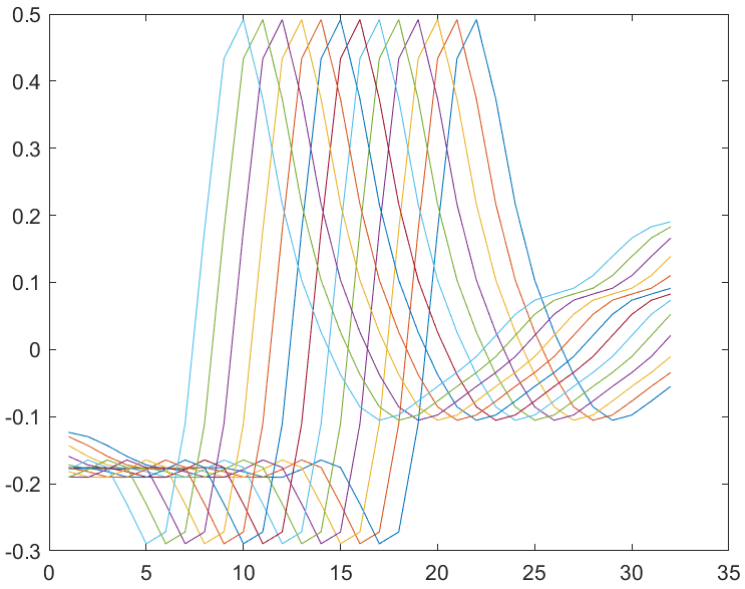

Fig. 6.5: Spike Replication for Data Augmentation.

6.3 NN Architectures

TensorFlow has been used to train and validate the NNs. The considered models have been trained through 50 epochs, with a learning rate equal to 0.001, and a batch size of 32. For the fully connected layers, dropout has been considered to reduce interdependent learning among neurons and avoid over-fitting. The chosen activation function is the Rectified Linear Unit (ReLU) which proved efficient and easy to model in hardware. As a loss function, the "categorical crossentropy" has been used, which is ideal for multi-class classification models where there are two or more output labels. A SoftMax layer has been used as an output layer, to convert a vector of 4 real numbers into a probability distribution of the 4 possible outcomes.

As already mentioned, the aim of this work is to embed the trained NN into an ASIC. Thus, a model of reduced size, with parameters that have a limited range, is desired. Parameters that are tightly concentrated around a specific value can result in a better and more "concise" binary representation, that will in turn, result in a smaller quantization error, which will be introduced by the hardware implementation of the NN. So, a regularization layer has been added during training: more specifically, L2 regularization has been used in TensorFlow by adding "*tensorflow.keras.regularizers.l2*" as a kernel regularizer in the model. L2 regularization forces parameters to be smaller but not exactly 0, by adding the "squared magnitude" of the coefficient as a penalty term to the loss function. The effect of regularization is shown in Fig. 6.6, where it is demonstrated that the distribution of parameters after regularization, became significantly more concentrated around 0.

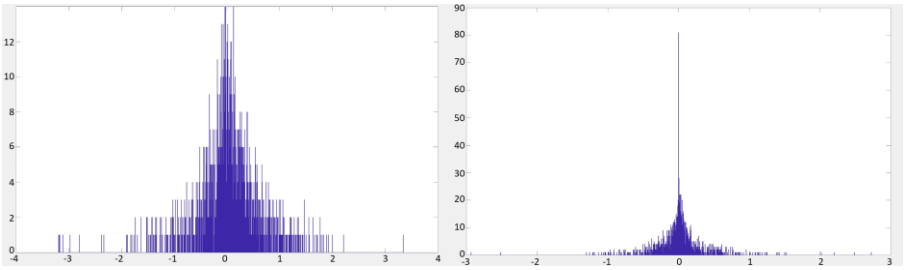

Fig. 6.6: Parameters distribution before (left) and after (right) regularization.

Table 6.2: Training accuracy, validation accuracy, and number of parameters for different NN architectures.

Neurons in				Training Accuracy [%]	Validation Accuracy [%]	Number of Parameters
Input Layer	Layer 1	Layer 2	Output Layer			
32	16	-	4	88	95	596
32	20	-	4	91	96	744
32	24	-	4	92	97	892
32	24	8	4	92	98	1028
32	24	16	4	94	98	1260

Table 6.2 shows some of the architectures that have been tested during the experimental phase. As expected, accuracy grows with the number of trainable parameters. The architecture chosen to be implemented in an ASIC, is the last one, that features two hidden layers, with 24 and 16 neurons respectively. The validation accuracy is obtained on a set of data that is not used during training, to effectively test the generalization ability of the model. The lower training accuracy can be attributed to the use of "dropout".

In Fig. 6.7 the training and validation accuracies, over the training epochs are shown, while in Fig. 6.8, a heat map is provided to visualize the performance of the network for each type of spike. After the training period, the network has achieved an excellent ability to predict the type of input signal. As it can be observed from the increasing tendency of the validation accuracy, after 50 epochs of training, the generalization ability of the model is still intact, showing no signs of over-fitting.

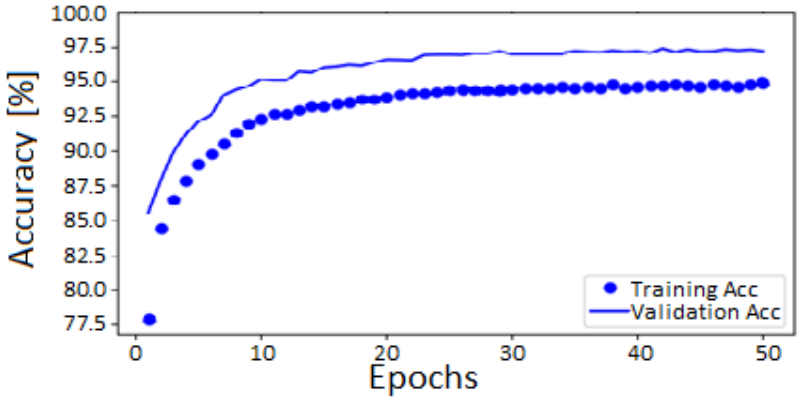

Fig. 6.7: Training and validation accuracies.

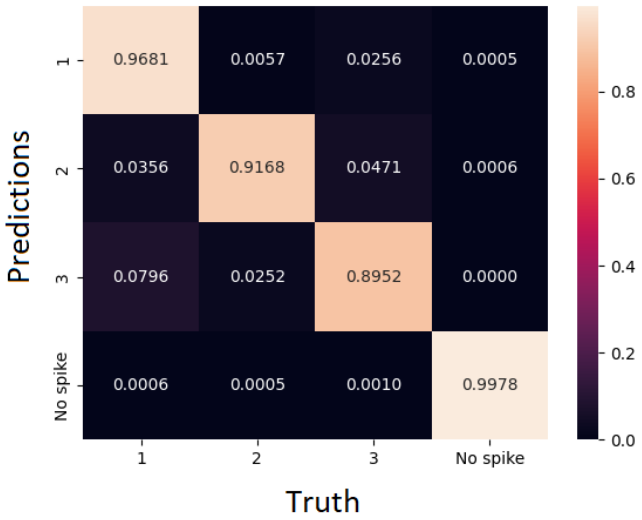

Fig. 6.8: Training and validation accuracies.

