UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

PH.D. THESIS IN

INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

VLSI CIRCUITS FOR APPROXIMATE COMPUTING

DARJN ESPOSITO

TUTOR: PROF. ANTONIO GIUSEPPE MARIA STROLLO

XXIX CICLO

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

Acknowledgments

At the end of this path I have to thank all the people involved in this valuable experience.
First of all, I would like to thank my tutor, professor Antonio Strollo, for his constant support. His enthusiasm, experience, technical rigor and elegance in solving problems have been of great example for me.

I also have to thank professors Davide De Caro and Nicola Petra, for their infinity patience to clarify my doubts and to always provide me answers to all my questions.
Many thanks to professor Ettore Napoli, for giving me the opportunity to help him at laboratory lectures of his course and for the stimulating discussions.
I want also to thank the remaining professors of Electronics Group for their priceless lessons.

I would like to thank professor Massimo Alioto of National University of Singapore, for the nice hospitality at Green IC group and for his enthusiasm in facing challenges. To this regard I want to thank the friends of the Green IC group: Orazio (for sharing his experience and suggestions and for the walks at midnight from lab to home), Enrico and his wife Federica (for the nice, albeit rare, moments of fun and for the nice Easter lunch), Sachin, Kien, Longyang (for the stimulating technical discussions), Priyankar, Santanu, Tata, Jim, Deepa, Isha, Prakhar (for his willingness to learn Italian ☺), Allan, Saurabh and Shubham. I big thank to my housemates Deepa, Pardha and Abhinav for making me feel at home.

I would like to thank the friends of the Electronics Group at DIETI: Pierluigi, Alessandro, Antonio, Gerardo, Grazia, Ilaria, Michele, Luca, Paolo, Gianpaolo, Francesco, Salvatore for their friendship and nice time spent together. In particular, the PhD course gave me the opportunity to meet Fabio Di Napoli, my desk mate, who is actually one of my best friends. I will never forget his *valiant* battle against a cockroach, in hotel room at Cagliari. Many thanks to Pasquale

Cennamo, for continuously sharing his valuable experience on PCB design and soldering.

I cannot forget (especially for the consequences ☺) to thank my girlfriend Giovanna, who has been and is a constant point of reference and support for me. I have to thank her for the ability to respect my choices, and for the fortitude in tolerating the six months of separation during my visiting at National University of Singapore.

A final thanks goes to my family for the serenity they gave me during the studies.

Chapter 1

Introduction

1.1 Approximate Computing: motivations

In the last years, important changes occurred in the VLSI world and, more in general, in the electronic community. The traditional Dennard scaling [1], which allowed, for around three decades, to obtain always smaller transistors with better performance, broke off in about 2005 [2]-[3].

While Moore's Law continues to provide increased transistors count (in 2014 Intel launched 14nm node [4]), the benefits deriving from smaller transistors sizes, diminished. Indeed reliability and energy issues (i.e. leakage power) arising with nanometer feature sizes, prevent threshold voltage to scale down and therefore operating voltage has remained at a constant value for several processor chip generation [4]. Moreover, in the race for always better performance, clock frequencies scaled up faster than dictated by Dennard scaling (Fig. 1) [3]. As a consequence of this phenomena, it was not possible to keep the power envelope constant from a generation to the successive (Fig. 2). The uninterrupted increase of chip power density (delayed by the clever low-power methodologies developed in the years [3]) stopped the clock frequency scaling, since processors attained the power wall for air cooling [3] (Fig. 1).

On the other hand, the computing workloads have profoundly changed, due to applications needing increased computing power to perform Recognition, Mining, Synthesis (RMS) tasks [5], to process always richer media and to interact with users and environment [6].

The need for always more computing power and the gain limitations due to the end of Dennard scaling (e.g. clock frequency saturation [7], Fig. 1) marked the transition to the multi-core architectures. These, while leveraging the inherent parallelism of many applications, started to show their limits, mainly due to thermal power. Indeed, in multi-core architectures, due to fixed power budget,

it is not possible to use all the cores at the maximum frequency and, as a result, going from a generation to the successive, the fraction of chip area which is *dark* (i.e. in idle state) increases, leading to the so-called *dark silicon* [8].

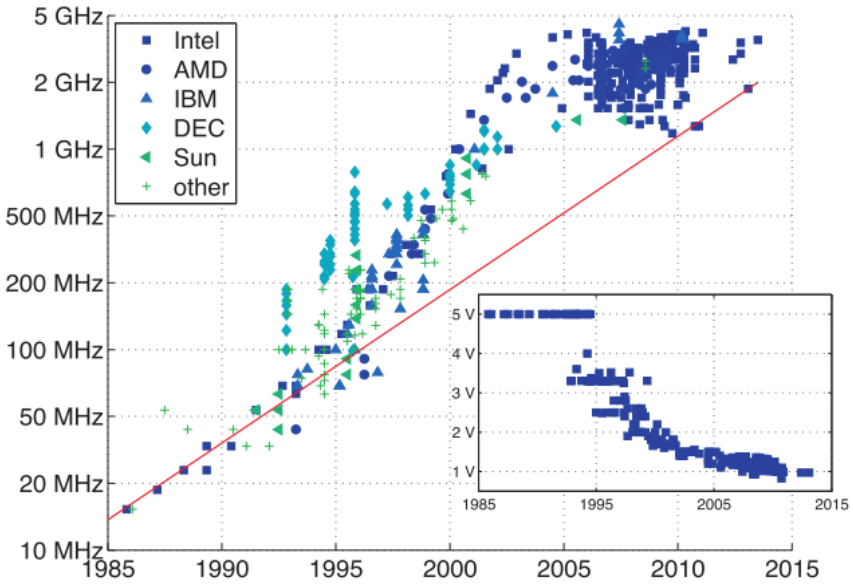

Fig. 1 Clock frequency vs year [3]. The red line represents the frequency scaling employing Dennard scaling [1]. In the insert, voltage vs year is reported. Note that, since about 2005, the scaling for both clock frequency and voltage saturated.

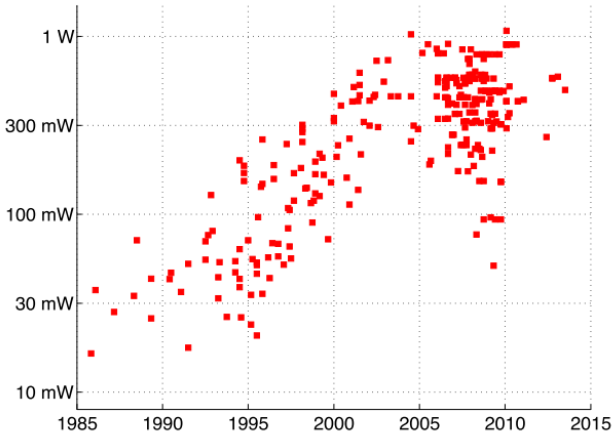

Fig. 2 Power density in mW/mm^2 vs year

Ensuring high performance, while meeting the power budget is a really challenging task. Therefore researches and designers started to search novel solutions to compute efficiently. One of the key to overcome this challenge is hardware specialization, which allows improving energy efficiency up to 2-3 orders of magnitude [3], with respect to general purpose solutions. In this context, one of the most promising solutions is given by the *Approximate Computing*. Indeed, many applications exhibit *approximation resiliency* or can tolerate small errors without compromise significantly the quality of their results [9]. As an example, in multimedia applications, small errors can be tolerated, due to the limited perceptual capabilities of humans [10]. For instance, this concept has been already used in compression algorithms for images, audio, video. Moreover in RMS applications, there no exist an unique golden result, while rather, a good enough answer [11]. Other applications operates on imprecise data inputs, as those collected by sensors.

Therefore Approximate Computing, breaking the dogma that computation must be error free [12], exploits error resiliency of applications to achieve better performance. Indeed, relaxing the correctness requirement, simpler, faster and/or more energy efficient circuits can be obtained.

Approximate Computing, therefore, allows enlarging the design space, introducing quality (or accuracy) as additional variable, enabling new possible trade-offs between quality, power, area and speed.

1.2 Research topics

Approximate computing literature is very broad, involving many layers of computation, spanning from approximate programming languages to inexact hardware [13]-[18].
My research activity is based on the design and optimization of approximate fundamental digital blocks.
As first topic, I have worked on adders, developing speculative (approximate) topologies of the main parallel-prefix adders. In order to use these topologies in common digital systems, speculative adders must include error detection and correction networks, leading to a variable-latency adder topology. I have worked on improvements of both error correction and detection networks.

In particular, part of my work has been devoted to develop an effective error correction technique for speculative adders working with signed operands.
As second topic I worked on digital building blocks for error tolerant applications. In this contest the requirement on the correctness of the results is relaxed, therefore computations can contain errors. I have investigated an error correction technique to contain the error rate when approximate adders deal with signed operands.
Moreover, a study about suitability of approximate adders in carry-save Multiply and Accumulate units, has been investigated.
In particular, Multiply and Accumulate (MAC) units constitute another recurrent arithmetic building block in digital systems. In this contest, I have worked on precision-scalable MACs, developing a novel, real-time (data-aware), error compensation technique. In this context I also developed a precision-scalable approximate MAC unit, in which the partial product matrix is compressed in an approximate way.
Moreover I have investigated precision-scalable topologies of standard cell memories (SCMs). These are recently emerged as an alternative to SRAM Macrocells for systems needing a large number of small embedded memories.
The precision-scalable MACs and SCMs can be part of a system that leverages approximations to improve energy efficiency.

1.3 Thesis outline

The thesis is organized as follows.
The chapter two reports my research activity about approximate adders. It is divided in two main parts, in the first one, after an introduction of parallel-prefix adders, the proposed speculative topologies are introduced. Moreover the error detection and correction techniques are introduced and investigated, for error-free applications. In the second part, approximate adders for error tolerant applications are illustrated. In particular an approximate adder is introduced, in which an error correction technique is proposed to decrease the error rate. An example for audio applications is shown. Moreover the suitability of approximate adders in carry-save MACs is investigated.
The chapter three is divided in two sections. In the first one my research activity on precision-scalable MACs is discussed. The second

section shows a precision-scalable topology of a standard cell memory (SCM).
In the chapter four a precision scalable approximate convolver for computer vision applications is discussed; this is composed of both the precision-scalable SCMs and MACs, shown in the chapter three.
The activities shown in chapters three and four have been investigated in collaboration with National University of Singapore, Green IC group, where I spent six months as visiting PhD student.

1.4 Publications

- **D. Esposito**, D. De Caro, E. Napoli, N. Petra, A. G. M. Strollo, "Variable Latency Speculative Han-Carlson Adder", IEEE Trans. on Circuits and Systems I: Regular Papers, vol.62, no.5, pp.1353-1361, 2015.
- **D. Esposito**, D. De Caro and A. G. M. Strollo, "Variable Latency Speculative Parallel Prefix Adders for Unsigned and Signed Operands," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 63, no. 8, pp. 1200-1209, Aug. 2016.
- E. Napoli, G. Castellano, D. De Caro, **D. Esposito**, N. Petra and A. G. M. Strollo, "A SISO Register Circuit Tailored for Input Data with Low Transition Probability," in IEEE Transactions on Computers, vol. 66, no. 1, pp. 45-51, Jan. 1 2017.
- E. Napoli; G. Castellano; D. De Caro; **D. Esposito**; N. Petra; A. G. M. Strollo, "Single Bit Filtering Circuit Implemented in a System for the Generation of Colored Noise," in IEEE Transactions on Circuits and Systems I: Regular Papers , vol.PP, no.99, pp.1-11.
- D. De Caro; E. Napoli; **D. Esposito**; G. Castellano; N. Petra; A. G. M. Strollo, "Minimizing Coefficients Wordlength for Piecewise-Polynomial Hardware Function Evaluation With Exact or Faithful Rounding," in IEEE Transactions on Circuits and Systems I: Regular Papers , vol.PP, no.99, pp.1-14.

- **D. Esposito**, D. De Caro, A. G. M. Strollo, “Speculative Parallel-Prefix Adders", Proceedings of 46th GE Conference, ISBN: 978-88-905519-2-5.
- **D. Esposito**, D. De Caro, M. De Martino, A. G. M. Strollo, “Variable Latency Speculative Han–Carlson Adders Topologies”, 11th Conference on PhD Research in Microelectronics and Electronics (IEEE PRIME 2015), pp.45-48, 2015.
- E. Napoli, G. Castellano, **D. Esposito** and A. G. M. Strollo, "Digital circuit for the generation of colored noise exploiting single bit pseudo random sequence," 2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS), Florianopolis, 2016, pp. 23-26.
- **D. Esposito**, A. G. M. Strollo, M. Alioto, “Power-Precision Scalable Latch Memories”, accepted at 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA.
- **D. Esposito**, D. De Caro, E. Napoli, N. Petra, A. G. M. Strollo, “On the Use of Approximate Adders in Carry-Save Multiplier-Accumulators”, accepted at 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA.

Chapter 2

Speculative Adders

2.1 Introduction

This chapter focuses on the design of approximate adders topologies. These kind of adders are usually indicated, in the literature, as "speculative adders" [19]-[22].

Speculative adders draw their motivations from the need to improve adders performance, which are usually limited by their long and rarely activated critical path, following a "better than worst case" approach [23].

Improving adders performance is indeed a challenging and useful task, since adders are ubiquitous in digital systems and a great research effort has been spent in the past in order to optimize their performance in the area-timing-power design space. The great number of different adder topologies is the result of that investigation effort. Going from the simpler carry-ripple, to carry-select, carry-lookahead and parallel-prefix adders topologies [24]-[26], it is possible to trade area and power for timing, as a function of the specification of the system in which the adders are called to operate.

Theoretical study about adders have shown that the speed bound of n-bit binary adders goes as $O(\log_2 n)$. This bound corresponds to the worst case path, which involves a carry propagation that, starting from the least significant bit (*LSB)*, reaches the most significant bit (*MSB)*, traversing the whole adder. A such condition, under uniformly distributed and uncorrelated operands assumption, is pretty rare. Indeed, in [20] authors show that the average carry propagation length has a logarithmic behavior with the adder size *n*. The Figure 1 reports the carry length distribution in a 32-bit adder, with operands uniformly distributed and uncorrelated. In this example it easy to locate the mean value of the distribution, being around five, confirming the behavior discussed in [20]. Therefore in a 32-bit adder,

under the abovementioned hypothesis, in the average case, the carry propagation length is significantly smaller than adder size *n*, allowing enough room to make "the common case faster".

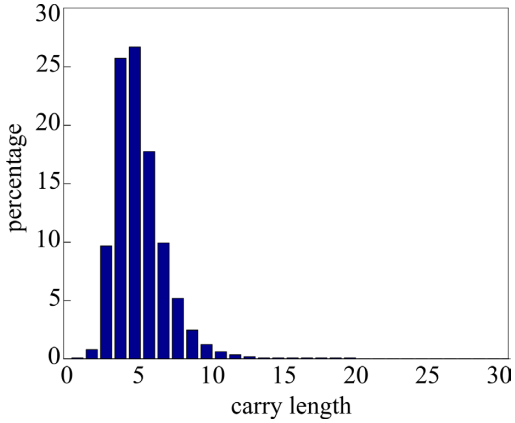

Fig. 1 Carry length in a 32-bit adder, with uniformly distributed and uncorrelated operands. The average carry length is around five.

2.2 Speculative adders: general overview

In this section a general representation of speculative adders is introduced. Moreover, error detection network will be discussed. This general representation will be used, in the following of the chapter, to discuss the proposed topologies. At the end of this paragraph a brief review of the state of art is discussed.

As previously discussed, in a *n*-bit adder, the *MSB* of the result depends on all the previous bits, since, in the worst case, a carry can be generated in the *LSB* and can traverse the whole adder. The Fig. 2 shows, in the employed representation, an *n*=18-bit adder, where the gray line treads the critical path. This is triggered by the following condition:

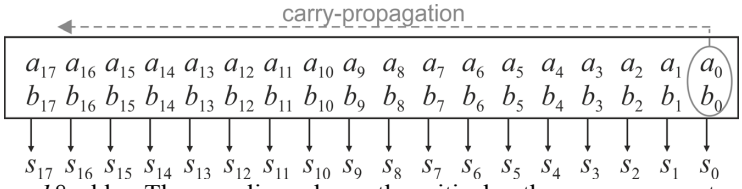

Fig. 2 *n=18* adder. The gray lines shows the critical path: a carry generates in the *LSB* and propagates up to the *MSB*. Therefore *s17* has to "wait" the carry propagation before being correctly evaluated.

$$\begin{cases} a_0 = b_0 = 1 \\ a_i \oplus b_i = 1, \ for\ i \in \{1,2,...,n-2\} \end{cases} \quad (2.1)$$

where $\oplus$ stands for the XOR operation. When this condition occurs the evaluation of the sum s requires the biggest delay.

As discussed in the previous paragraph, condition (2.1), under the assumption that a and b are uniformly distributed and uncorrelated, is pretty rare, since the average carry length is comparable with $\log_2 n$. Leveraging this observation, speculative adders assume that carry propagates no more than $p<n$ bits. Therefore each sum bit is predicted by considering only the p previous less significantly bits. This allows to repartition the adders into multiple, smaller, sub-adders operating in parallel [27], this turns into more efficient addition implementation, at the price of occasional errors. The Fig. 3 shows an example of speculative adder.

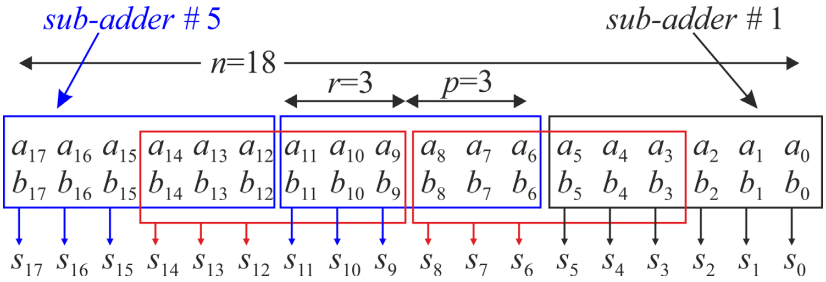

Fig. 3 The adder of Fig. 2 is partitioned into five sub-adders operating in parallel. Each sub-adder, considers only p previous bits to evaluate r sum bits.

In this figure, the adder of Fig. 2 has been subdivided into five sub-adders, each one, following the notation introduced in [28], produces r sum bits (with the exception of the first one, which produces $r+p$ output bit), "speculated" considering only p previous bits. The size of each sub-adder is $r+p$.

2.2.1 Error detection

Speculative adders are usually augmented with an error detection network [18], [20]-[22], [28], [30]. This network flags the errors due to mispredictions, allowing to adopt different correction techniques.

The Fig. 4 shows a possible error condition: a carry is generated at the bit position #1 and propagates for more than p (p=3, in this example) bits. As a consequence, the output from $s6$ to $s10$ are wrongly flipped, since they are calculated assuming that the carry-in of the second sub-adder is zero, due to speculation.

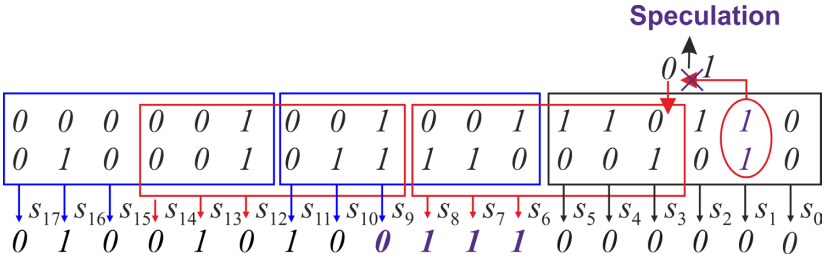

Fig. 4 Example of error condition: the carry generated at bit position #1 propagates for more than p bits, determining an incorrect result.

In order to formally determine a general error condition, let us introduce, in the following, the generate g_i and propagate p_i signals computed as:

$$g_i = a_i b_i \tag{2.2}$$

$$p_i = a_i \oplus b_i \tag{2.3}$$

The condition g_i=1 means that a carry is generated at bit i, while the condition p_i=1 means that a carry is propagated through bit i. The concept of generate and propagate can be extended to a block of contiguous bits, from bit k to bit i (with $k \le i$) as follows:

$$g_{[i:k]} = \begin{cases} g_i & \text{if } i = k \\ g_{[i:j]} + p_{[i:j]} g_{[l:k]} & \text{otherwise} \end{cases} \tag{2.4}$$

$$p_{[i:k]} = \begin{cases} p_i & \text{if } i = k \\ p_{[i:j]} p_{[l:k]} & \text{otherwise} \end{cases} \tag{2.5}$$

where: $i \ge l \ge j \ge k$, and + operator stands for logical *OR*.
The condition $g_{[i:k]}$=1 means that a carry is generated in the block k-i, while the condition $p_{[i:k]}$=1 means that a carry is propagated through the block. In the example of Fig. 4, g_1=1, p_2=1, determining $g_{[2:0]}$=1; moreover, being p_3=p_4=p_5=1, then $p_{[5:3]}$=1.

Let us observe that an error occurs whenever a sub-adder should receive a carry-in equal to one, due to generation and

propagation in the previous bits. Indeed, with reference to Fig. 4, the second sub-adder assumes a carry-in equal to zero, due to speculation, while, instead, the real carry-in is one. This condition can be expressed as:

$$p_{[5:3]}g_{[2:0]} \quad \text{error in sub-adder 2} \tag{2.6}$$

Similarly, the error condition in the third sub-adder can be expressed as:

$$p_{[8:6]}g_{[5:3]} \quad \text{error in sub-adder 3} \tag{2.7}$$

Note that the (2.7) does not include all the error conditions of sub-adder three, since it can give incorrect results also when (2.7) is not asserted, due to errors happened in previous sub-adders; this is the case of Fig. 4. The error conditions in the other sub-adders of Fig. 4 can be determined similarly.

Let us introduce a general Boolean equation for the error flag *Eu* [31]*:*

$$E_u = \sum_{i=2}^{M+1} p_{[(i-1)r+p-1:(i-2)r+r]}g_{[(i-1)r-1:(i-2)r]} \tag{2.8}$$

where

$$M = \left\lfloor \frac{n-p}{r} \right\rfloor \tag{2.9}$$

is the number of sub-adders and the $\sum$ symbol represents the logical OR. It is worthwhile observing that (2.8) is a necessary and sufficient error condition.

2.2.1.1 Error rate analysis

In [28] the error probability of speculative adders is expressed by means of a complex model. An approximate closed-form equation can be found following an approach similar to [18].

Let us observe that the first sub-adder, computing *r*+*p* outputs, is exact, therefore the error can occur in the remaining *M-1* sub-adders. For instance, in Fig. 3, we have an error in the second

sub-adder if (i) there is a carry-out coming from bit position #3 and (ii) this carry propagates across bits #4,5,6. In general for uniformly distributed operands, it can easily be demonstrated that condition (i) occurs with a probability of $(1/2)(1-2^{-r})$, while condition (ii) occurs with probability 2^{-p}. The error probability of each sub-adder is hence $\approx(1-2^{-r})2^{-(p+1)}$ and the probability of having an exact result is $\approx 1-(1-2^{-r})2^{-(p+1)}$. Since we have *M-1* inexact sub-adders, the overall error probability writes as:

$$p_{uniform} \simeq 1-\left(1-2^{-(p+1)}\left(1-2^{-r}\right)\right)^{M-1} \tag{2.10}$$

The data in Tab. I show that (2.10) gives results very close to the exact model of [28].

2.2.2 State of art

Different implementations of speculative adders have been proposed in the literature. In [20] the Almost Correct Adder (ACA-I) is implemented with *r=1* and exploits some hardware sharing between overlapping sub-adders. In [29] the Error Tolerant Adder Type II (ETA-II) is proposed and can be considered, in the representation of Fig. 3, as a speculative adder with *r=p*. In [18] the Accuracy Configurable Approximate Adder (ACAA) can be configured at runtime by changing circuit structure in order to tune accuracy; also in this case we have *r=p* in each sub-adder. The Speculative Carry-Select Adder (SCSA) of [30] also uses sub-adders and a similar carry prediction mechanism as ETA-II. The low-latency Generic Accuracy Configurable Adder (GeAr) [28] increases the design space by removing constraints on adder decomposition of previous approaches.

TABLE I. ERROR PROBABILITY FOR UNIFORMLY DISTRIBUITED INPUTS

Adder Config. (*n*,*r*,*p*)	**Error probability [28]**	**Error probability (2.10)**
(12,4,4)	2.9297%	2.9297%
(16,4,4)	5.770%	5.859%
(16,4,8)	0.1831%	0.1831%
(32,8,8)	0.3891%	0.3887%
(48,8,16)	0.0023%	0.0023%

The Carry Speculative Adder (CSPA) is proposed in [22] and implements a carry-select speculative adder, composed by sub-adders as shown in the general scheme of Fig. 3, with $r=p$.

2.3 Parallel-Prefix adders

In this paragraph parallel-prefix adders are briefly recalled, in order to introduce, in the following of the chapter, the proposed speculative parallel-prefix adders topologies.

The binary addition problem can be formulated as follows: given an n-bit augend $A=a_{n-1}a_{n-2}...a_0$ and an n-bit addend $B=b_{n-1}b_{n-2}...b_0$ generate the n-bit sum $S=s_{n-1}s_{n-2}...s_0$. Let us indicate as c_i the carry out of the i-th bit. The sum bit s_i and the carry c_i can be computed as follows:

$$s_i = a_i \oplus b_i \oplus c_{i-1} \tag{2.11}$$

$$c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1} \tag{2.12}$$

In prefix addition we use three stages to compute the sum: pre-processing, prefix-processing and post-processing.
In the pre-processing stage the generate g_i and propagate p_i signal are computed as in (2.2) and (2.3). The concept of generate and propagate can be extended to a block of contiguous bits, as shown in (2.4) and (2.5). In particular the signals defined by (2.4) are called *block generate* while the ones in (2.5) are called *block propagate*. Thus, for any bit i, the carry c_i can be expressed as:

$$c_i = g_{[i:0]} + p_{[i:0]} c_{-1} \tag{2.13}$$

where c_{-1} is the input carry of the n-bit adder. In the following, for the sake of simplicity, we assume that $c_{-1}=0$, so that (2.13) simplifies as:

$$c_i = g_{[i:0]} \tag{2.14}$$

The block generate and propagate terms are computed in the prefix-processing stage of the adder. To that purpose, the $(g_{[i:k]}, p_{[i:k]})$ couples are expressed with the help of the prefix operator ● defined as follows:

$$\left(g_{[i:k]}, p_{[i:k]}\right) = \left(g_{[i:j]}, p_{[i:j]}\right) \bullet \left(g_{[l:k]}, p_{[l:k]}\right) = \left(g_{[i:j]} + p_{[i:j]} g_{[l:k]}, p_{[i:j]} p_{[l:k]}\right) \tag{2.15}$$

where: $i \geq l \geq j \geq k$. The prefix operator has two important properties: it is associative and it is idempotent. These properties are exploited in the prefix-processing stage to speed-up the computation.

Finally, in the post-processing stage, the sum bit s_i are computed using (2.13) and:

$$s_i = p_i \oplus c_{i-1} \tag{2.16}$$

2.3.1 Kogge-Stone

The Kogge-Stone parallel-prefix stage [32] is shown in Fig. 5.

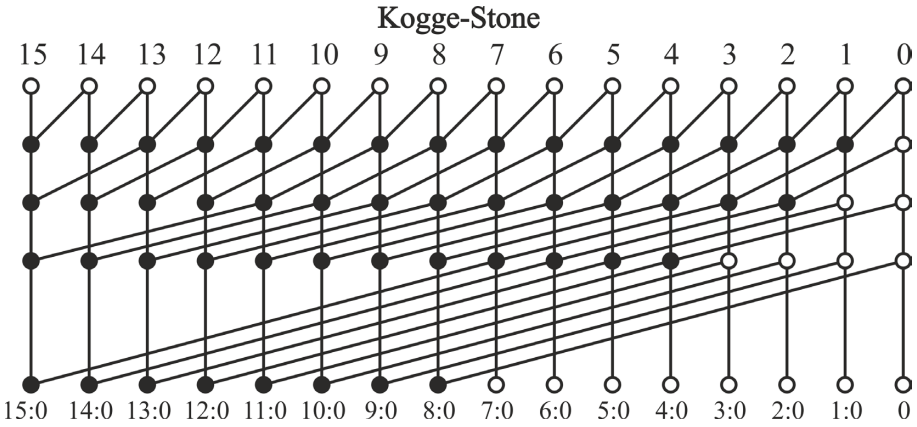

Fig. 5 *n=16* Kogge-Stone prefix-processing stage.

Here black dots represent the prefix operator (2.15), while white dots are simple placeholders. Kogge-Stone adder is composed by $\log_2(n)$ levels, and present a fanout of two, using a large number of cells and many wire track.

2.3.2 Brent-Kung

The Brent-Kung [33] adder topology is shown in Fig. 6. The number of black dots is lower than Kogge-Stone, while still presenting a fanout of two. This is achieved using additional levels. The total number of levels is $2\log_2(n)$-1.

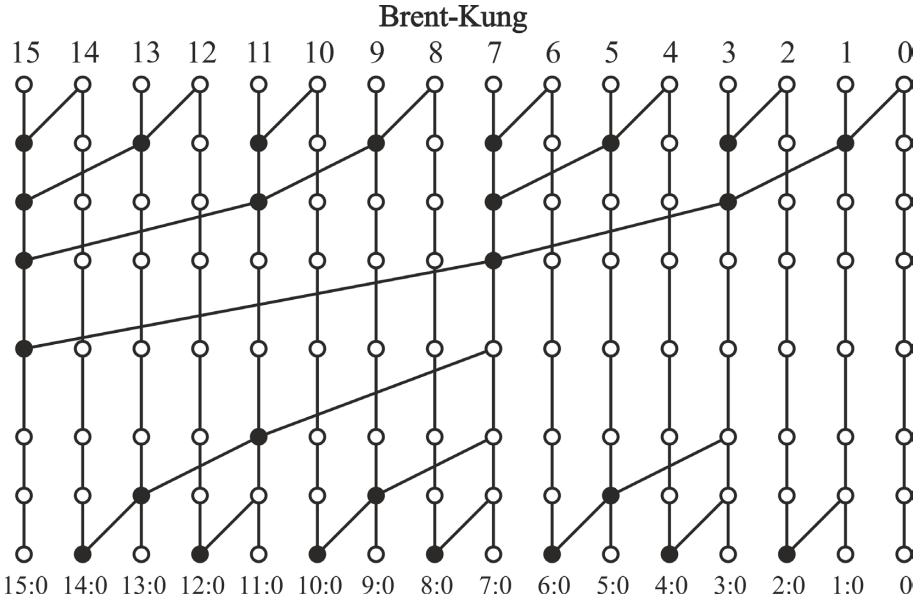

Fig. 6 *n=16* Brent-Kung prefix-processing stage.

2.3.3 Han-Carlson

A good trade-off between fanout, number of logic levels and number of black cells is given by Han-Carlson (Fig. 7). The outer rows of the Han-Carlson [34] topology are Brent-Kung graphs, while the inner rows are Kogge-Stone graphs. Han-Carlson adder exhibits an additional level with respect Kogge-Stone, being their total number equal to $\log_2(n)+1$.

2.3.4 Hybrid Han-Carlson

The Hybrid Han-Carlson [35] (Fig. 8) further decreases complexity with respect Han-Carlson, at a cost of two additionals levels with respect Kogge-Stone adder. The two outhers rows are Brent-Kung graphs, while the inner ones are Kogge-Stone graphs.

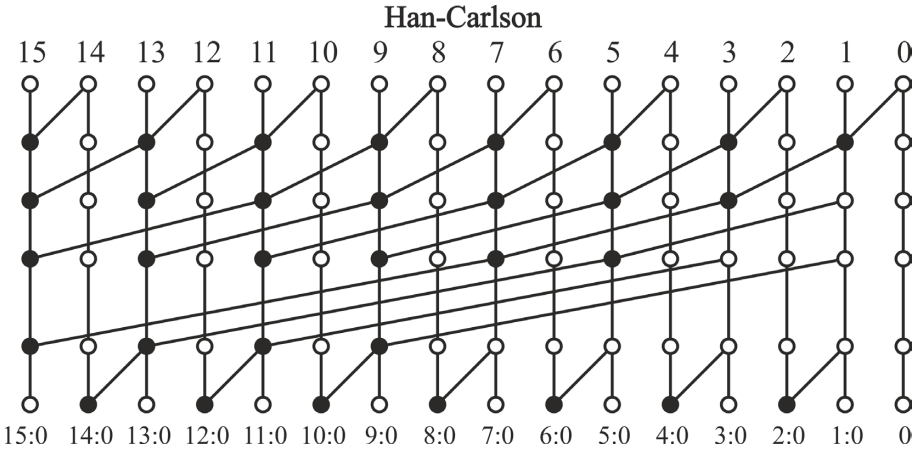

Fig. 7 *n=16* Han-Carlson prefix-processing stage.

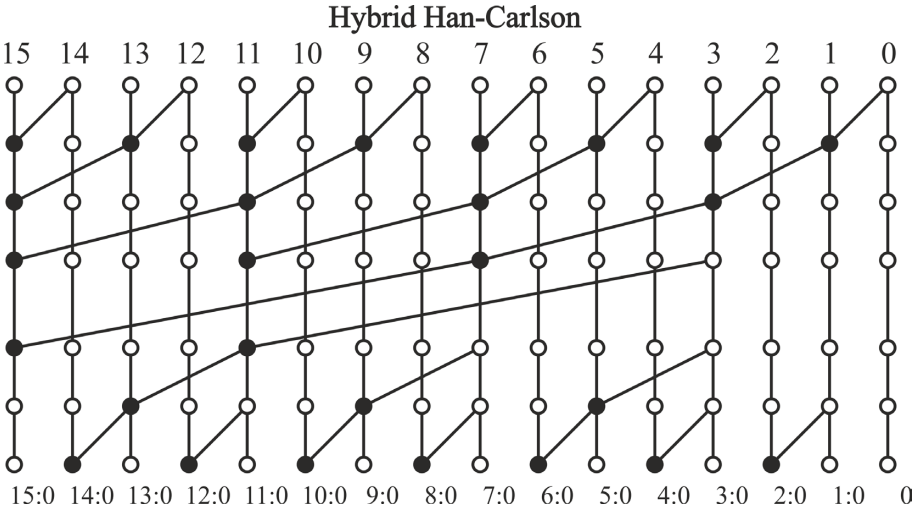

Fig. 8 *n=16* Hybrid Han-Carlson prefix-processing stage.

2.3.5 Sklansky

Sklansky [36] uses the minimum number $\log_2(n)$ of levels, but the fanout of black cells (implementing the prefix operator (7)) double at each level (Fig. 9).

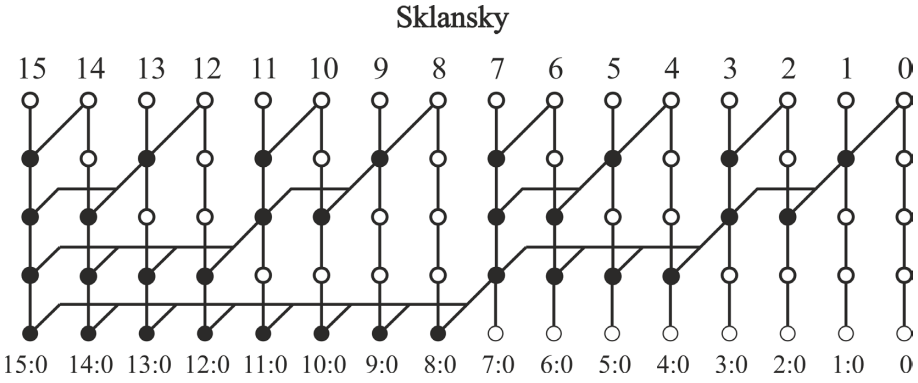

Fig. 9 *n=16* Sklansky prefix-processing stage.

2.3.6 Ladner-Fisher

Ladner-Fisher [37] (Fig. 10) adder represents and intermediate topology between Skalansky and Brent-Kung. Indeed, the first two level are Brent-Kung graphs, the intermediate levels are Sklansky graphs (with consequent increased fanout from one level to the successive) while the last one is a carry merge level, common in Brent-Kung, Han-Carlson and Hybrid Han-Carlson topologies. The toal number of levels is $\log_2(n)+1$.

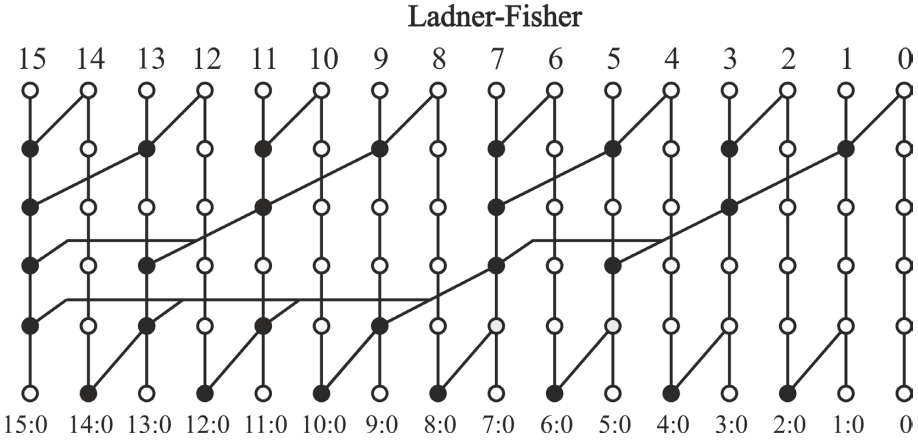

Fig. 10 *n=16* Ladner-Fisher prefix-processing stage.

2.3.7 Carry-Increment

Fig. 11 shows an example of a carry increment (C-I) adder (this is the prefix adder corresponding to the carry select architecture). The input operands are divided into w size groups (w=4 in Fig. 11), each one using a Kogge-Stone topology in this example. The result of a group is propagated to all the bits of the next group using a string of black cells aligned on the same row. The total number of levels is $\log_2(w) + n/w - 1$.

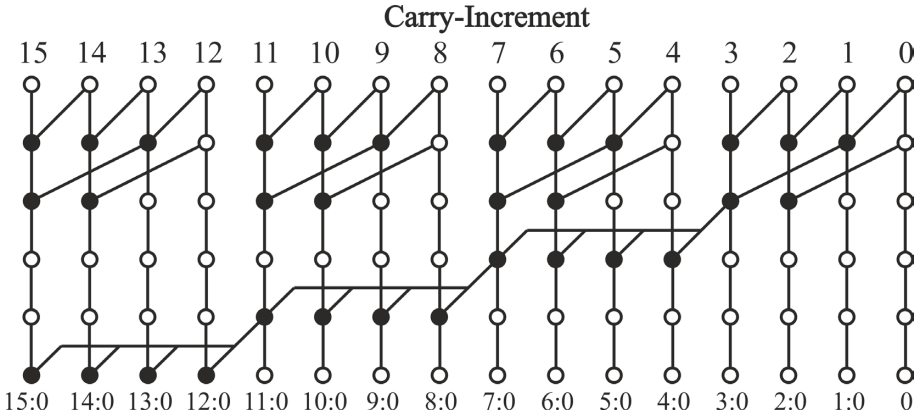

Fig. 11 *n=16* Carry-Increment prefix-processing stage.

2.4 Variable-Latency Speculative Adders

The general scheme of Variable-Latency Speculative Adders (VLSA) is shown in Fig. 12 [20], [22]. In this scheme, an approximate adder is augmented of an error detection network, as that discussed in paragraph 2.2.1, and of an error correction circuitry. The error detection asserts the signal E when the speculation fails. In this case an additional clock cycle is required to provide a correct result, through the error correction block. Therefore, the addition time is one clock cycle when speculation is correct, and two clock cycles when the speculation fails. We can define the *average addition time* T_{avg}, as follows:

$$T_{avg} = P_{Err} \cdot 2 \cdot T_{clk} + (1 - P_{Err}) \cdot T_{clk} = T_{clk}(1 + P_{Err}) \qquad (2.17)$$

where T_{clk} is the clock period and P_{Err} is the error probability of the speculative adder.