6.4 Hardware Implementation

6.4.1 Proposed Architecture

After training the network, with appropriate regularization to keep the values of the trainable parameters well concentrated around 0, the next step has been the implementation of the network in hardware. To that end, the architecture of the fully connected and softmax layers has been described in HDL.

The investigated network is shown in Fig. 6.9. It has 32 neurons in the input layer to process the 32-sampled windows, 24 and 16 neurons in the two hidden layers, and 4 neurons in the output for the classification task. Each neuron is responsible for adding the corresponding bias and products, before moving this sum to the activation function (in this case ReLU). For example, the output of neuron 1 from hidden layer 1 (the signal before the activation function), is the neuron's bias plus all the 32 products (weight times corresponding input), as shown in:

$$y = bias + \sum_{i=0}^{31} (weight_i \times input_i) \qquad 6.3$$

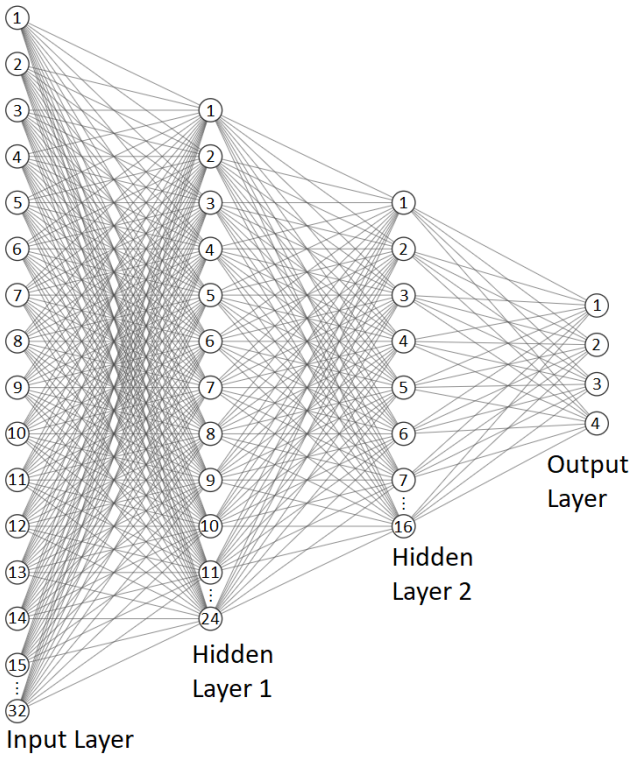

Fig. 6.9: Architecture of the proposed dense network.

As shown in eq. 6.3, the first neuron in the first hidden layer, processes the value of each input neuron. In a realistic scenario, the last sample of a given waveform (the one generated first timewise), is fed to the system first. The next sample follows until the waveform is completely processed by the system. Therefore, in the hardware implementation, the 32-sampled window is read one sample at a time and so, there is no need for a lot of parallel multiplications. Equation 6.3. is unfolded, and one addition is performed every clock cycle, as the products get accumulated. This results in a hardware effective implementation.

On the other hand, an increase in the critical delay is inevitable, since for synchronization purposes, each layer takes 32 clock cycles to produce an output. Fortunately, the achievable ASIC speeds are very high with respect to the frequencies of brainwaves. Thus, capturing one window every 32 samples, is considered enough to effectively measure brain activity.

Fig. 6.10 shows the implementation of an indicative hidden layer with 2 neurons. A universal 5-bit counter (that is able to count from 0 up to 31), informs the neurons as to which sample is processed at a given clock cycle, by controlling a multiplexer that provides the appropriate neuron with the corresponding weight. The chosen weight and the incoming sample are multiplied to generate the product, *pr*.

$$pr_i = weight_i \times input_i \quad (6.4)$$

The accumulation of the products is performed by adding a feedback signal to the current product:

$$pr\_acc_i = pr_i + feedback_i \quad (6.5)$$

The accumulated product, pr_acc_i, is stored in a register, and in the next clock cycle, it appears as $pr\_acc\_pl_i$:

$$pr\_acc\_pl_i = pr\_acc_{i-1} \quad (6.6)$$

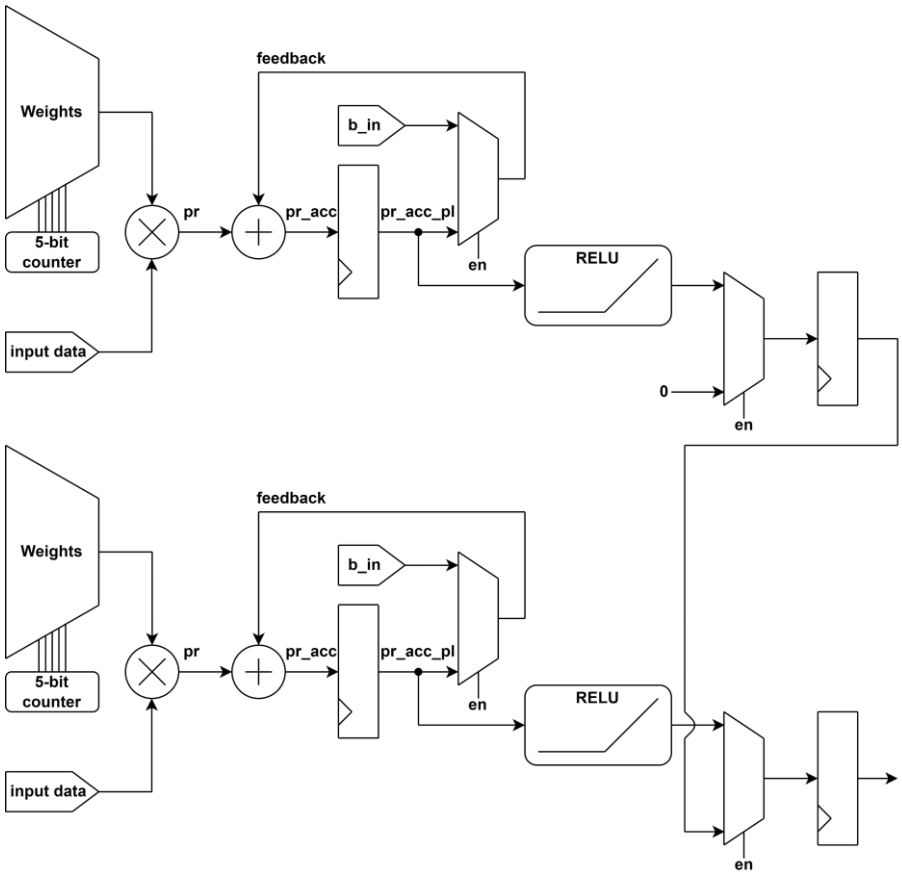

Fig. 6.10: Hardware implementation of a 2-neurons hidden layer.

This stored signal, that represents the previous accumulated products, is generally added to the current product. There is however one exception. When the first sample is fed to the neuron, the counter signal is "00000", which triggers the enable signal, *en*:

$$en = \overline{cntr[1]} \cdot \overline{cntr[2]} \cdot \overline{cntr[3]} \cdot \overline{cntr[4]} \cdot \overline{cntr[5]} \quad 6.7$$

Thus, instead of adding to the first product, the previous irrelevant value that remained stored in the register, a multiplexer chooses to add the neuron's bias. The multiplexer controls the *feedback*'s value, as shown in the following equation:

$$feedback_i = \begin{cases} bias & if\ en = 1 \\ pr\_acc\_pl_i & if\ en = 0 \end{cases} \quad 6.8$$

The accumulated products, that are stored in the register, are constantly fed to the activation function, which in this case is a simple ReLU. The implementation of ReLU is quite simple, as it basically zeros a negative argument, while leaving unaffected a positive one:

$$ReLU_i = \begin{cases} 0 & if\ pr\_acc\_pl_i < 0 \\ pr\_acc\_pl_i & if\ pr\_acc\_pl_i \geq 0 \end{cases} \quad 6.9$$

The enable signal, *en*, is used one more time to help determine the output of the neuron. As already mentioned, when the window is being processed, *en* is low. During that time, a second multiplexer sets the neuron's output to zero. However, when the first sample of a window, is being processed, *en* turns high. Apparently, at the same time the previous window is fully processed and ready to move to the next layer. So, as already mentioned, *en* turns high, and a shift register is utilized to feed the layer's output to the next layer, one sample at a time.

The last layer (after the 4-neurons output layer) is a custom hardmax layer. Unlike the softmax layer, it employs a "hard" modifier, as it sets to *high* the largest vector and to *low* all the rest.

6.4.2 Binary Signals Fixed-Width Implementation

The HDL is fully parameterized, thus enabling effortless modifications. The representation of the binary vectors is a critical issue, as it has a direct impact on the accuracy drop with respect to the accuracy obtained by the original model (98%). Choosing a short bit-width for the fractional part of the signals increases the quantization error, as the floating-point parameters are heavily truncated. On the

other hand, opting for longer signals results in heavy hardware requirements.

It has been shown that after the regularization, the integer part of all the obtained parameters can be described by three bits, without the risk of overflow. However, after exhaustive simulations, intermediate signals exceeding the range offered by three bits in the integer part, [-4,4), have been observed. Apart from aggressively adding one more bit at the left part of the radix, to tackle the occasional overflow problem, a more conservative solution has also been investigated. An overflow-detection and saturation module has been employed to single out the vectors that exceed the two limits. Once detected, these vectors are rounded to -4 and 4 (with the highest available precision) respectively.

Three different binary representations are considered in this work. The first one uses overflow detection and saturation when needed, as it has 3 bits before the radix point and 5 after. The second and third versions have 4 bits for the integer part to avoid overflowing signals, while for the fractional part, they use 6 and 8 bits respectively. Clearly, the longer the bit-width, the higher is the achievable accuracy, as the saturation and quantization errors are completely or partially overcome.

6.4.3 Approximate Computing and Performances

Multiplying circuits consume a considerable amount of power and chip area. Approximate computing provides a good compromise between precision and hardware resources for error resilient and mobile applications. Aiming at reducing the hardware requirements, the methodologies from [Str22], [Zac22], and [Esp18] have been applied to the multipliers for the NN. The best designs are presented in Table 6.3 and Fig. 6.11, while their configurations are shown in Table 6.4.

Designs 3.5_Ex, 4.6_Ex, and 4.8_Ex employ exact multipliers. Designs 4.8_ssm-m8 and 4.8_ssm-m10 use approximate signed 12×12 multipliers proposed in [Str22] with a segment of 8 and 10 bits respectively. The remaining designs use small 4×4 multiplying building blocks proposed in [Zac22], to recursively generate the 8×8, 10×10, and 12×12 required multipliers. For the 10×10 multipliers, 4:2 approximate compressors proposed in [Esp18] are also used.

Table 6.3: Absolute accuracies and electrical performances for circuits using different multipliers.

Multiplier Design		Accuracy [%]	Power [µW]	Area [µm²]	Min Delay [ps]
8bits	**3.5_Ex**	89.6	121.80	3969	665
	3.5_1	85.9	121.76	2987	599
10bits	**4.6_Ex**	95.5	153.80	5602	678
	4.6_1	91.3	153.70	4520	620
	4.6_2	87.0	153.67	3915	583
	4.6_3	85.2	153.66	3849	582
12bits	**4.8_Ex**	97.0	185.88	8964	745
	4.8_1	96.9	185.73	7638	725
	4.8_2	95.6	185.64	5912	647
	4.8_3	94.9	185.62	5534	642
	4.8_4	94.6	185.62	5475	641
	4.8_ssm-m8	96.2	185.68	6735	692
	4.8_ssm-m10	96.5	185.73	7490	716

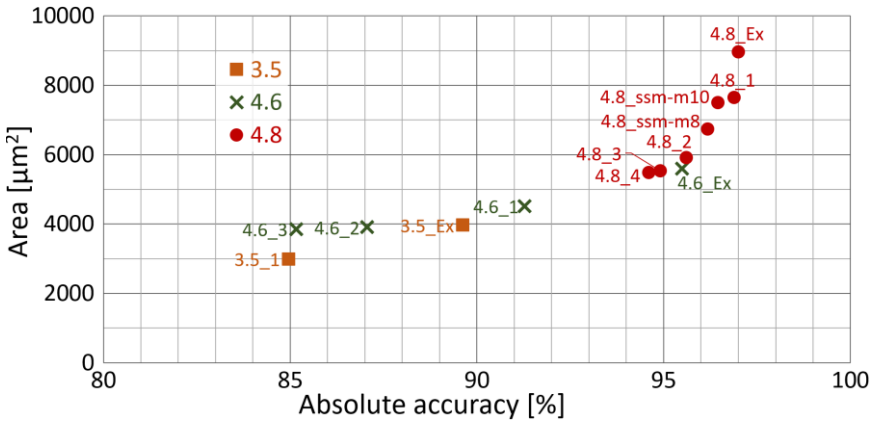

Fig. 6.11: Absolute Accuracy with respect to silicon area.

As shown in Fig. 6.11 and Table 6.3, the more bits used for signal representation, the higher is the achievable accuracy, due to the elimination of the overflow/saturation error and the moderation of the quantization error. Of course, power and area requirements increase as well. All circuits are synthesized for a 10ns timing constraint and simulated with the same set of 90.000 windows for the calculation of the power. It should be noted, that in this work the power is reported for

Table 6.4: Multiplier configurations for the best performing circuits.

Design		Partial Product Matrix part				
		High	**Mid High**	**Mid**	**Mid Low**	**Low**
8bits	**3.5_Ex**	Exact				
	3.5_1	Custom Exact		N2		Custom N1
10bits	**4.6_Ex**	Exact				
	4.6_1	Custom Exact	Exact		Exact / 4:2 AC	Not formed
	4.6_2	Custom Exact	N1		N1 / 4:2 AC	Not formed
	4.6_3	Custom Exact	N2		N2 / 4:2 AC	Not formed
12bits	**4.8_Ex**	Exact				
	4.8_1	Custom Exact	Exact	Exact	Exact	Not formed
	4.8_2	Custom Exact	Exact	N1	Or-Based	Not formed
	4.8_3	Custom Exact	N1	N2	Or-Based	Not formed
	4.8_4	Custom Exact	N2	Or-Based	Or-Based	Not formed
	4.8_ssm-m8	12×12 SSM, m=8				
	4.8_ssm-m10	12×12 SSM, m=10				

a single channel that receives a new window every 6 input samples, and a sample rate of 5kHz. Minimum delay refers to the minimum timing constraint at which the circuit can be synthesized with a non-negative slack. The use of approximate computing results in significant silicon area savings while power is mostly determined by the register and hence by the bit width of the signals.