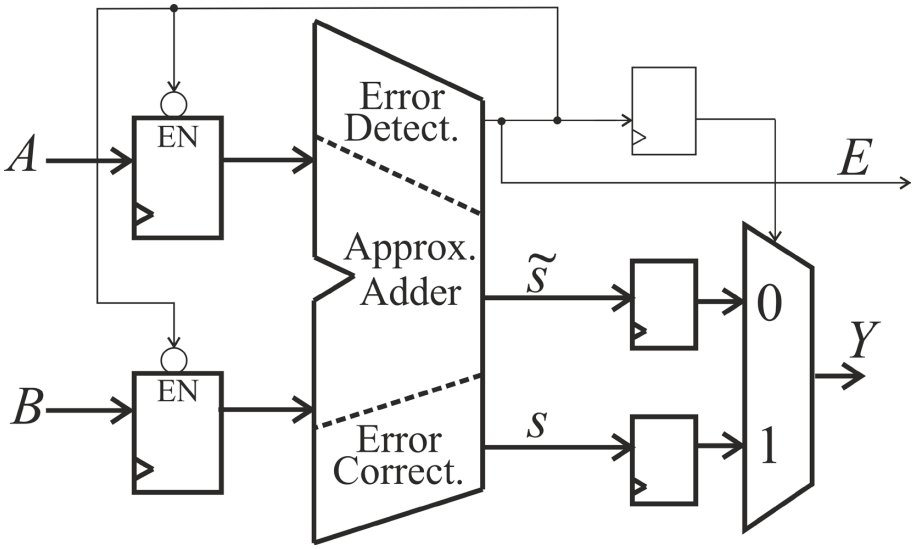

Fig. 12 General scheme of variable-latency speculative adder. When the speculative sum is incorrect, the *E* signal is asserted. The pipeline is therefore stalled and, in an additional clock cycle, the correct result is provided as output, with the help of an error correction stage.

Variable latency speculative prefix adders can be subdivided in five stages: pre-processing, speculative prefix processing, post-processing, error detection and error recovery. The error recovery stage is off the critical path, as it has two clock cycles to obtain the exact sum when speculation fails. In the following, this different stages are introduced.

2.4.1 Pre-processing

In the pre-processing stage the generate g_i and propagate p_i signals are computed as in (2.2) and (2.3). Note that this stage is employed also in classical, non-speculative, parallel-prefix adders, as already discussed in the paragraph 2.3.

2.4.2 Speculative prefix-processing

The speculative prefix-processing stage is one of the main difference compared with the standard prefix adders recalled in previous paragraph. Instead of computing all the $g_{[i:0]}$ and $p_{[i:0]}$ required in (2.13) to obtain the exact carry values, only a subset of block generate and propagate signals is calculated; in the post-processing stage approximate carry values are obtained from this subset. The output of the speculative prefix-processing stage will also be used in the error detection and in the error recovery stages discussed in the following.

As discussed in the paragraph 2.2, the basic assumption behind speculative prefix-processing stage is that carry signals propagate for no more than p bits, with $p<n$ and $p=O(log_2(n))$.

2.4.2.1 Han-Carlson topology

Han-Carlson (H-C) adder constitutes a good trade-off between fanout, number of logic levels and number of black cells. Because of this, Han-Carlson adder can achieve equal speed performance respect to Kogge-Stone adder, at lower power consumption and area [38]. Therefore it is interesting to implement a speculative Han-Carlson adder.

Moved by these reasons, we have generated a Han-Carlson speculative prefix-processing stage by deleting the last rows of the Kogge-Stone part of the adder [39]. As an example, the Fig. 13 shows the Han-Carlson adder of Fig. 7 in which the two Brent-Kung rows at the beginning and at the end of the graph are unchanged, while the last Kogge-Stone row is pruned. As shown in Fig. 13, the 9 rightmost output are exact (i.e. the values of $g_{[i:0]}$ and $p_{[i:0]}$ are calculated according to (2.13), for $i \in \{0,1,..,8\}$), the other outputs are instead speculative, since the obtained block-propagate and generate span only to a subset of the inputs. As an example, the tree in Fig. 13 yields $g_{[12:4]}$ and $p_{[12:4]}$, instead of $g_{[12:0]}$ and $p_{[12:0]}$ (Fig. 7).

In the following, we indicate as h the number of exact output computed by the speculative carry-tree and as l_j the block length of

the *j-th* speculative output (as an example, in Fig. 13 we have $h = 9$ and $l_{12} = 9$).

As shown in Fig. 13, the speculative outputs can be grouped, the member of the same group having a common black node parent (for example the output $(g_{[11:4]},p_{[11:4]})$ and $(g_{[12:4]},p_{[12:4]})$ have the same black node parent highlighted as 11:4).

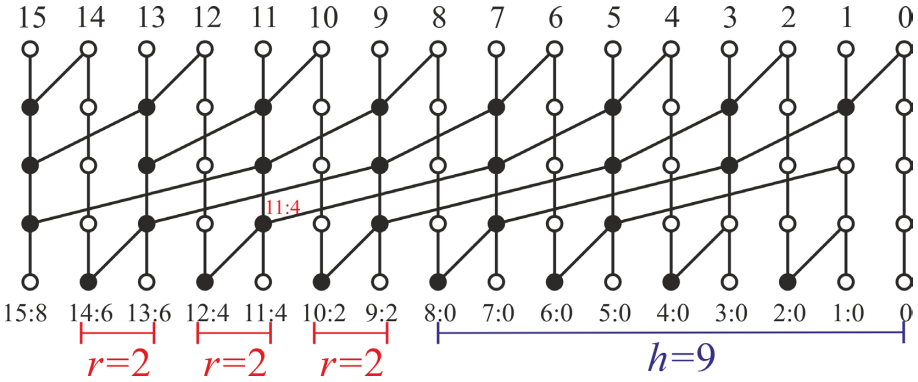

Fig. 13 Han-Carlson speculative prefix-processing stage. The last Kogge-Stone row of the n=16 bit graph is pruned.

The number of outputs belonging to the same group is indicated as r (conceptually is the same of the genral scheme of Fig. 3) in the following ($r=2$ in the example of Fig. 13). As it can be observed, the block length for the outputs of a same group varies from a minimum ($p_{min}=8$ in Fig. 13) to a maximum (p_{max}=9 in Fig. 13). We define:

$$p_{min} = \min(l_j);\ p_{max} = \max(l_j) \quad \text{for:} \quad j = \{h, h+1, ..., n-1\} \tag{2.18}$$

We have:

$$p_{max} = h = p_{min} + r - 1 \tag{2.19}$$

The last condition holds for all the prefix-processing stages investigated in the following, being related to the intrinsic symmetry of the graph.

2.4.2.2 Kogge-Stone topology

The Kogge-Stone (K-S) speculative prefix-processing stage has been proposed in [20], [40] and can be obtained by pruning the last levels of a traditional Kogge-Stone adder. In the example shown in

Fig. 14, the last level of a n=16 bit Kogge-Stone adder (Fig. 5) is pruned.

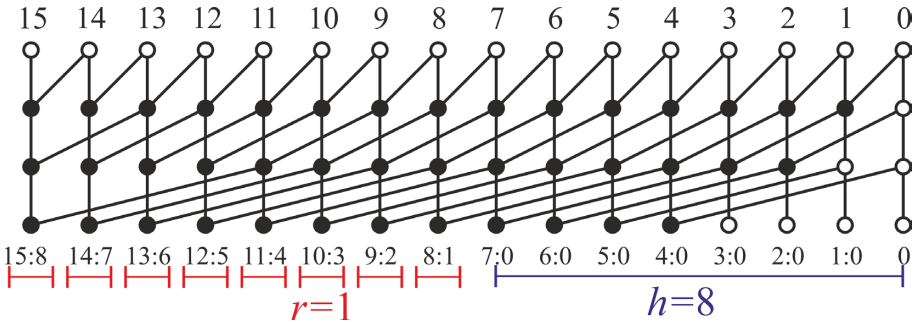

Fig. 14 Kogge-Stone speculative prefix-processing stage. The last row of a n=16 bit Kogge-Stone adder is pruned, resulting in a speculative prefix-processing stage with K=8.

The speculative Kogge-Stone tree of Fig. 14 is characterized by a $p_{min} = 8$, $h = 8$ and $r = 1$.

2.4.2.3 Brent-Kung topology

A speculative prefix-processing stage can be obtained by deleting the intermediate rows of the standard Brent-Kung (B-K) parallel-prefix graph [31]. Pruning the mid row eliminates just a single black cell and breaks the propagate chain only for the most significant bit; more interesting results are obtained by deleting a larger number of rows [39].

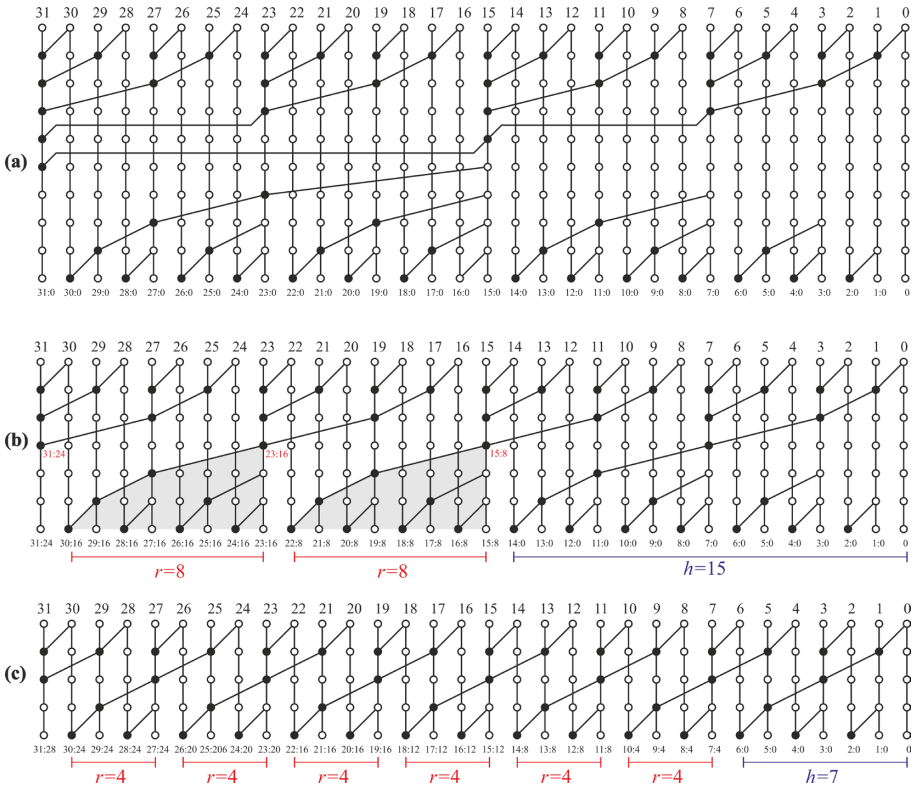

Fig. 15 32-bit Brent-Kung prefix-processing stages (a) Original topology; (b) three intermediate rows of the original graph are deleted; (c) five rows deleted.

As an example, Fig. 15 shows how can be obtained a 32-bit speculative B-K adder. Fig. 15(a) shows the original 32-bit B-K adder, while Fig. 15(b) displays what happens when three intermediate rows of the tree are deleted. In Fig. 15(b) we have h=15, p_{min}=r=8, l_{18}=11. The Fig. 15(c) shows the 32-bit B-K adder where five intermediate rows are deleted. In this case we have: h=p_{max}=7, r=4, p_{min}=4.

2.4.2.4 Hybrid Han-Carlson topology

The Hybrid Han-Carlson (HH-C) speculative prefix-processing stage is generated by keeping unchanged the rows at the beginning and at the end of the graph while deleting intermediate rows [41]. As an example, the Fig. 3 shows a 32-bit HH-C adder in which a single row is pruned, yielding a speculative stage with h=p_{max}=19, r=4, p_{min}=16.

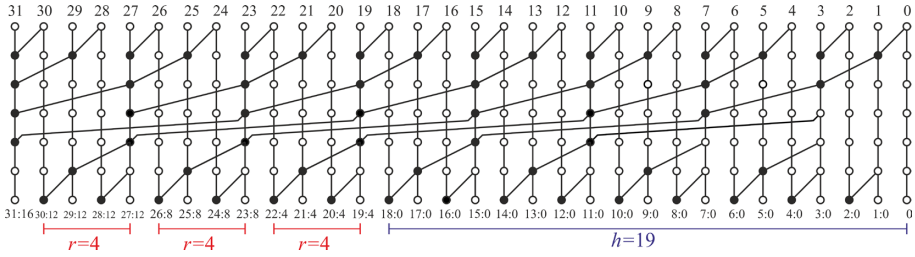

Fig. 16 32-bit speculative Hybrid Han-Carlson prefix-processing stage. The last Kogge-Stone row of the original graph is deleted

2.4.2.5 Carry-Increment topology

Fig. 17 shows the speculative version of the Carry-Increment (C-I) adder of Fig. 11. As it can be observed, the output of each group is propagated only to the next group (i.e. the third group is not linked to the first one; the fourth group is not linked to the second one and so on). In this way, the number of logic level reduces from 5 to 3. The speculative stage in Fig. 17 has h=7, r=4, p_{min}=4.

In general, the number of pruned levels P depends on group size w, and is equal to: $P=n/w-2$ [39].

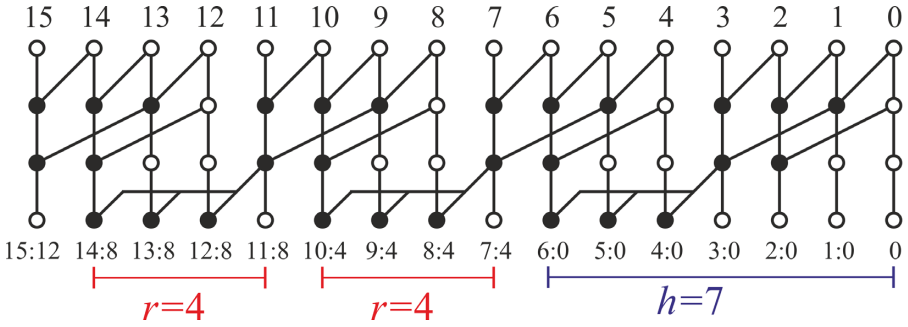

Fig. 17 Speculative architecture of $n=16$ Carry-Increment adder: the output of each group is propagated only to the next one

2.4.2.6 Ladner-Fischer topology

A speculative Ladner-Fischer (L-F) topology is obtained by deleting the intermediate levels between these ones and modifying the last survivor level [39]. As an example, the Fig. 18 shows a 16-bit L-F

adder in which the last intermediate level is pruned, yielding a speculative adder with h= 9, r=4, p_{min}=6.

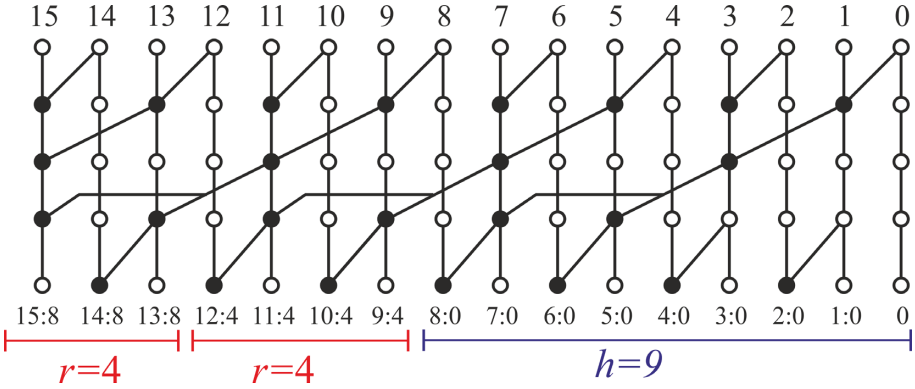

Fig. 18 16 bit Ladner-Fisher speculative prefix-processing stage.

2.4.2.7 Sklansky topology

The Sklansky (SK) speculative prefix-processing stage is obtained by deleting the last levels of the non-speculative topology. Fig. 19 shows a 16-bit example with h= 8, r=4, p_{min}=5

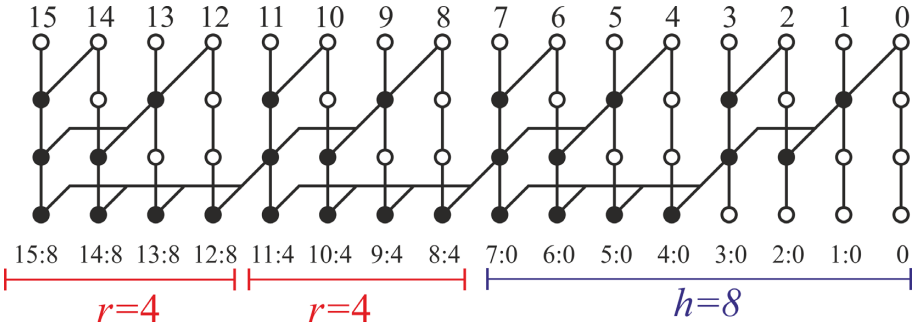

Fig. 19 16 bit Sklansky speculative prefix-processing stage.

2.4.2.8 Discussion

Table II summarizes the relations between p_{min}, r, the number of pruned levels P and the adder size n (h and p_{max} can be obtained from (2.19)). As it can be observed, p_{min} is related to n and P, with the exception of C-I topology, where p_{min} is equal to group size w. The parameter r is fixed for K-S, H-C and HH-C speculative topologies, while in B-K, C-I, L-F and SK it depends on p_{min}.

The number of levels of the speculative prefix-processing graphs is also reported in Table II. As it can be observed, a logarithmic behavior with p_{min} is exhibited.

2.4.3 Post-processing

In the post-processing stage we firstly compute the approximate carries, $\tilde{c}_i$, and then use them to obtain the approximate sum bits $\tilde{s}_i$ as follows:

$$\tilde{s}_i = p_i \oplus \tilde{c}_{i-1} \tag{2.20}$$

Similarly to (2.14), the approximate carries are obtained as the generate signals available in the last level of the prefix-processing stage. We have:

$$\tilde{c}_i = \begin{cases} g_{[i:0]} & \text{for: } i < h \\ g_{[i:i-l_i+1]} & \text{otherwise} \end{cases} \tag{2.21}$$

2.4.4 Error detection

The speculative carries are calculated making the following assumption:

$$c_i \textit{ results from propagation through no more than } l_i \textit{ bits} \tag{2.22}$$

In this way, for Kogge-Stone topology we assume that carry can

TAB. II –SPECULATIVE PREFIX STAGE PARAMETERS.

Topology	L_{min}	r	Number of Levels
Kogge-Stone	$n/2^P$	1	$\log_2(bL_{min})=\log_2 n - P$
Han-Carlson	$n/2^P$	2	$\log_2(bL_{min})=\log_2 n - P + 1$
Hybrid Han-Carlson	$n/2^P$	4	$\log_2(bL_{min})=\log_2 n - P + 2$
Brent-Kung	$n/2^{(P+1)/2}$	L_{min}	$\log_2(bL_{min})=2\log_2 n - P - 1$
Carry-Increment	w	L_{min}	$\log_2(2L_{min})=\log_2 w + 1$
Ladner-Fischer	$2+n/2^{(P+1)}$	L_{min}-2	$\lceil \log_2(2L_{min}) \rceil = \log_2 n - P + 1$
Sklansky	$1+n/2^{(P+1)}$	L_{min}-1	$\lceil \log_2(L_{min}) \rceil = \log_2 n - P$

propagate for no more than L_{min} bits, while for other topologies with $r>1$ we can tolerate, for some bit positions, carry propagation lengths longer than L_{min}, but always smaller than L_{max}.
The error condition E_u for the proposed topologies is expressed by (2.8), where, in (2.9) p must be interpreted as p_{min}.

Let us investigate, with some examples, the error detection logic for the proposed topologies.
For the Kogge-Stone topology of Fig. 14, the *Eu* is:

$$E_{u_{K-S}} = p_{[15:8]}g_7 + p_{[14:7]}g_6 + p_{[13:6]}g_5 + p_{[12:5]}g_4 + p_{[11:4]}g_3 + p_{[10:3]}g_2 + p_{[9:2]}g_1 + p_{[8:1]}g_0 \quad (2.23)$$

Considering Han-Carlson of Fig. 13 we have:

$$E_{u_{H-C}} = p_{[9:2]}g_{[1:0]} + p_{[11:4]}g_{[3:2]} + \ldots + p_{[15:8]}g_{[7:6]} \quad (2.24)$$

Comparing (2.24) and (2.23) we can observe that the number of terms to be OR-ed in order to compute E_u is halved in Han-Carlson topology. This derives from r value: in Han-Carlson two carries are "speculated" from the same parent node, therefore it is sufficient to check the error of the parent node only. This is formally expressed by (2.9).

We name "checking nodes" the nodes of the prefix-processing stage, whose outputs are needed to compute the error signal. The checking nodes for both the Kogge-Stone example of Fig. 14 and the Han–Carlson example of Fig. 13 are highlighted as white-red cells in Fig. 20.
As it can be observed, in Kogge-Stone some of the checking cells are at the last level of the graph; their output signals are available after three black cells delay. In Han-Carlson the critical checking cells are in the second last level of the graph and are also available after three black cells delay, in spite of the larger number of levels of the Han-Carlson prefix-processing stage. From the above observations, it can be concluded that error detection is sensibly simplified and potentially faster in Han-Carlson, compared to Kogge-Stone.

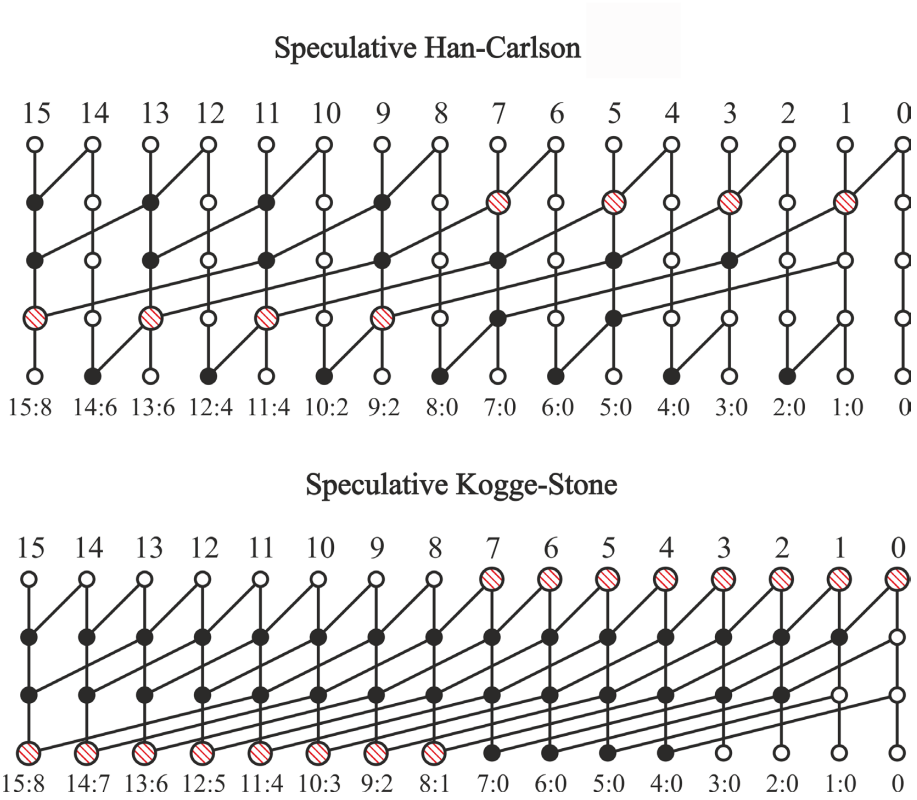

Fig. 20 The nodes of the prefix-processing stage, whose outputs are needed to compute the error signal, are named "checking nodes" and are highlighted as big hatched dots, for the topologies in Fig. 13-14.

In the case of Brent-Kung of Fig. 15(b) one has:

$$E_{u_{B-K}} = p_{[15:8]}g_{[7:0]} + p_{[23:16]}g_{[15:8]} + p_{[31:24]}g_{[23:16]} \tag{2.25}$$

whereas for the circuit of Fig. 15(c), with M=7 (2.9), we have:

$$E_u = p_{[7:4]}\, g_{[3:0]} + p_{[11:8]}\, g_{[7:4]} + p_{[15:12]}\, g_{[11:8]} + \cdots + p_{[27:24]}\, g_{[23:21]} + p_{[31:28]}\, g_{[27:24]} \tag{2.26}$$

In general, among the investigated topologies, the K-S one exhibits the most complex error detection stage, involving the OR of n-p_{min} terms. The H-C halves the number of terms to be OR-ed, while HH-C topology further decrease complexity, needing n-$p_{min}/4$ terms. Brent-Kung, Carry-Increment, Sklansky and Ladner-Fischer

speculative topologies shows a variable r value, depending on L_{min} (see Tab. I), but they are general more effective than K-S.

Another important aspect to be considered is delay. In K-S some of the checking nodes are at the last level of the graph, and are hence late-arriving signals. The situation is better in H-C and C-I, where the critical checking nodes are in the second last level of the graph. The best configurations are, from this point of view, B-K and HH-C, in which the checking nodes are located in the middle of the tree [39].

As an additional note, the need of driving the gates of the error detection stage increases the fanout of the checking cells, slowing the speculative prefix-processing stage.

2.4.5 Error Correction

The error correction stage computes the exact carry signals (2.14), to be used to be used for $i > h$, in case of misprediction.

The error correction stage is composed by the levels of the prefix-processing stage pruned to obtain the speculative adder. The Fig. 21 shows the error correction stage of the proposed speculative Han-Carlson adder; the error correction of the others topologies can be obtained similarly.

It can be observed that the inclusion of the error correction stage increases the fanout of some of the cells of the speculative prefix-processing stage, with adverse effect on adder speed.

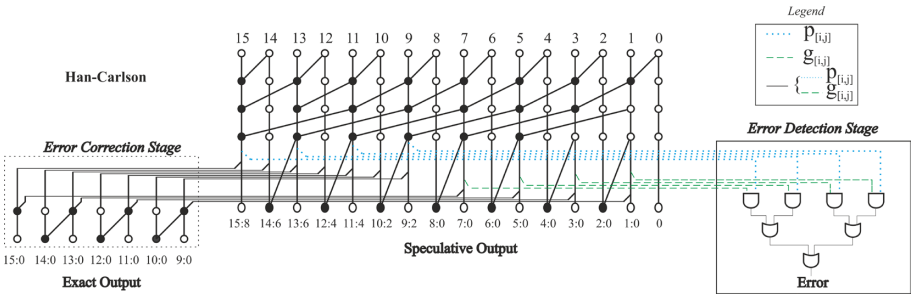

Fig. 21 Error correction and detection stages for the proposed speculative Han-Carlson adder of Fig. 13.

2.5 Signed operands

Unfortunately, the assumption made on operands statistics are not always verified. Indeed many applications make use of 2's complement representation, in which, due to the sign extension, long carry propagations can arise more likely than the case of uniformly distributed inputs.

This issue is exacerbated, as discussed in [42], by the fact that, in practical applications using 2's complement representation, small numbers appear more frequently than large ones. As an example, the Fig. 22 shows the probability density function for an audio signal (a 13s fragment of a pop song with 16-bit 2's complement encoding). As it can be observed the signal is far from being normally distributed and instead it follows closely a Gaussian shape. This comes in agreement with [30], which assumes a Gaussian distribution to capture the behavior observed in [42], through software profiling.

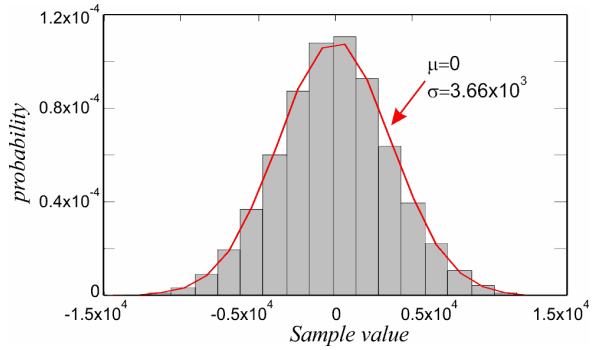

Fig. 22 Probability density function for a 13s fragment of a pop song (16-bit 2's complement encoding). The red curve is a Gaussian fit.

In presence of distribution like that in Fig. 2, long carry propagation can arise when summing two small number with opposite signs. In this condition the carry propagation length is comparable with the adder size, significantly increasing the error rate.

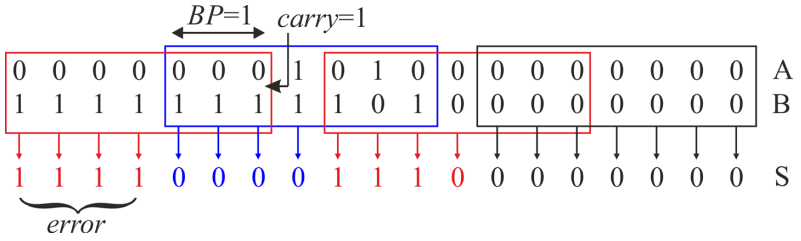

Fig. 23 Operation of the speculative adder of Fig. 3, when the two operand values are: A=2688_{10} and B=-704_{10}. An error occurs in the four MSBs of the result.

The Fig. 23 reports an error condition occurring when two operands, with opposite signs, are added using a speculative adder. In this figure the two operands are: A=2688_{10} and B=-704_{10}. As it can be observed, (i) the carry propagates along the most-significant r+p bits of the adder (r+p=6 in the example of Fig. 23) and (ii) there is a carry-out from bit position n-(r+p)-1 (i.e. from bit position #11 in the example of Fig. 23). Thus an error occurs and the adder output is computed as: Y=-30784_{10} instead: of 1984_{10}.

2.5.1 Error rate analysis

In this section a lower bound for error rate of speculative adders in presence of Gaussian distributed operands is shown.

An error condition like that represented in Fig. 23 can be summarized as:

$$\begin{cases} 0 < A < Q \\ -Q < B < 0 \\ A > B \end{cases} \tag{2.27}$$

where

$$P = 2^{n-(r+p)} \tag{2.28}$$

In fact, since 0<A<Q and -Q<B<0 the r+p MSBs of A are all zeros and the r+p MSBs of B are all ones (and the carry propagates along the most-significant bits of the adder). Moreover, since the result is positive (A>B) there is certainly a carry-out from bit position n-(r+p)-1. Another error condition occurs also when the roles of A and B are swapped:

$$\begin{cases} 0 < B < Q \\ -Q < A < 0 \\ B > A \end{cases} \tag{2.29}$$

In conclusion, we have an error whenever A and B have opposite signs, $A+B>0$ and both $|A|$ and $|B|$ are smaller than P. The most evident case is: A=1 and B=-1 that, in the adder of Fig. 23, gives as a result -64.

The probability of the conditions (2.27) and (2.29) is quite small in the case of uniformly distributed operands (in this case the probability that the $r+p$ MSBs of an operand are all one or all zero is $2^{-(r+p)}$). Instead, when the operands are Gaussian distributed the probability of (2.27) and (2.29) can increase significantly.

Let us observe that the two conditions (2.27) and (2.29) happen with the same probability, therefore, naming p_A the probability of the case ($0<A<Q$, $-Q<B<0$, $A+B>0$), the probability p_1 of the conditions (2.27) and (2.29) is given by $p_1 = 2p_A$. The probability of $0<A<Q$, under our hypothesis writes as:

$$\Pr(0 < A < P) = \frac{1}{2} erf\left(\frac{Q}{\sigma\sqrt{2}}\right) \tag{2.30}$$

where *erf* is the error function and σ is the standard deviation of the Gaussian distribution (assumed to have zero mean). Moreover, using Gaussian symmetry, we have

$$\Pr(-P < B < 0) = \Pr(0 < A < B) \tag{2.31}$$

while the probability of A+B>0 is equal to 1/2. By using the proprieties of jointly independent Gaussian variables (the operands are assumed uncorrelated) and exploiting the symmetry of the problem, the probability $p1$ can be expressed as the product of the single events [43]:

$$p_1 = 2p_A = \frac{1}{4} erf^2\left(\frac{Q}{\sigma\sqrt{2}}\right) \tag{2.32}$$

It is important to note that (2.32) is only a lower bound of the actual error probability, since there are other error conditions not included in (2.27), (2.29). For instance, an error can occur also in the case when

A>Q and *B*<-*Q*, or even independently from the sign and the size of the operand. Nevertheless, these additional error components are often negligible compared to (2.32).

The Fig. 24 shows simulated error probabilities for three different adder configurations. For each adder, several simulations with Gaussian distributed operands have been performed, by varying the standard deviation σ of the distribution. One million randomly generated inputs have been simulated for each σ value. As it can be observed, the simulated error probability is much larger compared to the case of uniformly distributed inputs. The equation (2.32) predicts fairly well the numerical results; it can be observed that $p_{gaussian} \rightarrow 0.25$ when $Q/\sigma >> 1$ that is: $\sigma << Q$. The largest deviations between (2.32) and numerical simulations occur in the case n=16, r=4, p=4 where the adder error probability is quite large (about 6%) also for uniformly distributed inputs.

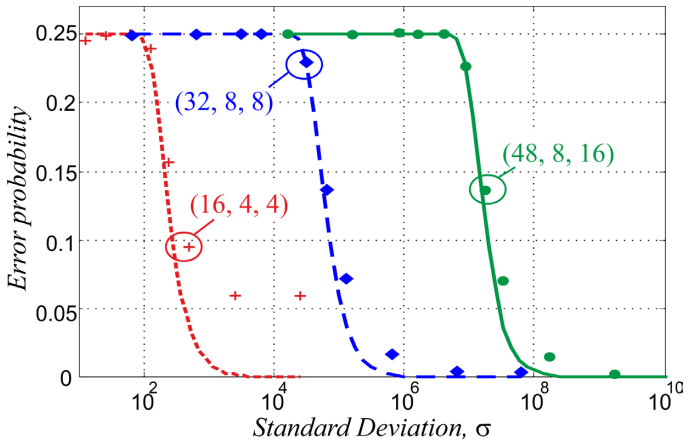

Fig. 24 Error probability for three adder configurations, with Gaussian input distribution. The number in brackets are: *n,r,p* respectively. Lines: equation (2.32); dots: simulations.

2.6 Variable-Latency Speculative Adders for signed operands

In presence of Gaussian distributed operands, as discussed above, the average carry propagation length increases significantly,

therefore the assumption, made in the case of uniformly distributed operands, that the carry propagates no more than $p<n$ bits fails. As a consequence the error rate increases significantly (Fig. 24), degrading the average addition time (2.17).

In order to address this issue, both [30] and [42] introduce a global carry signal, triggered in presence of a carry chain longer than p. This global carry is taken into account in the approximate adder by using additional logic levels that negatively affects overall performance of the speculative adder.

2.6.1 Speculative assumption for signed operands

The speculative assumption (2.22), employed in the case of uniformly distributed operands, fails when working with signed operands. It is, therefore, needed to determine a novel assumption to mitigate the error rate increase.

In Fig. 25(a) (where h=7 is assumed). two decimal numbers 39 and –47 are summed and the carry is killed in bit #3; in this case the speculating condition (2.22) is verified and we can use (2.21) to compute carries without errors.

On the other hand (Fig. 25(b)) summing the two decimal numbers 47 and -39 the carry generates in bit #3 and then propagates up to the most significant bit. The speculating condition (2.22) fails in this case, which is flagged as an error, with E_u=1. Note that in this case the carries from c_3 through c_{n-1} are all one, while the sum bits from position 4 through n-1 are all zero.

Thus, to reduce error rate, we assume a different, less stringent, speculating assumption to obtain $\tilde{c}_i$ for $i \geq h$:

$$\begin{aligned} &c_i \text{ results from propagation through no more than } l_i \text{ bits} \\ &OR \\ &c_i = 1 \text{ due to carry generation in the less significant } h \text{ bits, followed by propagation through the most significant } n-1-h \text{ bits } (\text{i.e. from } h+1 \text{ to } n-1) \end{aligned} \tag{2.33}$$

If the first condition in (2.33) is true, then E_u in (2.8) is zero and we can use (2.21) to compute carries. This covers cases like the one in Fig. 25(a).

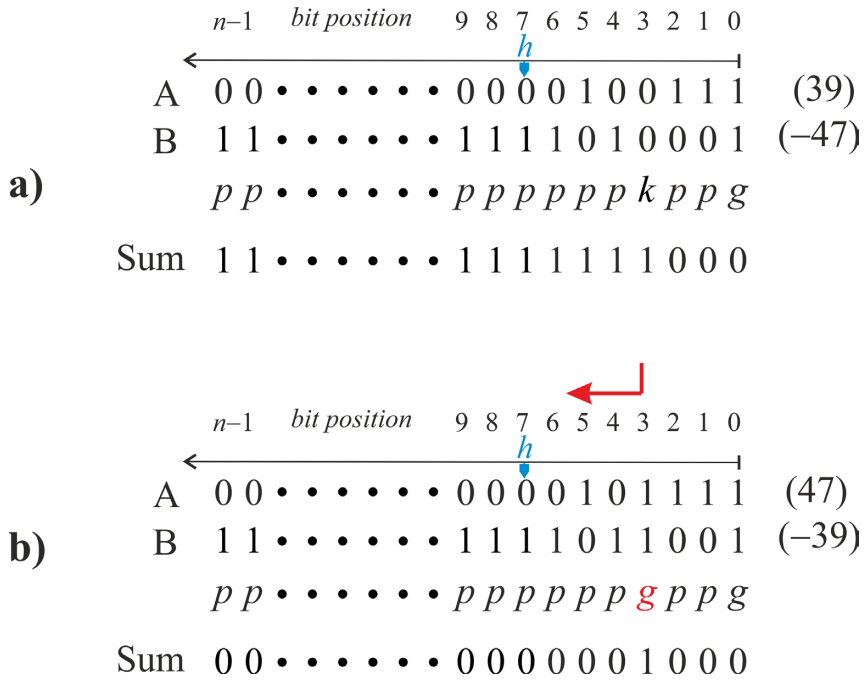

Fig. 25 Sum of two small operands, with opposite signs and absolute value smaller than 2^{h+1} (h=7 is assumed). (a) When the sum is negative (2.22) is verified and (2.21) gives carries without errors. (b) When the sum is positive (2.22) fails. The carries from position h to n-1 are high while the sum bits from position h+1 to n-1 are zero.

The second condition in (2.33) flags the presence of a long carry chain and covers cases like the one in Fig. 25(b). The second condition in (2.33) is verified when γ=1, where:

$$\gamma = E_u \cdot p_{[n-1:h+1]} \tag{2.34}$$

Let us consider, as an example, the B-K speculative prefix-processing stage of Fig. 15(b). From (2.25) and exploiting the condition: $p_{[i:j]}g_{[k:j]}=0$ for: $k \le i$, the equation (2.34) becomes:

$$\gamma = p_{[31:8]} \cdot \left[p_{[7:4]}\, g_{[3:0]} + p_{[11:8]}\, g_{[7:4]} \right] \tag{2.35}$$

The last equation can be rewritten as:

$$\gamma = p_{[31:8]} \cdot g_{[7:0]} \tag{2.36}$$

In general, it can be shown that:

$$\gamma = E_u \cdot p_{[n-1:h+1]} = p_{[n-1:h+1]} \cdot g_{[h:0]} \tag{2.37}$$

Therefore, if (2.34) is asserted, we know that all the carries from position h to n-1 are high. Moreover, since all p_i (2.3) are high for i=h+1..n-1, we also know that all the sum bits from position h+1 to n-1 are zero.

The cases not considered in (2.33) result in a misprediction and require two clock cycles for error correction. From the previous discussion, the error condition is given by

$$E_s = E_u \cdot \bar{\gamma} = E_u \cdot \overline{p_{[n-1:h+1]}} \tag{2.38}$$

The architecture of the resulting speculative adder is shown in Fig. 26. It can be seen that this architecture is more effective than those reported in [30], [42], because when condition (2.34) is asserted the computation of output bits is not needed (the sum bits from position h+1 to n-1 are zero).

As final remark, please note that condition (2.33) includes (2.22), therefore the resulting speculative adder can be used also in presence of uniformly distributed operands without no degradation in terms of error probability.

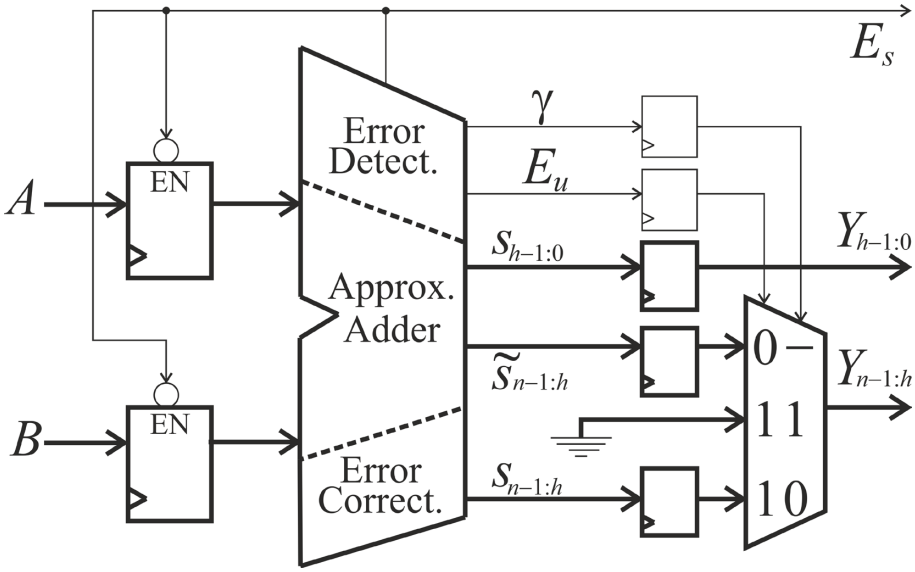

Fig. 26 Proposed Variable Latency speculative adder for applications using 2's complement representation. For E_u=1, γ=1 the speculative output is known beforehand to be 0.