The proposed approach offers a variety of design choices that favorably compare against the state of the art. As an example, design 4.8_2 achieves 95.6% accuracy and reduces silicon area by 34% with respect to 4.8_Ex which is the most accurate, offering only 1.4% more accuracy. The proposed circuits are more power hungry than [Sag22] and [Sagge22] for example, but they provide both detection and sorting in a single integrated solution.

6.5 Summary

Observing brain activity through low-power implantable BMIs is not only possible nowadays, but these techniques provide also cleaner results with respect to non-invasive methods, in terms of SNR. Spikes in brainwaves may indicate seizure activity in patients with a predisposition toward epilepsy. Thus, implantable spike detectors can

be employed to extract and transmit only the valuable neural information needed, instead of the whole raw recording.

Machine Learning approaches have been employed, to detect and categorize spikes in simulated brain activity. The network training and testing data have been provided by NeuroCube, a tool presented in [Cam13] that generates synthetic, yet realistic and labelled data with different levels and types of noise. The same data has been checked with a simpler threshold approach, exhibiting non-satisfying results in high noise recordings.

The chosen network model has been developed for implementation in an integrated circuit. Thus, a small architecture with parameters that have a limited range, have been critical points in this work. To that end, smaller 32-sampled input windows have been generated, and parameters regularization has been performed.

A balanced dataset while training the network is essential. Therefore, a dataset made up by 25% normal brain activity, 25% spikes of “Type-A”, 25% spikes of “Type-B”, and 25% spikes of “Type-C”, has been used. New spikes have been generated for data augmentation, by moderately shifting the spike-windows. Finally, the whole dataset has been divided into two sets: 75% for training and 25% for validation.

The considered networks have been trained using TensorFlow. As an activation function, the ReLU has been proved to be efficient and at the same time, easy to implement in hardware. The “categorical crossentropy” has been used as a loss function for this multi-label classification task.

As shown in Fig. 6.9, the chosen model has 32 neurons in the input layer to process the 32 sampled windows, two hidden layers with 24 and 16 neurons respectively, and an output layer of 4 neurons, one for each output label. That results in 1260 trainable parameters and an accuracy equal to 98%.

In a real application, the data would arrive to the model sequentially. Therefore, in the hardware implementation of the network, each neuron is -simply put- a “multiply and accumulate” unit (MAC). In each clock cycle, an input is multiplied by the corresponding weight and all these products are accumulated. At the last clock cycle, the accumulated products pass through the activation function to a shift register, that is responsible of providing the next layer with one sample at a time.

Choosing the most efficient data representation has been the next challenge. It has been demonstrated, that after regularization, using 4 bits for the integer part of all vectors, is enough to avoid overflowing numbers. Furthermore, as shown in table 6.3, using 8 bits for the fractional part and exact multipliers, results in a very small (1.5%) drop in accuracy, due to the inevitable quantization error. Approximate multipliers manage to effectively mitigate the hardware burden and a variety of circuits offering different precision-hardware cost tradeoff have been presented. The different versions of the circuit reach an accuracy range from 85% to 97%, while occupying 3000μm^2 to 9000μm^2 respectively.

Conclusions and future work

The scope of this dissertation is the development of approximate arithmetic circuits. A neural network aiming to detect and categorize unusual behavior in synthesized brain activity is also presented. The network has been described in hardware language in order to be integrated in an ASIC. The innate error factor in machine learning and the vast number of required multiplications in a NN, make introducing approximate computing to this application, a viable option. In this way, a significant reduction in silicon area and power dissipation can be achieved. Producing minimally invasive and long-lasting implantable devices are of course major concerns.

Two main categories of approximate binary multipliers are highlighted: multipliers using approximate compressors and multipliers using recursive architectures with smaller approximate modules.

Multipliers that use approximate compressors, act on the PPM reduction phase, and aim to approximately encode the information in less bits. Specifically, the compressors presented in chapter 3, encode the information of n bits of the same PPM column, in $m = \lceil n/2 \rceil$ output bits, again in the same column. For that reason, they are called single-weight approximate compressors (SWACs). Two types of compressors are presented: one aiming at the minimization of hardware resources while introducing reasonable error, and another type that introduces the minimum error possible, while trying to keep hardware resources as low as possible. All circuits avoid the use of bulky and slow XOR gates. The sizes of the developed compressors are 3:2, 4:2, 5:3, and 6:3. Compressors of higher sizes are composed of smaller designs. The presented compressors are used to form approximate multipliers of various sizes, as dictated by the proposed compressor allocation strategy. The proposed multipliers exhibit competitive error vs hardware trade-off when synthesized in a 14 nm FinFET technology and when tested in an error resilient application.

Recursive multipliers use small elementary approximate multiplier blocks (2×2 or 4×4), suitably assembled to design larger multipliers. In this work, five approximate 4×4 multipliers, with

different error vs hardware trade-off, are proposed. The smaller blocks are obtained by a systematic simplification (or even truncation) of the sum and carry terms, while attempting to partially compensate for the introduced error, at the same time. As always, XOR gates are avoided as much as possible. The proposed designs, an exact, and an OR-based multiplier are used recursively scale up to 8×8, 16×16 and 32×32 approximate multipliers. The proposed multipliers and competitive circuits found in the literature, have been synthesized using a commercial 14nm FinFET standard cell library. The proposed designs exhibit a great tradeoff between power reduction and precision, as is also confirmed by image filtering applications and a pre-trained convolutional neural network.

Binary squaring is often needed in digital signal processing. Although it is nothing more than a special multiplication case, it is demonstrated that the partial product matrix of a number multiplied by itself, has inherent symmetries that can be exploited. The folded PPM is considerably simpler, thus justifying the development of dedicated squaring designs. Approximate 8-bit binary squarers, able to outperform the state of the art, are proposed. They are obtained by recursively exploiting 4-bit approximate multipliers and squarers. To that end, two novel 4-bit approximate squaring circuits, obtained by simplifying the expressions in the folded partial product matrix, are developed. The electrical characteristics of the synthesized circuits, as well as their error behavior, demonstrate that the proposed designs overcome the state of the art in terms of power vs. precision. The two considered applications, signal demodulation, and RMS calculation, confirm these results.

Approximate computing can be effectively used in machine learning applications. The innate probabilistic nature of ML, and the huge number of multiplications in dense networks, make approximate multipliers appropriate for the inference phase of a trained network. In this way, a significant computational burden can be eliminated.