2.7 Signed operands model

The operation of VLAs is based on specific assumption on the length of carry chains. The actual statistical distribution of input values, therefore, strongly affects error probability and hence the performances of VLAs. In [18], [20], [27]-[29], [39], [42], [45] an uniform distribution for the input operands is assumed, which is an acceptable approximation of practical cases when the input values represent unsigned numbers [46]. In [42] profiling of software programs running on a physical machine was carried-out in order to obtain statistics with real-world workloads, using 2's complement representation. In [30] it is found that assuming a Gaussian distribution allows capturing the basics behavior of carry chains length distribution obtained in [42].

In order to asses proposed topology with signed operands we employ mathematical distributions to approximate the distribution found in practical workloads. We assume that in a certain percentage of cases the operands A, B represent integer values uniformly distributed in $[-2^{n-1}, 2^{n-1}-1]$, while in the remaining cases they follow a Gaussian distribution with mean $\mu=0$ and standard deviation σ. Thus, the operands are obtained as follows [31]:

$$(A,B) \in \begin{cases} U(-2^{n-1},\ 2^{n-1}-1) & \text{in the } 100u\% \text{ of cases} \\ G(0,\sigma) & \text{in the remaining cases} \end{cases} \quad (2.39)$$

where U is the uniform distribution and G the normal one. For a given wordlength n, the model (2.39) has only two parameters, u and σ.

For $u=1$ each bit of both inputs A, B has an equal probability of being zero (this is the model to be used for workloads where input values represent unsigned numbers).

For $u<1$, the elements taken from the Gaussian distribution capture the carry chains behavior found in [42]. The value of σ is typically much smaller than 2^{n-1}, in order to represent operations that are performed between small operands (frequently found in practical workloads [42]).

Fig. 27 shows histograms relative to maximum carry propagation length in a 32-bit adder, obtained from (2.39) using 50,000 test vectors. Here a carry chain is defined as a generate

followed by propagates bit; the carry propagation length is k+1 whenever a generate followed by k propagates bit is found. The example of Fig. 25(a) has a carry propagation length of 3, while in Fig. 25(b) the maximum carry propagation length is n–3.

It can be observed that for uniform distribution the length of carry chain is always sensibly smaller than adder size. When 50% of inputs are taken from Gaussian distribution with σ=256 (Fig. 27(b)), a bimodal distribution is observed with an appreciable portion of carry chains is as long as the adder size; by increasing σ the second peak of the distribution moves to the left (Fig. 27(c)).

2.8 Error rate results

In this paragraph the error rate values of the investigated topologies are shown, assuming operands described by model (2.39). For VLSA the error rate assumes significant relevance, affecting the average addition time (2.17).

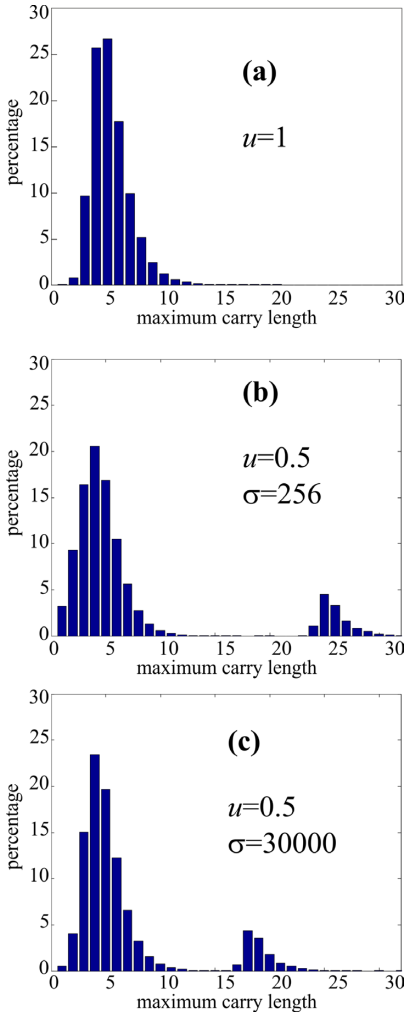

Fig. 27 Histogram of maximum carry propagation length obtained with model (2.39) , for a 32 bit adder. (a) uniform distribution; (b) half uniform, half Gaussian with σ=256; (b) half uniform, half Gaussian with σ=30000

The error rate values have been evaluated performing Monte Carlo simulations with a 2% relative error and a 99% confidence level. The simulated error probability values using the three distributions in Fig. 27 are reported in Table III for operand size n=32 and n=64. Error values larger than 10% are highlighted in red. For each topology, and for each n value, two VLSAs have been considered, with different p_{min} levels. The corresponding values of h=p_{max} and r are also reported in Table III. The grayed rows report

results for the H-C and K-S topologies designed to operate with unsigned operands (Fig. 12), to compare the error probability reduction, when operating with signed operands, of the topology proposed in Fig. 26.

Let us firstly focus on the column u=1, corresponding to uniformly distributed operands. The following observations can be drawn:

a) For uniformly distributed operands, the error probability of reported speculative adders assume low values. The maximum is given by Sklansky topology which exhibits a value of 8.65×10^{-2}. This is due by the value of p_{min} of this topology, being equal to 5, while the other topologies are generally reported with p_{min} values around 8. This show that the error probability as an exponential dependence on $p_{min,}$ indeed the reported error probability for Sklansky topology having $p_{min}=9$ decreases significantly to 1.93×10^{-3}. Each term OR-ed in (2.8), in fact, has a probability of being one that decreases exponentially with L_{min}. Moreover it can be observed that the topologies with the same p_{min} but higher r perform better in terms of error rate (compare in Tab. III, the H-C and K-S topology having the same p_{min}). This can be interpreted as follows: in Kogge-Stone speculative prefix stage all the carries are computed independently from each other, instead in Han-Carlson, half of the carries (those in even bit-positions) are calculated from "parents" carries (those in odd bit-positions), through an additional level of the tree. This reduces error probability (if a parent carry is correct the "child" carry will be correct, too) [39]. This is formally expressed by (2.9) showing that increasing r reduces the terms to be OR-ed in E_u (2.8). Observe that for uniformly distributed operands, the VLSAs of Fig. 12 have the same error rate of the corresponding circuits using the architecture of Fig. 26. Actually, for this distribution of input values, the probability of having $p_{[n-1:h+1]}=1$ is extremely low and hence, from (2.38), $E_s\approx E_u$.

From the last two columns in Table III, corresponding to distributions of Fig. 27(b) and Fig. 27(c), the following observations can be drawn:

TAB. III – ERROR PROBABILITY VALUES FOR SPECULATIVE ADDERS.

Adder Topology	**n**	p_{min}	p_{max}	r	**Error Probability**		
					u=1	u=0.5, σ=256	u=0.5, σ=30000
C-I	32	8	15	8	3.90×10^{-3}	1.95×10^{-3}	2.34×10^{-3}
Hyb. H-C	32	8	11	4	9.11×10^{-3}	4.60×10^{-3}	1.22×10^{-1}
		16	19	4	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$
B-K	32	8	15	8	3.90×10^{-3}	1.95×10^{-3}	2.34×10^{-3}
H-C	32	8	9	2	1.61×10^{-2}	8.12×10^{-3}	1.33×10^{-1}
		16	17	2	3.80×10^{-5}	$<10^{-5}$	$<10^{-5}$
K-S	32	8	8	1	2.25×10^{-2}	2.20×10^{-2}	1.38×10^{-1}
		16	16	1	5.36×10^{-5}	$<10^{-5}$	$<10^{-5}$
L-F	32	6	9	4	4.41×10^{-2}	2.21×10^{-2}	1.51×10^{-1}
		10	17	8	9.58×10^{-4}	4.78×10^{-4}	4.78×10^{-4}
S-K	32	5	8	4	8.65×10^{-2}	5.61×10^{-2}	1.79×10^{-1}
		9	16	8	1.93×10^{-3}	9.72×10^{-4}	9.80×10^{-4}
H-C	32	8	9	2	1.61×10^{-2}	1.35×10^{-1}	1.36×10^{-1}
		16	17	2	3.80×10^{-5}	1.25×10^{-1}	1.19×10^{-1}
K-S	32	8	8	1	2.25×10^{-2}	1.37×10^{-1}	1.40×10^{-1}
		16	16	1	5.36×10^{-5}	1.25×10^{-1}	1.19×10^{-1}
C-I	64	8	15	8	1.17×10^{-2}	5.80×10^{-3}	1.33×10^{-2}
		16	31	16	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$
Hyb. H-C	64	8	11	4	2.38×10^{-2}	1.18×10^{-2}	1.28×10^{-1}
		16	19	4	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$
B-K	64	8	15	8	1.17×10^{-2}	5.80×10^{-3}	1.33×10^{-2}
		16	19	16	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$
H-C	64	8	9	2	3.91×10^{-2}	1.96×10^{-2}	1.44×10^{-1}
		16	17	2	1.32×10^{-4}	$<10^{-5}$	$<10^{-5}$
K-S	64	8	8	1	5.27×10^{-2}	3.75×10^{-2}	1.53×10^{-1}
		16	16	1	1.82×10^{-4}	9.01×10^{-5}	9.32×10^{-5}
L-F	64	6	9	4	9.80×10^{-2}	5.00×10^{-2}	1.78×10^{-1}
		10	17	8	2.91×10^{-3}	1.45×10^{-3}	1.45×10^{-3}
S-K	64	9	16	8	5.91×10^{-3}	2.90×10^{-3}	2.91×10^{-3}
		17	32	16	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$
H-C	64	8	9	2	3.91×10^{-2}	1.45×10^{-1}	1.47×10^{-1}
		16	17	2	1.32×10^{-4}	1.25×10^{-1}	1.25×10^{-1}
K-S	64	8	8	1	5.27×10^{-2}	1.52×10^{-1}	1.54×10^{-1}
		16	16	1	1.82×10^{-4}	1.25×10^{-1}	1.25×10^{-1}

b) The error rate of VLSAs designed for unsigned operands (Fig. 12) drastically increases, ranging from 12% to 18%, and is almost independent from p_{min}. These architectures, therefore, are unsuitable for application using 2's complement representation.

c) The VLSAs designed for signed operands (Fig. 26) perform quite well with the distribution of Fig. 27(b). The error probability is actually lower than the case of uniformly distributed operands. Due to the low σ value, in fact, carry generation for Gaussian distributed operands takes place with high probability in the less-significant p_{max} bits of the adder and is correctly taken into account by the proposed architecture.

d) When the distribution of Fig. 27(c) is considered, the error probability increases significantly. In this case, due to the larger σ value, only the architectures with $p_{max} \geq 15$ have a high probability of catching the propagation chains arising for Gaussian distributed operands. This result clearly shows that the best architecture for a VLSA is strongly related to the assumption made on the input signal statistics.

2.9 Synthesis results

The investigated topologies have been described, along with their speculative counterparts, in Verilog HDL and synthesized with Cadence RTL Compiler using UMC 65nm library, for 32bit, 64bit and 128bit operands. The adders have been described in a structural way, instantiating operators (2.15). The error detection and correction networks have been described in behavioral way. The RTL Compiler synthesis directive *synthesize –to_mapped –effort high* has been employed to design the adders at their maximum speed. The *set_multicycle_path* synthesis command was used to mark the non-speculative outputs of the speculative adders. The dynamic power dissipation is evaluated after synthesis by extracting the nodes activities from a back-annotated simulation.

2.9.1 The optimal p_{min} choice

The variable latency speculative adders depend on the parameter p_{min}. Therefore a key point in the design of variable latency

speculative adders is the choice of the optimal p_{min} parameter. This choice involves a trade-off between the error probability and the speed of the speculative addition. Indeed, by increasing p_{min} the error rate decreases, with positive effects on T_{avg} (2.17), but the speculative carry tree slows down, since a little number of levels is pruned.

In order to investigate this trade-off, multiple synthesis of the proposed topologies have been performed, for different p_{min} values. The Fig. 28, shows, as an example, the synthesis results for the Han-Carlson speculative adders, by varying the synthesis timing constraint. The Fig. 28 shows the results for 32-bit adders, the Fig. 29 for 64-bit adders and the Fig. 30 for 128-bit adders. The x-axis reports the average addition time (2.17), which accounts for error probability. The distribution employed in Figs. 28-29-30 is represented in Fig. 27(a), while the VLSA topology is that suitable for unsigned operands (Fig. 12). As observable, the implementations with $p_{min}=4$ and $p_{min}=n/2$ reveals ineffective because of the high error rate ($p_{min}=4$) or of the little number of pruned levels ($p_{min}=n/2$).

For 32 bit (Fig. 28) the optimum value is p_{min} =8; this value of p_{min} is also the best choice for n=64 bit (Fig. 29). For n=128 bit (Fig. 30) both p_{min}=8 and p_{min}=16 give similar performance.

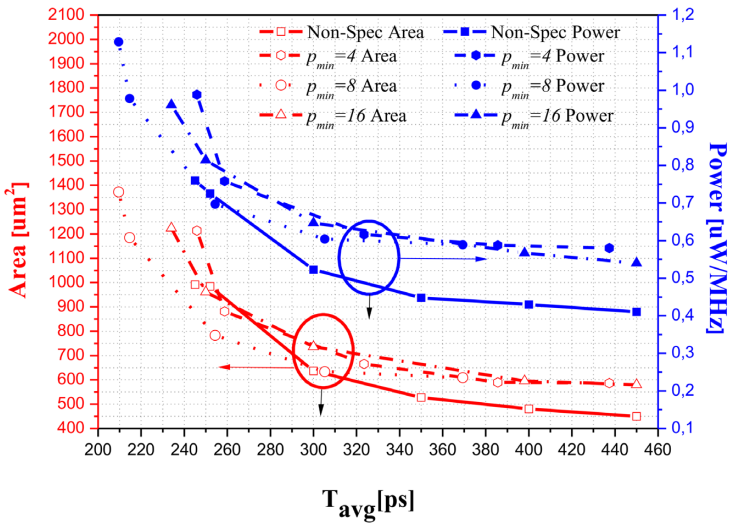

Fig. 28 Area and power of 32-bit speculative and non-speculative Han-Carlson adders as a function of the timing constraint. The performance are shown for different p_{min} values.

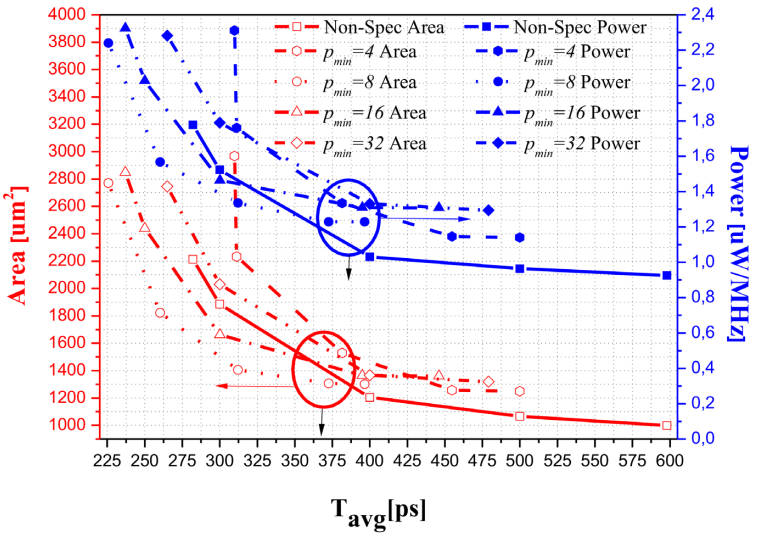

Fig. 29 Area and power of 64-bit speculative and non-speculative Han-Carlson adders as a function of the timing constraint. The performance are shown for different p_{min} values.

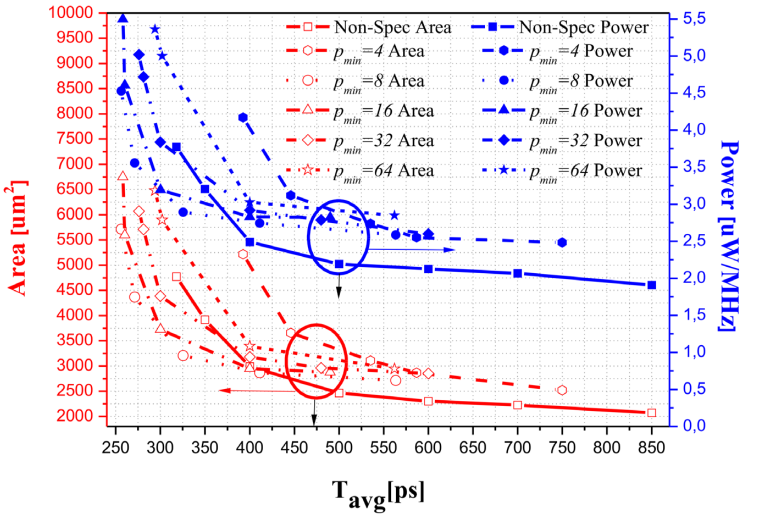

Fig. 30 Area and power of 128-bit speculative and non-speculative Han-Carlson adders as a function of the timing constraint. The performance are shown for different p_{min} values.

Observe that in these figures also the non-speculative topologies performance as reported, for comparison. To this regard, we can observe that the speculative topologies offer are particularly effective in terms of speed, allowing to reduce the minimum achievable delay. As an example, in the 64-bit case, the minimum achievable delay is about 280 ps for the non-speculative adder and reduces up to 225 ps in the variable latency architecture.

The analysis of Area Occupation and Power Dissipation shows that speculative adders are not effective for large average delay. As the timing constraint imposed during synthesis is made tighter speculative adders become advantageous. For instance, in the 64-bit case, speculative Han-Carlson adder results in a lower Area for T_{avg} lower than 385 ps and also in a lower Power Dissipation for T_{avg}<350ps. For T_{avg}=300 ps, the non-speculative adder presents an area of 1885 μm^2 and a power of 1.52 μW/MHz, while the variable latency adder exhibits an area of 1500 μm^2 (20% reduction) and a power of about 1.39 μW/MHz (9% reduction).

2.9.2 Comparison among investigated topologies

In this paragraph the investigated topologies are compared with the aim to determine the most effective one. Moreover, in order to compare the proposed VLSAs suitable for signed operands (Fig. 26) with the previously proposed global carry approach [30], [42], the distribution of Fig. 27(b) is employed.

Each VLSA has been synthesized and, as discussed in the previous paragraph, the topologies with optimal p_{min} have been employed in order to perform the comparison. We have found that p_{min}=8 is the best solution for all the investigated cases, with the exception of L-F and SK topologies. Optimal values for L-F are p_{min}=6 (32 and 64 bit) and p_{min}=10 (128 bit). Optimal values for SK are p_{min}=5 (32 bit) and p_{min}=9 (64 and 128 bit).

The results are reported in the Fig. 31 and Fig. 32, showing, respectively, the area and the dynamic power versus T_{avg} (2.17). Note that, in addition to the topologies of Fig. 26, the results of Kogge-Stone and Han-Carlson speculative topologies, designed to operate with unsigned operands (Fig. 12), are reported, to show how

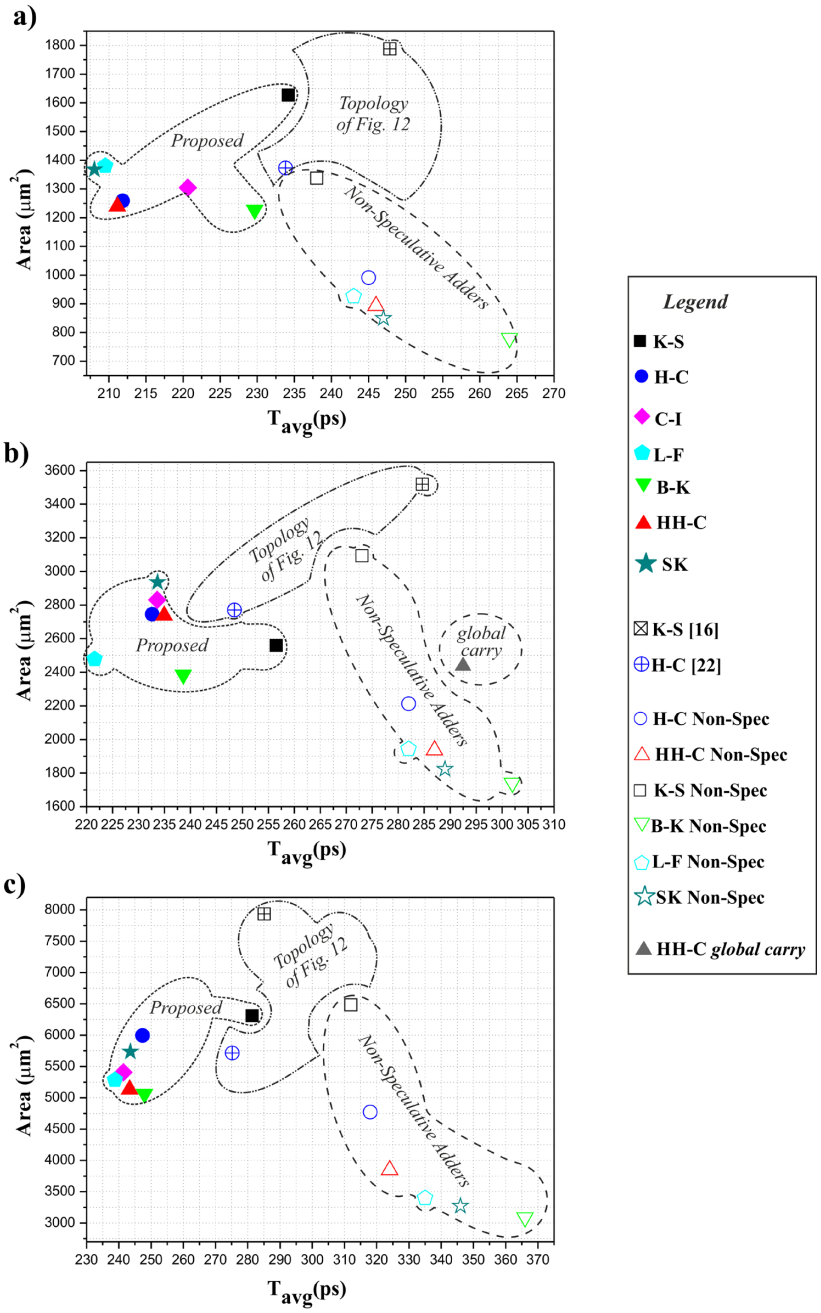

Fig. 31 Area of speculative and non-speculative adders. Each topology is reported at the minimum synthesizable clock period. (a) 32-bit, (b) 64-bit, (c) 128-bit.

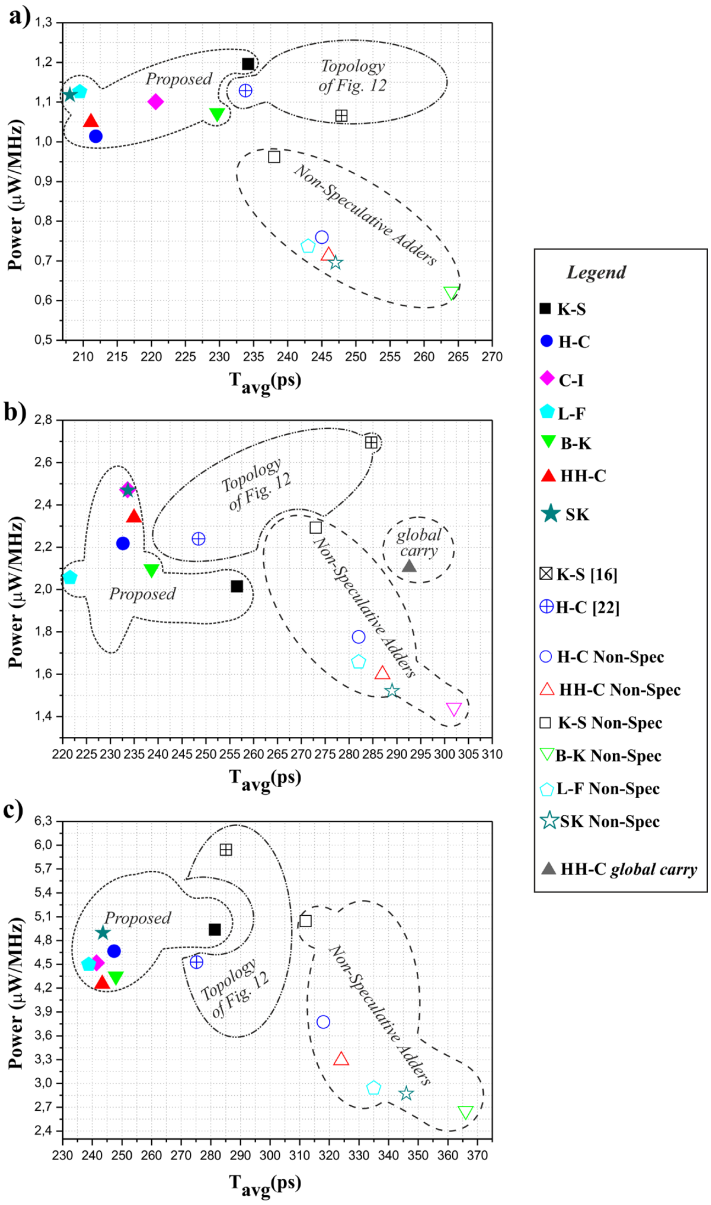

Fig. 32 Power of speculative and non-speculative adders. Each topology is reported at the minimum synthesizable clock period. (a) 32-bit, (b) 64-bit, (c) 128-bit.

their performance are affected in presence of signed operands. Moreover, for the 64-bit case the HH-C speculative topology with the global carry technique proposed in [30] and [42] is reported. In the Figs. 31-32 each topology is reported at its minimum achievable delay.

As it can be observed, proposed VLSAs are faster than non-speculative counterparts. For 128-bit, the addition time reduces of about 20%, from more than 310ps (Kogge-Stone) to less than 240ps; the improvement is less evident, but not negligible, also for 64-bit and 32-bit adders. This improvement in speed corresponds also to a reduction in area (more evident for 128 and 64bit adders). Power reduces in 128bit adder (compared to non-speculative K-S), while slightly increases for 32bit case.

The topologies of Fig. 12, while being effective with unsigned operands, reveal ineffective when operating with signed operands, due to the large error rate, as reported in the gray rows of Tab. III. Moreover, as shown in the Figs. 31(b)-32(b) the global-carry technique is also inefficient. Similar considerations holds investigating the results for different parallel-prefix speculative topologies. In particular, global-carry technique, while providing error probability similar to the ones obtainable with proposed technique, reveals ineffective in terms of circuit, due to the large fanout of the global carry signal that increases power and slows down the adders.

In the investigated cases the speculative L-F adder is the fastest one. This result is particularly evident in the 64-bit case and is due to the fact that the speculative prefix-processing stage mitigates the fan-out issues, since the maximum fan-out in this topology, approximatively, halves every time a level is pruned.

In terms of area and power consumption, in the 32-bit case, H-C and HH-C VLSAs are the most effective topologies. For 64-bit and 128-bit adders, the C-I and the B-K VLSAs are also competitive.

Note that the speculative C-I topology has a logarithmic delay behavior (with respect to operand size n), see Fig. 17. Hence, this topology performs quite well as VLA, while the original one has a critical path traversing all the adder blocks and is hence much less effective.

2.10 Approximate Adders with error correction for error tolerant applications

As discussed in chapter 1, approximate computing leverages error resiliency of applications to improve circuits performance. The assumption made in the previous paragraphs, to provide an always error free result to the system, employing a two-cycles error correction, can be relaxed in presence of error tolerant applications.

In such scenario, the speculative output is always provided to the system and, eventually, corrected to alleviate the error rate or the error magnitude, but, in this case, the error correction, involving simple operations, typically happens in the same clock cycle in which the result is provided.

In this paragraph an approximate adder is discussed, which reduces error rate in presence of signed operands. As discussed in the paragraph 2.5, in real applications the numbers are represented assuming a 2'complements representation, as a consequence the operands are no longer uniformly distributed, following, instead, a Gaussian distribution (Fig. 22). As a result, the carry propagation length significantly increases, raising the speculative adders error rate to values that compromise significantly the quality of results, also in error tolerant applications (the Fig. 24 reveals that the error rate can reach 25% of errors in presence of signed operands).

As shown by [42], in practical applications small values appears more likely than higher values. Employing this observation, an error correction technique is proposed to reduce error rate in presence of Gaussian distributed operands [43].

As discussed in the paragraph 2.5.1, an error condition associated with signed operands happens whenever the operands A and B have opposite signs (first condition), $A+B>0$ (second condition) and $|A|$ and $|B|$ are sufficiently small (third condition). Note that in this condition the speculative adder provide an erroneous negative results, since the carry propagation cannot happen, due to speculation. Moreover, since $|A|$ and $|B|$ are small, also their sum will be small. Note that, in addition to the error condition above, many others error conditions are possible, but, in a signed scenario, these should happen

with lower probability, due to the observations that small numbers appears more likely than higher ones.

The proposed adders is reported in Fig. 33. As you can see, to the speculative adder, composed of three sub-adders, is added some error correction logic. This logic is designed to detect the discussed error condition: the MSB of the operands A and B are XOR-ed in order to check that they have opposed signs (first condition); the MSB of the last sub-adder reveals if the sum is negative (second condition); the *nor* of the L MSBs of the first sub-adder is computed to check the third conditions. Indeed, if the output of the *nor* is high we have a hint that the sum is positive and smaller than $2^{(r+p-L)}$. When the three conditions are true we have three hints that allows us to speculate that we are in the case of positive sum of small numbers with opposite signs, in this case the output is assumed to be the positive value produced by the first sub-adder. Indeed, in this condition, the signal E_1 is low and all the outputs from the second to the last sub-adders are driven to zero. Similarly, we check the L MSBs of the second sub-adder: if they are all zero, A and B have opposite signs and the sum is negative, the signal E_2 goes to zero and all the outputs from the third through the last sub-adder are driven to zero.

2.10.1 Error rate and quality of results

The proposed architecture is able to catch the errors due to the Gaussian distributed operands and to correct them. The Fig. 34 shows the simulated error probabilities for three adder configurations, as a

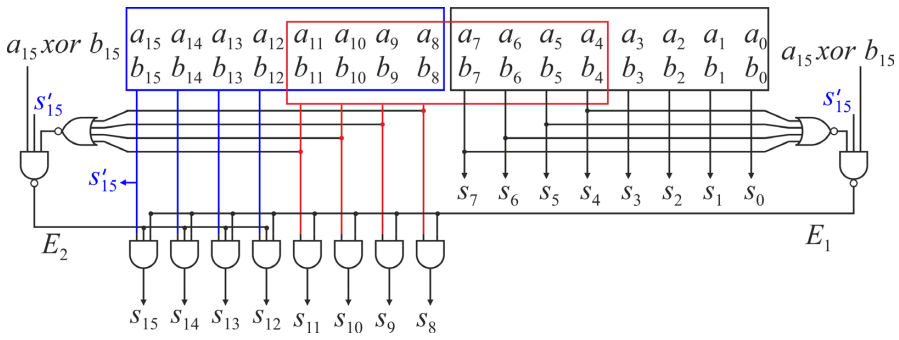

Fig. 33 Proposed approximate adder with output error correction circuit. Here n=16, r=4, p=4, L=4.

function of the standard deviation σ of the Gaussian distribution. As it can be observed the error rate is significantly reduced for a wide range of σ values (compare with Fig. 24).

In the case of uniformly distributed inputs, the resulting error probabilities are reported in Tab. IV. By comparing with Tab. I, we can observe that proposed adders increase error rate in presence of uniformly distributed operands. As shown in second and third row of Tab. IV, the error probability for uniformly distributed operands can be mitigated by increasing L (at a cost in terms of hardware complexity, see Fig. 33). The possibility that signals E_1 or E_2 becomes erroneously low, in fact, decreases exponentially with L.

In order to understand the impact of the error correction on a real application, we have simulated a simple audio processing system, in which an echo is added to an audio signal, by adding a waveform replica delayed by about ¼ of second. The obtained results are shown

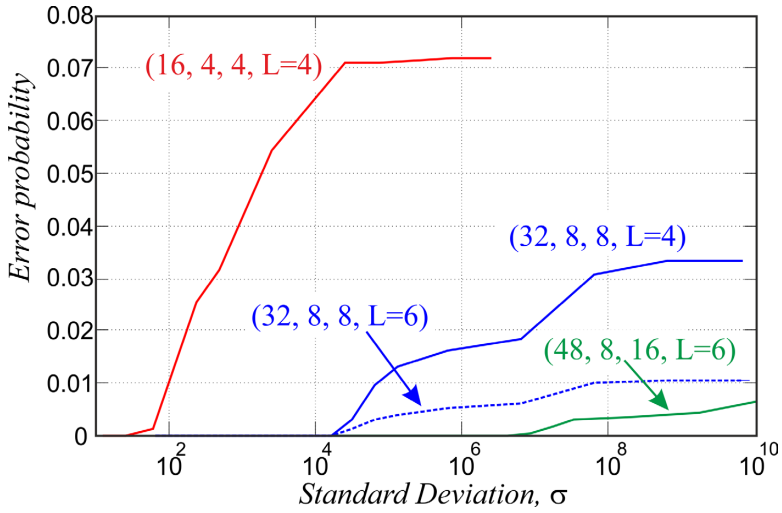

Fig. 34 Simulated error probability of the adder architecture proposed in Fig. 33. The input distribution is assumed to be Gaussian.

TABLE IV. ERROR PROBABILITY FOR UNIFORMLY DISTRIBUITED INPUTS IN PROPOSED SPECULATIVE ADDERS

Adder Config. (*n*,*r*,*p*) L	**Error probability**
(16,4,4) L=4	7.15%
(32,8,8) L=4	3.30%
(32,8,8) L=6	1.07%
(48,8,16) L=6	0.78%

TABLE V. PERFORMANCE OF APPROXIMATE ADDERS IN AN AUDIO PROCESSING EXAMPLE

Adder Config. (*n*,*r*,*p*)	Error %	SNR (dB)
Without correction (24,6,6)	13.98%	8.58
Corrected (24,6,6) L=6	0.67%	21.78
Without correction (24,5,9)	3.46%	14.64
Corrected (24,5,9) L=6	0.01%	40.20
Without correction (24,6,6)	13.98%	8.58
Corrected (24,4,12) L=6	0%	∞

TABLE VI. VLSI IMPLEMENTATION RESULTS

Adder Config. (*n*,*r*,*p*) L	Delay (ns)	Power (μW/MHz)	Power×Delay (fJ/MHz)
(16,4,4) L=4	0.401	0.258	0.103
16bit standard	0.598	0.182	0.109
(32,8,8) L=4	0.701	0.516	0.362
32bit standard	1.13	0.397	0.447
(48,8,16) L=6	0.981	0.861	0.845
48bit standard	1.81	0.593	1.07

in Tab. V, where a 13sec. fragment of a song is used as a test vector and 24-bit adders are considered. Waveform processing requires about 415,000 additions. The signal to noise ratio obtained with the standard adder is quite low when using 12-bit and 14-bit sub-adders (cases (24,6,6) and (24,5,9) in Tab. V), reaching 20dB only when using 16-bit sub-adders. The proposed adder with output correction, instead, shows sensibly better performances, giving an error-free result for the n=24, r=4, p=12, L=6 configuration.

2.10.2 Synthesis results

The proposed adders have been synthesized in UMC 65nm technology, along with the non-corrected adders. Cadence RTL Compiler has been employed to synthesize the design. The adders have been described in Verilog HDL, using + operator the describe the sub-adders of each speculative adder. The RTL Compiler synthesis directive *synthesize –to_mapped –effort high* has been employed. Power is calculated by simulating the synthesized circuits, with SDF annotation. The synthesis results are reported in Tab. VI. The proposed adder with output correction is faster than standard (error-free) circuit; on the other hand, its power dissipation is higher.

The power-delay product is better in the proposed circuit, especially for 32-bit and 48-bit implementations.

2.11 Approximate Adders in Carry-Save Multiplier-Accumulators

In this paragraph the use of approximate adders as final adder of carry-save multiplier-accumulators (MACs) is investigated. The MACs are basic building blocks in digital signal processing applications. We will focus, in this paragraph, on image processing applications. In this context the MACs are usually employed to perform convolutions, which are a basic operation for image filtering applications.

Many papers have introduced approximate adders for image processing application, but their investigations are done making assumptions that are hardly verified in practical applications. As an example, [47] employs approximate adders for image processing applications, but the multiplications and subtraction are performed by exact functional units. Similar assumption are done in [18].

As previously discussed, the use of approximate adders and the consequent performance improvement strictly depends on the input statistics of the approximate adder, therefore on the application. We propose a design flow to opportunely design the approximate adders accounting for the input distribution.

The MAC employed in this paper is constituted by a carry-save Wallace tree for partial product compression (for a detailed discussion about multipliers and MACs the reader can make reference to chapter 3). The resulting output, in carry-save format, is added using an approximated carry-propagate adder.

In the following, the proposed design flow is discussed.

2.11.1 Design flow

We start the MAC design describing its architecture in HDL language. The HDL code is then read and optimized by the synthesizer. In this phase the synthesizer detects the arithmetic operator and performs datapath extraction to opportunely transform the arithmetic operators into optimized blocks. In this phase

carry-save arithmetic is usually employed in order to optimize datapath performance. A final carry-propagate adder is then employed to sum the output of the carry-save stage, obtaining the final result.

Unfortunately, describing the MAC operation as `y<=A+B*C`, the designer is not able to access, after the synthesis, to the carry-save signals to be summed by the final adder. To overcome this issue, we employ arithmetic IP components to describe the MAC. In particular, in Cadence RTL Compiler synthesizer the designer can use the ChipWare IP Components [48], in Synopsys Design Compiler the designer can use DesignWare Building Block IP [49].

The resulting architecture is shown in the Fig. 35, where the component `CW_multp` is the partial product multiplier, while the `CW_csa` is the carry save adder, used to sum an 18-bit addend to the carry-save multiplier outputs.

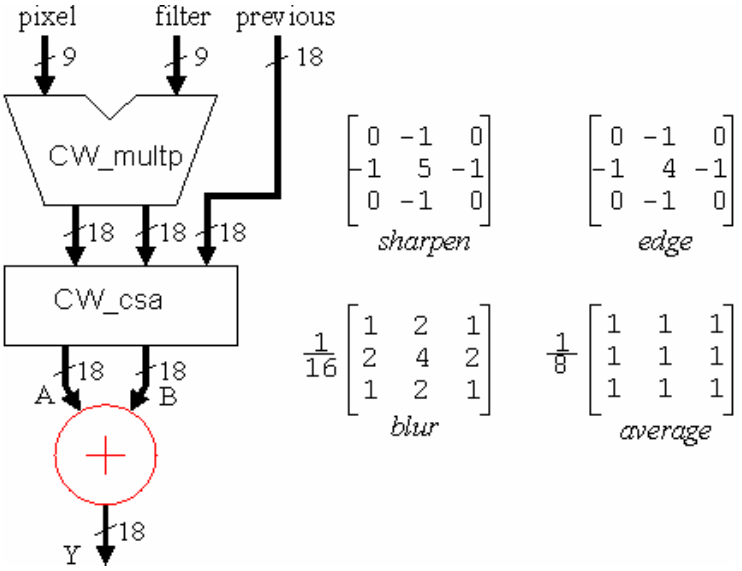

Fig. 35 MAC structure. CW_multp is the partial product multiplier, while CW_csa is the carry save adder. The final carry propagate adder is highlighted in red. On the right some of the considered 3x3 kernels are reported.

We employ a 9x9 signed multiplier, in which the MSB of the pixel operand is zero. The filter coefficients are signed and use 4 fractional bits (LSB=2^{-4}) to represents the commonly used 3x3 kernels (as those represented in Fig. 35). Please note that the datapath is dimensioned to avoid overflow while performing image filtering. The

carry-propagate adder highlighted in red in Fig. 35 produces the MAC output Y.

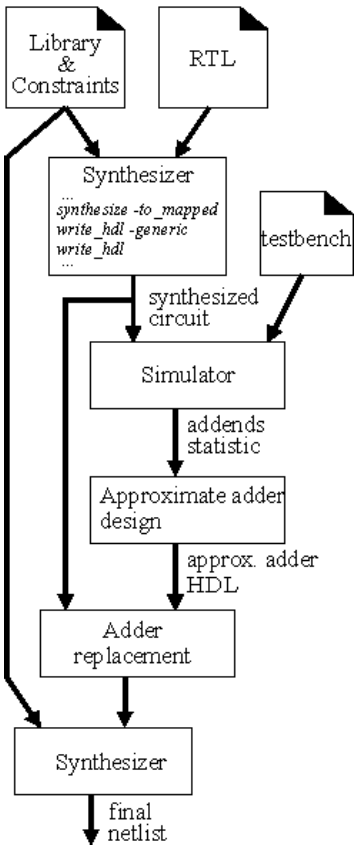

Fig. 36 Proposed design flow.

The proposed design flow is shown in Fig. 36. We start with an HDL description which employes IP blocks, as in Fig. 35, then we apply constraints and synthesize the circuit. Then the synthesized netlist is simulated in order to extract the statistics of the two addends *A* and *B* (Fig. 35) summed by the final propagate adder. It is worthwhile observing that this simulation cannot be done before of the synthesis. Indeed the synthesizers selects the appropriate architecture for the partial product multiplier as a function of the applied constraints. Therefore the values of *A* and *B* post-synthesis may differ from those obtained with a pre-synthesis simulation (although their sum remains the same).

The Fig. 37 shows the statistics of the carry-save operands A and B when and edge filtering operation is performed on the test Lena image. Unlike previously discussed, the distribution is neither uniform nor Gaussian, as a consequence the formulas presented in literature to estimate the error probability, reveals inadequate to predict the error probability of the proposed MAC.

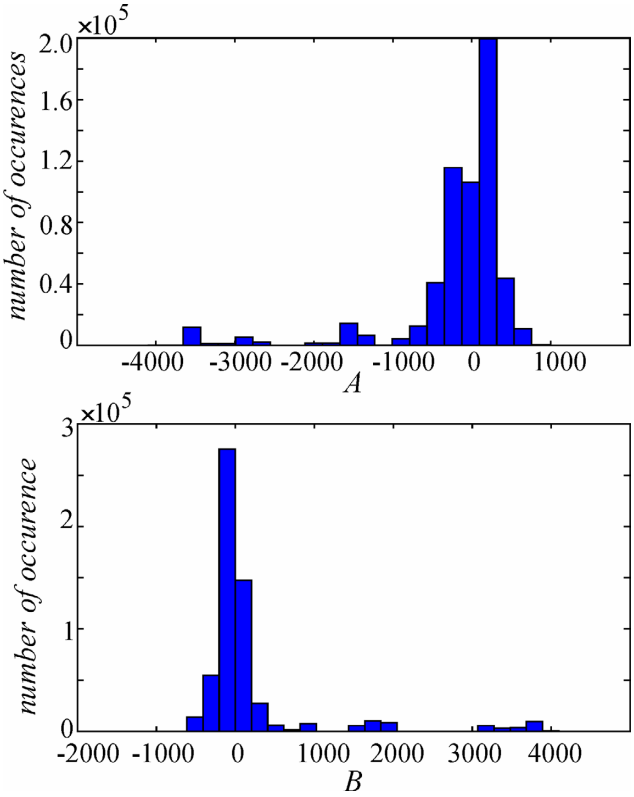

Fig. 37 Distributions of the inputs A and B of the carry-propagate adder while performing an edge filtering on the Lena test image.

In order to design the approximate adder, we numerically extract from simulated data two set of values: the probability of having a carry propagation from bit i to bit j (indicated as $p_{i,j}$) and the probability of having a carry generation from bit i to bit j (indicated as $g_{i,j}$). From these values, we can compute numerically the error probability of the approximate adder. Let us consider the adder shown in Fig. 38.

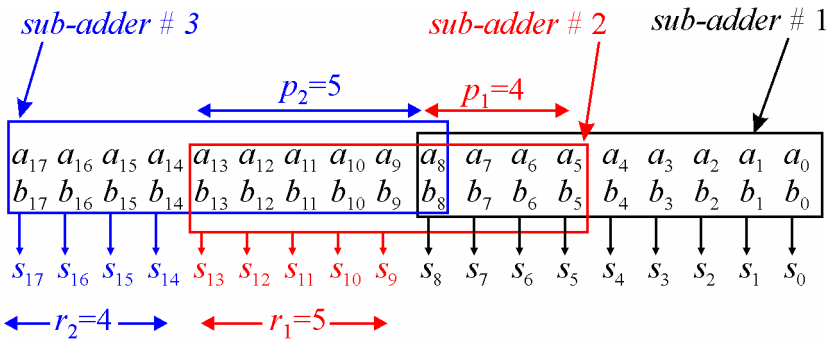

Fig. 38 Speculative adder segmentation. A n=18 bit adder is segmented using three sub-adders. Each sub-adder produces r sum bits that contribute to the final result and employs p bits to predict the carry.

As usually, the first sub-adder is exact, while the second one gives erroneous results if the condition $g_{[4:0]}\ p_{[8:5]}$ is asserted. This condition has a probability $P(E_1)=P(g_{[4:0]}\ p_{[8:5]})$. In the third sub-adder adder an error occurs with probability and this occurs with a probability $P(E_2)=P(g_{[7:0]}\ p_{[13:8]})$. The overall error probability is given by $P(E)=P(E_1 \cup E_2)=P(E_1)+P(E_2)-P(E_1 \cap E_2)$. Simple calculations yield, in this example, $P(E)=g_{0,4}\ p_{5,8}+g_{5,7}\ p_{8,13}$. By generalizing this approach, we are able to quickly investigate different approximate adder configurations, after characterizing the distributions of A and B signals.

After the design of the approximate adder, we modify the netlist by substituting the exact adder with the approximate one. Another synthesis and optimization step yields the final netlist of the MAC.

2.11.2 Quality of results

The approximate adders of Fig. 38 has been employed as final adder of Fig. 35. This adder exhibits $P(E)\approx 0.09$ with the statistics of Fig. 37. %. Note that, after convolution, each pixel value is limited to an 8-bit integer value, i.e. any negative value is limited to zero while any positive value is saturated to 255. Therefore, not every error in the convolution results in an erroneous pixel in the output image. The Fig. 39 shows the original Lena image, while the Fig. 40-41 the filtered ones. The image quality can be quantified with the Structural

Similarity Index, SSIM [50]; a value of SSIM=1 means perfect similarity between two images.

Fig. 39 Original "Lena" image.

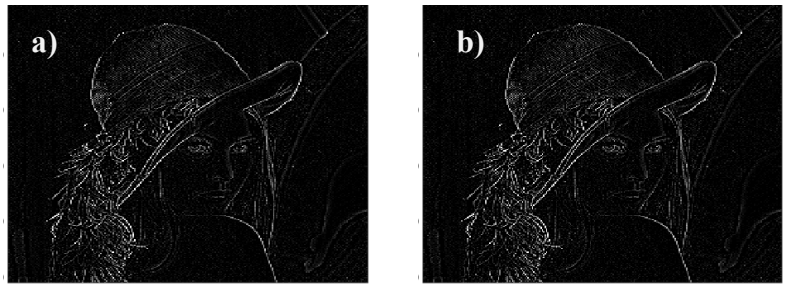

Fig. 40 Edge filtered image. (a) exact adder; (b) approximate adder

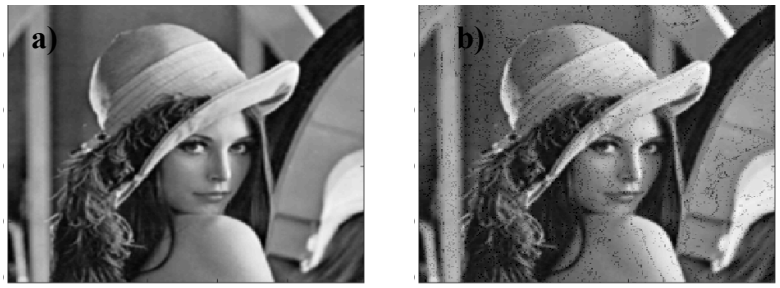

Fig. 41 Blur filtered image. (a) exact adder; (b) approximate adder

For the edge filtered image (Fig. 40), the SSIM=0.98, therefore the approximate adder performs well with the edge filter. With the blur filter, the use of approximate adder results instead in sensible image noise in the form of spurious black pixels, with SSIM=0.81 for the two images in Fig. 41(a) and 41(b).

2.11.3 Pixels skipping

The Fig. 41(b) has shown a significant degradation of image quality, due to spurious black pixels. It is possible to sensibly improve the quality of image affected by this kind of "noise" due to approximation, augmenting the approximate adder with a simple circuitry to detect the error by checking the carry-out of sub-adders #2 and #3 and the carries c_9 and c_{14}. Indeed, as an example, in the second sub-adder an error occurs if the carry-out of sub-adder#1 is high, while the carry c_9 computed by sub-adder#2 is low.

When an error is detected, the convolution is skipped and the previous filtered pixel is outputted. As shown in Fig. 42 this simple technique reveals effective, with SSIM=0.98 for the two images in Fig. 41(a) and 42.

Fig. 42 Blur filtered image with approximate adder and pixels skipping technique.

2.11.4 Synthesis results

The discussed MAC has been implemented in STM 28 nm technology, standard V_T, typical corner. We have imposed constraint aimed to obtain a minimum area, lowe-power design.

The Tab. VII shows the VLSI implementation results, using Cadence RTL Compiler. As shown, the MAC with the approximate adder allows improving the speed by 19%, with a 4-5% increase in power and area, respectively. By performing voltage scaling we can trade power for speed as shown in the last row of Tab. VII. By reducing supply voltage to 0.91 a power saving of 14% can be achieved, while keeping the same delay of the MAC with exact adder. This improvement in performance, while noticeable, is probably less than one would expect.

TABLE VII. VLSI IMPLEMENTATION RESULTS

Design	V_{DD} [V]	Minimum period [ns]	Area [μm^2]	Norm. Power [$\mu W/MHz$]
MAC with Exact adder	1.0	1.38	324	1.46
MAC with Approximate adder	1.0	1.11 (-19%)	341 (+5%)	1.52 (+4%)
MAC with Approximate adder Voltage scaled	0.91	1.38	341	1.26 (-14%)

To better investigate this behavior, the Fig. 43 shows the arrival times for the inputs of the carry propagate adder. As it can be observed, the signals corresponding to the middle bits of the addends arrive later than the others [51]. This partly overcomes the speed advantages related to sub-adder decomposition of approximate adders.

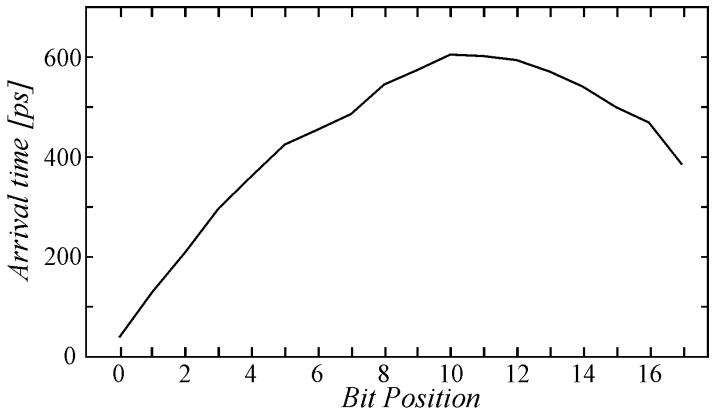

Fig. 43 Arrival times of carry propagate adder inputs.

2.12 Conclusions

In this chapter my research activity about speculative adders have been discussed.

The first part of the chapter is devoted to discussing speculative adders for error-free applications. In this contest, the speculative adders are augmented with a two-cycles error correction mechanism, acting as variable latency adders. Numerous variable latency speculative parallel-prefix adders topologies have been

proposed, for high speed and error-free applications. Moreover, the case in which speculative adders deals with 2'complements representation has been investigated. For this case, the error rate increases significantly, reducing the effectivity of speculative adders. Therefore, a technique which allows keeping the error rate low, also in presence of 2'complements representations, has been proposed. This technique allows reducing hardware overhead with respect other solutions proposed in the literature. The implementation results shows that variable latency speculative adders allows improving performance (speed and power) when the highest speed is desired, otherwise the standard, non-speculative adders, remains the best choice. It is also worthwhile observing that variable-latency speculative adder general scheme require to put the selection multiplexer after the output register, therefore they eat into the next clock cycle, reducing the amount of calculations that can be done there and the benefit compared to non-speculative adders.

In the second part of the chapter, speculative adders for errors tolerant applications are discussed. In particular a speculative adder is proposed with employs an error correction circuitry allowing to reduce significantly the error rate in presence of Gaussian distributed operands, which is the way to model 2'complements represented signals. The proposed error correction increases significantly the quality of results in error tolerant applications, like audio processing. The VLSI implementation results show that the power-delay product improves with respect standard (non-speculative) adders.

Moreover, a study about the use of approximate adders in carry-save multiply and accumulate units has been conducted. An approximate adders has been employed in the final stage of a Wallace tree carry-save MAC unit, designed for image filtering applications. It has been shown that in a typical image processing application the inputs of the carry-propagate adder are far from being uniformly or Gaussian distributed. Therefore formulas proposed in the literature for estimate error probability, while giving an important insight on approximate adders operation, are inadequate to judge the actual performance of the MAC. Therefore a design flow has been proposed to accurately choose the approximate adder architecture as a function of the application. The image filtering results have shown that, for some kernel filters, the approximation results in significant noise, affecting the overall image quality. To mitigate this phenomena, a

simple technique has been employed which allows skipping the erroneous pixel, using the previous, corrected one. This significantly improves quality. The VLSI implementation results show that the use of speculative adders as final adder in MAC units allows saving 15% of power in voltage scaled mode. Moreover, it has been observed that this gain can increase if the speculative adder is designed accounting for the non-uniform arrival time of the carry-save signals.

Chapter 3

Precision-scalable units

3.1 Introduction

In this chapter my research activity about precision-scalable units is discussed. Precision-scalable units fit in the approximate computing framework, allowing improving computation efficiency at expense of quality degradation. In particular, the peculiarity of precision-scalable units is the ability to change the precision level at runtime. This allows adapting the precision level of the unit with the precision requirements of a given application. Indeed, as discussed in [52], the same application can tolerate different precision levels during its computation; moreover the degree of resilience of an application strongly depends on the input data being processed [53].

Precision-scalable units and systems have been proposed in the last years. In [52] a precision-scalable processor is proposed, for Support Vector Machine applications. The processor implements precision-scaling at algorithm, architecture and circuit level. A quality estimator and a PID controller allows to automatically control the precision level as a function of the input dataset, in order to keep a given quality level. *Raha et al.* [53] propose precision-scalable adders whose precision level is managed, at run-time, through an heuristic algorithm, as a function of the input dataset. The precision-scalable unit is employed in an MPEG encoder. In [54] a precision-scalable deep learning core is proposed. The authors implements bit-truncation to save switching energy in the arithmetic units and to speed up the computation. The increased speed deriving by reduced precision can be turned into power saving by performing voltage scaling. The voltage and accuracy scaling is performed at run-time during the feedforward path of a state of art CNN. The resulting technique is named by authors as “Dynamic Voltage Accuracy Scaling” (DVAS). Precision-scalable concept has been also employed in memories. In [55] *Frustaci et al* propose an SRAM which can dynamically trade

power for quality. In [56] authors focus on reducing the power dissipation due to off-chip memories of precision-scalable systems, arguing that most of the power is spent in off-chip memory accesses. The authors therefore propose a run-time memory controller and a memory access scheme to opportunely manage and read the data as a function of the precision of the system.

In this chapter two precision-scalable units are shown. Firstly, after an introduction of binary multiplication and an overview of error compensation techniques in truncated multiplier, a precision-scalable data-aware Multiply And Accumulate (MAC) unit is discussed. This MAC unit employs a programmable truncated multiplier [57] in which a novel real-time data-aware error compensation technique is proposed. Secondly, a precision-scalable standard cell memory (SCM) architecture is proposed.

In the following, the proposed precision-scalable data-aware MAC unit is discussed.

3.2 Binary multiplication

Binary multiplication is a fundamental operation in many digital signal processing and machine learning algorithms. Multiplication operation, being more complex than addition, results in area and power hungry VLSI implementations.

Due to its inherent area consuming nature, serial multiplication has been widely used in the past. Nowadays, being the area a secondary figure of merit in modern VLSI, parallel multiplication has replaced serial multiplication.

The multiplication operation involves two steps: partial product generation and their summation. Assuming an *MxN*-bit multiplication, *N* partial products of *M* bits each must be opportunely shifted and added. Partial products summation has been widely investigated in the past, being the core of the multiplication operation and, in the following, the main contributions are briefly reported.

Low-power multiplier implementations have been widely investigated. Among these, truncated multiplier represents an effective way of trade accuracy for speed and power, indeed, recently, Synopsys has introduced the “internal rounding” datapath synthesis directive [58]-[59], which allows to easily implement a truncated multiplier . A brief overview of error compensation techniques in

truncated multipliers will be reported before introducing the proposed precision-scalable data-aware Multiply And Accumulate unit.

3.2.1 Partial Product generation

Partial product generation is the first step of the multiplication operation. In the following, without loss of generality, let us assume the followings conditions:

a. The operands *X, Y* are *n*-bit numbers;

b. The output *P* is on *2n-bit* (full-width multipliers);

c. The output *Pt* is on *n-bit* (truncated multipliers).

In the following three different partial product matrix are examined: Unsigned Matrix, Two's Complement Matrix, Mixed-Operands Matrix.

3.2.1.1 Unsigned Matrix

In case of unsigned multiplication, in addition to conditions *a, b, c,* let us assume that the operands are fractional unsigned numbers in the range $[0,1)$:

$$X = \sum_{i=1}^{n} x_i \cdot 2^{-i} \tag{3.1}$$

$$Y = \sum_{i=1}^{n} x_i \cdot 2^{-i} \tag{3.2}$$

The multiplier result can be trivially expressed as:

$$P = \sum_{i=1}^{2n} p_i \cdot 2^{-i} = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i y_j \cdot 2^{-i-j} \tag{3.3}$$

The above equation states that the multiplier output is the weighted sum of the partial products $x_i y_j$, whose computation requires n^2 AND gates. The Fig 1 reports the resulting partial product matrix (PPM).

Weights	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8								
	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8								
									x_1y_8	x_2y_8	x_3y_8	x_4y_8	x_5y_8	x_6y_8	x_7y_8	x_8y_8
								x_1y_7	x_2y_7	x_3y_7	x_4y_7	x_5y_7	x_6y_7	x_7y_7	x_8y_7	
							x_1y_6	x_2y_6	x_3y_6	x_4y_6	x_5y_6	x_6y_6	x_7y_6	x_8y_6		
						x_1y_5	x_2y_5	x_3y_5	x_4y_5	x_5y_5	x_6y_5	x_7y_5	x_8y_5			
					x_1y_4	x_2y_4	x_3y_4	x_4y_4	x_5y_4	x_6y_4	x_7y_4	x_8y_4				
				x_1y_3	x_2y_3	x_3y_3	x_4y_3	x_5y_3	x_6y_3	x_7y_3	x_8y_3					
			x_1y_2	x_2y_2	x_3y_2	x_4y_2	x_5y_2	x_6y_2	x_7y_2	x_8y_2						
		x_1y_1	x_2y_1	x_3y_1	x_4y_1	x_5y_1	x_6y_1	x_7y_1	x_8y_1							
	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}

Fig. 1 Partial product matrix for 8x8 unsigned multiplier.

3.2.1.2 Two's Complement Matrix

Let us derive the PPM in the case in which the operands *X, Y* are represented in two's complement representation:

$$X = -x_1 \cdot 2^{-1} + \sum_{i=2}^{n} x_i \cdot 2^{-i} \tag{3.4}$$

$$Y = -y_1 \cdot 2^{-1} + \sum_{i=2}^{n} y_i \cdot 2^{-i} \tag{3.5}$$

The output *P*, in this case, writes as:

$$\begin{aligned} P = -p_1 \cdot 2^{-1} + \sum_{i=2}^{2n} p_i \cdot 2^{-i} = x_1 y_1 \cdot 2^{-2} + \sum_{i=2}^{n}\sum_{j=2}^{n} x_i y_j \cdot 2^{-i-j} \\ - \sum_{i=2}^{n} x_1 y_i \cdot 2^{-i-1} - \sum_{i=2}^{n} y_1 x_i \cdot 2^{-i-1} \end{aligned} \tag{3.6}$$

Note that (3.6) contains negative terms. Instead of doing a subtraction it is possible to sum the two's complement of the negative terms (Baugh-Wooley multiplier [60]). First of all, we have to extend the last two terms of (3.6) in the output representation, performing a zero padding (the negative terms extend in binary weights from 2^{-3} to 2^{-n-1},while the output extends from 2^{-1} to 2^{-2n}, please note that padding operation is needed in order to be able to add the negative terms with the other ones) . Therefore, complementing all the bits and adding one LSB, we have:

$$-\sum_{i=2}^{n} x_1 y_i \cdot 2^{-i-1} = \sum_{i=2}^{n} \overline{x_1 y_i} \cdot 2^{-i-1} + 2^{n-1} + 2^{-2} \tag{3.7}$$

$$-\sum_{i=2}^{n} y_1 x_i \cdot 2^{-i-1} = \sum_{i=2}^{n} \overline{y_1 x_i} \cdot 2^{-i-1} + 2^{n-1} + 2^{-2} \tag{3.8}$$

Substituting (3.7) and (3.8) in (3.6) we have:

$$P = 2^{-1} + 2^{-n} + \sum_{i=2}^{n}\sum_{j=2}^{n} x_i y_j \cdot 2^{-i-j} + \sum_{i=2}^{n}\left(\overline{x_i y_1} + \overline{x_1 y_j}\right) \cdot 2^{-i-j} \tag{3.9}$$

The Fig. 2 shows the resulting PPM. The implementation of the PPM for two's complement multiplication requires $(2n-2)$ NAND gates and $\left((n-1)^2+1\right)$ AND gates.

Weights	-2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8								
	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8								
								1	$\overline{x_1y_8}$	x_2y_8	x_3y_8	x_4y_8	x_5y_8	x_6y_8	x_7y_8	x_8y_8
								$\overline{x_1y_7}$	x_2y_7	x_3y_7	x_4y_7	x_5y_7	x_6y_7	x_7y_7	x_8y_7	
							$\overline{x_1y_6}$	x_2y_6	x_3y_6	x_4y_6	x_5y_6	x_6y_6	x_7y_6	x_8y_6		
						$\overline{x_1y_5}$	x_2y_5	x_3y_5	x_4y_5	x_5y_5	x_6y_5	x_7y_5	x_8y_5			
					$\overline{x_1y_4}$	x_2y_4	x_3y_4	x_4y_4	x_5y_4	x_6y_4	x_7y_4	x_8y_4				
				$\overline{x_1y_3}$	x_2y_3	x_3y_3	x_4y_3	x_5y_3	x_6y_3	x_7y_3	x_8y_3					
			$\overline{x_1y_2}$	x_2y_2	x_3y_2	x_4y_2	x_5y_2	x_6y_2	x_7y_2	x_8y_2						
	1	$\overline{x_1y_1}$	x_2y_1	x_3y_1	x_4y_1	x_5y_1	x_6y_1	x_7y_1	x_8y_1							
	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}

Fig. 2 Partial product matrix for 8x8 two's complement multiplier.

3.2.1.3 Mixed-Operands Matrix

Let us assume that the *X* operand is signed, represented in two's complement, while the *Y* operand is unsigned:

$$X = -x_1 \cdot 2^{-1} + \sum_{i=2}^{n} x_i \cdot 2^{-i} \tag{3.10}$$

$$Y = \sum_{i=1}^{n} y_i \cdot 2^{-i} \tag{3.11}$$

In this case the result *P* becomes:

$$P = -p_1 \cdot 2^{-1} + \sum_{i=2}^{2n} p_i \cdot 2^{-i} = \sum_{i=2}^{n}\sum_{j=1}^{n} x_i y_j \cdot 2^{-i-j} - \sum_{i=1}^{n} x_1 y_i \cdot 2^{-i-j} \tag{3.12}$$

As done the for the Two's Complement Matrix (above paragraph), the (3.12) can be written as summation of positive terms, as follows:

$$P = 2^{-1} + 2^{-n-1} + \sum_{i=2}^{n}\sum_{j=1}^{n} x_i y_j \cdot 2^{-i-j} + \sum_{i=1}^{n} \overline{x_1 y_i} \cdot 2^{-i-1} \qquad (3.13)$$

The resulting PPM (Fig. 3) implementation requires n NAND gates and $\left(n\left(n-1\right)\right)$ AND gates.

Weights	-2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8								
	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8								
								$\overline{x_1y_7}$	$\overline{x_1y_8}$	x_2y_8	x_3y_8	x_4y_8	x_5y_8	x_6y_8	x_7y_8	x_8y_8
							$\overline{x_1y_6}$	x_2y_6	x_2y_7	x_3y_7	x_4y_7	x_5y_7	x_6y_7	x_7y_7	x_8y_7	
						$\overline{x_1y_5}$	x_2y_5	x_3y_5	x_3y_6	x_4y_6	x_5y_6	x_6y_6	x_7y_6	x_8y_6		
					$\overline{x_1y_4}$	x_2y_4	x_3y_4	x_4y_4	x_4y_5	x_5y_5	x_6y_5	x_7y_5	x_8y_5			
				$\overline{x_1y_3}$	x_2y_3	x_3y_3	x_4y_3	x_5y_3	x_5y_4	x_6y_4	x_7y_4	x_8y_4				
			$\overline{x_1y_2}$	x_2y_2	x_3y_2	x_4y_2	x_5y_2	x_6y_2	x_6y_3	x_7y_3	x_8y_3					
		$\overline{x_1y_1}$	x_2y_1	x_3y_1	x_4y_1	x_5y_1	x_6y_1	x_7y_1	x_7y_2	x_8y_2						
	1								x_8y_1							
									1							
	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}

Fig. 3 Partial product matrix for 8x8 mixed-operands multiplier.

3.2.2 Array multiplier

Array multipliers constitute a particular type of parallel multiplier, highly suitable for VLSI implementation, due to its regularity and to reduced wire tracks, going from one full-adder to the contiguous one.

In Fig. 4 is reported an example of such multiplier, in the case of unsigned PPM. As it can be observed each cell in Fig. 4 receives a different couple x_iy_j (3.3). The array is composed by AND gates (for the partial product generation) and by MFAs and MHAs gates, which are respectively, full-adder (FA) cells and half-adder (HA) cells which include an AND gate for the generation of the remainder partial products x_iy_j, these ones are then summed with the carry and the sum bits coming from previous MFAs and HFAs. To this regard, please note that the array of Fig. 4 employs carry-save technique to avoid carry propagation: each full adder acts as a compressor (3,2) taking three inputs and producing a sum and a carry, which is therefore not propagated, but "saved" in the array and opportunely mixed in the following row to reduce the delay. Note that the LSB of the result are

directly produced by the array, while the MSBs are obtained through a final carry-propagate adder.

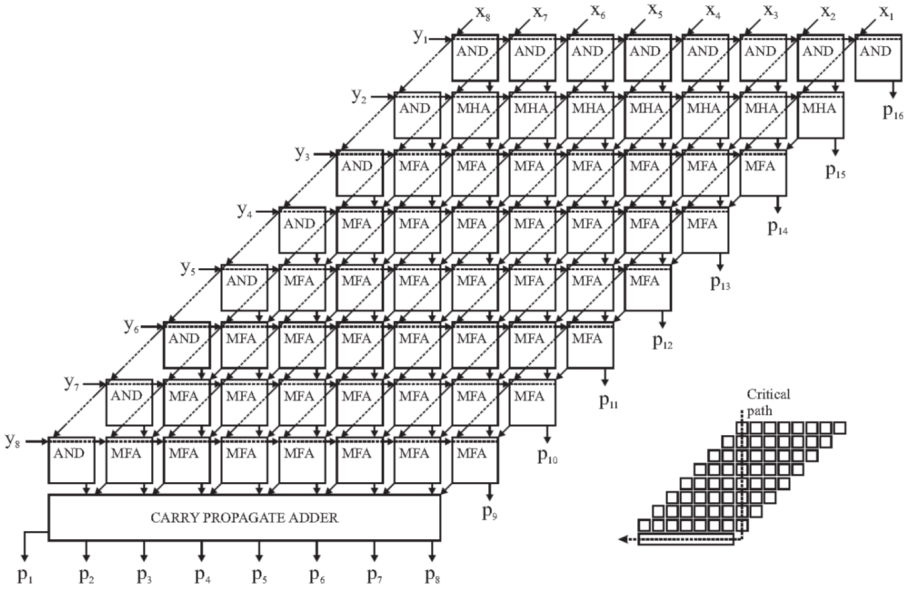

Fig. 4 Unsigned 8x8 array multiplier.

In the bottom-right corner of Fig. 4 the array critical path is reported. It involves the higher column of array, therefore array multiplier delay goes linearly with *n*. Similar considerations apply to the cases of Two's Complement PPM and Mixed-Operands PPM.

3.2.3 Tree multipliers

As shown above, the carry-save array multiplier delay linearly increases with operands size *n*. High-speed topologies (e.g. parallel-prefix adders) for the final carry-propagate adder (also named as "Vector Merging Adder" in this context) can be implemented, making the array delay the bottleneck for multiplier speed. To this regard, faster multipliers can be obtained using tree multipliers.

3.2.3.1 Wallace reduction tree

Wallace [61] proposed to organize the PPM rows in group of three partial products and to use a full-adder to "compress" three partial products into two outputs, a carry and a sum bit, adopting a

carry-save approach. In this way, in the first step a good number of partial product are summed in parallel in carry-save format, speeding up the multiplication (with the same principle of carry-lookahead and parallel-prefix adders: start to compute in parallel, without waiting for the late-arrival signals, as long as you can). In the cases in which two partial products remains in a given row, an half-adder is used. In the second step the partial products resulting from the first step are newly grouped, with the same principle, and compressed using FAs. The height of the matrix from one step to the next, reduces approximately of 1.5 (thanks to the compressing action of FAs). The Fig. 5 shows the various compression steps in a 8x8 multiplier, implementing Wallace reduction three; in this figure the dots represent partial products and outputs of FAs and HAs.

The Wallace reduction tree allows achieving a logarithmic delay, being the number of reduction levels, using (3,2) compressors given by [26]:

$$\left\lceil \log_{3/2}\left(\frac{n}{2}\right)\right\rceil \tag{3.14}$$

3.2.3.2 Dadda reduction tree

The Wallace tree can be optimized in terms of number of FAs and HAs as proposed by Dadda [62]. At each reduction step, the height of the PPM (which is given by the height of the highest column) follows the Dadda's series:

$$\begin{aligned} &d_1 = 2; \\ &d_{j+1} = \left\lfloor \frac{3}{2} d_j \right\rfloor \quad ; j > 1 \end{aligned} \tag{3.15}$$

Therefore, in Dadda tree, a full-adder or an half-adder is placed only where strictly needed, in order to follow (3.15): if, in the *j-th* step, a column of the PPM has an height $h \le d_j$, the column is maintained unchanged for the next step. The Fig. 6 shows an example of Dadda's reduction tree, to an 8x8 multiplier.

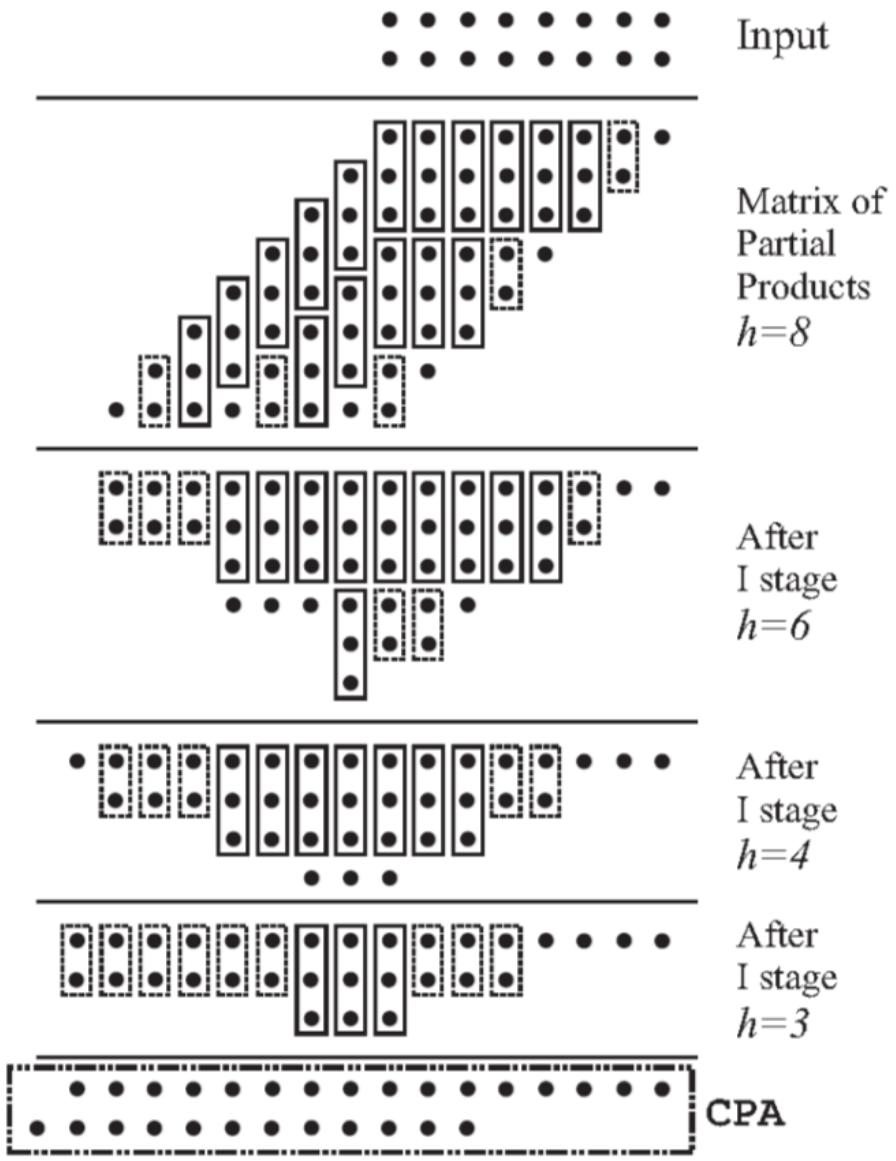

Fig. 5 Wallace reduction tree for 8x8 PPM. Full rectangle: full-adder; Dashed rectangle: half-adder. *h* is e partial product matrix in each step.

While the number of FAs and HAs is reduced with respect Wallace tree, the VMA is generally longer in Dadda multiplier.

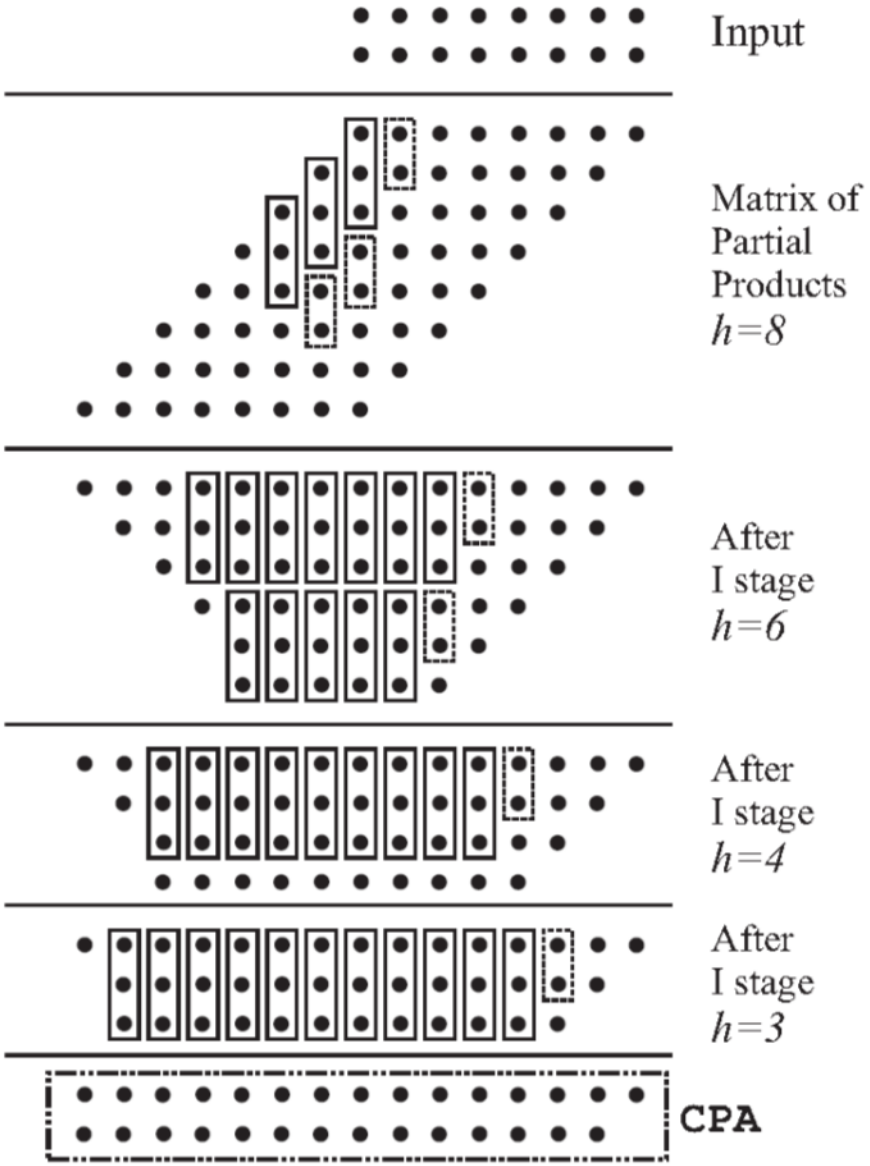

Fig. 6 Dadda reduction tree for 8x8 PPM. Full rectangle: full-adder; Dashed rectangle: half-adder. *h* is e partial product matrix in each step. The number of FAs and HAs is reduced with respect Fig. 5.

3.2.3.3 Three Dimensional Minimization (TDM) method

In [51] *Oklobdzija et al.* propose a further optimization of the reduction tree of partial products. After the introduction, due to *Dadda [62],* of the counters concept (a full-adder can be seen as a ones-counter, since its output indicate, in binary form, the number of ones at the input of the full adder), the researcher deeply investigated the implementation of bigger counters, such as (4,2) [63] and (9,2) [64] compressors. In [51] authors state that also when considering big compressors, such as (4,2), as a single cell, they can be always seen as composed by full-adders and that, operating a proper interconnection, the compressors composed by full-adders have the same speed of that treated as single cell. Therefore the optimization key is the interconnection between full-adders.

Starting from these observations, authors define an algorithm for optimized partial product reduction. In this algorithm a single big compressor is employed, composed by the appropriate connection of full adders. The connection takes into account the fact that the delay from an input to an output of a full adder is not the same. Therefore, accounting for the different arrival time of the signal in the array and for the different timing arcs characterizing a given full-adder (and half adder) the authors propose an algorithm which is able to employ fast inputs and outputs in the critical paths of the PPM and slow input and outputs in the non-critical path of the PPM. As a result, the delay of the tree is optimized.

3.2.4 Truncated multipliers

As discussed above, the multiplication of two n-bit operands results in a $2n$-bit output. In many applications this bit growth is avoided because such incremented precision is unnecessary, with benefits in terms of downstream hardware complexity.

As shown in Fig. 7, the PPM can be separated in two main regions: the Least Significant Part (LSP) which contains the partial products belonging to the n less significant columns of the PPM and the Most Significant Part (MSP) which includes the partial products belonging to the $n-1$ most significant columns. With this notation, the full-width (exact) output P can be expressed as:

$$P = S_{MSP} + S_{LSP} \tag{3.16}$$

where S_{MSP} and S_{LSP} represent, respectively, the weighed sum of the MSP and LSP elements.

The simple, but most expensive way to produce an n-bit output Pt is to truncate the full-width multiplier output P, the resulting multiplier is usually named as *full-rounded multiplier* [65]:

$$Pt = trunc_n\left(S_{MSP} + S_{LSP} + \frac{LSB}{2}\right) \tag{3.17}$$

Where $trunc_n$ means that n least significant bit are truncated, while LSB is the weight of lest significant bit of the truncated output Pt. Please note that in this case the multiplication operation is exact, and the error is exclusively introduced by the rounding operation. This results in little area and power savings, but in the smallest error $e = P - Pt$. Under the assumption of independent and uniformly distributed operands, the mean and the variance of the error e are:

$$\mu_{round} = E\left[e_{round}\right] = 0 \tag{3.18}$$

$$\varepsilon^2_{round} = E\left[e^2_{round}\right] = \frac{1}{12}LSB^2 \tag{3.19}$$

The Fig. 7 shows an unsigned full-rounded multiplier PPM.