A dense network has been developed to detect and categorize spikes in simulated brain activity. The aim of this work is to implement the model in an ASIC and therefore, constraints like low number of trainable parameters and well concentrated parameters have been taken under consideration during the experimental phase. The final architecture has 32 neurons in the input layer to receive the 32-sampled

input windows, 24 and 16 neurons in two hidden layers, and 4 output neurons, each of which corresponds to one of the four states: “Type-A spike”, “Type-B spike”, “Type-C spike”, and “No spike”. After training in TensorFlow the 1260 available parameters, the obtained accuracy is 98%. The model’s architecture has been described in hardware language (Verilog). It has been demonstrated, that using 12 bits to represent all the vectors in the circuit (4 before the radix point and 8 after), results in no overflowing numbers and a small quantization error, that causes accuracy to drop from 98% to 97%.

Results demonstrate that approximate multipliers can be effectively used instead of exact multipliers, without drastically threatening the circuit’s prediction ability. As an example, design 4.8_2 achieves 34% reduction in silicon area, while sacrificing accuracy by 1.4% with respect to 4.8_Ex, which uses exact multipliers.

Future tasks include further testing of approximate designs, to better investigate the sweet spot of power-area saving vs accuracy drop. Moreover, quantization aware training techniques are intended for investigation, hoping to achieve higher accuracies for lower bit-width representations.

Literature

[Car12] S. Carrara, S.s. Ghoreishizadeh, J. Olivo, I. Taurino, C. Baj-Rossi, A. Cavallini, M. O. Beeck, C. Dehollain, W. Burleson, F. Moussy, A. Guiseppi-Elie, and G. Micheli, "Fully Integrated Biochip Platforms for Advanced Healthcare," Sensors (Basel, Switzerland), vol. 12, Dec 2012, doi: 10.3390/s120811013.

[Baz12] K. Bazaka and M. Jacob, "Implantable Devices: Issues and Challenges," Electronics. vol. 2, pp. 1-34, Mar 2012, doi: 10.3390/electronics2010001.

[Han13] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," 18th IEEE European Test Symposium (ETS), pp. 1-6, May 2013, doi: 10.1109/ETS.2013.6569370.

[Liu20] W. Liu, F. Lombardi and M. Schulte, "Approximate Computing: From Circuits to Applications [Scanning the Issue]," in Proceedings of the IEEE, vol. 108, no. 12, pp. 2103-2107, Dec. 2020, doi: 10.1109/JPROC.2020.3033361.

[Gup11] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan and K. Roy, "IMPACT: IMPrecise adders for low-power approximate computing," IEEE/ACM International Symposium on Low Power Electronics and Design, Fukuoka, Japan, pp. 409-414, 2011, doi: 10.1109/ISLPED.2011.5993675.

[Seo20] H. Seo, Y.S. Yang and Y. Kim, "Design and Analysis of an Approximate Adder with Hybrid Error Reduction," Electronics. vol. 9, no. 471, 2020, doi: https://doi.org/10.3390/electronics9030471.

[Seo21] H. Seo, J. Lee, H. Seok and Y. Kim, "Design of an Accuracy Enhanced Imprecise Adder with Half Adder-based Approximation," 18th International SoC Design Conference (ISOCC), Jeju Island, Korea, pp. 153-154, 2021, doi: 10.1109/ISOCC53507.2021.9613888.

[Ram19] M. Ramasamy, G. Narmadha and S. Deivasigamani, "Carry based approximate full adder for low power approximate computing," 7th International Conference on Smart Computing & Communications (ICSCC), Sarawak, Malaysia, pp. 1-4, 2019, doi: 10.1109/ICSCC.2019.8843644.

[Soa19] L. B. Soares, M. M. A. da Rosa, C. M. Diniz, E. A. C. da Costa and S. Bampi, "Design Methodology to Explore Hybrid Approximate Adders for Energy-Efficient Image and Video Processing Accelerators," in IEEE Trans. on Circuits and Systems I: Regular Papers, vol. 66, no. 6, pp. 2137-2150, June 2019, doi: 10.1109/TCSI.2019.2892588.

[Gup13] V. Gupta, D. Mohapatra, A. Raghunathan and K. Roy, "Low-Power Digital Signal Processing Using Approximate Adders," in IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 32, no. 1, pp. 124-137, Jan. 2013, doi: 10.1109/TCAD.2012.2217962.

[Che18] L. Chen, J. Han, W. Liu, P. Montuschi and F. Lombardi, "Design, Evaluation and Application of Approximate High-Radix Dividers," in IEEE Trans. on Multi-Scale Computing Systems, vol. 4, no. 3, pp. 299-312, 2018, doi: 10.1109/TMSCS.2018.2817608.

[Hor14] M. Horowitz, "Computing's energy problem (and what we can do about it)", Proc. IEEE Int. Solid-State Circuits Conf., pp. 10-14, Feb. 2014.

[Jia20] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu and J. Han, "Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications," in Proc. of the IEEE, vol. 108, no. 12, pp. 2108-2135, Dec. 2020, doi: 10.1109/JPROC.2020.3006451.

[Liu18] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi and F. Lombardi, "Design and Evaluation of Approximate Logarithmic Multipliers for Low Power Error-Tolerant Applications," in IEEE Trans. on Circuits and Systems I: Regular Papers, vol. 65, no. 9, pp. 2856-2868, Sept. 2018, doi: 10.1109/TCSI.2018.2792902.

[Kim19] M. S. Kim, A. A. D. Barrio, L. T. Oliveira, R. Hermida and N. Bagherzadeh, "Efficient Mitchell's Approximate Log Multipliers for Convolutional Neural Networks," in IEEE Trans. on Computers, vol. 68, no. 5, pp. 660-675, 1 May 2019, doi: 10.1109/TC.2018.2880742.

[Lot21] U. Lotrič, R. Pilipović and P. Bulić, "A Hybrid Radix-4 and Approximate Logarithmic Multiplier for Energy Efficient Image Processing," in Electronics for Low-Size Low-Power Sensors and Systems: From Custom Design to Embedded Solutions, vol. 10, no. 10, May 2021, doi: 10.3390/electronics10101175.

[Str22] A. G. M. Strollo, E. Napoli, D. De Caro, N. Petra, G. Saggese and G. Di Meo, "Approximate Multipliers Using Static Segmentation: Error Analysis and Improvements," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 69, no. 6, pp. 2449-2462, June 2022, doi: 10.1109/TCSI.2022.3152921.

[Yan18] T. Yang, T. Ukezono and T. Sato, "A low-power high-speed accuracy-controllable approximate multiplier design," 2018 23rd Asia and South Pacific Design Automation Conf. (ASP-DAC), 2018, pp. 605-610, doi: 10.1109/ASPDAC.2018.8297389.

[Češ18] M. Češka, J. Matyáš, V. Mrazek, L. Sekanina, Z. Vasicek and T. Vojnar, "ADAC: Automated Design of Approximate Circuits," in Computer Aided Verification. CAV 2018. Lecture Notes in Computer Science, vol. 10981, July 2018, doi: https://doi.org/10.1007/978-3-319-96145-3_35.

[Ull18] S. Ullah, S. S. Murthy and A. Kumar, "SMApproxLib: Library of FPGA-based Approximate Multipliers," 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), 2018, pp. 1-6, doi: 10.1109/DAC.2018.8465845.

[Mra20] V. Mrazek, L. Sekanina and Z. Vasicek, "Libraries of Approximate Circuits: Automated Design and Application in CNN Accelerators," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems,

vol. 10, no. 4, pp. 406-418, Dec. 2020, doi: 10.1109/JETCAS.2020.3032495.