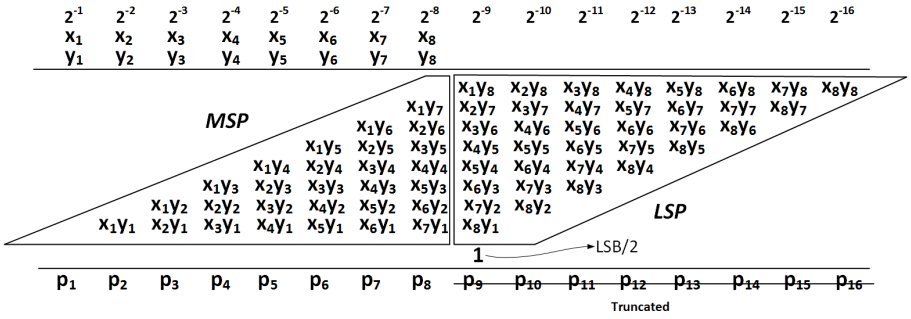

Fig. 7 Full-rounded unsigned multiplier.

Significantly higher performance improvements, at a cost of decreased accuracy, can be obtained discarding the partial products belonging to the LSP. Between these two extreme cases, a multitude of techniques have been proposed that discard part of the LSP partial products. These techniques propose also error compensation circuits.

Let us indicate the h most significant columns of the LSP as LSP_{major}, while the remaining $n_{eq}=n-h$ columns of the LSP are indicated as LSP_{minor}. The leftmost column of the LSP_{minor} is called Input Correction (IC) [65]. Please note that h is a design parameter ranging from $h=n$ to $h=0$. The Fig. 8 shows the different regions of the PPM in a truncated multiplier. The LSP_{minor} partial products are discarded and their contribution is estimated by means of the IC column.

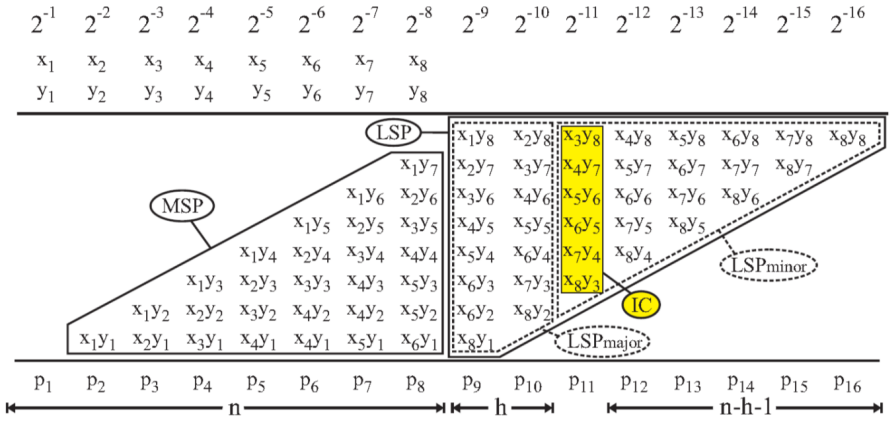

Fig. 8 PPM organization for truncated multipliers. The LSP_{minor} terms are discarded to save area and power. Their contribution is estimated by means of the IC column. h is a design parameter.

The Fig. 9 reports the general scheme of truncated multipliers. As shown in this figure, the PPM of a truncated multiplier involves the terms belonging to MSP and LSP_{major}, while the LSP_{minor} is estimated by the compensation function *f(IC)*; the multiplier output is then rounded, producing an *n-bit* output.

The compensation techniques can be divided into two sets: Constant Correction Methods (CCM) where the compensation function f is a constant and Variable Correction Methods (VCM). In the following an overview of these two different approach is given.

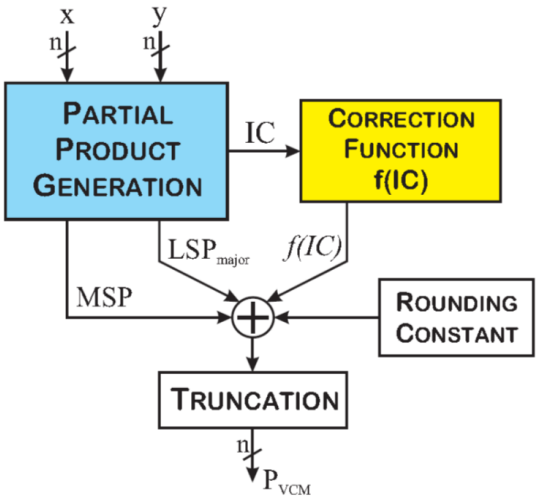

Fig. 9 General scheme of truncated multipliers.

3.2.4.1 Constant Correction Methods

In Constant Correction Methods (CCMs) the compensation function f is a constant:

$$Pt_{CCM} = trunc_n\left(S_{MSP} + S_{LSP_{major}} + \text{constant}\right) \tag{3.20}$$

A first discussion about truncated multipliers is reported in [66]. In this work the author proposes two method for error compensation. In the first one a constant estimates the mean value of LSP_{minor}, under the assumption of independent and uniformly distributed inputs. In the second one some terms of the IC are considered to estimate the sum of the correlated terms in the LSP_{minor}. The estimation is always done in terms of average value, using conditional probabilities and independent and uniformly distributed inputs assumption. This method, exploiting correlation between IC column and LSP_{minor} lowers the error, at expense of a more complex error correction circuit. The approach of [66] is expanded by [67], incorporating in the constant also the compensation of the rounding error.

In *Kidambi et al.* [68] a truncated multiplier with constant compensation is proposed. The multiplier discards all the partial

products belonging to the LSP. These terms are compensated with a fixed bias, which is always evaluated as average value of the neglected columns, assuming independent and uniformly distributed inputs. The resulting truncated multiplier, while reducing of about 50% area and power with respect a full-width multiplier, shows a large error that rapidly increase with *n*.

3.2.4.2 Variable Correction Methods

Truncated multipliers accuracy can be significantly increased, with some hardware overhead, employing a Variable Correction Method. In this case the truncated output writes as:

$$Pt_{VCM} = trunc_n\left(S_{MSP} + S_{LSP_{major}} + f(IC)\right) \tag{3.21}$$

As shown in Fig. 9, the compensation function *f(IC)* tries to compensate the neglected LSP_{minor} terms. Several solution for the compensation function *f* have been proposed.

In [69]-[71] the authors extend the method proposed in [66], leveraging the correlation between the *IC* column and the LSP_{minor}. In particular, the *IC* terms are "sensed" and the correction term is adjusted as a function on the *IC* sum: if all the *IC* terms as zero, due to the correlations, the LSP_{minor} will have a high number of zeroed partial products and therefore the correction term is decreased to zero. In the opposite case in which the *IC* column contains all ones, the correction term is adjusted to a maximum value. In [72] the authors propose a hybrid method, in which part of *IC* terms are used to obtain a variable term, while a constant term is obtained as a function of the remaining *IC* terms. In [73] the previous methods are improved by summing to the rightmost column of the LSP_{major} the sum of all the *IC* terms plus a further correcting bit. Authors in [74] propose a correction method for truncated multipliers with *h=0,* manipulating the *IC* terms by means of AND-OR operation. The resulting circuit implementation is slow, being characterized by a ripple architecture, moreover the mean and mean square error are significant. *Curticapean et al.* [75] provide a modified version of the [74], improving accuracy, but the error correction circuit is still based on a slow ripple architecture. In [76] *Van et al.* generalize the correction proposed in [74], considering in the compensation function either the *IC* terms or

their complements. In [77] a compensation function that minimize mean-square error or maximum absolute error is proposed. The compensation function is obtained heuristically. The solution provided by *Strollo et al.*in [77] is simplified in terms of hardware complexity in [78]. In [65] *Petra et al.* offer a closed-form solution for mean square error minimization. This optimal compensation function has a quadratic dependence on the terms of the *IC*. In this paper a sub-optimal compensation function, best suited for hardware implementation, which linearizes the optimal one is proposed also. The sub-optimal linear compensation function is further investigated in [79], in which the effect of the quantization of the coefficients of the linear compensation function is examined. In [80] *De Caro et al.* examine the problem of minimizing the maximum absolute error (MAE). To this regard the linear compensation function approach, developed in [65], [79], is employed to determine a novel linear compensation function that minimizes the MAE. The proposed approach is then expanded also to multiply and accumulate (MAC) units.

3.3 Precision-scalable MAC Unit

In this paragraph the proposed precision-scalable MAC unit, along with the proposed real-time data-aware compensation technique are discussed.

MACs units are the basic units of DSP processors, since a multitude of algorithms are based on Sum Of Products (SOP) (FIR filtering, image processing, machine learning). Recently, MACs units constitute the core of deep learning accelerators, being SOP a central operation in many machine learning algorithms, such as Convolutional Neural Networks (CNN). Due to the compute intensive role that SOP has in CNN algorithms, developing energy-efficient MAC operation is of great importance, facilitating the implementation of machine learning algorithms in mobile, battery-operated devices.

The starting point of my research activity regarding precision-scalable MAC unit is the work proposed by *de la Guia Solaz et al.* [57] which discusses about a programmable truncated multiplier. In this paper the authors propose a multiplier for general purpose systems which can be “programmed”, in the sense that its accuracy can be controlled with a fine-grain approach, as a function of the

application. The accuracy is modified by choosing the number of column to discard in the PPM of a multiplier. In this way the accuracy is traded for dynamic power. In this work, for the first time, the truncation is expanded also in the MSP region. The Fig. 10 shows the PPM proposed in [57]. As it can be observed, the signal t_j with *j* ranging from *0* to *2n-2* allows disabling each column of the PPM. The disabling is implemented imposing $t_j = 0$, as a result, all the partial products of the *j*-th column will be freezed to zero, eliminating any switching activity in the column.

Authors in [57] employ a Constant Compensation Method (CCM), in order to compensate for the neglected partial products. The compensation constant is calculated under the assumption of independent and uniformly distributed operands, with the same approach of [68]. Please note that, being the number of discarded columns imposed at run-time, different compensation constants must be provided as a function of the actual number of discarded columns. As a consequence, in [57] authors perform the compensation in software, observing that providing the constants for the different truncation levels can be hardware-expensive, reducing the power improvement achievable with the truncation. The precision-scalable multiplier is embedded in [57] in a DSP system and the number of column to be neglected is fixed through an internal register, whose value is controlled by a dedicated instruction set.

Starting from [57], my research activity focused on two contributions: i) developing, for the first time, an hardware compensation method for programmable truncated multipliers; ii) implement a data-aware, low-power, compensation technique. Regarding this second point, it is worthwhile observing that the Constant Compensation Method employed in [57], being based on specific assumption regarding input operands (independence and uniform distribution) does not meet the flexibility paradigm which constitutes the baseline for precision-scalable units: the ability to adapt to datasets statistics. Therefore my research activity focused on implement an adaptive ("data-aware"), low-power, compensation technique for precision scalable-units.

3.3.1 Real-Time Data-Aware Compensation Technique

The key motivation of precision-scalable unit is the ability to improve energy efficiency at expense of quality degradation of the results. In the case of precision-scalable unit, maximizing the accuracy achieved at a given precision level, without affecting the dissipated power, is of paramount importance, allowing performing a more aggressive precision scaling to trade the increased accuracy for power.

In the case of precision-scalable truncated multipliers this observation translates into an higher number of neglected column (i.e. less dynamic power dissipation) when the accuracy is maximized. Therefore an energy-efficient, precise, compensation technique is required. In [57] this challenge is solved by performing a software compensation, in which the multiplier result is corrected with the addition of a compensation constant estimated assuming that the neglected partial products are independent and uniformly distributed.

In this paragraph, the proposed low-power, real-time data-aware compensation technique is discussed. The basic idea to obtain a data-aware compensation is to "sense" the error done with a given precision level. In order to achieve a low-power circuital implementation, the sensing is performed every $F \gg 1$ multiplication.

Let's obtain an expression that describes the error in function of the precision level, in a precision-scalable truncated multiplier with mixed operands PPM. The precision-scaled output can be written as follows:

$$P_t(nt) = SE(nt) + \sum_{i=2}^{n} x_i y_j \cdot r(i+j) \cdot 2^{-i-j} + \sum_{i=1}^{n} \overline{x_1 y_j} \cdot r(i+1) \cdot 2^{-i-1} + K \tag{3.22}$$

where n_t is the number of discarded columns (as an example, n_t=0 means $t_j = 1 \ \ \forall j \in \{0, 2n-1\}$ therefore no column is discarded (compare with Fig. 10), while n_t=2 means $t_j = 1 \ \ \forall j \in \{2, 2n-1\}; t_0 = t_1 = 0$), *r(i+j)* is defined as follows:

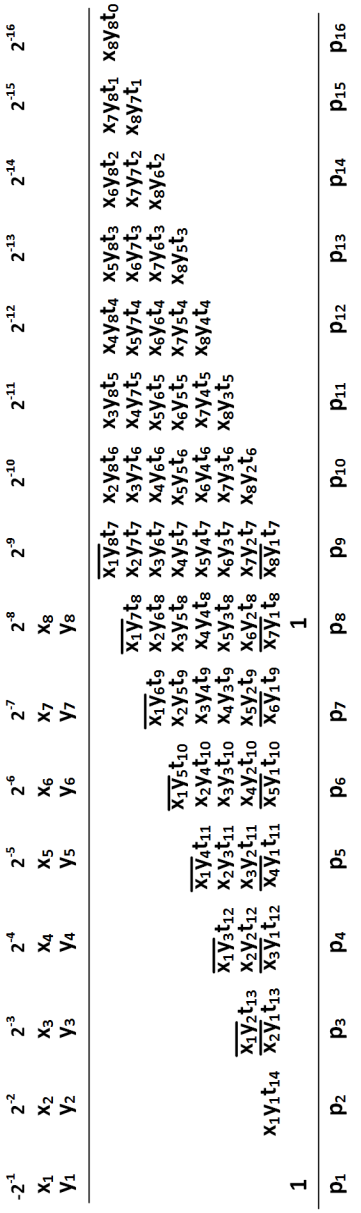

Fig. 10 Precision-scalable PPM for signed operands. The t_j signals allows to discarding the PPM columns with a fine-grain approach.

$$r(i+j)=\begin{cases}0 & if\ i+j>2n-nt\\ 1 & if\ i+j\le 2n-nt\end{cases} \tag{3.23}$$

$SE(n_t)$ is the sign-extension prevention constant, which is a function of the number of discarded columns, n_t:

$$SE\left(n_t\right)=\begin{cases} 2^{-1}+2^{-n-1} & if\ n_t<n \\ 2^{-1}+2^{-2n+n_t} & if\ n\le n_t\le 2n-2 \\ 0 & if\ n=2n-1 \end{cases} \tag{3.24}$$

while K is the compensation function, defined in the following. The exact output P, is expressed as follows:

$$P=SE+\sum_{i=2}^{n}\sum_{j=1}^{n}x_i y_j\cdot 2^{-i-j}+\sum_{i=1}^{n}\overline{x_1 y_i}\cdot 2^{-i-1} \tag{3.25}$$

being $SE=2^{-1}+2^{-n-1}$ as stated by (3.13). The MAC output can be expressed, for the precision-scalable circuit, as:

$$Z_t=trunc_{n+g}\left(\sum_{i=1}^{M}P_t\left(n_t\right)\right) \tag{3.26}$$

And for the exact circuit as:

$$Z=trunc_{n+g}\left(\sum_{i=1}^{M}P\right) \tag{3.27}$$

being M the number of multiplications to be accumulated and g the number of eventually employed guard-bit (n+g bits are truncated, providing an n-bits output). While (3.22) and (3.25) describe the multiplication operation, the (3.26) and (3.27) describe the accumulation one. Note that, with respect to (3.17), (3.20), (3.21) the truncation operation is not performed at the output of the multiplier, but at output of the MAC unit, avoiding, in this way, the accumulation of the truncation error.

It is worthwhile observing that, while the accumulation is done in an accurate way for both the precision-scaled and the exact MAC, the error source for the precision-scaled unit is given by the truncated multiplication (3.22). This error can be obtained as follows:

$$e(n_t) = P - P_t(n_t) = SE - SE(n_t) + \sum_{i=2}^{n}\sum_{j=1}^{n} x_i y_j \cdot \overline{r(i+j)} \cdot 2^{-i-j} + \sum_{i=1}^{n} \overline{x_1 y_j} \cdot \overline{r(i+1)} \cdot 2^{-i-j} - K \quad (3.28)$$

where

$$\overline{r(w)} = 1 - r(w) = \begin{cases} 1 & if\ w > 2n - n_t \\ 0 & if\ w \le 2n - n_t \end{cases} \quad (3.29)$$

The (3.28), (3.29) state that, substantially, the error can be calculated using a dual PPM, in which only the columns deactivated in the precision-scalable multiplier are activated.
In the proposed real-time, data-aware compensation technique, the error *e*, computed by the dual PPM, is sampled with an aggressive subsampling period *F*, and each *L* samples a mean error e_m is evaluated:

$$e_m(u) = \frac{e(u-(L-1)F) + e(u-(L-2)F) + \ldots + e(u)}{L} \quad (3.30)$$

The (3.30) states that the mean error e_m, at a given clock period *u* is evaluated accumulating the actual sample and the previous *L-1* ones, each of them far from the contiguous of *F* clock periods, due to the subsampling.
It is worthwhile recalling that the mean square error (*MSE*) is directly related to mean error μ (whose e_m constitutes an estimate), as follows:

$$MSE = \sigma^2 + \mu^2 \quad (3.31)$$

where σ^2 is the error variance. Therefore compensating the mean error represents a first step for *MSE* minimization. This observation constitutes the baseline of all the Constant Correction Methods. Please note that the MSE is directly related to *PSNR* definition, which represents a standard quality metric for images:

$$PSNR = 20\log_{10}\left(\frac{MAX\{I\}}{\sqrt{MSE}}\right) \tag{3.32}$$

In (3.32) $MAX\{I\}$ represents the maximum pixel value in the image I (for an 8-bits grayscale image $MAX\{I\} = 255$). In the following, we impose the mean error compensation. The difference with Constant Correction Methods is that in the proposed approach the mean error is estimated on the actual data statistics, therefore the compensation term K is adaptive and the compensation term can be defined "data-aware".

Therefore, the mean error is compared with zero: if the comparison is true (e_m=0) the compensation K is not updated, while if the mean error e_m is not zero the compensation term is updated as follows:

$$K(u) = \begin{cases} K(u - L\cdot F) & if \quad e_m(u) = 0 \\ K(u - L\cdot F) + e_m(u) & if \quad e_m(u) \neq 0 \end{cases} \tag{3.33}$$

Note that the update of e_m and K happens every $L\cdot F$ clock cycles, due to the subsampling and the accumulation. When $e_m \neq 0$ the compensation term is corrected summing the previous K to the actual mean error, in this way, if $e_m < 0$, therefore the truncated product is over-compensated (compare with (3.28)), the compensation terms is decreased, in order to decrease the over-compensation error and vice versa when $e_m > 0$.

3.3.1.1 Circuit Overview

In this paragraph a circuital overview is discussed. The Fig. 11 shows a top-level view of the proposed approach.

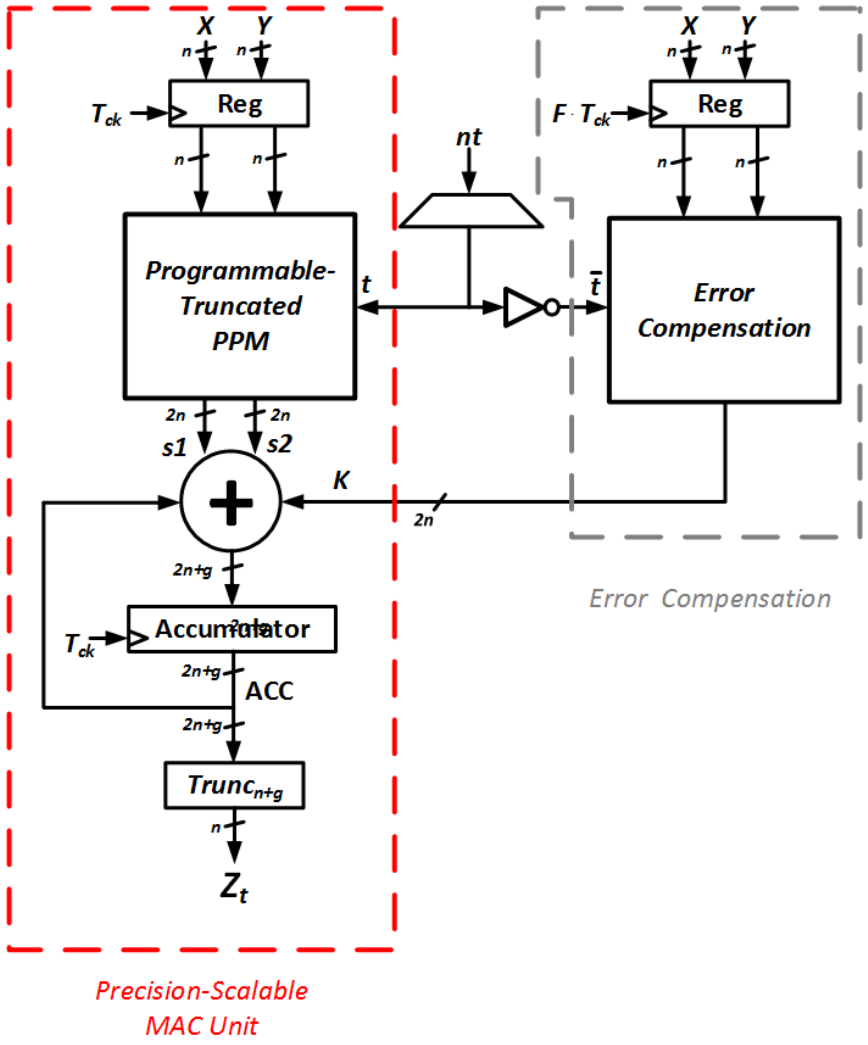

Fig. 11 Top level view of the proposed circuit. Note that the Error Compensation block works at scaled frequency and that provides the compensation term *K* to the Precision-Scalable MAC Unit.

The input operands *X, Y* are fed with a *Tck* clock period to the programmable-truncated PPM, in which n_t columns are deactivated. A combinatory circuit acts as decoder to translate the n_t signals into *2n-1* t_j signals of Fig. 10. The programmable-truncated PPM can be implemented as TDM, carry-save tree, providing two outputs in carry-save format *s1* and *s2*, to be added in order to complete the

multiplication. These two signals are added with the compensation term *K* provided by the error compensation circuit, and with the output of the accumulator register, in order to perform a MAC operation. The result is then truncated in order to provide an *n*-bits output (3.22), (3.26). Regarding the error compensation circuit, the operands *X* and *Y* are sampled with a clock period *F* times bigger than *Tck*. Note that, in order to compute the error *e,* the columns neglected in the programmable-truncated PPM must be taken into account, therefore the signal *t* passes through a logic inverter and is fed to the Error-Compensation block, in order to implement (3.28). The mean error computation e_m (3.30) and the *K* control (3.33) are also implemented in this block. Let us discuss, in more details, the Precision-Scalable MAC Unit and the Error Compensation blocks.

3.3.1.1.1 Precision-Scalable MAC Unit

The Precision-Scalable MAC Unit (Fig. 11) is composed by the Programmable-Truncated PPM, a multi-operands adder and an accumulation register.

The Programmable-Truncated MAC Unit is similar to that represented in Fig. 10, with the exception that no sign extension prevention constant is introduced. Indeed, as stated by (3.24), the sign extension (*SE*) constant is function of the number of discarded columns n_t, therefore its implementation requires a LUT. Note that a constant in the PPM can be easily accounted in the carry-save tree implementation with negligible increased complexity; this is not the case for $SE(n_t)$, assuming different values as a function of the discarded columns: in this case an additional row of partial products must be accounted, with consequent lower performance in terms of speed and power. In order to avoid this overhead, the LUT is not implemented, and the $SE(n_t)$ is controlled by the Error Compensation circuit, embedding its value in the compensation term *K*. At the reset of the circuit, the compensation term *K* is initialized to the *SE* constant corresponding to the condition $n_t = 0$. Its value is then adjusted automatically as a function of n_t, thanks to the constraint $e_m = 0$ (3.33) . As a result the expression for $P_t(nt)$ (3.22) becomes:

$$P_t(nt) = \sum_{i=2}^{n} x_i y_j \cdot r(i+j) \cdot 2^{-i-j} + \sum_{i=1}^{n} \overline{x_1 y_j} \cdot r(i+1) \cdot 2^{-i-1} + K \tag{3.34}$$

while the expression for error calculation (3.28) modifies as follows:

$$e(n_t) = SE + \sum_{i=2}^{n} \sum_{j=1}^{n} x_i y_j \cdot \overline{r(i+j)} \cdot 2^{-i-j} + \sum_{i=1}^{n} \overline{x_1 y_j} \cdot \overline{r(i+1)} \cdot 2^{-i-j} - K \tag{3.35}$$

Note that also in the (3.35) the LUT is avoided.

The Programmable-Truncated PPM provides as output the two signals *s1* and *s2* in carry-save format. These two signals must be added with the compensation term *K* and with the output of the accumulator *ACC* through a multi-operand adder; these two signals are sign extended with *g* guard bit before entering in the carry-save tree (the multi-operand adder can be implemented as carry-save tree). Note that, as observed for $SE(n_t)$, the addition of the compensation term represents a significant overhead, requiring an additional row in the tree, of length $2n+g$, slowing down the circuit and increasing the power dissipation. This overhead can be avoided, initializing, at the start of the *M* multiplications (compare (3.26)) , the accumulator register with the compensation term *K* multiplied *M* times. This requires an additional multiplexer in the Precision-Scalable MAC Unit (which is simpler and faster than a row of full-adders) and a constant multiplier (implemented with shifts and additions) in the Error Compensation block. Note that this multiplier, while increasing the area occupation, does not impact the energy efficiency of the circuit, due to the low working frequency of the Error Compensation block ($F \gg 1$). The resulting scheme for the Precision-Scalable MAC Unit is shown in Fig. 12, where the *rst_acc* signal becomes high when a new set of *M* multiplication starts and goes down in the next clock cycle. Note that a register has been added at the output of the unit; this register can be easily clock-gated and enabled when a set of *M* multiplications is computed.

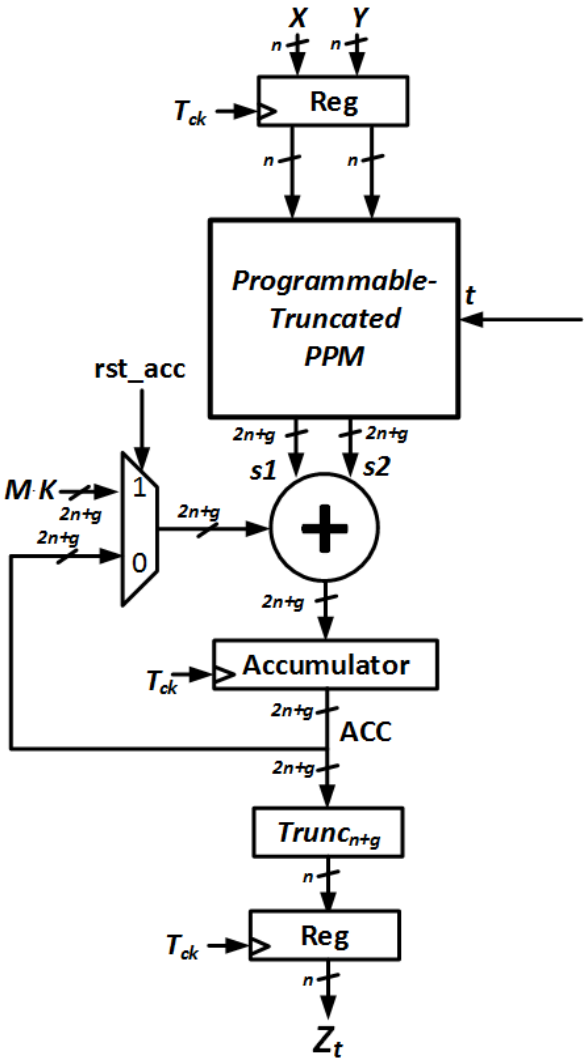

Fig. 12 Precision-Scalable MAC Unit. Note that the compensation terms is initialized at each new set of *M* multiplications through the multiplexer.

3.3.1.1.2 Error Compensation

The Error Compensation block is shown in detail in the Fig. 13. After a sampling stage, implemented with a register operating with a clock period equal to $F \cdot T_{ck}$, the sampled inputs and the $\bar{t}$

signal are fed to a Programmable-Truncated PPM that is similar to that in the Precision-Scalable MAC Unit, with the exception of an additional row, accounting for *SE* (3.35). The two outputs in carry save format *s1* and *s2* and the actual correction term $K\left(u - L \cdot F\right)$ are summed in order to compute the error *e*. In addition to them, an extra signal is added, which is the output of the accumulator, needed to perform the mean operation. Note that this multi-operand adder can be, as previously discussed, implemented as a carry-save tree, in which g_e guard bit are included, due to the accumulation of the *L* samples. At the end of the accumulation (i.e. when *L* samples are accumulated), a division by *L* must be performed (3.30). In order to save hardware and power, we impose the constraint that *L* is a two power. In this way the division can be trivially implemented as a variation of representation followed by LSB truncation, therefore no extra-hardware is needed, being the implementation a wiring matter. In order to avoid unnecessary switching in the downstream circuits, a multiplexer is inserted, clamping to zero its output when the accumulation is not completed (*RSTcntK=0)*. The e_m signal is then compared with zero in order to implement the control mechanism (3.33). Note that the output of the adder is multiplied by *M*. This multiplier can be significantly simplified, as already discussed, being a constant multiplier. Observe, moreover, that the final multiplexer can be eliminated if the clock of the output register is produced by a CGIC (Clock Gate Integrated Cell), whose enable is the signal *!Update*, with beneficial effects on the dynamic power dissipation.

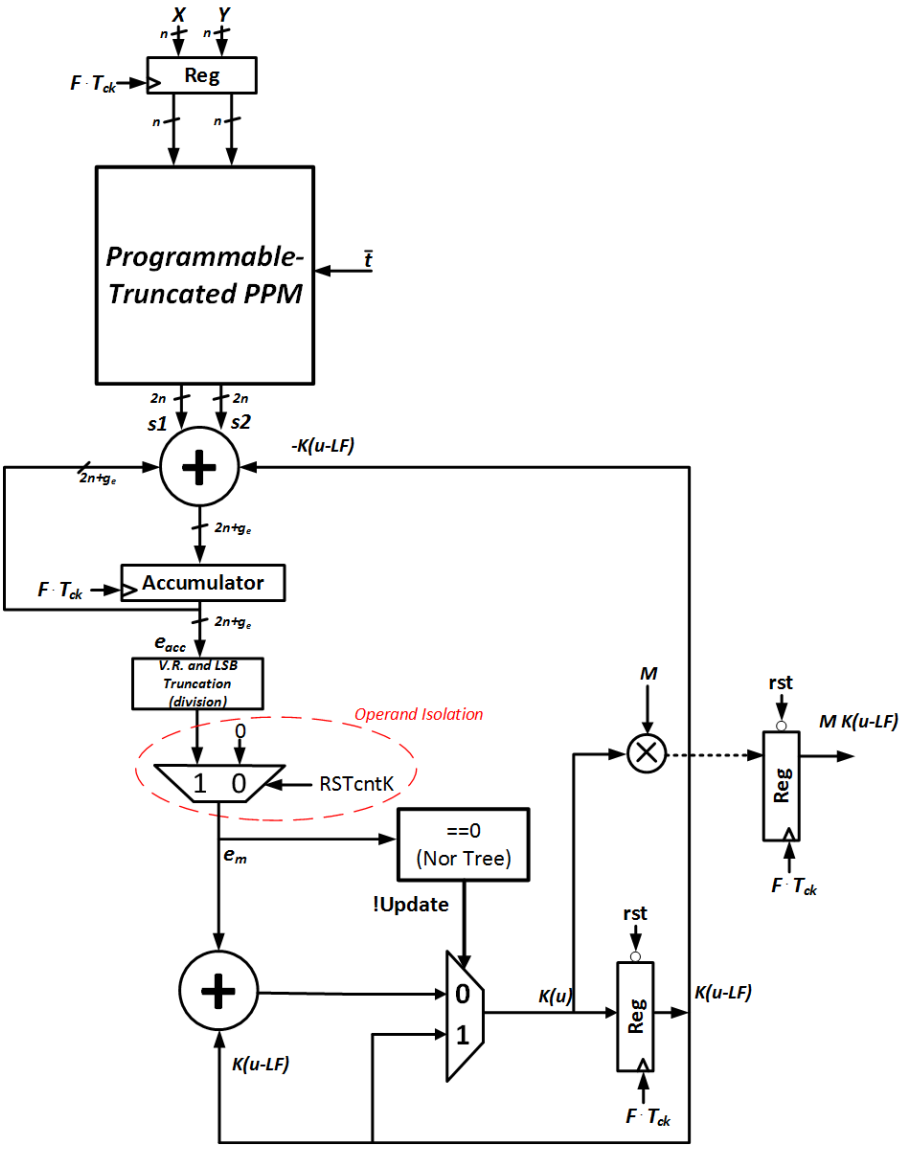

Fig. 13 Error Compensation circuit. The discarded column in the Precision-Scalable MAC Unit are now activated in order to compute the committed error. The error (subsampled with a factor F, note the input register) is then accumulated and the mean error is evaluated. A control circuit, in function of the actual mean error manage the compensation term value. The actual compensation term is then multiplied by M though a constant multiplier and given to the Precision-Scalable MAC Unit.

3.3.1.2 Parameters choice

In this paragraph an heuristic analysis on image processing applications is carried out in order to establish the values of *F* and *L* parameters able to assure the best quality-energy ratio.

3.3.1.2.1 2D Convolution

Before analyzing the results of the analysis, let us briefly introduce the 2D convolution, which is a standard operation in computer vision applications. It allows to filter an image through a filter, that, in this context, is usually named as "kernel". The filtering process happens performing a 2D convolution: the kernel, usually represented as a squared matrix *qxq*, is overlapped on all the possible *qxq* image windows, and, for each window, a SOP operation between kernel coefficients and homologous image pixels is performed, producing, as output, a filtered pixel, as represented in Fig. 14.

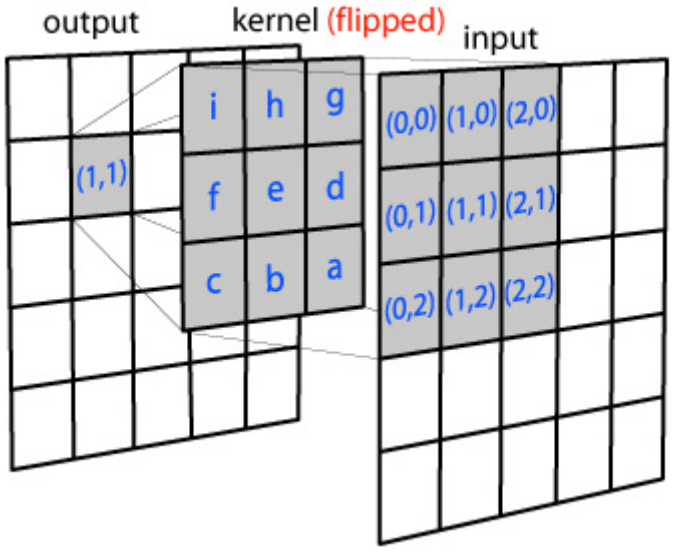

Fig. 14 Filtering of an image window through a kernel. A single pixel is produced, corresponding to the central position of the window.

The operation can be formally expressed as [81]:

$$h[i,j] = I[i,j] \otimes k[i,j] = \sum_{r=1}^{q} \sum_{c=1}^{q} I[r,c] \cdot k[i-r, j-c] \quad (3.36)$$

where I is the original image, h the filtered one, k is the kernel and (i,j) is a generic pixel. The (3.36) must be extended to the remaining (i',j') pixels belonging to the original image.

3.3.1.2.2 Case study for Gaussian kernel

In order to find out the effect on the image quality due to different choices of parameters F and L, the 2D convolution between "Lena" image and the following Gaussian kernel has been investigated:

$$k_g = \begin{bmatrix} 0.0392 & 0.0398 & 0.0400 & 0.0398 & 0.0392 \\ 0.0398 & 0.0404 & 0.0406 & 0.0404 & 0.0398 \\ 0.0400 & 0.0406 & 0.0408 & 0.0406 & 0.0400 \\ 0.0398 & 0.0404 & 0.0406 & 0.0404 & 0.0398 \\ 0.0392 & 0.0398 & 0.0400 & 0.0398 & 0.0392 \end{bmatrix} \quad (3.37)$$

The (3.37) kernel has been obtained through the Matlab command `fspecial('gaussian',5,10)`. In the following, the result for $n_t=6$ are presented. Please note that the following considerations are pretty general and the following behavior is observed with others kernels and n_t values.

The Fig. 15 shows PSNR results for the convolution between Lena image and (3.37) kernel, when six columns of the PPM are neglected. Please observe that L is varied in the $[4;64]$ range, while F in $[2;2048]$. The results for the case in which the error is not compensated and is compensated assuming the uniform distribution are also shown in this figure, for comparison. Please note that x-axis, in Fig. 15 is in logarithmic scale.

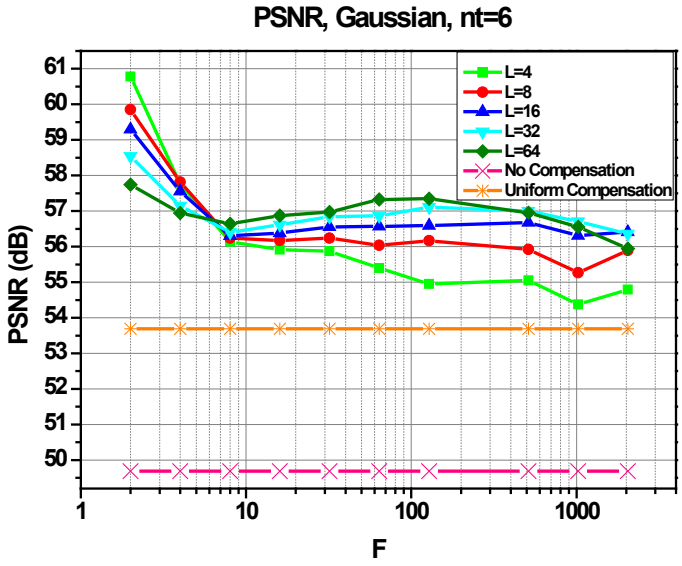

Fig. 15 PSNR result for the proposed approach and for the Uniform and No Compensation ones, in the case of Gaussian kernel. For the proposed approach different value of L are shown, while the x-axis is subsampling factor F.

As observable in Fig. 15, for $F<10$ the PSNR is more than 10dB higher than the case of no compensation and more than 7dB higher than that of uniform compensation. Unfortunately, a such relatively frequent subsampling factor highly impacts on energy efficiency of the proposed MAC unit. A key observation is that the PSNR, after a rapid decrease, remains approximately flat for $F>10$, decreasing only for very high subsampling values (approximately for $F>512$). This behavior is found also with other filters analyzed. Note, moreover, that in the flat region, the PSNR tends to increase with L (the opposite happens for $F<10$), this can be explained with the fact that in this region, due to the high subsampling value, the sampled errors e (3.28) tend to be uncorrelated, therefore the mean error e_m tends to be a bad estimation of the real mean error μ, when the number of accumulated samples L is too low. Instead, in the region with $F<10$, the samples tend to be correlated, therefore the mean error e_m is a good estimation also when L is low: in this region, keeping L low offers the best results because the response time of the

control decreases with L (and F), allowing to rapidly compensate the errors made.

As a result of these observations, to improve energy-efficiency, the proposed compensation technique will work in the flat region. In particular, the following parameters have been chosen:

$$\begin{cases} F^* = 128 \\ L^* = 64 \end{cases} \tag{3.38}$$

Note that such choice increases PSNR of more than 3dB with respect uniform compensation and of more than 7dB with respect no compensation, for the case of Fig. 15. In the following more detailed results are given for different kernels, images and n_t values.

3.3.1.3 Results

In this paragraph the quality and VLSI implementation results are discussed.

3.3.1.3.1 Quality results

The proposed real-time data-aware compensation technique has been assessed on different images and kernels. A comparison with the cases of No Compensation and Uniform Compensation is provided. The following three test images have been considered for the next quality results:

Fig. 16 Cameraman test image, resolution: 204x204.

Fig. 17 Lena test image, resolution: 223x292.

Fig. 18 Airplane test image, resolution: 288x511.