[Bal22] P. Balasubramanian, R. Nayar, O. Min and D.L. Maskell, "Approximator: A Software Tool for Automatic Generation of Approximate Arithmetic Circuits," in Computers, vol. 11, no. 1, Jan. 2022, https://doi.org/10.3390/computers11010011.

[Vah19] S. Vahdat, M. Kamal, A. Afzali-Kusha and M. Pedram, "TOSAM: An Energy-Efficient Truncation and Rounding-Based Scalable Approximate Multiplier," in IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 5, pp. 1161-1173, May 2019, doi: 10.1109/TVLSI.2018.2890712.

[Fru20] F. Frustaci, S. Perri, P. Corsonello and M. Alioto, "Approximate Multipliers with Dynamic Truncation for Energy Reduction via Graceful Quality Degradation," in IEEE Trans. on Circuits and Systems II: Express Briefs, vol. 67, no. 12, pp. 3427-3431, Dec. 2020, doi: 10.1109/TCSII.2020.2999131.

[Wal64] C. S. Wallace, "A Suggestion for a Fast Multiplier," in IEEE Transactions on Electronic Computers, vol. EC-13, no. 1, pp. 14-17, Feb. 1964, doi: 10.1109/PGEC.1964.263830.

[Dad83] L. Dadda, "Some schemes for fast serial input multipliers," 1983 IEEE 6th Symposium on Computer Arithmetic (ARITH), 1983, pp. 52-59, doi: 10.1109/ARITH.1983.6158074.

[Okl96] V. G. Oklobdzija, D. Villeger and S. S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," in IEEE Transactions on Computers, vol. 45, no. 3, pp. 294-306, March 1996, doi: 10.1109/12.485568.

[Kel09] D. R. Kelly, B. J. Phillips and S. Al-Sarawi, "Approximate signed binary integer multipliers for arithmetic data value speculation", Proc. DASIP Conf., pp. 97-104, 2009.

[Cil14] A. Cilardo, D. De Caro, N. Petra, F. Caserta, N. Mazzocca, E. Napoli, "High Speed Speculative Multipliers Based on Speculative Carry-Save Tree," in IEEE Trans. on Circuits and Systems I: Regular Papers, vol. 61, no. 12, pp. 3426-3435, Dec. 2014, doi: 10.1109/TCSI.2014.2337231.

[Qiq17] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov and A. Yakovlev, "Energy-efficient approximate multiplier design using bit significance-driven logic compression," Design, Automation & Test in Europe Conf. & Exhibition (DATE), 2017, pp. 7-12, doi: 10.23919/DATE.2017.7926950.

[Esp17] D. Esposito, A. G. M. Strollo and M. Alioto, "Low-power approximate MAC unit," 2017 13th Conf. on Ph.D. Research in Microelectronics and Electronics (PRIME), 2017, pp. 81-84, doi: 10.1109/PRIME.2017.7974112.

[Guo18] Y. Guo, H. Sun, L. Guo and S. Kimura, "Low-Cost Approximate Multiplier Design using Probability-Driven Inexact Compressors," 2018 IEEE Asia Pacific Conf. on Circuits and Systems (APCCAS), 2018, pp. 291-294, doi: 10.1109/APCCAS.2018.8605570.

[Ans18] M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han, "Low-Power Approximate Multipliers Using Encoded Partial Products and Approximate Compressors," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 8, no. 3, pp. 404-416, Sept. 2018, doi: 10.1109/JETCAS.2018.2832204.

[Esp18] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro and N. Petra, "Approximate Multipliers Based on New Approximate Compressors," in IEEE Trans. on Circuits and Systems I: Regular Papers, vol. 65, no. 12, pp. 4169-4182, Dec. 2018, doi: 10.1109/TCSI.2018.2839266.

[Mad19] L. Maddisetty, R. K. Senapati, J. V. R. Ravindra “Training Neural Network as Approximate 4:2 Compressor applying Machine Learning Algorithms for Accuracy Comparison,” in Int. Journal of

Advanced Trends in Computer Science and Engineering, vol 8, no. 2, pp. 211-215, 2019, doi: 10.30534/ijatcse/2019/17822019.

[Sab19] F. Sabetzadeh, M. H. Moaiyeri and M. Ahmadinejad, "A Majority-Based Imprecise Multiplier for Ultra-Efficient Approximate Image Multiplication," in IEEE Trans. on Circuits and Systems I: Regular Papers, vol. 66, no. 11, pp. 4200-4208, Nov. 2019, doi: 10.1109/TCSI.2019.2918241.

[Eda20] P. J. Edavoor, S. Raveendran and A. D. Rahulkar, "Approximate Multiplier Design Using Novel Dual-Stage 4:2 Com-pressors," in IEEE Access, vol. 8, pp. 48337-48351, 2020, doi: 10.1109/ACCESS.2020.2978773.

[StNa20] A. G. M. Strollo, E. Napoli, D. De Caro, N. Petra and G. D. Meo, "Comparison and Extension of Approximate 4-2 Compressors for Low-Power Approximate Multipliers," in IEEE Trans. on Circuits and Systems I: Regular Papers, vol. 67, no. 9, pp. 3021-3034, Sept. 2020, doi: 10.1109/TCSI.2020.2988353.

[StDe20] A. G. M. Strollo, D. De Caro, E. Napoli, N. Petra and G. Di Meo, "Low-Power Approximate Multiplier with Error Recovery using a New Approximate 4-2 Compressor," 2020 IEEE Int. Symp. on Circuits and Systems (ISCAS), 2020, pp. 1-4, doi: 10.1109/ISCAS45731.2020.9180767.

[Zak20] P. Zakian, R. N. Asli, “An efficient design of low-power and high-speed approximate compressor in FinFET technology,” in Computers & Electrical Engineering, vol. 86, Sept 2020, doi: 10.1016/j.compeleceng.2020.106651.

[Pei21] H. Pei, X. Yi, H. Zhou and Y. He, "Design of Ultra-Low Power Consumption Approximate 4–2 Compressors Based on the Compensation Characteristic," in IEEE Trans. on Circuits and Systems II: Express Briefs, vol. 68, no. 1, pp. 461-465, Jan. 2021, doi: 10.1109/TCSII.2020.3004929.

[Kha21] M. Khaleqi, M. Ahmadinejad, M. H. Moaiyeri, “Ultraefficient imprecise multipliers based on innovative 4:2 approximate compressors,” in Int. Journal of Circuit Theory and Applications, vol. 49, no. 1, pp. 169-184, 2021, doi: 10.1002/cta.2876.

[Kul11] P. Kulkarni, P. Gupta and M. Ercegovac, "Trading Accuracy for Power with an Underdesigned Multiplier Architecture," 2011 24th Int. Conf. on VLSI Design, 2011, pp. 346-351, doi: 10.1109/VLSID.2011.51.

[Reh16] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, J. Henkel and J. Henkel, "Architectural-space exploration of approximate multipliers," 2016 IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD), 2016, pp. 1-8, doi: 10.1145/2966986.2967005.