The following kernels have been considered:

$$k_{average} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.39}$$

$$k_{blur} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.40}$$

$$k_{emboss} = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{3.41}$$

$$k_{sharp} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{3.42}$$

$$k_{sobel\_v} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{3.43}$$

In addition to Average (3.39), Blur (3.40), Emboss (3.41), Sharp (3.42), Sobel Vertical (3.43), Gaussian (3.37), two kernels extracted from an actual Convolutional Neural Network (CNN) (VGG-F [82]) have been employed, one of size 11x11 “VGG 11x11”, another of size 3x3 “VGG3x3”. For simplicity, only the values of VGG3x3 are reported:

$$k_{VGG3x3} = \begin{bmatrix} 0.0254 & -0.0044 & -0.0174 \\ 0.0223 & -0.0126 & -0.0066 \\ 0.0146 & -0.0005 & 0.0108 \end{bmatrix} \tag{3.44}$$

Note that the values of CNNs kernels are the result of a training procedure. The Kernels have been quantized assuming different representation on 8-bits, since of their significant difference in terms of absolute value. From an hardware perspective this corresponds to employ a shifter after the MAC operation, this is a common design choice in all practical Digital Signal Processors.

In the following tables, the quality results, expressed in terms of PSNR (3.32) and Structural Similarity Index (SSIM) [50] are shown for the employed kernels and for Lena, Airplane and Cameraman test image. In the tables, the maximum PSNR and SSIM for each row (corresponding to a given n_l) are reported in red, while the maximum values higher, respectively, than 3 *dB* and 0.1 (please note that SSIM is dimensionless, while PSNR is expressed in *dB*) , for PSNR and SSIM with respect the second best value in the row, are reported in bold red.

Table I shows quality results for Average kernel. As it can be observed, the proposed approach exhibits, almost always, the best PSNR, while in terms of SSIM negligible differences appear between proposed and uniform approaches, while the results corresponding to the No Compensation show worse performance. In some cases, especially for aggressive precision-scaling levels, proposed approach allows increasing PSNR of more than 3°*dB*, up to 10 *dB*.

The results corresponding to Blur kernel are shown in Table II. In this case, the proposed approach always shows the best result in terms of PSNR, with an improvement more than 3°*dB* in almost all cases, with a peak of 16°*dB*. In terms of SSIM negligible differences exist with uniform approach (note that No Compensation approach offers good results when non aggressive precision-scaling is performed).

Table III shows the performance for Emboss Kernel. In this case proposed approach offers always the best results both in terms of PSNR and SSIM, where results are significant improved with respect Uniform and No Compensation case. Note that, in terms of PSNR, the Uniform approach exhibits significantly lower performance than No Compensation one. Also in terms of SSIM, No Compensation offers better performance than Uniform approach, when a non-aggressive precision scaling is performed. The Uniform approach tends to be unsuitable for this kind of kernel, due to the presence of a integer coefficients, resulting in a significant number of zeroed LSB. As a result uniform approach results in an over-compensation, which degrades SSIM and PSNR in higher measure.

In Table IV quality performance for Gaussian Kernel are reported. Also in this case the proposed approach exhibits the best PSNR values, with improvement up to 9°*dB*. In terms of SSIM the performance are comparable with that of Uniform approach, while for

the No Compensation one, performance rapidly decreases with high number of discarded columns.

For the case of Sharp kernel, the results are shown in Tab. V. Also with this kernel, proposed technique exhibits significantly better PSNR and SSIM results with respect the other two approaches. Note that No Compensation approach offers better PSNR performance with respect Uniform one, while the results are about the same in terms of SSIM: in both the case the SSIM rapidly decreases increasing n_t.

Table VI reports the result for Sobel Vertical Kernel. In this case the consideration are similar to that made for Emboss Kernel: proposed approach shows the best performance in terms of PSNR and SSIM and the Uniform approach fails to compensate the errors, performing worse than the No Compensation one.

The Table VII shows the results for the VGG11x11 kernel. In this case similar results are obtained using proposed approach and Uniform one. In particular, proposed technique often results in better PSNR, while the SSIM is often better for the case of uniform compensation, however no significant differences are observed. Note that No Compensation approach is the worst one in this case, rapidly degrading its performance quality when a significant number of columns is neglected. Similar considerations apply to VGG3x3 (Table VIII), in which the performance between proposed and uniform approach are almost the same.

At the end of this discussion it is worthwhile observing that, in some cases the No Compensation approach offers better results than uniform one and in other cases vice versa, while proposed approach exhibits, in all the analyzed cases, always significant better or, at least, comparable quality performance with respect the best among No Compensation and uniform approaches, showing an adaptive capacity to the dataset, which constitutes the basic motivation of the proposed approach. Observe, moreover, that, both PSNR and SSIM measure the quality of image (although with different theoretical considerations) , but, while the SSIM is intrinsically related to image processing applications, the PSNR, being a ratio between a constant and the root mean square error (3.32) is a rigorous measure of the mathematical error done. With this consideration it is worthwhile observing that proposed approach, improving especially the PSNR (due to the constrain on the mean error imposed in (3.33)), offers the smaller

mathematical error. This allows to employ proposed technique also for other application contexts.

As final remark, note that the quality dependence on the kernel is stronger than that on the image.

TABLE I. QUALITY RESULTS FOR AVERAGE KERNEL (F=128, L=64)

n_t	Image	Proposed		No Compensation		Uniform	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
6	Lena	57.45	0.99	50.55	0.99	56.59	0.99
7	Lena	53.05	0.98	42.88	0.99	53.23	0.98
8	Lena	47.49	0.97	33.70	0.97	46.79	0.97
9	Lena	41.94	0.93	27.23	0.93	40.80	0.94
10	Lena	35.51	0.83	20.84	0.78	33.55	0.84
11	Lena	**28.31**	0.66	14.63	0.46	24.32	0.68
12	Lena	**23.35**	0.50	9.53	0.12	17.73	0.50
13	Lena	**19.28**	0.26	7.69	0.00	13.44	0.23
6	Cameraman	56.78	0.96	50.80	0.92	55.72	0.95
7	Cameraman	51.87	0.92	43.27	0.92	52.59	0.92
8	Cameraman	46.31	0.84	34.53	0.84	47.17	0.85
9	Cameraman	39.59	0.73	26.89	0.70	38.91	0.75
10	Cameraman	34.05	0.65	21.20	0.57	33.46	0.66
11	Cameraman	27.92	0.53	15.89	0.44	26.82	0.56
12	Cameraman	21.14	0.40	9.91	0.20	18.76	0.44
13	Cameraman	**16.68**	0.29	5.82	0.00	13.77	0.31
6	Airplane	57.10	0.96	50.19	0.95	56.76	0.96
7	Airplane	53.64	0.94	43.76	0.94	53.45	0.94
8	Airplane	48.44	0.90	33.40	0.91	45.66	0.90
9	Airplane	**40.87**	0.83	25.80	0.83	35.94	0.83
10	Airplane	**36.04**	0.68	20.56	0.67	32.35	0.69
11	Airplane	**31.12**	0.55	14.66	0.49	24.79	0.55
12	Airplane	**26.32**	0.49	8.40	0.29	16.57	0.48
13	Airplane	**19.41**	0.29	2.97	0.00	9.24	0.23

TABLE II. QUALITY RESULTS FOR BLUR KERNEL (F=128, L=64)

n_t	Image	Proposed		No Compensation		Uniform	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
6	Lena	**56.18**	0.99	50.97	0.99	41.01	0.99
7	Lena	**52.32**	0.98	43.95	0.98	34.10	0.98
8	Lena	**45.54**	0.96	30.06	0.96	28.04	0.96
9	Lena	**38.11**	0.89	21.58	0.84	23.51	0.88
10	Lena	**31.20**	0.75	14.54	0.50	19.96	0.74
11	Lena	**24.58**	0.57	9.24	0.11	17.56	0.56
12	Lena	**20.77**	0.39	7.67	0.00	15.23	0.38
13	Lena	**17.50**	0.16	7.67	0.00	12.24	0.17
6	Cameraman	**55.03**	0.94	51.07	0.92	41.05	0.95
7	Cameraman	**50.43**	0.88	43.90	0.92	34.04	0.86
8	Cameraman	**43.12**	0.81	29.96	0.78	28.03	0.80
9	Cameraman	**35.90**	0.70	22.05	0.62	23.48	0.69
10	Cameraman	**29.43**	0.60	15.18	0.46	19.52	0.58
11	Cameraman	**23.25**	0.47	9.30	0.17	15.57	0.46
12	Cameraman	**18.12**	0.33	5.78	0.00	13.38	0.35
13	Cameraman	14.53	0.19	5.78	0.00	11.90	0.20
6	Airplane	**56.03**	0.96	50.90	0.94	41.04	0.95
7	Airplane	**51.96**	0.92	43.79	0.93	34.16	0.92
8	Airplane	**45.84**	0.87	29.83	0.88	28.19	0.88
9	Airplane	**38.54**	0.76	21.20	0.76	23.92	0.77
10	Airplane	**32.67**	0.61	14.62	0.56	19.40	0.62
11	Airplane	**28.30**	0.50	8.27	0.27	16.93	0.49
12	Airplane	**22.45**	0.40	2.94	0.00	16.42	0.40
13	Airplane	16.28	0.19	2.94	0.00	15.85	0.16

TABLE III. QUALITY RESULTS FOR EMBOSS KERNEL (F=128, L=64)

n_t	Image	Proposed		No Compensation		Uniform	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
6	Lena	∞	**1**	∞	**1**	29.77	0.71
7	Lena	**48.56**	0.98	41.61	0.91	22.51	0.52
8	Lena	**41.73**	**0.94**	26.66	0.37	16.01	0.38
9	Lena	**35.40**	**0.84**	20.80	0.15	10.51	0.24
10	Lena	**28.32**	**0.65**	17.00	0.04	5.63	0.14
11	Lena	**21.25**	**0.44**	14.92	0.01	1.83	0.06
12	Lena	14.06	**0.29**	14.65	0.00	1.14	0.03
13	Lena	7.27	0.11	**14.65**	0.00	1.72	0.03
6	Cameraman	∞	1	∞	**1**	29.19	0.58
7	Cameraman	**46.89**	**0.82**	42.57	0.71	22.03	0.44
8	Cameraman	**40.43**	**0.69**	28.03	0.38	15.73	0.32
9	Cameraman	**33.64**	**0.60**	21.59	0.22	10.38	0.23
10	Cameraman	**26.82**	**0.53**	16.81	0.11	5.65	0.15
11	Cameraman	**20.53**	**0.42**	13.39	0.02	1.79	0.07
12	Cameraman	13.87	**0.29**	12.96	0.02	1.08	0.03
13	Cameraman	13.06	**0.18**	12.96	0.02	0.75	0.03
6	Airplane	∞	**1**	∞	**1**	29.46	0.55
7	Airplane	**48.84**	**0.94**	42.35	0.76	22.12	0.40
8	Airplane	**41.94**	**0.84**	28.37	0.31	15.71	0.28
9	Airplane	**35.16**	**0.67**	22.01	0.16	10.29	0.19
10	Airplane	**27.60**	**0.48**	17.52	0.05	5.68	0.12
11	Airplane	**21.51**	**0.35**	15.23	0.01	2.04	0.06
12	Airplane	16.62	**0.21**	14.52	0.00	0.68	0.02
13	Airplane	9.80	**0.16**	**14.52**	0.00	0.00	0.05

TABLE IV. QUALITY RESULTS FOR GAUSSIAN KERNEL (F=128, L=64)

n_t	Image	Proposed		No Compensation		Uniform	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
6	Lena	**57.35**	0.99	49.49	0.98	53.69	0.98
7	Lena	53.17	0.98	41.79	0.98	51.51	0.98
8	Lena	48.42	0.97	32.62	0.97	47.39	0.97
9	Lena	42.91	0.95	26.15	0.94	41.72	0.96
10	Lena	37.81	0.87	19.89	0.81	37.46	0.89
11	Lena	29.84	0.71	14.07	0.46	28.93	0.74
12	Lena	25.85	0.59	9.21	0.10	23.07	0.60
13	Lena	**20.08**	0.32	7.84	0	16.71	0.30
6	Cameraman	**56.87**	0.95	49.86	0.92	53.25	0.92
7	Cameraman	**52.52**	0.91	42.64	0.92	49.77	0.92
8	Cameraman	47.77	0.87	33.30	0.87	45.49	0.87
9	Cameraman	40.97	0.77	25.86	0.74	40.51	0.79
10	Cameraman	36.03	0.68	20.25	0.60	35.56	0.70
11	Cameraman	30.41	0.59	15.36	0.44	29.96	0.60
12	Cameraman	22.47	0.46	8.67	0.15	20.46	0.50
13	Cameraman	20.30	0.37	6.02	0.00	19.43	0.39
6	Airplane	**57.16**	0.94	49.60	0.93	53.73	0.93
7	Airplane	**54.25**	0.92	42.54	0.91	50.66	0.92
8	Airplane	48.67	0.89	31.91	0.90	48.01	0.90
9	Airplane	40.70	0.82	24.65	0.86	39.03	0.86
10	Airplane	38.16	0.70	19.81	0.71	37.55	0.74
11	Airplane	**30.55**	0.56	13.30	0.52	27.29	0.60
12	Airplane	**26.73**	0.49	7.78	0.29	20.89	0.54
13	Airplane	**22.20**	0.34	3.09	0.00	13.93	0.34

TABLE V. QUALITY RESULTS FOR SHARP KERNEL (F=128, L=64)

n_t	Image	Proposed		No Compensation		Uniform	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
6	Lena	**35.96**	0.96	26.96	0.95	17.99	0.91
7	Lena	**26.77**	**0.82**	16.65	0.69	11.81	0.69
8	Lena	**20.22**	**0.62**	8.02	0.10	6.39	0.38
9	Lena	**14.14**	**0.41**	6.91	0.00	4.15	0.10
10	Lena	9.22	**0.24**	6.84	0.00	3.90	0.02
11	Lena	6.46	**0.11**	6.84	0.00	3.98	0.01
12	Lena	4.65	0.01	6.84	0.00	4.31	0.01
13	Lena	4.58	0.00	6.84	0.00	4.31	0.00
6	Cameraman	**34.34**	0.77	27.07	0.74	18.10	0.71
7	Cameraman	**25.41**	0.62	16.99	0.55	12.01	0.52
8	Cameraman	**18.15**	**0.53**	6.91	0.15	7.92	0.32
9	Cameraman	**12.97**	**0.41**	5.16	0.01	5.35	0.10
10	Cameraman	**9.35**	**0.26**	5.09	0.00	4.48	0.02
11	Cameraman	5.17	0.09	5.09	0.00	4.53	0.03
12	Cameraman	3.61	0.00	5.09	0.00	5.02	0.00
13	Cameraman	3.61	0.00	5.09	0.00	4.53	0.00
6	Airplane	**35.96**	0.92	26.63	0.92	18.01	0.89
7	Airplane	**26.71**	0.71	16.05	0.67	13.64	0.51
8	Airplane	**19.91**	**0.52**	5.14	0.17	10.16	0.23
9	Airplane	**13.17**	**0.33**	2.79	0.00	8.35	0.06
10	Airplane	**9.43**	**0.20**	2.75	0.00	7.97	0.01
11	Airplane	6.40	0.09	2.75	0.00	8.07	0.01
12	Airplane	5.77	0.01	2.75	0.00	8.43	0.00
13	Airplane	2.95	0.00	2.75	0.00	2.55	0.00

TABLE VI. QUALITY RESULTS FOR SOBEL VERTICAL KERNEL (F=128, L=64)

n_t	Image	Proposed		No Compensation		Uniform	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
6	Lena	**47.17**	0.99	41.47	0.98	22.70	0.58
7	Lena	**37.11**	**0.95**	28.44	0.79	16.61	0.33
8	Lena	**29.52**	**0.86**	14.44	0.33	10.69	0.15
9	Lena	**21.98**	**0.68**	6.03	0.05	5.53	0.01
10	Lena	**15.66**	**0.47**	1.39	0.00	1.84	0.00
11	Lena	**10.31**	**0.28**	1.39	0.00	1.47	0.00
12	Lena	**4.96**	**0.12**	1.39	0.00	1.45	0.00
13	Lena	3.49	0.02	1.39	0.00	1.45	0.00
6	Cameraman	**47.04**	0.92	41.70	0.84	22.77	0.48
7	Cameraman	**36.07**	**0.69**	28.96	0.56	16.31	0.35
8	Cameraman	**27.24**	**0.59**	14.26	0.30	10.33	0.21
9	Cameraman	**19.44**	**0.51**	5.69	0.10	5.20	0.08
10	Cameraman	**13.92**	**0.42**	1.07	0.00	1.50	0.00
11	Cameraman	**9.89**	**0.27**	1.06	0.00	1.10	0.00
12	Cameraman	**4.54**	**0.10**	1.11	0.00	1.11	0.00
13	Cameraman	**4.31**	0.05	1.11	0.00	1.12	0.00
6	Airplane	**47.35**	0.97	41.56	0.92	22.97	0.36
7	Airplane	**37.51**	**0.85**	28.68	0.56	16.47	0.25
8	Airplane	**30.23**	**0.67**	14.18	0.23	10.30	0.14
9	Airplane	**22.78**	**0.47**	5.46	0.04	5.17	0.02
10	Airplane	**15.90**	**0.31**	0.95	0.00	1.60	0.00
11	Airplane	**11.24**	**0.23**	0.95	0.00	0.93	0.00
12	Airplane	**9.43**	**0.12**	0.95	0.00	0.93	0.00
13	Airplane	3.15	0.02	0.95	0.00	0.93	0.00

TABLE VII. QUALITY RESULTS FOR VGG11x11 KERNEL (F=128, L=64)

n_t	Image	Proposed		No Compensation		Uniform	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
6	Lena	56.33	0.96	45.85	0.75	55.48	0.96
7	Lena	50.01	0.90	39.23	0.59	48.72	0.89
8	Lena	44.19	0.80	32.45	0.41	42.83	0.80
9	Lena	38.21	0.66	28.07	0.31	36.88	0.66
10	Lena	32.08	0.50	25.96	0.27	30.93	0.51
11	Lena	26.73	0.36	25.71	0.27	25.47	0.39
12	Lena	22.57	0.26	25.71	0.27	21.68	0.27
6	Cameraman	56.49	0.95	47.96	0.82	55.02	0.96
7	Cameraman	49.87	0.89	41.39	0.74	47.74	0.88
8	Cameraman	43.21	0.78	33.69	0.60	41.09	0.81
9	Cameraman	37.32	0.61	28.44	0.48	34.99	0.64
10	Cameraman	30.61	0.48	25.00	0.43	28.70	0.50
11	Cameraman	22.84	0.37	23.40	0.41	21.44	**0.51**
12	Cameraman	18.00	0.31	23.39	0.41	16.79	0.37
6	Airplane	58.57	0.99	47.63	0.90	57.55	0.99
7	Airplane	52.29	0.97	40.45	0.80	50.83	0.98
8	Airplane	45.75	0.93	33.07	0.65	45.04	0.93
9	Airplane	39.38	0.82	28.43	0.57	39.62	0.85
10	Airplane	33.08	0.63	25.85	0.54	34.46	**0.74**
11	Airplane	27.91	0.50	24.54	0.53	29.66	**0.65**
12	Airplane	24.85	0.44	24.42	0.53	25.50	**0.57**

TABLE VIII. QUALITY RESULTS FOR VGG3x3 KERNEL (F=128, L=64)

n_t	Image	Proposed		No Compensation		Uniform	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
6	Lena	63.38	0.91	55.47	0.65	63.26	0.91
7	Lena	59.33	0.79	51.70	0.51	58.35	0.77
8	Lena	54.81	0.64	45.64	0.45	54.41	0.63
9	Lena	51.79	0.53	40.02	0.17	51.23	0.54
10	Lena	47.35	0.43	37.65	0.01	45.97	0.43
11	Lena	42.81	0.29	37.35	0.00	41.65	0.28
12	Lena	39.32	0.18	37.34	0.00	39.76	0.19
6	Cameraman	63.01	0.91	55.61	0.70	63.25	0.91
7	Cameraman	59.50	0.83	52.09	0.65	60.08	0.84
8	Cameraman	53.93	0.59	46.16	0.57	54.12	0.64
9	Cameraman	50.91	0.52	40.63	0.36	51.45	0.57
10	Cameraman	46.20	0.45	36.07	0.08	46.26	0.49
11	Cameraman	40.58	0.32	35.45	0.02	39.58	0.38
12	Cameraman	36.84	0.16	35.38	0.02	36.07	0.23
6	Airplane	63.50	0.90	55.84	0.59	63.72	0.90
7	Airplane	60.11	0.78	50.86	0.51	59.64	0.76
8	Airplane	55.21	0.57	44.66	0.47	53.27	0.52
9	Airplane	52.09	0.47	39.25	0.34	50.94	0.53
10	Airplane	**47.87**	0.36	33.48	0.04	43.21	0.37
11	Airplane	**43.27**	0.27	32.84	0.00	37.80	0.24
12	Airplane	**41.75**	0.20	32.82	0.00	35.95	0.17

3.3.1.3.2 VLSI Implementation results

In this paragraph the VLSI implementation results, in 40nm TSMC technology are presented. The proposed

precision-scalable MAC Unit of Fig. 13 has been described in Verilog HDL along with a precision scalable MAC Unit with no compensation and with a standard MAC Unit with fixed, full precision. The multipliers PPM has been described with the help of a Matlab script, implementing the TDM algorithm [51] starting from the characterization of the standard cell library (.LIB file). In order to perform a conservative comparison, the electrical performances are compared with the precision-scalable MAC unit with no compensation. Indeed, from the synthesis and simulations done, in terms of energy efficiency, the precision-scalable MAC unit implementing a compensation technique pay an overhead, with respect the precision-scalable MAC unit with no mechanism of compensation, due to the additional logic for embedding the compensation term into the carry-save tree. This overhead is the same for both the proposed compensation approach and the uniform one. Moreover, from a quality perspective, proposed approach exhibits significantly better or equally with respect uniform one. Therefore no improvement in the quality-energy tradeoff is expected when using uniform compensation. It is, instead, interesting investigating how the tradeoff is affected when moving from a no compensation approach to the proposed one. The circuits have been synthesized with Cadence RTL Compiler, with a clock constraint of 1.5 ns. The RTL Compiler synthesis directive *synthesize –to_mapped –effort high* has been employed. Moreover, a Physical Layout Estimation approach has been followed to estimate the wire parasitic. The Error Compensation block has been synthesized using HVT standard cells to optimize leakage increase, while all others circuits (including the Precision-Scalable MAC) have been synthesized at LVT. The power has been evaluated from a VCD back-annotated post-synthesis simulation. The simulation performs the real convolution operation on the Lena image and with some of the filters analyzed in the previous paragraph.

In Tab. IX are reported the area occupation and power leakage results. The area overhead for the MAC with no compensation is of 12% and this is due to the additional AND gates in the PPM for freezing the partial products. For the proposed approach, as expected, the area overhead is huge, being the circuit footprint more than doubled; this is essentially due to the Compensation Circuit (Fig. 11). Regarding leakage, thanks to the usage of HVT cells in the Error

Compensation block, the overhead does not follow the area one, being equal to 48% (if LVT cells are employed in the Error Compensation block the leakage increment becomes 148%). Note that the leakage increment can be an issue only when operating with deeply scaled voltage supply. If this is the case, it is worthwhile observing that the entire approach of introducing AND gates in the PPM to avoid the switching of the partial products can be inefficient, being targeted to decrease dynamic power at expense of the leakage one.

In Tab. X the dynamic power dissipation in full-precision mode is reported. Here *B* stands for Blur kernel, *E* for Emboss, *G* for Gaussian, *S V* for Sobel Vertical. Regarding power dissipation, in full precision mode, as expected, both the circuits increase the dissipated power, up to 9% for the no compensation circuit, and up to 24% for the proposed circuit. Note that the contribution to dynamic power due to the Error Compensation circuit accounts for less than 0.5% of the total dynamic dissipation, due to the high subsampling factor *F*.

TABLE IX. IMPLEMENTATION RESULTS FOR FULL-PRECISION MODE

Circuit	**Area** [μm^2]	**Power Leakage** [µW]
Standard	865	0.838
No Comp.	976 (+12%)	0.923 (+10%)
Proposed	2243 (+159%)	1.238 (+48%)

TABLE X. IMPLEMENTATION RESULTS FOR FULL-PRECISION MODE

Circuit	**Dynamic Power [µW/MHz]**					
	B	*E*	*G*	*S V*	*VGG 11x11*	*VGG 3x3*
Standard	1.447	1.221	1.577	1.367	1.670	1.918
No Comp	1.548 (+7%)	1.335 (+9%)	1.715 (+9%)	1.452 (+6%)	1.782 (+7%)	2.027 (+6%)
Proposed	1.798 (+24%)	1.548 (+27%)	1.828 (+16%)	1.682 (+23%)	1.910 (+14%)	2.226 (+16%)

In order to have a clear insight of the quality-power tradeoff, the energy reduction with respect standard MAC against the quality (PSNR and SSIM) is reported in the next graphs, for both the MAC unit with no compensation the one with proposed real-time data-aware approach.

The Fig. 19 reports the tradeoff curve for a Blur kernel. The x-axis represents the power saving with respect a standard MAC unit, while a the y-axis is reported the PSNR (Fig. 19 (a)) and the SSIM (Fig. 19 (b)). The blue line represents proposed approach, while the green one the no compensation one. Note that the higher is the curve the better is the performance, allowing the same power saving with higher quality. Note that for both the PSNR and SSIM cases, quality degrades gracefully for the proposed approach. For the no compensation one there is an abrupt degradation if n_t is increased. Note that for the PSNR metric, for power saving up to 5% the no compensation one is the best choice, assuring higher PSNR than proposed approach. When a more consistent power saving is desired, the quality in the no compensation cases degrades, while the proposed approach is the best choice. Similar considerations hold for the SSIM (Fig. 19 (b)). Note that, while for the full-precision mode the proposed MAC and the one with no compensation exhibits very different power dissipation (Tab. X), this difference tends to be negligible cutting more columns (as an example in Fig. 19, for $n_t \geq 11$ the power saving has the same value for both the circuits). The Fig. 20 shows the filtered Lena image, with both the proposed and no compensation MAC units, for the point with $n_t = 11$ (circled in red in the Fig. 19 (a)-(b)). At a price of significant degradation, around 22% of power can be saved. Note that for the case of Fig. 20 (b) (image resulting from no compensation) the image is almost totally compromised, imposing a bound to the power saving achievable.

The Fig. 21 shows the tradeoff curve for Sobel Vertical kernel. As previously observed the proposed approach degrades gracefully. In this case for both the PSNR (Fig. 21 (a)) and the SSIM (Fig. 21 (b)), in the region where a power saving is achieved (power saved < 0), the proposed MAC is always the best choice. Note that, with respect previous kernel (Fig. 19) the curves are left-shifted, therefore with the

same target quality a lesser power saving is achieved. This can be explained observing that kernels with integers number, quantized on 8-bits have a higher number of zeroed coefficients in the LSBs. This is the case of Sobel Vertical kernel. In this case, the zeroed LSBs of the coefficients reduce the switching in the least significant columns of the PPM of the standard MAC unit, therefore an advantage in terms of power using truncated multipliers can only be obtained with an aggressive precision scaling. Moreover, the sobel filter is an highpass filter therefore the error, which in the proposed case is almost at zero mean (3.33) (therefore having components at high frequency), is not filtered out, decreasing the quality performance. The Fig. 22 shows the filtered Lena images for $n_t = 10$, corresponding to the circled red points in Fig. 21. Note that the Fig. 22 (a) filtered with the proposed MAC exhibits some noisy pixels in the background but the edge are preserved, while for the case of no compensation Fig. 22 (b) in addition to a "complemented" intensity (pixels in the background from black are transformed in white) some edges details disappears. In correspondence of this points the power reduction is lesser than 5%.

The Fig. 23 reports the quality-power curve for the VGG 3x3 kernel. In this case the proposed approach always outperforms the no compensation one. In particular, the degradation appears to be really gracefully both in terms of PSNR and SSIM, decreasing almost linearly with the discarded columns. For power saving around 20% the difference in terms of quality is emphasized. In Fig. 24 are reported the Lena filtered image, for $n_t = 10$, corresponding to the red circles in Fig. 23. Note that in the case of no compensation (Fig. 24 (b)) the image filtering operation is completely compromised. For the image filtered with proposed MAC (Fig. 24 (a)) the degraded quality is traded for an about 27% reduction.

The filters investigated in this paragraph well reassume the three kernel typologies that can be found in practical application: (i) kernels like Blur and Sharp belong to the first typology for which the proposed approach make sense for aggressive precision-scaling; (ii) filters like Emboss, Sobel Vertical and Sobel Horizontal belong to the second typology, where, due to a significant number of zeroed LSBs, the power reduction is minimal; (iii) filters like Gaussian, Average, VGG 3x3 and VGG 11x11 belongs to the third category, in which the

proposed approach is effective in every point of the trade-off curve quality-power.

It is worthwhile observing that proposed approach allows increasing the precision-scaling range, enabling for an extra-gain in terms of power.

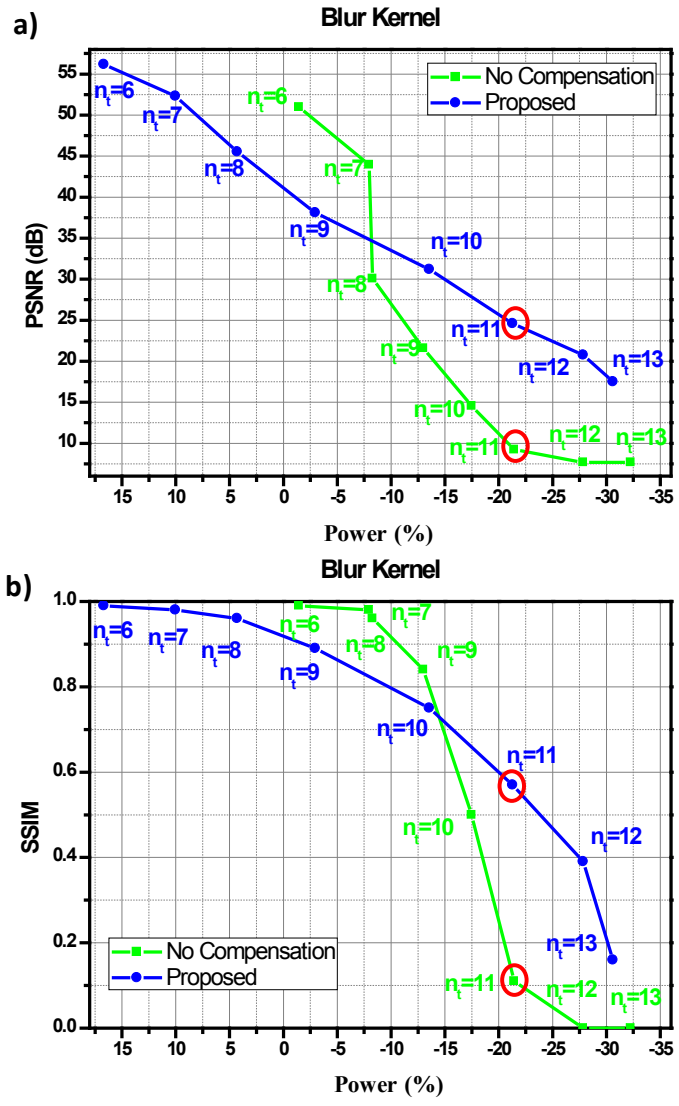

Fig. 19 Quality-Power tradeoff curves. (a) PSNR is employed as quality metric; (b) SSIM is employed as quality metric. Kernel: Blur.

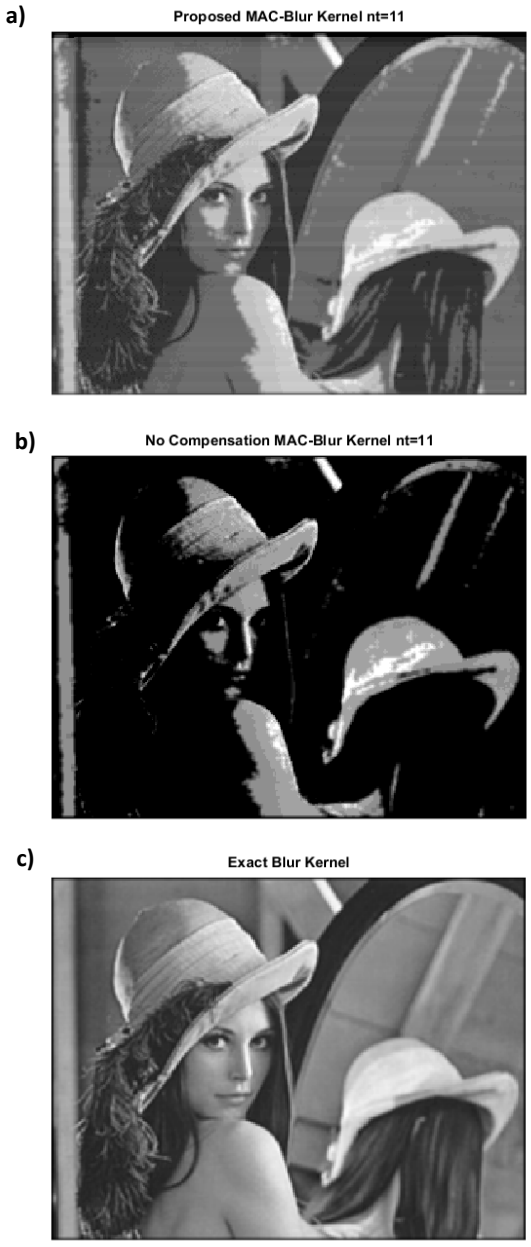

Fig. 20 (a) Filtered image with proposed MAC; (b) Filtered image with MAC with no compensation; (c) Exact image. Kernel: Blur.

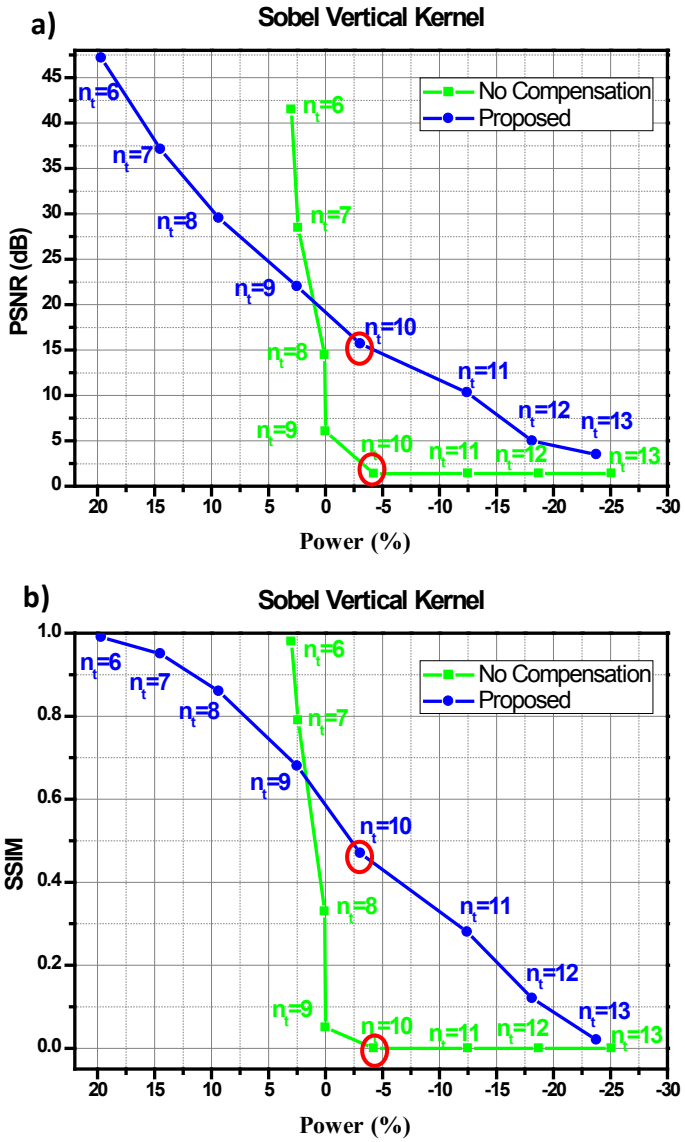

Fig. 21 Quality-Power tradeoff curves. (a) PSNR is employed as quality metric; (b) SSIM is employed as quality metric. Kernel: Sobel Vertical.

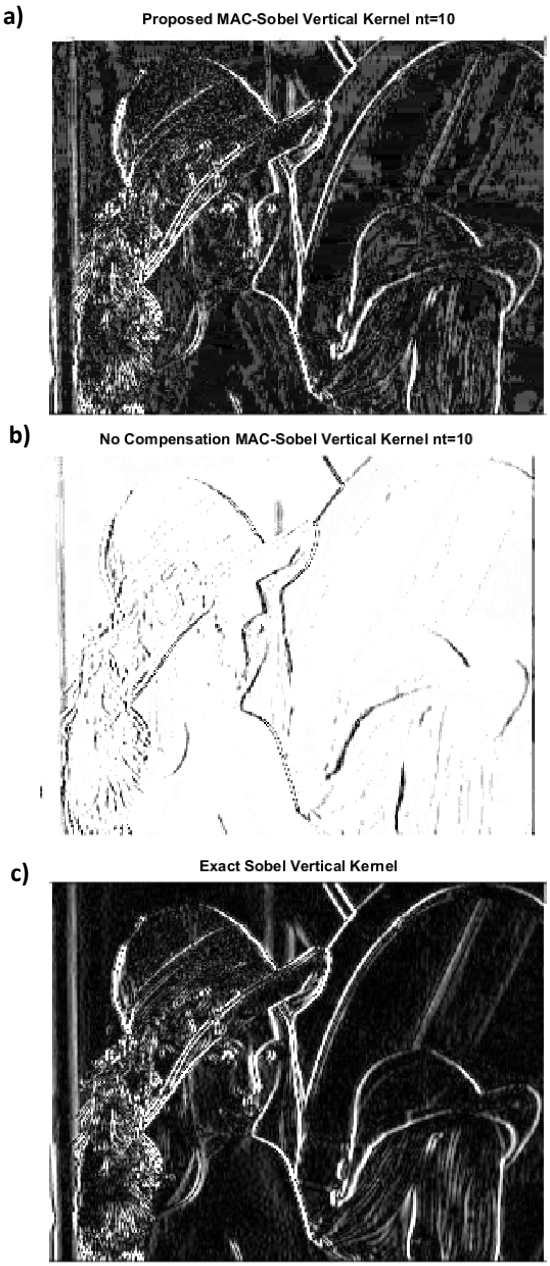

Fig. 22 (a) Filtered image with proposed MAC; (b) Filtered image with MAC with no compensation; (c) Exact image. Kernel: Sobel Vertical.

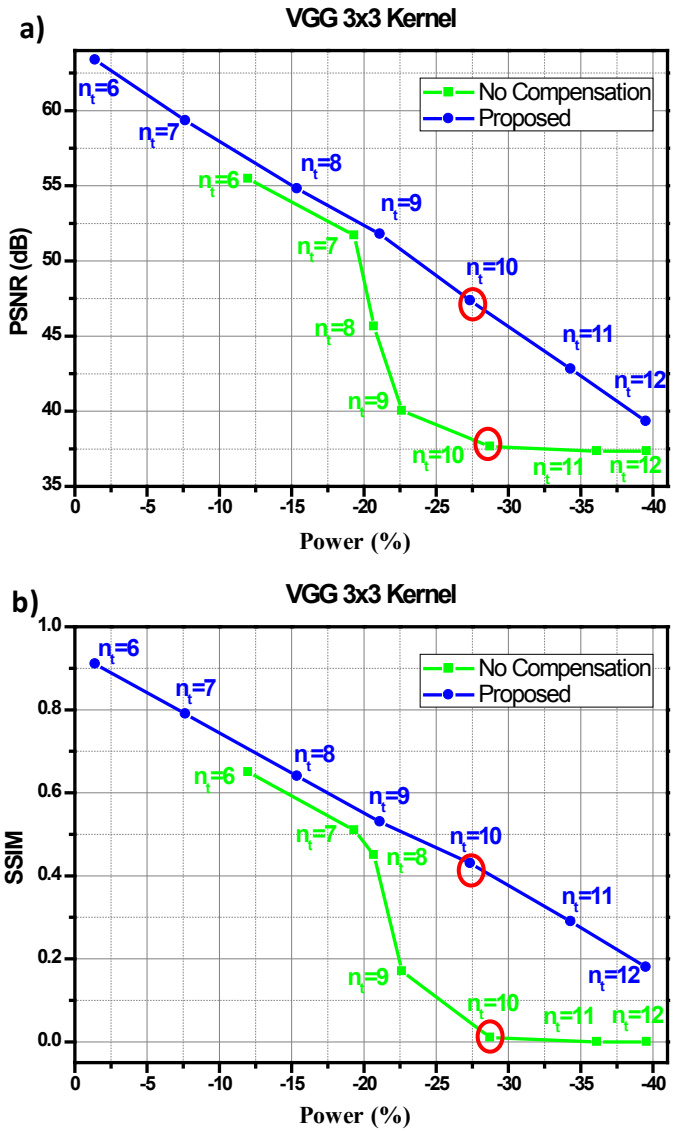

Fig. 23 Quality-Power tradeoff curves. (a) PSNR is employed as quality metric; (b) SSIM is employed as quality metric. Kernel: VGG 3x3.

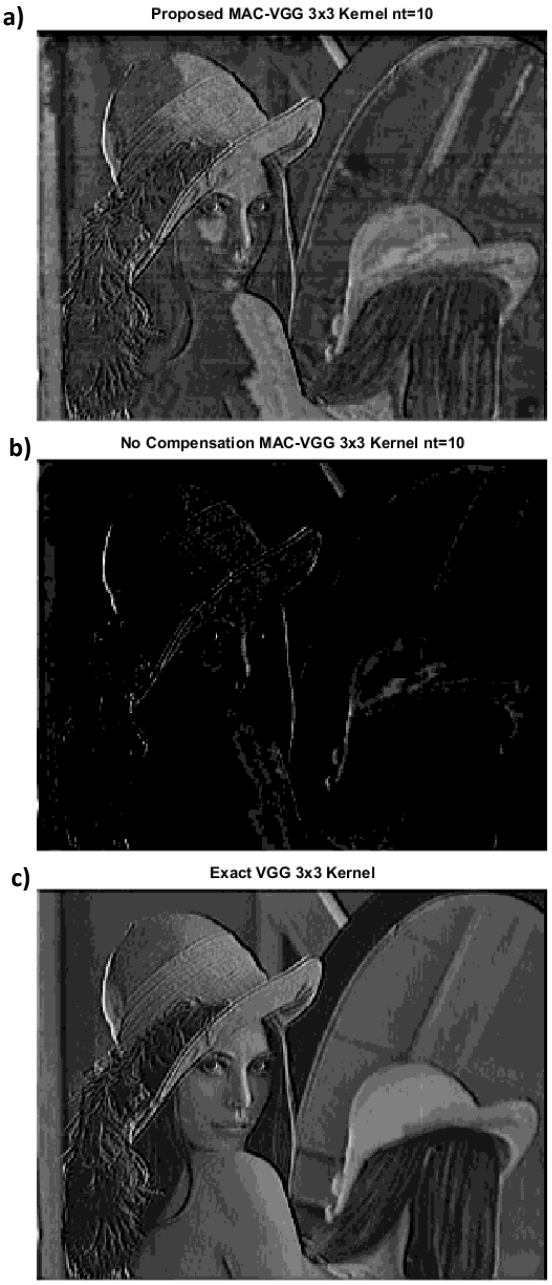

Fig. 24 (a) Filtered image with proposed MAC; (b) Filtered image with MAC with no compensation; (c) Exact image. Kernel: VGG 3x3.

3.4 Precision-scalable Approximate MAC Unit

In this paragraph a precision-scalable Approximate MAC Unit is discussed. Conversely to the MAC Unit discussed in the paragraph 3.3, here the partial product matrix is compressed in an approximate way, employing OR gates in place of half adders. This approximate MAC can be employed in systems area-and-power constrained. Indeed, with respect to the MAC unit shown in the paragraph 3.3, the Approximate MAC Unit shows significant power savings, due to the compression step and to the precision scalability, at a price of a reduced quality. Indeed, also when no column is discarded (full-precision modality), the proposed Approximate MAC unit provide to the system erroneous (approximated) results, resulting from the approximate partial product matrix (OR gates in place of half-adders), making it unsuitable in systems where the quality constraint significantly varies over the time. Note that the compression step, reducing the gates count, improves leakage and area performance with respect standard MAC unit. Therefore, in order to keep bounded the area occupation, a uniform compensation method is employed. It is worthwhile observing that the proposed Real Time Data Aware Compensation Technique (paragraph 3.3.1) can also be employed, in systems where the area and leakage overhead can be tolerated.

Recently, researches focused on energy-efficient multipliers implementation [45], [57], [65], [83]. In [65] a truncated multiplier with variable compensation method is proposed. The least significant columns of the multiplier partial products matrix are discarded and a compensation function to minimize the resulting mean square error is employed. In [57] a programmable truncated multiplier is proposed. The partial product columns can be dynamically freezed, with a fine grain approach, in order to save energy when the quality constraint can be relaxed. The resulting error is compensated in software. Authors in [83] propose an approximate multiplier obtained reducing the maximum height of the partial product matrix by means of compression of partial product terms. Three different designs are presented, in which two, three and four partial products are reduced into one term using OR gates. In this paper the precision is fixed at design time, therefore the energy quality tradeoff cannot be

dynamically tuned as function of the input data and application. Moreover, the error resulting from the compression is not compensated, limiting the quality performance especially when three or more terms are compressed into one.

In this paragraph a precision scalable approximate Multiply and Accumulate (MAC) unit is proposed for computer vision applications. In the proposed MAC unit, leveraging the observations made in [45], the height of the partial product matrix is preliminarily compressed (at design time). The resulting compressed matrix is then implemented as precision scalable carry save tree, similarly to [57]. In order to reduce the error deriving from (i) compression and (ii) dynamic precision scalability, a compensation term is inserted in the accumulation loop.

3.4.1 Partial product recoding and compression

In the following a signed for unsigned multiplier will be considered. The Fig. 25 (a) shows the Partial Product Matrix (PPM) of a signed for unsigned N=8 bits Baugh Wooley multiplier, where X is the signed operand and Y the unsigned one. The operands are considered fractional, with MSB of magnitude 2^{-1}.

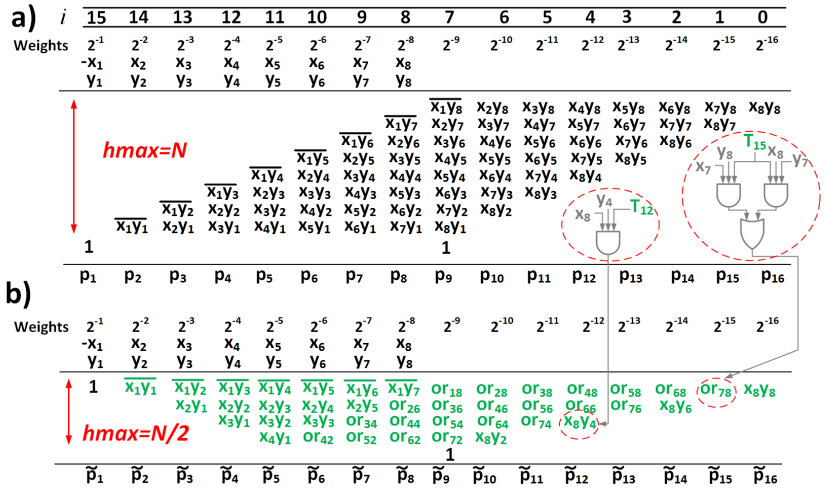

Fig. 25(a) Partial Product Matrix (PPM) of a 8-bits signed for unsigned multiplier. (b) Resulting PPM after recoding, compression and precision-scalability.

The multiplier critical path is related to the maximum height of the PPM, being equal to N. Note that the sign extension prevention constant (whose value is for a signed for unsigned multiplier), being a constant term, generally does not impact on the multiplier delay, implemented as carry save tree. In order to compress the PPM (i.e. decrease the maximum height of the matrix), a recoding of the partial product terms can be employed. As observed in [45], two generic partial product $x_i y_j$ and $x_{i-k} y_{j+k}$ belonging to the *i+j-th* column can be recoded as follows:

$$A_{i,j} = x_i y_j \; AND \; x_{i-k} y_{j+k} \qquad (3.45)$$

$$O_{i,j} = x_i y_j \; OR \; x_{i-k} y_{j+k} \qquad (3.46)$$

being their sum $A_{i,j} + O_{i,j} = x_i y_j + x_{i-k} y_{j+k}$. Under the hypothesis that the input bits xi and yj are uniformly and independently distributed, the probability to be high is $(1/4)^2 = 0.0625$ for $A_{i,j}$ and $7/16$ for $O_{i,j}$ [45]. The recoding, while constituting an overhead due to the AND (3.45), OR (3.46) additional gates, allows us to neglect the low-probability terms (3.45). Note that the direct elimination of a partial product $x_i y_j$, avoiding the recoding, involves the elimination of high probability terms, being $1/4 = 0.25$ the probability of a partial product to be high. Neglecting the $A_{i,j}$ terms, the approximate sum can be expressed as:

$$x_i y_j + x_{i-k} y_{j+k} \simeq O_{i,j} \quad (3.47)$$

In this way, we are approximating an half adder with an OR gate, with consequent area, delay and power improvement, at a price of an error on both the carry out and the sum, when both the partial products are high. Note that also in [83], an half adder is approximated by means of an OR gate. Authors in[83], however, propose to approximate a full adder with a 3 inputs OR gate (acting as 3:1 compressor) and to use a 4 input OR gate as 4:1 compressor. The error deriving from these choices, in our opinion, can be significant, affecting the overall quality, therefore only compression using the (3.47) is employed in our multiplier.

Employing the recoding and compressing as in (3.47) allows halving the maximum height of the PPM. While, for the terms in the least significant columns (the last N columns), the compression of the maximum possible terms using (3.47) is beneficial in terms of energy quality tradeoff (contributing minimally to the output due to their weight), in the most significant columns the best choice is to compress selectively only the minimum terms of each column in order to keep the maximum height of the compressed matrix equal to N/2. If we indicate as $i=0$ the least significant PPM column (Fig. 25) , the compression can be extended up to the i^**-th* column, given by:

$$i^* = \frac{3N-4}{2} \quad (3.48)$$

The number of OR gates needed in each column is given by the following expression:

$$Nor(i) = \begin{cases} \left\lceil \frac{i}{2} \right\rceil & for\ \ 0 \le i \le N-1 \\ \frac{3N-2(i+1)}{2} & for\ N \le i \le i^* \end{cases} \quad (3.49)$$

The (3.49) suggests to compress as much as possible in the least significant columns ($0 \le i \le N-1$) and as less as possible to keep the height of matrix equal to N/2, in the most significant ones ($N \le i \le i^*$). For the PPM of Fig. 25 (a), $i^* = 10$ and $Nor(10) = 1$ therefore only one OR gate is needed in the $i^* = 10$ column to keep the max height of the compressed PPM bounded to *N/2*.

3.4.2 Precision-scalability

In order to meet the different quality constraints (i.e. different error resiliency) during the elaboration of a given application, the compressed PPM must be able to scale its precision at run time.
In this paper a fine-grain approach is employed, as proposed in [57]. In particular, each column *i-th* of the PPM can be selectively discarded, by means of an additional control signal T_i. When $T_i = 1$ the column is accounted in the multiplication, while, when $T_i = 0$ all the

partial products belonging to the *i-th* column are freezed to zero, in order to save dynamic power. An external signal *nt*, representing the number of columns (starting from the least significant one) to be discarded, codifies the signals T_i: as an example, if $nt = 0$ the $2N-1$ T_i signals will be all ones (no column is discarded), while if $nt = 8$ the signals $T_i = 0$ *for* $0 \le i \le nt-1$. Employing the precision-scalability requires to modify the (3.47) as follows:

$$x_i y_j t_{i+j} + x_{i-k} y_{j+k} t_{i+j} \simeq O_{i,j} \qquad (3.50)$$

Note that the (3.50) can be easily mapped with an AO33 standard cell. The remaining non-compressed partial product terms are AND-ed with the control signals T_i, requiring a 3-input AND in place of a standard 2-input AND for the partial products computation. The resulting precision scalable PPM, after the recoding and compression steps, is shown in Fig. 25 (b).

3.4.3 Error Analysis and Compensation

In this section an error analysis is presented in order to determine an error compensation expression. The errors result from (i) compression (ii) precision scalability. Therefore an expression of the compensation terms, in function of the number of discarded columns nt will be provided.

Let us assume that no column is actually discarded ($n_t = 0$). In this condition the errors result from the compression step only. From the (3.45)-(3.47) we know that the mean error committed when an half adder is substituted by an OR gate is equal to the probability of $A_{i,j}$ to be high, which equals to $(1/4)^2$. Therefore the mean error committed due to the compression of the *i-th* column with $0 \le i \le N-2$, can be expressed, using the (3.49), as:

$$E_{C1}(i) = \left\lceil \frac{i}{2} \right\rceil \cdot \left(\frac{1}{4}\right)^2 \cdot 2^{-2N+i} \quad for \quad 0 \le i \le N-2 \qquad (3.51)$$

In the column $i = N-1$, having the maximum height, all the partial products must be compressed (compare ((3.49))), therefore also the NAND partial product (deriving from Baugh Wooley multiplier) is compressed. It can be easily shown that, when the compression

involves this partial product, the deriving error is higher, being equal to $3(1/4)^2$. This observation suggests us that, when (3.49) is used to determine the number of employed OR gates, the compression of the NAND partial products should be avoided whenever possible. The mean error associated to the compression of the $i = N-1$ column is given by:

$$E_{C2} = \left[\left\lfloor \frac{N-1}{2} \right\rfloor \cdot \left(\frac{1}{4}\right)^2 + 3\left(\frac{1}{4}\right)^2 \right] \cdot 2^{-N-1} \tag{3.52}$$

The mean error deriving from the compression of the most significant columns (up to i^* (3.48)) is expressed, using (3.49), by:

$$E_{C3}(i) = \frac{3N - 2(i+1)}{2} \cdot \left(\frac{1}{4}\right)^2 \cdot 2^{-2N+i} \ \ for\ N \le i \le i^* \tag{3.53}$$

Let us assume, now, that no compression is performed, therefore the error is only caused by precision scalability. Following an analysis similar to [57], the mean error resulting from the precision scalability can be expressed as:

$$E_P(nt) = \begin{cases} for\ nt \le N-1: \\ \sum_{m=1}^{nt} m \cdot 2^{m-3-2N} \\ for\ N \le nt \le 2N-1: \\ E_P(N-1) + \sum_{m=N}^{nt} (2+2N-m) \cdot 2^{m-3-2N} \end{cases} \tag{3.54}$$

Combining (3.51)-(3.54), the resulting mean error, when the matrix is compressed and *nt* columns are discarded, is obtained as:

$$E = \begin{cases} E_P(nt) + \sum_{i=nt}^{N-2} E_{C1}(i) + \sum_{i=N}^{i^*} E_{C3}(i) + E_{C2} \ \ for\ nt \le N-1 \\ E_P(nt) + \sum_{i=nt}^{i^*} E_{C3}(i) \ \ for\ N \le nt \le 2N-1 \end{cases} \tag{3.55}$$

A first attempt to compensate the error resulting from above approximations is achieved by compensating the mean error [65]; this

allows increasing quality metric such as mean-square-error (MSE) and PSNR. Therefore the following compensation term will be employed:

$$K(nt) = E \tag{3.56}$$

3.4.4 Approximate MAC Unit architecture

The discussed precision-scalable approximate PPM is embedded in a MAC unit. The resulting circuit is shown in Fig. 26. The precision-scalable PPM is implemented as a TDM carry-save tree and provides as output the two signals *s1* and *s2* in carry-save format. These signals are then accumulated and the resulting output Z_t is obtained truncating the $N+g$ least significant bits of the accumulator output *ACC*, where *g* is the number of guard-bit implemented to avoid the overflow in the accumulation loop. Note that the *nt* signal, drives both the precision-control signals *T* and the LUTs. The LUTs are used to store the compensation terms (3.56). Note that, in order to avoid an additional row in the multi-operand accumulation adder (implemented, in turn,. as carry-save tree followed by a final vector merging adder) and therefore an energy overhead due to the compensation, the compensation term is initialized at the start of each convolutional kernel. This requires to store in the LUTs the *K(nt)* values, multiplied for the number of multiplications contained in a convolutional kernel (e.g. 9 multiplications in a 3x3 kernel and 25 multiplications in a 5x5 kernel). A simple control circuit, starting from the *Kernel_size* signal (indicating the size of the kernel), initializes the accumulation register to *K(nt)* through the *rst_acc* signal, at each new kernel. Note that the LUTs minimally affect the dynamic power consumption, since *nt* is supposed to vary with a frequency $f_{nt} \ll f_{clk}$.

3.4.5 VLSI Implementation results

The proposed circuit has been described in Verilog HDL and synthesized in TSMC 40nm technology using Cadence RTL Complier. For comparison purpose the three versions of the approximate multipliers proposed in [83] have been implemented and embedded in a MAC unit.

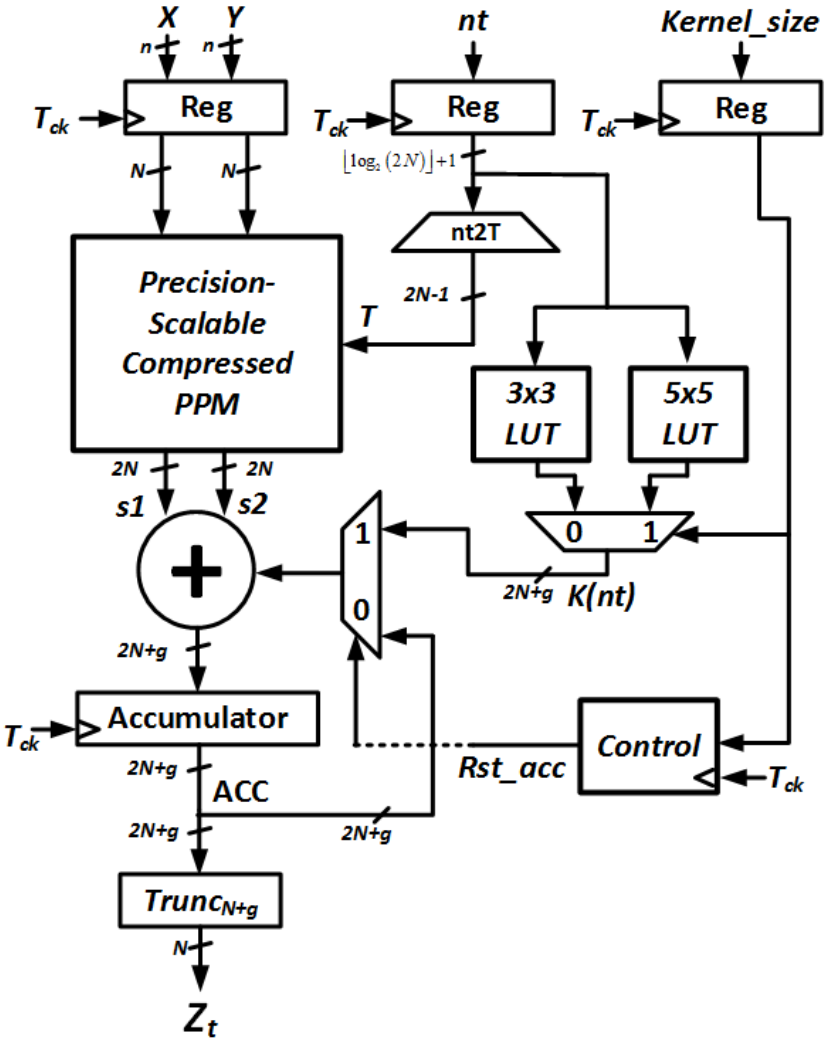

Fig. 26 Proposed precision scalable approximate MAC architecture. The LUTs store the compensation terms *K(nt)*.

In the following we will denote respectively as "*DATE 17 L=2*", "*DATE 17 L=3*", "*DATE 17 L=4*", the version with the PPM compressed using 2 inputs, 3 inputs and 4 inputs OR gates [83]. A classic full precision MAC unit has also been implemented as reference to evaluate the power improvement using the approximate MAC units.

TABLE XI. VLSI IMPLEMENTATION RESULTS-@VDD=1.1V

Circuit	**F [GHz]**	**Area [μm^2]**	**$P_{LEAKAGE}$ [μW]**
Classic MAC	1.11	1867	1.686
DATE17-L=2	1.11	1111(-41%)	1.041 (-38%)
DATE17-L=3	1.11	903(-52%)	0.802(-52%)
DATE17-L=4	1.11	880(-53%)	0.756 (-55%)
This paper	1.11	1741(-7%)	1.404(-17%)

The Table XI shows the area and leakage performance of the implemented MAC units. The period constraint has been imposed at 0.9 ns, which is the minimum period at which the Classic MAC unit meets the timing constraint with zero setup slack (the approximate MACs are faster), while power dissipation has been evaluated from VCD and SDF-based post-synthesis simulations.

While the circuits proposed in [83] exhibit substantial area and leakage improvements, up to 52% and 55% respectively, the proposed MAC shows moderate improvements, enhancing the area of 7% and the leakage of 17%. Compared with *DATE17-L=2* circuit, which adopts a similar compression step, the reduced area and leakage savings are due to the LUTs overhead, needed to implement the compensation mechanism.

The Fig. 27 shows the power-quality tradeoff when the "cameraman" image is filtered with a 3x3 gaussian kernel. In the graph of Fig. 3 the x-axis reports the power saved with respect the Classic MAC, while the y-axis reports the PSNR. The blue line reports the power at different quality levels for the proposed MAC, obtained by varying *nt*. In this graph, the higher is the curve, the better is the trade-off, achieving the same power saving at higher quality. When little columns are neglected, the PSNR is about 2dB higher (point a in Fig. 27) than *DATE17-L=2* circuit, at the same or better power saving, thanks to the compensation (3.56). When the precision in scaled down the proposed circuit outperforms the counterparts achieving a power reduction of 53% with tolerable quality degradation (point b).

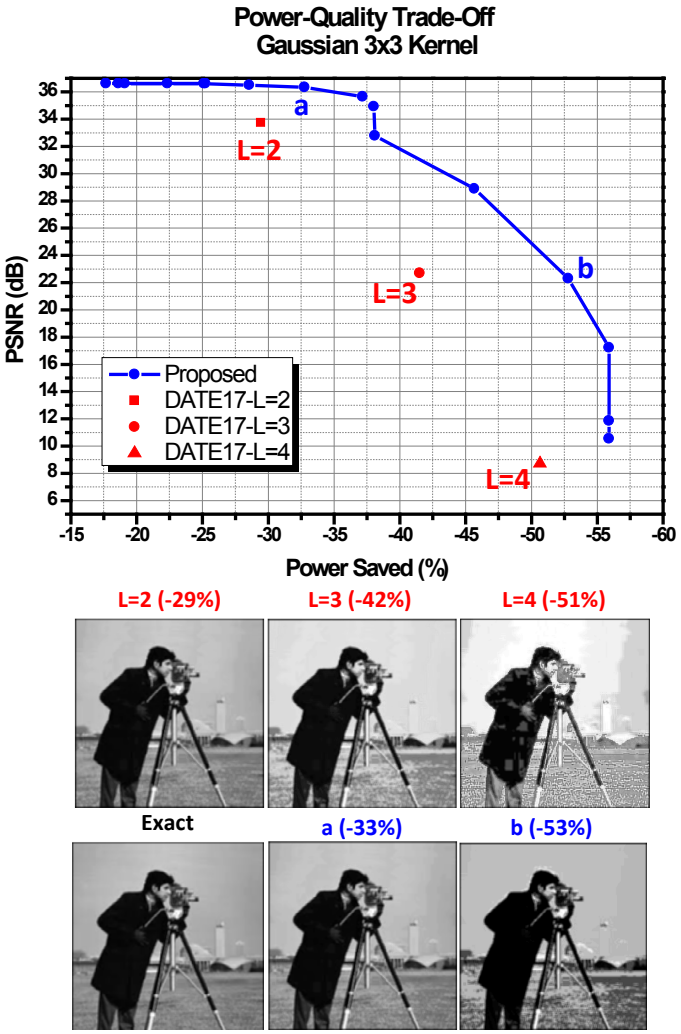

Fig. 27 Quality-Power trade-off for a gaussian 3x3 kernel. Due to precision scalability and error compensation the proposed MAC outperforms the fixed-precision circuits proposed in [83].

3.5 Precision-scalable Latch Memory

In this paragraph a precision-scalable Latch Memory is discussed. The proposed precision-scalable latch memory can be part of a precision-scalable system. In these kind of systems, the data can be incorrect, due to scaled precision, therefore the memories in which

these data are stored can be made precision-scalable to further increase approximations efficiency.

Standard Cell Memories (SCMs) have been introduced for the first time in [84] and they represent an interesting alternative to SRAM Macrocells (MMs) to implement embedded memories. The storage functionality is assured by a matrix of flip-flops or latches. SCMs can be described using HDL languages and easily synthesized, being composed by standard cells. This gives high flexibility, since the memory features (number of ports, number of words, number of bit per words) can be easily decided at design time, according to the specific system needs, without the limitations imposed by the usage of a memory generator, in terms of words and word lengths [85]. Moreover, SCMs can be placed using standard CAD tools and therefore merged with logic blocks, improving data locality with consequent reduction of wiring and parasitics (i.e. improved energy efficiency and timing).

In [84] authors report that SCMs offer area and energy reduction, with respect MMs, for storage capacity up to 1kbit, while, for bigger storage capacity, SCMs become bigger than MMs, but still exhibit a better energy efficiency. In [86] SCMs are proposed as an alternative to full-custom sub-Vt SRAM Macrocells for systems operating in deeply scaled voltage supply. An extensive analysis, targeted for ultra-low voltage applications and corroborated by ASIC measurements, is reported in [85], showing that up to 4-6 kbit SCMs exhibit better than sub-Vt SRAMs. A controlled placement design methodology, as a part of the standard digital design flow, is proposed in [87], optimizing placement density and power dissipation, due to the reduced wire length.

3.5.1 Latch Memory architecture

The SCM storage element can be a D flip-flop or a D-latch. In the following, we will refer to latch-based SCMs, since, compared to flip-flop based SCMs, they are more area efficient and the timing can be improved exploiting time borrowing [87]. We will examine dual-port memories, with a word access scheme. Fig. 28 shows the standard latch-memory architecture, as proposed in [84]. SCM are composed by three main blocks: (i) write logic, (ii) read logic, (iii)

storage matrix. In the following, we briefly discuss about write and read logic implementations.

3.5.1.1 Write logic

As shown in Fig. 28, the storage matrix is composed by *R* rows and *C* columns. *C* latches belonging to the same row constitute a word; at each word an address is assigned. Let us suppose that new data must be written at the first row of the storage matrix reported in the trivial example of Fig. 28 (here R=C=2). The corresponding writing address will be *waddr=0* (assuming to associate address 0 to the first word, address 1 to the second one and address *R*-1 to the last one). In the hypothesis that the write enable is high (*we=1*) the new data *wdin* is sampled on the rising edge of the gated clock *clk_din* by the input flip-flops (note that when *wen* is zero these flip-flops are gated). The captured data must be written in the corresponding row, identified by *waddr*. The selection of the appropriate word occurs as follows. The write address decoder (*WAD*) produces a one-hot output (note that if *wen*=0 the WAD switching is inhibited due to AND gate), constituted by the *R strobe* signals of Fig. 28 (in our example *waddr=0* implies *strobe1=1*, *strobe2=0*). The asserted strobe signal, in turn, activates one of the R enable signals (*en* in Fig. 28), with the help of a clock gating integrated cell (CGIC, constituted by a transparent-low latch and an AND gate, Fig. 25). The new data is then latched in the memory. With respect to the topology proposed in [84] the input flip-flops are introduced to improve the setup timing constraint on the *wdin* data (this constraint must be met by the external circuit feeding *wdin*) [87]. It is worthwhile observing that the input flip-flops are rising edge triggered and the storage latch are high level transparent. Therefore, the clock to output delay of the input registers (which can be significant due to the high fan-out) occupies part of the latch transparent window. This tight setup constraint is nevertheless mitigated by the time borrowing capacity of the latch. This design choice relaxes the setup constraint on the CGIC that drive *en* signals (this path is generally the critical one, due to the WAD). Note that other write approaches have been previously introduced in [84], involving the usage of tristate buffers, but this solution has not been considered, being less attractive in terms of area and power.

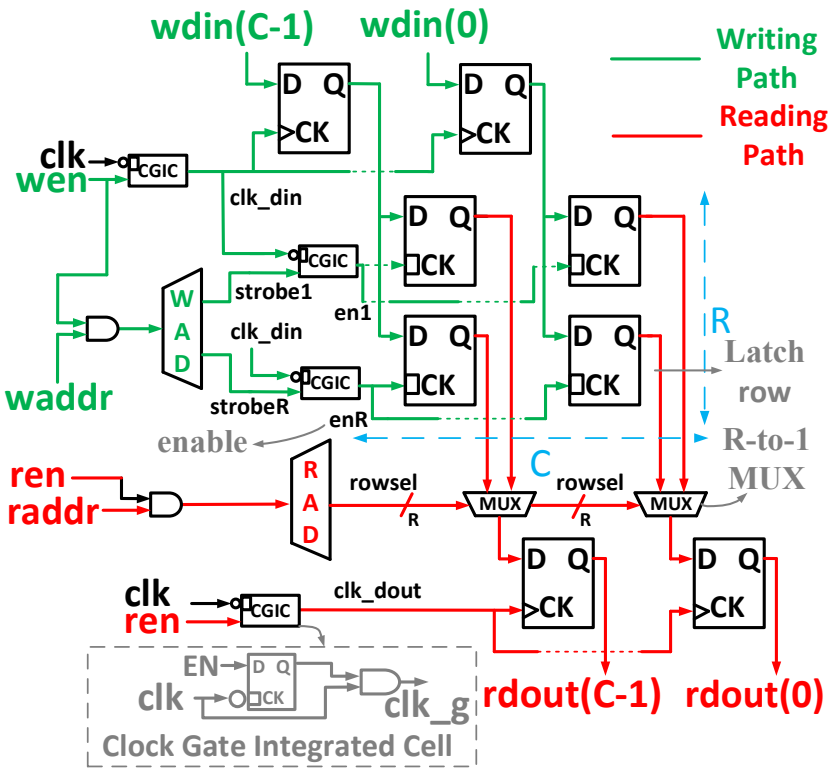

Fig. 28 Standard Cell Memory [71]. The storage matrix is composed by RxC latches (R=C=2 in this figure). One out R latches rows is written activating the corresponding enable signal. The reading involve an Rto1 multiplexer whose selection signal is produced the read address decoder (RAD).

3.5.1.2 Read logic

With reference to Fig. 28, the read operation involves a read address decoder (RAD), which has the same behavior of the WAD, producing one-hot output bus, indicated as *rowsel* in Fig. 28, corresponding to the word to be read. As discussed in [84], the read multiplexer is energy efficient if its selection signal is a one-hot bus: in this case the mux is easily implemented with a first level of AND gates, performing the logic AND between each *rowsel* selection input and the corresponding data bit. The outputs of this AND plane are fed into an OR gates tree. Note that, as done for the WAD, the RAD is gated with an AND gate, filtering read address *(raddr)* variations

when the read operation is disabled (*ren=0*), this saves the RAD logic to useless switching. The mux output is then sampled by a flip-flops stage, in order to provide a synchronized output to the successive logic stages.

3.5.2 Precision-scalable architecture

In this paragraph, the proposed precision-scalable latch-memory architecture is discussed. The fundamental idea is to group the *C* latches belonging to every row in one or more precision-scalable groups (PSG), which can be activated or deactivated as function of the desired quality or of the input data statistic [52]-[54]. The number and the size (number of latches in a given group) of the PSG can be decided at design time, with minimal engineering effort, depending on the level of precision scalability of the system in which the memory are embedded, on the word size and, of course, on the application. The proposed architecture requires modifications of write logic, storage matrix and read logic, as detailed in the following.

3.5.2.1 Write logic and storage matrix modification

Fig. 29 reports the storage matrix and the input registers write logic of the proposed architecture. In this example C=3, while R=2. Note that an additional input, indicated as *g1,* is shown in Fig. 29.

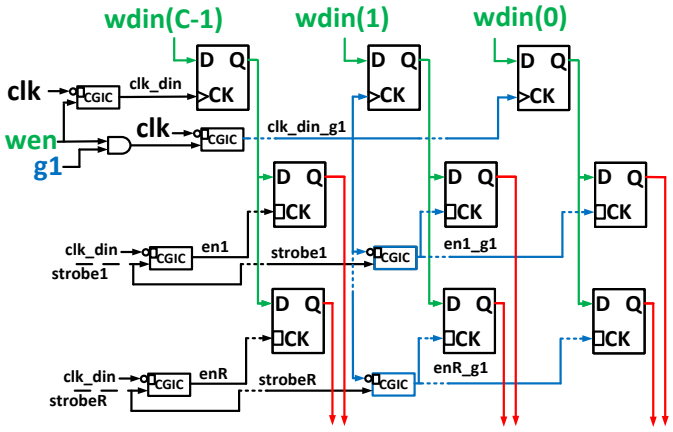

Fig. 29 Storage matrix and register logic of the proposed precision-scalable architecture. Additional CGIC cells are needed.

This signal allows to scale down precision (*g1=0*). In this example, we assume to have a single PSG, with a group size (GS) of 2 (we are grouping and conditionally disabling 2 LSBs of the memory). Focusing on the input register logic, we observe that a CGIC cell, whose enable is the AND between *wen* and *g1*, allows to gate the clock of the registers *wdin(1), wdin(0)* and of all latches belonging to the PSG (blue wires in Fig. 29). In this way, when *g1=0,* the input registers are not updated (the clock *clk_din_g1* will be always low), avoiding to write in the latches of the PSG. This saves the dynamic power associated to: (i) the updating of the input registers, (ii) the updating of the latches, (iii) the switching of the input clock capacitance of flip-flops and latch, (iiii) the switching of the clock buffers belonging to the gated clock in the PGS (blue wires in Fig. 29). The drawback of this approach is given by the need of additional CGIC cells in the storage matrix. These additional CGIC cells are reported in blue in Fig. 29. They take as enable signal the corresponding *strobe* signal of the row (coming from the WAD, as in the architecture previously discussed) and the gated clock *clk_din_g1*. In this way, when *g1=1* (full precision modality), *clk_din_g1* is enabled and the new data is written only in the latches belonging to the desired row, thanks to the *strobe* signals. The number of additional CGIC cells is a function of the Number of Precision Scalable Groups (NPGS) and the number of rows *R*:

$$N_{CGIC1} = NPGS \cdot (1 + R) \tag{3.57}$$

The increased number of CGIC cells represents an overhead in terms of area, leakage and dynamic power when operating in full precision mode. It is worthwhile observing that, thanks to the dual stage clock gating, the additional CGIC in the storage matrix contribute to dynamic power only when the signal *clk_din_g1* is enabled (full precision mode and enabled writing).

3.5.2.2 Read logic modifications

Fig. 30 shows the proposed read logic modifications, relative to the example of Fig. 28. As previously discussed, when *g1=0* all the latches in the PSG are clock (enable) gated, therefore their output, which constitute the input of the read logic mux is freezed. However

this does not prevent the mux to switch, since, when *ren=1*, the read address changes, causing commutations of the *rowsignal* and useless switching of the mux belonging to the precision-scaled group. Therefore, the row signal is gated with the precision control input *g1*, before to act as selection signal for the mux of the PSG; this involves *R* additional AND gates per group. Again, these additional AND gates, while saving dynamic power when operating in scaled precision mode, represent an overhead in terms of area and leakage; moreover they contribute to dynamic power when operating in full precision mode. As done for the input registers, the output registers are also gated using an additional CGIC cell per group, whose enable is the AND of the *ren* and the *g1* signal. This also allows avoiding switching activities in the successive logic stages. The total number of additional CGIC cells, accounting also for these last ones, is given by:

$$N_{CGIC} = NPSG \cdot (2 + \mathrm{R}) \simeq NPSG \cdot R \tag{3.58}$$

Therefore, in practical applications, the additional number of CGIC cells grows linearly with the number of precision scalable groups.

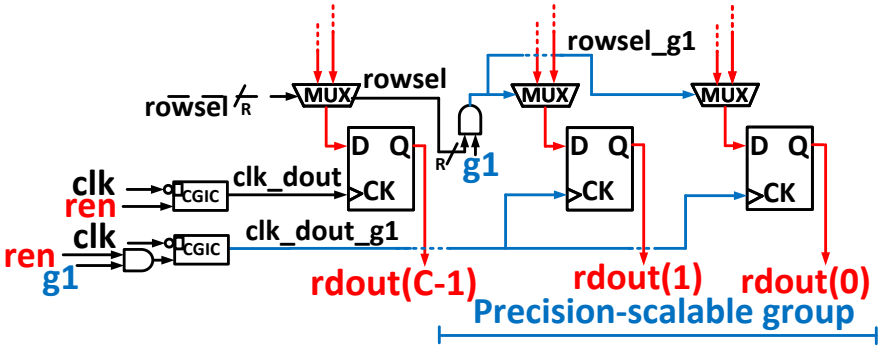

Fig. 30 Proposed precision-scalable read logic. The mux belonging to PSGs are operand-isolated when the corresponding PSG is disabled.

As a final note, some applications may require the *rdout* bits belonging to PSGs to be zero when operating in scaled precision mode. This can be obtained by a simple modification of the read logic of Fig. 30 postponing the gating of the output registers of one clock cycle (in this way the zero value produced by the mux can be sampled by the registers). This requires only an additional OR and two flip-flops per PSG, with a negligible overhead.

3.5.3 VLSI implementation results

In order to investigate the performance of the proposed architecture and the power-quality trade-off, the Precision-Scalable Latch Memory has been implemented in 40nm TSMC technology and simulated assuming to be embedded in a computer vision system, as frame buffer. We have implemented the proposed circuit with word size C equal to 8 and 16 and with R spanning from 32 to 128. Higher storage capabilities can be obtained by arranging more banks in parallel and adopting a 2-dimensional addressing scheme. For the memories with C=8 a single PSG ("PSG1") has been employed, grouping four LSBs. For C=16 two PSGs have been implemented: PSG1 going from bit 0 (LSB) to 6 and PSG2 going from bit 7 to 11.

Proposed and Classical SCMs (Fig. 28) have been described in Verilog HDL with the aid of Matlab scripts. The CGIC cells found in the standard cell library have been directly instantiated into the Verilog netlist. The circuits have been synthesized using Cadence RTL Compiler, imposing a clock period constraint of 1 ns. To this regard, the RTL Compiler synthesis directive *synthesize –to_mapped –effort high* has been employed; moreover the Physical Layout Estimation flow has been followed in order to estimate wire parasitics. Please note that for circuits using clock gating, the timing of the enable signals of the CGIC cells is not accurately accounted during synthesis phase, because of the lack of the clock tree. Therefore, in order to accurately the timing, the circuits have been automatically placed and routed using Cadence Encounter. Power dissipation has been evaluated from VCD and SDF-based post-layout simulations, in which the memory control signals (*ren*, *wen, raddr, waddr*) have been generated by a FSM, implementing the control for a convolution operation (image filtering). The memory has been periodically written with pixels belonging to real images encoded on 8-bits (for the memories with C=8) and 16-bits (for the memories with C=16).

The Tab. XII shows the post-layout performance of classic and proposed latch memories. Area and leakage overhead is about 16% and is due, essentially, to the addition of CGIC cells as shown by equation (3.58).

The Tab. XIII shows power performance. In this table P_{RW} and P_R refer to dynamic power evaluated by assuming a periodic writing

and no writing operation, respectively. In full-precision mode, the dynamic power overhead due to the additional CGIC cells is significant during read-write operations (in the range17%-27%) while it is much lower during read-only operation. The dynamic power is strongly reduced in the proposed memory, when the precision is scaled down. During read-write operations the improvement in power compared to the standard topology of Fig. 28 reaches 29% in the case *C*=8 and 56% for *C*=16. When only read operations are done, power saving increases up to 66%. The image quality, quantified with the Structural Similarity Index, SSIM [50], is also reported in Tab. XIII. A value of SSIM=1 means perfect similarity. For C=16 and a single PSG shut down, SSIM=0.99 and hence image quality is practically unaffected by precision scaling. In the other cases, SSIM is about 0.64 showing a certain amount of image degradation.

TABLE XII. POST-LAYOUT IMPLEMENTATION RESULTS

Topology (R,C)	**F [GHz]**	**Area [μm^2]**	**$P_{LEAKAGE}$ [μW]**
Classic 64,8	1.0	3622	1.875
Proposed 64,8	1.0	4066 (+12%)	2.183 (+16%)
Classic 128,8	1.0	7595	4.103
Proposed 128,8	1.0	8653 (+14%)	4.756 (+16%)
Classic 32,16	1.0	3359	1.71
Proposed 32,16	1.0	3767 (+12%)	2.052 (+20%)
Classic 64,16	1.0	7032	3.498
Proposed 64,16	1.0	8016 (+14%)	3.969 (+13%)

TABLE XIII. POST-LAYOUT IMPLEMENTATION RESULTS

R,C	**Standard architect.**		**Proposed full precis.**		**Proposed PSG2=1; PSG1=D**			**Proposed PSG2=D; SG1=D**		
	P_{RW} [mW]	**P_R** [mW]	**P_{RW}** [mW]	**P_R** [mW]	**P_{RW}** [mW]	**P_R** [mW]	**SSIM**	**P_{RW}** [mW]	**P_R** [mW]	**SSIM**
64,8	0.425	0.307	0.516 +21%	0.309 +0.7%	0.302 –29%	0.170 –44%	0.64	-	-	-
128,8	0.515	0.289	0.653 +27%	0.307 +6%	0.386 –25%	0.193 –33%	0.64	-	-	-
32,16	0.433	0.319	0.535 +21%	0.333 +4%	0.354 –20%	0.219 –31%	0.99	0.216 –51%	0.141 –56%	0.65
64,16	0.654	0.446	0.768 +17%	0.455 +2%	0.506 –23%	0.266 –40%	0.99	0.287 –56%	0.151 –66%	0.65

To further investigate this aspect, the Fig. 31 shows the quality degradation obtained after applying a sharp filtering to an 8-bit depth grayscale image, read from the proposed memory with PSG disabled. While quality degradation is perceptible, it is still adequate for several error tolerant applications.