[GSK18] Y. Guo, H. Sun and S. Kimura, “Design of power and area efficient lower-part-OR approximate multiplier,” TENCON 2018 - IEEE Region 10 Conf., 2018, pp. 2110-2115, doi:10.1109/TENCON.2018.8650108.

[Gil19] G. A. Gillani, M. A. Hanif, B. Verstoep, S. H. Gerez, M. Shafique and A. B. J. Kokkeler, "MACISH: Designing Approximate MAC Accelerators with Internal-Self-Healing," in IEEE Access, vol. 7, pp. 77142-77160, 2019, doi: 10.1109/ACCESS.2019.2920335.

[War20] H. Waris, C. Wang, W. Liu, J. Han and F. Lombardi, “Hybrid partial product-based high-performance approximate recursive multipliers,” in IEEE Trans. on Emerging Topics in Computing, 2020, doi:10.1109/TETC.2020.3013977.

[Yan20] Z. Yang, X. Li and J. Yang, "Power Efficient and High-Accuracy Approximate Multiplier with Error Correction," in Journal of Circuits, Systems and Computers, vol. 29, no. 15, Dec. 2020, doi: 10.1142/S0218126620502412.

[War21] H. Waris, C. Wang, C. Xu and W. Liu, "AxRMs: Approximate Recursive Multipliers using High-Performance Building Blocks," in IEEE Trans. on

Emerging Topics in Computing, 2021, doi: 10.1109/TETC.2021.3096515.

[Nun22] I. Nunziata, E. Zacharelos, G. Saggese, A. M. G. Strollo and E. Napoli, "Approximate Recursive Multipliers Using Carry Truncation and Error Compensation," 17th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Villasimius, SU, Italy, pp. 137-140, 2022, doi: 10.1109/PRIME55000.2022.9816787.

[Zac22] E. Zacharelos, I. Nunziata, G. Saggese, A. G. M. Strollo and E. Napoli, "Approximate Recursive Multipliers Using Low Power Building Blocks," in IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 3, pp. 1315-1330, 1 July-Sept. 2022, doi: 10.1109/TETC.2022.3186240.

[Ans20] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek and J. Han, "Improving the Accuracy and Hardware Efficiency of Neural Networks Using Approximate Multipliers," in IEEE Trans. on Very Large-Scale Integration (VLSI) Systems, vol. 28, no. 2, pp. 317-328, Feb. 2020, doi: 10.1109/TVLSI.2019.2940943.

[LeC10] Y. LeCun, C. Cortes, and C. Burges. (2010). MNIST hand-written digit database. AT&T Labs. [Online]. Available: http://yann.lecun.com/exdb/mnist

[Net11] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning,” in Proc. NIPS Workshop Deep Learn. Un-supervised Feature Learn., 2011, p. 5. Dataset available online: http://ufldl.stanford.edu/housenumbers.

[Ahm22] M. Ahmadinejad, and M. Moaiyeri, "Energy- and Quality-Efficient Approximate Multipliers for Neural Network and Image Processing Applications" in IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 02, pp. 1105-1116, 2022, doi: 10.1109/TETC.2021.3072666

[Gil18] G. A. Gillani, M. A. Hanif, M. Krone, S. H. Gerez, M. Shafique and A. B. J. Kokkeler, "SquASH: Approximate Square-Accumulate with Self-Healing," in IEEE Access, vol. 6, pp. 49112-49128, 2018, doi: 10.1109/ACCESS.2018.2868036.

[She02] M.H. Sheu and S.-Hon Lin, "Fast compensative design approach for the approximate squaring function," in IEEE Journal of Solid-State Circuits, vol. 37, no. 1, pp. 95-97, Jan. 2002, doi: 10.1109/4.974551.

[Lan06] J. M. P. Langlois and D. Al-Khalili, "Carry-free approximate squaring functions with O(n) complexity and O(1) delay," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 53, no. 5, pp. 374-378, May 2006, doi: 10.1109/TCSII.2006.873364.

[Man20] K. Manikantta Reddy, M. H. Vasantha, Y. B. Nithin Kumar and D. Dwivedi, "Design of Approximate Booth Squarer for Error-Tolerant Computing," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 5, pp. 1230-1241, May 2020, doi: 10.1109/TVLSI.2020.2976131.

[Ans22] M. S. Ansari, B. F. Cockburn and J. Han, "Low-Power Approximate Logarithmic Squaring Circuit Design for DSP Applications," in IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 1, pp. 500-506, 1 Jan.-March 2022, doi: 10.1109/TETC.2020.2989699.

[Gar10] V. Garofalo, M. Coppola, D. De Caro, E. Napoli, N. Petra and A. G. M. Strollo, "A novel truncated squarer with linear compensation function," Proceedings of 2010 IEEE International Symposium on Circuits and Systems, 2010, pp. 4157-4160, doi: 10.1109/ISCAS.2010.5537591.

[Sha15] B. Shao and P. Li, "Array-Based Approximate Arithmetic Computing: A General Model and Applications to Multiplier and Squarer Design," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 62, no. 4, pp. 1081-1090, April 2015, doi: 10.1109/TCSI.2015.2388839.

[Pet14] N. Petra, D. De Caro, V. Garofalo, E. Napoli, A. G.M. Strollo, Truncated squarer with minimum mean-square error, Microelectronics Journal, vol. 45, no. 6, 2014, 799-804, https://doi.org/10.1016/j.mejo.2014.02.018.

[Noe89] A. S. Noetzel, "An interpolating memory unit for function evaluation: analysis and design," in IEEE Transactions on Computers, vol. 38, no. 3, pp. 377-384, March 1989, doi: 10.1109/12.21124.

[Vit67] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," in IEEE Transactions on Information Theory, vol. 13, no. 2, pp. 260-269, April 1967, doi: 10.1109/TIT.1967.1054010.

[Xu16] Q. Xu, T. Mytkowicz and N. S. Kim, "Approximate Computing: A Survey," in IEEE Design & Test, vol. 33, no. 1, pp. 8-22, Feb. 2016, doi: 10.1109/MDAT.2015.2505723

[Chi13] V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," 2013 50th ACM/EDAC/IEEE Design Automation Conf. (DAC), 2013, pp. 1-9, doi: 10.1145/2463209.2488873.

[Avr14] A. Avramović, Z. Babić, D. Raič, D. Strle, and P. Bulić, “An approximate logarithmic squaring circuit with error compensation for DSP applications,” in Microelectronics Journal, vol 45, no 3, pp. 263-271, 2014, https:// doi.org/10.1016/j.mejo.2014.01.005.

[Sol89] M. R. Soleymani, and S. D. Morgera, "A fast MMSE encoding technique for vector quantization," in IEEE Transactions on Communications, vol. 37, no. 6, pp. 656-659, June 1989, doi: 10.1109/26.31152.

[Kol98] R. K. Kolagotla, W. R. Griesbach, H.R. Srinivas, (1998). VLSI implementation of 350 MHz 0.35 /spl mu/m 8 bit merged squarer. Electronics Letters, 34, pp. 47-48.

[Bui14] S. Bui and J. E. Stine, "Additional optimizations for parallel squarer units," 2014 IEEE International

Symposium on Circuits and Systems (ISCAS), 2014, pp. 361-364, doi: 10.1109/ISCAS.2014.6865140.

[Das16] D.K. Das, and A. Banerjee, "A New Squarer Design with Reduced Area and Delay," in IET Computers & Digital Techniques, 2016, doi: 10. 10.1049/iet-cdt.2015.0170.

[Yoo97] J.-T. Yoo, K. F. Smith, and G. Gopalakrishnan,"A fast parallel squarerbased on divide-and-conquer," IEEE J. Solid-State Circuits, vol. 32, no.6, pp. 909–912, Jun. 1997.

[Car01] D. De Caro, and A.G.M. Strollo, "Parallel squarer using Booth-folding technique," in Electronics Letters. 37, pp 346 – 347, 2001, doi: 10.1049/el:20010241

[Str03] A. G. M. Strollo and D. De Caro, "Booth folding encoding for high performance squarer circuits," in IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 50, no. 5, pp. 250-254, May 2003, doi: 10.1109/TCSII.2003.810574.

[Dea69] K.J. Dean, "Cellular logical array for obtaining the square of a binary number." In Electronics Letters 5, 1969.

[Sha91] M. Shamanna, S. Whitaker, and J. Canaris. "Cellular Logic Array for Computation of Squares", in 3rd NASA Symposium on VLSI Design, 1991.

[Cha99] J.K. Chapin, K.A. Moxon, R.S. Markowitz, and M.A.L. Nicolelis. "Real-time control of a robot arm using simultaneously recorded neurons in the motor cortex", in Nature Neuroscience, vol. 2, no. 7, pp. 664 - 670, July 1999, doi: 10.1038/10223.

[Leb06] M.A. Lebedev, and M.A. Nicolelis, "Brain-machine interfaces: past, present and future", in Trends in neurosciences, vol. 29, no. 9, pp. 536 - 546, 2006, doi: 10.1016/j.tins.2006.07.004.

[Rap21] A. B. Rapeaux, and T. G. Constandinou, "Implantable brain machine interfaces: first-in-human studies, technology challenges and trends", in Current Opinion in Biotechnology, vol. 72, pp. 102-111, 2021, doi: https://doi.org/10.1016/j.copbio.2021.10.001.

[Sah21] S. Saha, K.A. Mamun, K.I.U. Ahmed, R. Mostafa, G.R. Naik, S. Darvishi, A.H. Khandoker, M. Baumert, “Progress in brain computer interface: challenges and potentials”, in Frontiers in Systems Neuroscience, vol. 15, p. 4, 2021, doi: 10.3389/fnsys.2021.578875

[Fra19] J.A. Frank, M.-J. Antonini, and P. Anikeeva, “Next-generation interfaces for studying neural function”, in Nature Biotechnology, vol 37, pp. 1013-1023, 2019, doi: 10.1038/s41587-019-0198-8

[Shaer15] M.A. Shaeri and A. M. Sodagar, "A Method for Compression of Intra-Cortically-Recorded Neural Signals dedicated to Implantable Brain-Machine Interfaces," in IEEE Trans. on Neural Systems and Rehabilitation Engineering, vol. 23, no. 3, pp. 485-497, May 2015, doi: 10.1109/TNSRE.2014.2355139.

[Sag22] G. Saggese, E. Zacharelos and A. G. M. Strollo, "Low Power Spike Detector for Brain-Silicon Interface using Differential Amplitude Slope Operator," 2022 17th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Villasimius, SU, Italy, 2022, pp. 301-304, doi: 10.1109/PRIME55000.2022.9816758.

[Sagge22] G. Saggese, A.G.M. Strollo, "Low-Power Energy-Based Spike Detector ASIC for Implantable Multichannel BMIs," in Electronics, vol. 11, 2022, doi: https://doi.org/10.3390/electronics11182943.

[Sag21] G. Saggese et al., "Comparison of Sneo-Based Neural Spike Detection Algorithms for Implantable Multi-Transistor Array Biosensors," in Electronics, vol. 10, pp. 410, 2021, doi: 10.3390/electronics10040410.

[Bar14] D. Y. Barsakcioglu et al., "An Analogue Front-End Model for Developing Neural Spike Sorting Systems," in IEEE Trans. on Biomedical Circuits and Systems, vol. 8, no. 2, April 2014.

[Riz09] M. Rizk et al, "A fully implantable 96-channel neural data acquisition system," in Journal of Neural Engineering, vol. 6, May 2009.

[Nav14] J. Navajas et al, "Minimum requirements for accurate and efficient real-time on-chip spike sorting," in

Journal of Neuroscience Methods, vol. 230, Apr. 2014, doi: https://doi.org/10.1016/j.jneumeth.2014.04.018.

[Nie16] J. Niediek, J. Bostrom, C.E. Elger, and F. Mormann, "Reliable analysis of single-unit recordings from the human brain under noisy conditions: tracking neurons over hours," in PloS one, vol. 11, no. 12, Dec. 2016.

[Reh19] M. S. Rehman et al., "SpikeDeeptector: A deep-learning based method for detection of neural spiking activity," in Journal of Neural Engineering, vol 16, July 2019, doi: 10.1088/1741-2552/ab1e63.

[Tam21] M. Tambaro et al., "A scalable spike detection method for implantable high-density multielectrode array," PRIME 2021, pp. 1-4, July 2021.

[Fre16] Z. Frehlick, I. Williams and T. G. Constandinou, "Improving neural spike sorting performance using template enhancement," in IEEE Biomedical Circuits and Systems Conference (BioCAS), Shanghai, China, pp. 524-527, Oct. 2016, doi: 10.1109/BioCAS.2016.7833847.

[Cam19] S. L. Cam, H. Tran, R. Ranta and V. Louis-Dorr, "A Bayesian approach for Simultaneous Spike Extraction and Sorting," in 2019 9th International IEEE/EMBS Conference on Neural Engineering (NER), San Francisco, CA, USA, 2019, doi: 10.1109/NER.2019.8716888.

[Rác20] M. Rácz et al., "Spike detection and sorting with deep learning," in Journal of Neural Engineering, vol 17, no. 1, Jan. 2020.

[Reh21] M. S. Rehman et al., "SpikeDeep-Classifier: A deep-learning based fully automatic offline spike sorting algorithm," in Journal of Neural Engineering, vol 18, Febr. 2021, doi: 10.1088/1741-2552/abc8d4.

[Cam13] M.L.A. Camuñas, and R. Q. Quiroga, “A detailed and fast model of extracellular recordings”, in Neural Computation, vol 25, no. 5, pp. 1191-1212, 2013, doi: https://doi.org/10.1162/NECO_a_00433

[Jia22] T. Jiang, D. Wu, F. Gao, J. Cao, S. Dai, J. Liu, and Y. Li, “Improved Spike Detection Algorithm Based on

Multi-Template Matching and Feature Extraction", in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69, no. 1, pp. 249-253, Jan. 2022, doi: 10.1109/TCSII.2021.3092141

[Hao21] H. Hao, J. Chen, A. G. Richardson, J. Van der Spiegel and F. Aflatouni, "A 10.8 µW Neural Signal Recorder and Processor With Unsupervised Analog Classifier for Spike Sorting", in IEEE Transactions on Biomedical Circuits and Systems, vol. 15, no. 2, pp. 351-364, April 2021, doi: 10.1109/TBCAS.2021.3076147.