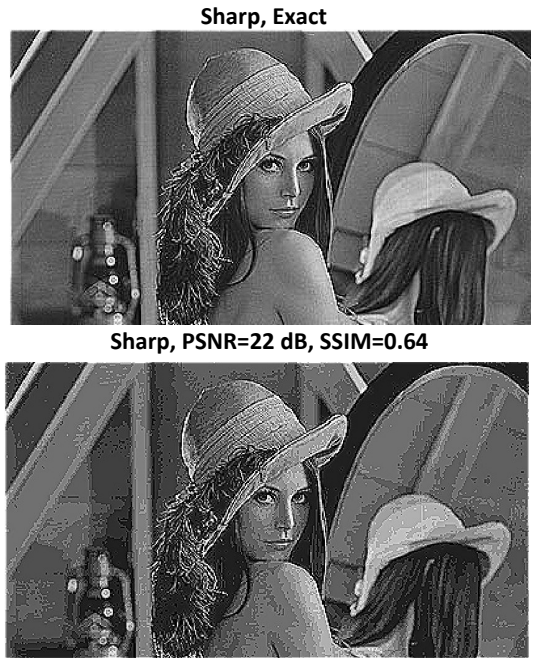

Fig. 28 Quality degradation due to the scaled precision in the proposed memory, during a sharp filtering operation.

3.6 Conclusions

In this chapter my research activity regarding precision-scalable units has been discussed. The first part of the chapter focuses on truncated multipliers. This are usually employed in DSP applications to trade performance with accuracy, in a fixed way. The state of art error compensation technique for truncated multipliers have been discussed. Then, a precision-scalable truncated multiplier, introduced recently in the literature, has been introduced. This can be used in precision scalable-systems to account for the different precision needs and error resiliency of processed datasets. For this kind of precision-scalable multipliers no error compensation technique

have been proposed in hardware. Therefore an hardware compensation technique have been proposed, able to adapt to different dataset. The proposed compensation technique is named as Real-Time Data-Aware Compensation technique, since it sense, which a given subsampling period, the committed errors, without making any statistical assumption on the error done. In this way the error can be compensated, independently from the particular dataset statistic. A compensation circuit is devoted to calculate the errors and adjust the value of the compensation term, on the basis of the mean error done. In order to achieve high energy efficiency the subsampling period of the error must be aggressive. The proposed approach has been employed in a precision-scalable Multiply and Accumulate Unit, for computer vision applications.

The comparison with the other compensation technique shows that proposed approach is able to adapt to different dataset providing significantly improved quality results where the other technique fails and equal quality results where other compensation technique alternate their domain.

In terms of quality-power tradeoff, proposed MAC Unit with Real-Time Data-Aware compensation technique exhibits improved quality-energy performance for deeply-scaled precision level, allowing to shift the energy bound toward higher energy efficiency levels. The proposed MAC has been employed on some Convolutional Neural Network kernels showing an always better energy-quality tradeoff than precision-scalable MAC with no compensation technique.

Moreover, a precision-scalable Approximate MAC unit has been proposed. The MAC employs partial products recoding and compression to achieve energy efficiency. The precision-scalability allows tuning the precision level according to the incoming dataset. A low-overhead compensation technique is employed to coarsely compensate the committed errors. The proposed circuit allows reaching 53% power reduction with tolerable image quality degradation.

In the second part of the chapter a precision-scalable latch-based memory has been discussed. This memory can be used as embedded memory in precision-scalable systems.

When the precision is scaled down, the dynamic power dissipation can be reduced more than 60%, compared to the classical Standard Cell Memory topology, with tolerable image quality degradation.

Chapter 4

A Precision-scalable Approximate Convolver

4.1 Introduction

In this last chapter a precision-scalable Approximate Convolver is discussed. 2D Convolution is a basic and compute-intensive operation in many computer vision tasks such as image processing and machine learning. Recently, Convolutional Neural Networks are achieving huge interest, due to their significantly better accuracy results (moreover the availability of GPU acceleration has drastically reduced their training time, allowing massive usage) with respect standard approaches, in typical machine learning tasks (image classification and segmentation, gesture recognition, object detection, speech recognition, etc.). In CNN the convolutions account for more than 90% of overall computation, dominating runtime and energy consumption [88]. In this context, hardware acceleration plays a critical role, allowing real time operations with optimized energy consumption. As discussed in [3], the energy due to the memory drastically impacts on the system energy budget, therefore, in the design of an accelerator, both the datapath and the on-chip (and offchip) energy contribution, must be accounted for and optimized.

In this chapter, the discussed precision-scalable Latch Memory (chapter 3, paragraph 3.5) and precision-scalable Approximate MAC Unit (chapter 3, paragraph 3.4) are employed in order to implement a convolver for computer vision applications and machine learning. The MAC unit allows mapping in hardware the 2D-convolution operation, common to computer vision and machine learning tasks. By employing the precision-scalable Latch Memory, the energy benefit deriving from the approximate datapath (Approximate MAC Unit), is not hidden by the memory energy consumption, since, when the Approximate MAC works in precision-scaled modality, also the memory can scale its precision and therefore its energy dissipation.

4.2 Architecture

In this paragraph the architecture of the precision-scalable Approximate Convolver is described. Let us recall that in the 2D-convolution operation (3.36), a convolutional window of size *qxq*, representable as the kernel matrix, is slid on the image, as shown in Fig. 1, where two convolutional windows W1 and W2 are shown.

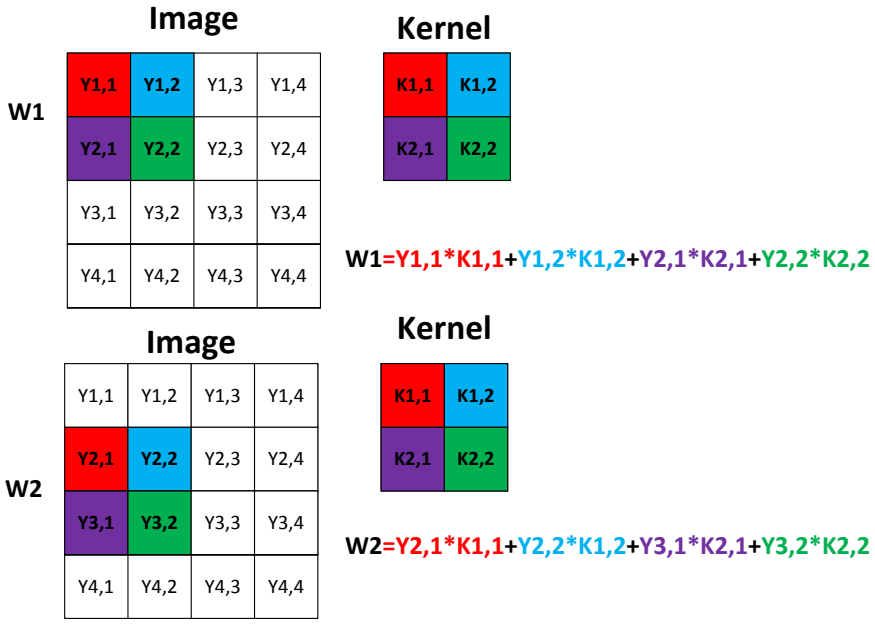

Fig. 1 The kernel is slid on the image. Therefore the filtering of the whole image requires multiple convolutional windows (like W1 and W2) .

The redundant nature of the 2D-convolution operation allows exploiting data reuse, in order to reduce the access to both on-chip and off-chip memory [89]. Moreover, since multiple convolutional windows are required to filter the whole input image (Fig. 1), this allows to exploit parallelism in order to achieve real-time operation. Therefore in the proposed architecture each convolutional window is mapped on a single Approximate MAC Unit. The resulting architecture is shown in Fig. 2. Here, three Approximate MAC Units are considered, organized on three rows. Each of them receives one operand (image pixel) from the corresponding row memory, while the

other operand is shared between the MAC of the column and is provided by a memory storing the kernel coefficients.

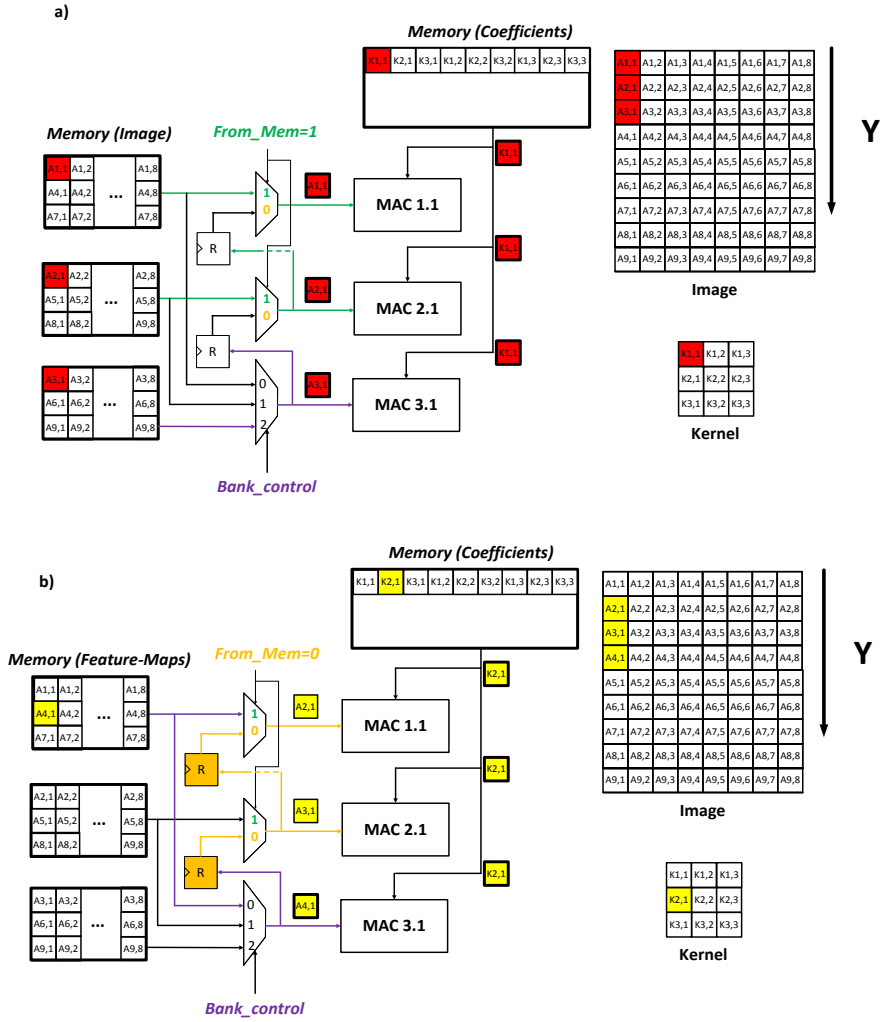

Fig. 2 (a) A new convolution is started. Three image pixels, corresponding to three convolutional windows computed in parallel through the three MACs, are loaded from the rows memory, together with a kernel coefficient. (b) second step of the convolution: two image pixels have been read in the previous clock cycle, therefore their value is read from the auxiliary registers with the help of the mux logic, exploiting, in this way, the data redundancy characterizing 2D-convolution operation.

The working principle is described in the following. Let us observe that the employed topology allows parallelizing the

convolutional window along the Y dimension (Fig. 2). The Fig. 2 (a) shows the start of a convolutional operation: three image pixels are read from the three rows memory together with a kernel coefficient, read from the memory on the top. The pixels and coefficient read are highlighted on the right side of the Fig. 2 (a). Each of the three MACs perform the multiplication between the corresponding image pixel and the shared kernel pixel. Therefore different pixels, displaced along the Y dimension are multiplied by the MACs with the same kernel coefficient. This means that each MAC is computing a different convolutional window, each of them displaced along Y (compare with Fig. 1, showing that the same kernel coefficient *K1,1* is multiplied with *Y1,1* and *Y2,1*, belonging to two different and vertically displaced convolutional windows W1 and W2). In the next clock cycle (Fig. 2 (b)) one novel image pixel is loaded from the memory and processed by the MAC at third row, while the remaining two pixels, being already read at the previous clock cycle, are shifted up through the registers (highlighted in yellow) and the mux logic, exploiting data reuse. A novel kernel coefficient is instead needed at each clock cycle. At the tenth clock cycle (the kernel is 3x3 in this example and we are accounting for the latency of the accumulation register) three filtered pixel are produced by the three MACs, each one is the result of the corresponding convolutional window, as shown in Fig. 3. Note that when a new column of the convolutional windows starts, all the image pixels are read from the rows memories, while, in the other cases only one image pixel is read from one pixel memory and provided to the MAC in the last row. The described mechanism assumes that the image pixels are memorized in the memory rows in interleaving, in particular, pixels belonging to the same image row must be stored in the same row memory, while two contiguous pixel row must be stored into different (preferably contiguous to facilitate the control circuit) rows memories. The Fig. 4 shows the data moving during the convolution operation. The bolded edges represents the pixel loaded from the rows memories in a given clock cycle.

The architecture shown in Figs. 2-3 can be extended on more columns, allowing to filter the same image with different kernels, at the same time. This scenario is typical in Convolutional Neural Networks. In this way the data reuse is further exploited. The Fig. 5 shows the resulting 2D architecture, in which *LxH* MAC Units are

employed, while $L+H$ memories are needed. Once the convolution is completed, the resulting H outputs per column, must be provided as output. To this regard, the MAC Units can employ a register after the accumulation one.

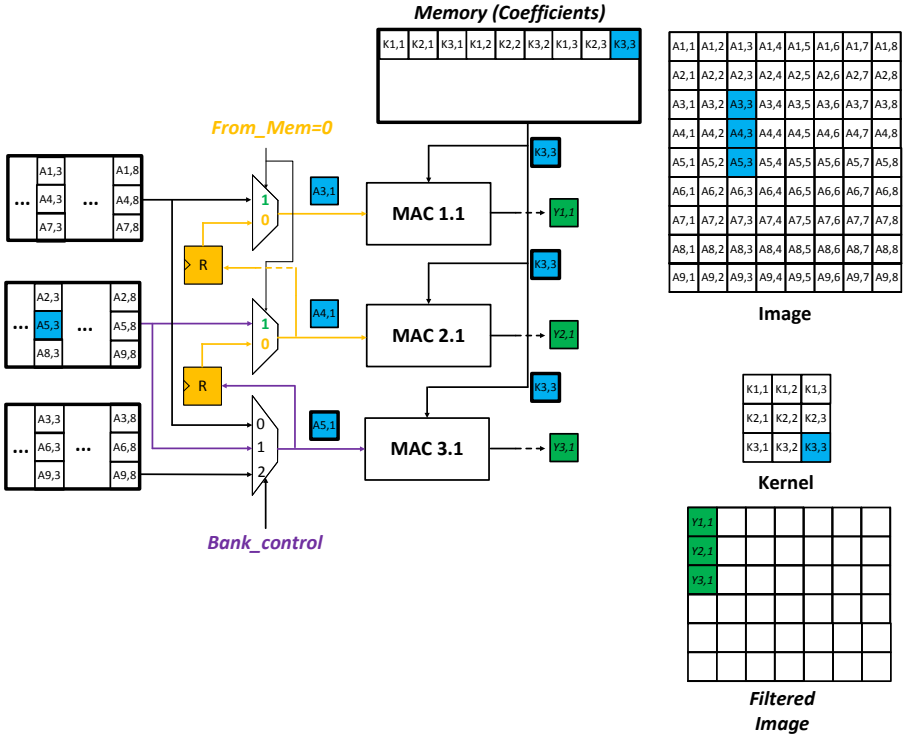

Fig. 3 End of the convolution. Three outputs, corresponding to the three convolutional windows parallelized, are produced.

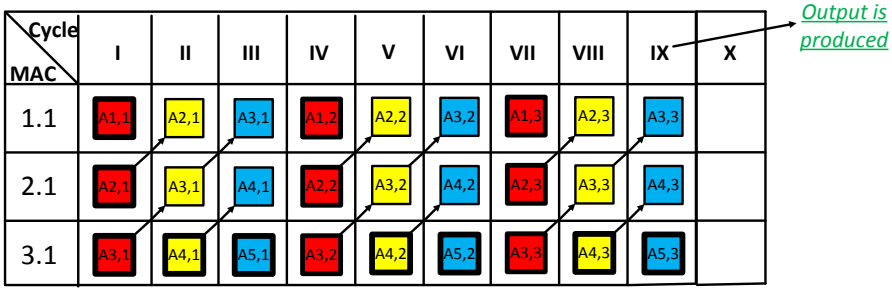

Fig. 4 Data moving between the MAC at different clock cycles. The bold edges indicates pixels that, in a given clock cycle, are read from the memory, the arrows highlight reused pixels.

This register is normally clock-gated and is enabled when a new result from convolution must be stored (therefore each q^2 clock cycles, being the kernel matrix qxq). Therefore, at the end of each convolution H filtered pixels are stored into the corresponding MAC output register for q^2 clock cycles, allowing to provide to the output the H pixels through a multiplexer logic, if the following condition holds:

$$q^2 \geq H \tag{4.1}$$

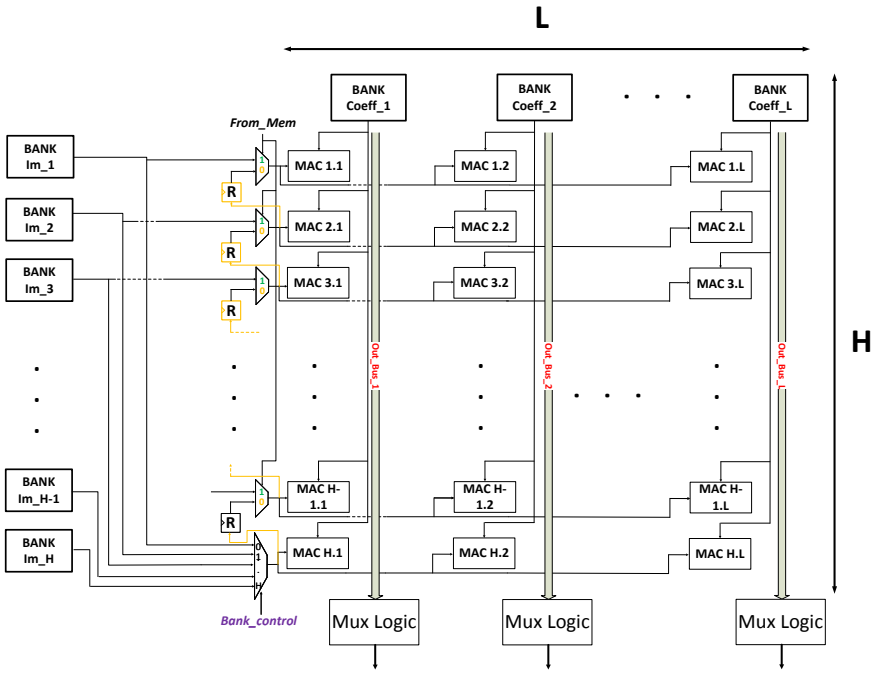

Fig. 5 Convolver Architecture. *LxH* MACs are employed, while *L+H* memories are needed.

4.3 Precision-scalability

A discussion about the combined precision-scalability of Latch Memory ad MAC Unit is needed. As for chapter 3, *nt* represents the number of discarded least significant columns of the compressed PPM, while the Latch Memory employs a single Precision Scalable Group (PSG), allowing to freeze (to zero, as discussed in chapter 3) 4 least significant bits.

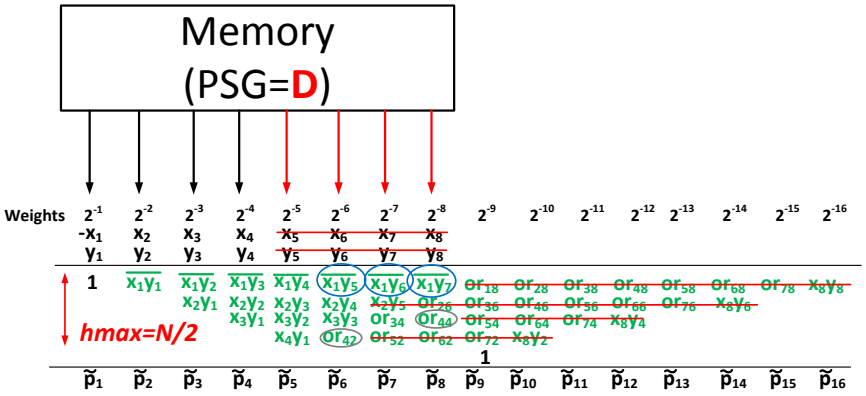

Fig. 6 Resulting compressed PPM when the memories precision is scaled down.

The Fig. 6 shows the compressed PPM when the Latch Memories operate in scaled precision modality. The red-barred terms are deleted due to the precision reduction of the memory. The gray circled terms are OR-compressed partial products where one of the two is zero (therefore surviving only the non-zeroed partial product), while the blue circled terms are 1, due to the NAND. We can observe that , when PSG=**D**, varying *nt* from 0 to 8 has no effect on the compressed partial product matrix, being the terms already zeroed by the memory. In the case of PSG=**D**, *nt=8* corresponds to the maximum precision of the Approximate MAC. Note that corrected compensation constant must be calculated for *nt=8-9-10*. The calculation of this constant can be done by following the discussion of the paragraph 3.43 (chapter 3). Note that, however the blue circled terms offers a partial compensation, being 1. It is worthwhile observing that for $nt \geq 11$ no modifications of the PPM occurs due the memory, therefore the compensation constant is the same of that determined in the paragraph 3.43 (chapter 3).

Note that while in this paragraph we have assumed that both the memories are precision-scaled, hybrid scenarios can be adopted, when only on the of the two memories operates at reduced precision. However, in the following of the chapter we will assume that both the Latch Memories providing the operands to the MACs scale down the precision.

4.4 VLSI Implementation results

The proposed architecture has been described in Verilog HDL and synthesized in TSMC 40nm CMOS technology using Cadence RTL Compiler. In our case $H = L = 4$, therefore 16 precision-scalable Approximate 8-bit MAC Units (chapter 3, paragraph 3.4) have been employed, while 4 precision-scalable 0.512Kbit Latch Memory (chapter 3, paragraph 3.5) have been employed as frame buffer for the image pixels and 4 precision-scalable 0.256Kbit Latch Memory have been employed to store the four kernels (note that 0.256Kbit allows storing up to 5x5 kernels, represented on 8-bits). Obviously, other choice for H and L can be employed, depending on the application and on area, power and throughput constraints. A finite state machine has been implemented in order to manage the memories control signals to implement the convolution operation. A full-precision convolver, employing standard MAC Units and Latch Memories has also been described and synthesized, in order to evaluate the performance improvements. The Tab. I shows the area and leakage performance of the proposed circuits.

TABLE I. VLSI IMPLEMENTATION RESULTS-@VDD=1.1V

Circuit	F [GHz]	Area [μm^2]	$P_{LEAKAGE}$ [μW]
Classic Convolver	1.0	34100	28.897
Proposed Convolver	1.0	38329(+12%)	32.161(+11%)

As shown in Tab. I the proposed precision-scalable Approximate Convolver exhibits an area overhead of 12% and a similar leakage overhead. Note that this overhead is due to the precision-scalable Latch Memory (compare Tab. XII of chapter 3).

TABLE II. POWER DISSIPATION-@VDD=1.1V; F=1GHZ

Circuit	PSG	nt	$P_{DYNAMIC}$ [mW]
Classic Convolver	/	/	59.528
Proposed Convolver	1	0	55.578(-7%)
Proposed Convolver	1	4	52.597(-12%)
Proposed Convolver	1	8	46.064(-23%)
Proposed Convolver	1	11	40.587(-32%)
Proposed Convolver	1	12	37.770(-37%)
Proposed Convolver	**D**	8	31.249(-48%)
Proposed Convolver	**D**	11	26.150(-56%)
Proposed Convolver	**D**	12	23.389(-61%)

The Tab. II shows the dynamic power improvements for different precision levels, when the Lena image is convolved with a Gaussian filter. Note that at working clock frequency (1GHz) the dynamic power dissipation is about three order of magnitude higher than leakage power (Tab. I). Therefore in the comparison of Tab. II the improvement expressed in terms of dynamic power corresponds to the improvement in terms of total power dissipation. We can observe that in case of fullprecision (PSG=1; nt=0) there is a little power saving (7%) due to the approximate compression of the partial product matrix of the Approximate MAC Units, allowing to compensate the overhead due to the precision-scalable Latch Memories (compare Tab. XII of chapter 3). When the memories are in full-precision (PSG=1) and only the precision of the Approximate MAC Unit is decreased, power savings up to 32% can be achieved with tolerable image quality degradation (*nt=11*, corresponding to point b in Fig. 27, chapter 3). It is worthwhile observing that for the case in which only the Approximate MAC Unit was considered (chapter 3), for *nt=11* a power saving of 53% could be achieved. Therefore the memories partially hide the power saving achieved using approximate arithmetic blocks. When the precision-scalable Latch Memories operate in scaled precision-mode (PSG=**D**), significant power savings are observed, allowing a 56% power saving with tolerable quality degradation (as discussed in the previous paragraph, there is no difference in terms of discarded partial products in the PPM between the conditions (PSG=**D**, nt=11) and (PSG=1, nt=11), both corresponds to point b of Fig. 27 of chapter 3, therefore for nt=11 there is no reason to keep the memory in full-precision mode).

4.5 Conclusions

A precision scalable Approximate convolver has been discussed in this chapter. The convolver employes the precision-scalable Latch Memory and the precisionscalable Approximate MAC Unit discussed in the chapter 3, to develop a convolutor for computer vision and machine learning applications. Due to the precision-scalability of the memories, the precision-scalable Approximate Units energy improvement are not

hidden. Indeed, when Latch Memories operate in scaled precision modality, the power saving, with respect standard convolver is increased up to 56%, with tolerable image quality degradation.

References

[1] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," in IEEE Journal of Solid-State Circuits, vol. 9, no. 5, pp. 256-268, Oct 1974.

[2] "Advancing Moore's Law —The Road to 14 nm", Intel presentation,[Online], Available: http://www.intel.com/content/www/us/en/silicon-innovations/advancing-moores-law-in-2014-presentation.html.

[3] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, San Francisco, CA, 2014, pp. 10-14.

[4] C. Martin, "Multicore Processors: Challenges Opportunities Emerging Trends", *Proceedings Embedded World Conference 2014*, 25–27 February 2014.

[5] Y. K. Chen *et al*., "Convergence of Recognition, Mining, and Synthesis Workloads and Its Implications," in *Proceedings of the IEEE*, vol. 96, no. 5, pp. 790-807, May 2008.

[6] S. Venkataramani, S. T. Chakradhar, K. Roy and A. Raghunathan, "Computing approximately, and efficiently," *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, 2015, pp. 748-751.

[7] "ISSCC 2016 Tech Trends" [Online], Available: http://isscc.org/doc/2016/ISSCC2016_TechTrends.pdf

[8] H. Y. Cheng, J. Zhan, J. Zhao, Y. Xie, J. Sampson and M. J. Irwin, "Core vs. uncore: The heart of darkness," *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, San Francisco, CA, 2015, pp. 1-6.

[9] 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 Conference (DAC)*, Austin, TX, 2013, pp. 1-9.

[10] 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.

[11] S. Venkataramani, S. T. Chakradhar, K. Roy and A. Raghunathan, "Approximate computing and the quest for computing efficiency," *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, San Francisco, CA, 2015, pp. 1-6.

[12] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies (C.E. Shannon and J. Mc-Carthy eds.), Princeton Univ. Press, Princeton, N.J.*, 1956.

[13] A. Yazdanbakhsh *et al*., "Axilog: Language support for approximate hardware design," *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, 2015, pp. 812-817.

[14] M. Carbin, S. Misalovic, M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware", Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications.

[15] A. Sampson *et al,* "EnerJ: Approximate data types for safe and general low-power computation", Proceedings of the 2011 ACM SIGPLAN international conference on Object oriented programming systems languages & applications.

[16] V. Gupta, D. Mohapatra, A. Raghunathan and K. Roy, "Low-Power Digital Signal Processing Using Approximate Adders," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124-137, Jan. 2013.

[17] Shih-Lien Lu, "Speeding up processing with approximation circuits," in *Computer*, vol. 37, no. 3, pp. 67-73, Mar 2004.

[18] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," *DAC Design Automation Conference 2012*, San Francisco, CA, 2012, pp. 820-825.

[19] T. Liu and S. Lu, "Performance improvement with circuit-level speculation," *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, Monterey, CA, 2000, pp. 348-355.

[20] A. K. Verma, P. Brisk, P. Ienne, "Variable latency speculative

addition: A new paradigm for arithmetic circuit design", *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, pp. 1250-1255, Mar. 2008.

[21] V. Camus, J. Schlachter and C. Enz, "Energy-efficient inexact speculative adder with high performance and accuracy control," *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, Lisbon, 2015, pp. 45-48.

[22] I. C. Lin, Y. M. Yang and C. C. Lin, "High-Performance Low-Power Carry Speculative Addition With Variable Latency," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1591-1603, Sept. 2015.

[23] T. Austin, V. Bertacco, D. Blaauw and T. Mudge, "Opportunities and challenges for better than worst-case design," *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005.*, 2005, pp. I/2-I/7 Vol. 1.

[24] I. Koren, *Computer Arithmetic Algorithms*, Natick, MA, USA: A K Peters, 2002.

[25] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Design*. New York: Oxford Univ. Press, 2000.

[26] N. H. E. Weste, D. M. Harris, *CMOS VLSI Design*, 4th ed., Addison-Wesley.

[27] H. Jiang, J. Han, F. Lombardi, "A comparative review and evaluation of approximate adders", Proc. of Great Lakes Symposium on VLSI, pp. 343-348, Pittsburgh, 2015.

[28] M. Shafique, W. Ahmad, R. Hafiz, J. Henkel, "A low-latency generic accuracy configurable adder", proc. of Design Automation Conf., DAC'15, pp. 86:1-86:6, June 2015, San Francisco.

[29] N. Zhu, W. L. Goh, G. Wang, K. S. Yeo, "An enhanced low-power high-speed adder for error tolerant applications", proc. of International Symp. on Integrated Circuits, (ISIC), pp. 67-69, 2009.

[30] K. Du, P. Varman, K. Mohanram, "High performance reliable variable latency carry select addition," Design, Automation and Test in Europe DATE'12, pp.1257-1262, March 2012.

[31] D. Esposito, D. De Caro and A. G. M. Strollo, "Variable Latency Speculative Parallel Prefix Adders for Unsigned and Signed Operands," in *IEEE Transactions on Circuits and Systems*

I: Regular Papers, vol. 63, no. 8, pp. 1200-1209, Aug. 2016.

[32] Kogge, Peter M.; Stone, Harold S., "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," Computers, IEEE Transactions on , vol.C-22, no.8, pp.786,793, Aug. 1973.

[33] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," in *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260-264, March 1982.

[34] T. Han; D. A. Carlson, "Fast area-efficient VLSI adders," *Computer Arithmetic (ARITH), 1987 IEEE 8th Symposium on*, pp.49,56, 18-21 May 1987.

[35] S. M. Sudhakar, K. P. Chidambaram, E. E. Swartzlander, "Hybrid Han-Carlson adder," IEEE 55th Intern. Midwest Symposium on Circuits and Systems (MWSCAS), pp.818-821, 5-8 Aug. 2012.

[36] J. Sklansky, "Conditional-sum addition logic," IRE Trans. Electronic Computers, vol. EC-9, pp. 226-231, June 1960.

[37] R.E. Ladner, M.J. Fischer, "Parallel Prefix Computation," *Journal of the ACM*, Vol. 27, No 4, October 1980, pp 831-838.

[38] Mathew, S.K.; Krishnamurthy, R.K.; Anders, M.A.; Rios, R.; Mistry, K.R.; Soumyanath, K., "Sub-500-ps 64-b ALUs in 0.18-μm SOI/bulk CMOS: design and scaling trends," *Solid-State Circuits, IEEE Journal of* , vol.36, no.11, pp.1636,1646, Nov 2001.

[39] D. Esposito, D. De Caro, E. Napoli, N. Petra and A. G. M. Strollo, "Variable Latency Speculative Han-Carlson Adder," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 5, pp. 1353-1361, May 2015.

[40] Nowick, S.M., "Design of a low-latency asynchronous adder using speculative completion," *Computers and Digital Techniques, IEE Proceedings* - , vol.143, no.5, pp.301,307, Sep 1996.

[41] D. Esposito, D. De Caro, M. De Martino and A. G. M. Strollo, "Variable latency speculative Han-Carlson adders topologies," *2015 11th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, Glasgow, 2015, pp. 45-48.

[42] A. Cilardo, "A new speculative addition architecture suitable for two's complement operations," *Design, Automation & Test in*

Europe Conference & Exhibition, DATE '09, pp.664-669, April 2009.

[43] D. Esposito, G. Castellano, D. De Caro, E. Napoli, N. Petra and A. G. M. Strollo, "Approximate adder with output correction for error tolerant applications and Gaussian distributed inputs," *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, Montreal, QC, 2016, pp. 1970-1973.

[44] K. Du, P. Varman, K. Mohanram, "Static window addition: a new paradigm for the design of variable latency adders," *Proc. Intl. Conf. on Computer Design*, pp. 455–456, 2011.

[45] A. Cilardo, D. De Caro, N. Petra, F. Caserta, N. Mazzocca, E. Napoli, A.G.M. Strollo, "High Speed Speculative Multipliers Based on Speculative Carry-Save Tree," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol.61, no.12, pp.3426,3435, Dec. 2014.

[46] D. Kelly, J. Phillips, "Arithmetic data value speculation," in *Advances In Computer Systems Architecture*, Lecture Notes in Computer Science, pp. 353–366, 2005.

[47] C. Liu, J. Han, F. Lombardi, "An analytical framework for evaluating the error characteristics of approximate adders", IEEE Trans. on Computer, vol. 64, n. 5, pp.1268-1281, May 2015.

[48] ChipWare IP Components in Encounter® RTL Compiler, Cadence, Product Version 14.2, August 2015.

[49] Synopsys DesignWare Building Block IP User Guide, 2009.

[50] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," in IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, April 2004.

[51] V. G. Oklobdzija, D. Villeger, S. S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach", IEEE Transactions on Computers, Vol. 45 n. 3, March 1996, pp.294-306.

[52] V. Chippa, A. Raghunathan, K. Roy and S. Chakradhar, "Dynamic effort scaling: Managing the quality-efficiency tradeoff," *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, New York, NY, 2011, pp. 603-608.

[53] Raha, H. Jayakumar and V. Raghunathan, "Input-Based

Dynamic Reconfiguration of Approximate Arithmetic Units for Video Encoding," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 3, pp. 846-857, March 2016.

[54] B. Moons and M. Verhelst, "DVAS: Dynamic Voltage Accuracy Scaling for increased energy-efficiency in approximate computing," *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Rome, 2015, pp. 237-242.

[55] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester and M. Alioto, "SRAM for Error-Tolerant Applications With Dynamic Energy-Quality Management in 28 nm CMOS," in *IEEE Journal of Solid-State Circuits*, vol. 50, no. 5, pp. 1310-1323, May 2015.

[56] Y. Tian, Qian Zhang, Ting Wang, Feng Yuan, Qiang Xu, "ApproxMA: Approximate Memory Access for Dynamic Precision Scaling", *ACM Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pp. 337-442.

[57] M. de la Guia Solaz, W. Han and R. Conway, "A Flexible Low Power DSP With a Programmable Truncated Multiplier," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 11, pp. 2555-2568, Nov. 2012.

[58] R. Zimmerman, "Getting the Most from Synthesis to Improve your Datapath QoR", *Proceedings of Synopsys Users Group 2012,* Germany, 2012.

[59] Synopsys Design Compiler Synthesis Commands, version G-2012.06, June 2012.

[60] C. Baugh, B. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Transaction on Computers,* vol. C-22, no. 12, Dec. 1973, pp. 1045-1047.

[61] 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.

[62] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349-356, Jan. 1965.

[63] A. Weinberger, "4:2 Carry-Save Adder Module," *IBM Technical Disclosure Bull.*, vol. 23, Jan. 1981.

[64] P. Song, G. De Micheli, "Circuits and Architecture Trade-Offs for High Speed Multiplication," *IEEE J. Solid State Circuits,*

vol. 26, no. 9, Sept. 1991.

[65] N. Petra, D. De Caro, V. Garofalo, E. Napoli and A. G. M. Strollo, "Truncated Binary Multipliers With Variable Correction and Minimum Mean Square Error," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 6, pp. 1312-1325, June 2010.

[66] Y. C. Lim, "Single-precision multiplier with reduced circuit complexity for signal processing applications," *IEEE Trans. Comput.*, vol. 41, no. 10, pp. 1333–1336, Oct. 1992.

[67] M. J. Schulte and E. E. Swartzlander, "Truncated multiplication with correction constant [for DSP]," *Proceedings of IEEE Workshop on VLSI Signal Processing*, Veldhoven, 1993, pp. 388-396.

[68] S. S. Kidambi, F. El-Guibaly and A. Antoniou, "Area-efficient multipliers for digital signal processing applications," in *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 2, pp. 90-95, Feb 1996.

[69] E. J. King and E. E. Swartzlander Jr., "Data dependent truncated scheme for parallel multiplication," in *Proc. 31th Asilomar Conf. on Signals, Circuits Syst.*, 1997, pp. 1178–1182.

[70] E. E. Swartzlander, "Truncated multiplication with approximate rounding," *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers (Cat. No.CH37020)*, Pacific Grove, CA, USA, 1999, pp. 1480-1483 vol.2.

[71] M. J. Schulte, J. E. Stine and J. G. Jansen, "Reduced power dissipation through truncated multiplication," *Proceedings IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, Como, 1999, pp. 61-69.

[72] J. E. Stine and O. M. Duverne, "Variations on truncated multiplication," *Euromicro Symposium on Digital System Design, 2003. Proceedings.*, Belek-Antalya, Turkey, 2003, pp. 112-119.

[73] H. Park and E. E. Swartzlander, "Truncated Multiplication with Symmetric Correction," *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, 2006, pp. 931-934.

[74] Jer Min Jou, Shiann Rong Kuang and Ren Der Chen, "Design of low-error fixed-width multipliers for DSP applications,"

in *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, no. 6, pp. 836-842, Jun 1999.

[75] F. Curticapean and J. Niittylahti, "A hardware efficient direct digital frequency synthesizer," *ICECS 2001. 8th IEEE International Conference on Electronics, Circuits and Systems (Cat. No.01EX483)*, 2001, pp. 51-54 vol.1.

[76] Lan-Da Van, Shuenn-Shyang Wang and Wu-Shiung Feng, "Design of the lower error fixed-width multiplier and its application," in *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 10, pp. 1112-1118, Oct 2000.A. G. M. Strollo, N. Petra and D. De Caro, "Dual-Tree Error Compensation for High Performance Fixed-Width Multipliers," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 52, no. 8, pp. 501-507, Aug. 2005.

[77] A. G. M. Strollo, N. Petra and D. De Caro, "Dual-Tree Error Compensation for High Performance Fixed-Width Multipliers," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 52, no. 8, pp. 501-507, Aug. 2005.

[78] S. R. Kuang and J. P. Wang, "Low-error configurable truncated multipliers for multiply-accumulate applications," in *Electronics Letters*, vol. 42, no. 16, pp. 904-905, August 3, 2006.

[79] N. Petra, D. De Caro, V. Garofalo, E. Napoli and A. G. M. Strollo, "Design of Fixed-Width Multipliers With Linear Compensation Function," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 5, pp. 947-960, May 2011.

[80] D. De Caro, N. Petra, A. G. M. Strollo, F. Tessitore and E. Napoli, "Fixed-Width Multipliers and Multipliers-Accumulators With Min-Max Approximation Error," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 9, pp. 2375-2388, Sept. 2013.

[81] R. Jain, R. Kasturi, B. G. Schunck, *Machine Vision,* McGraw-Hill, 1995.

[82] K. Chatfield, K. Simonyan, A. Vedaldi, A. Zisserman, “Return of the Devil in Details: Devolving Deep into Convolutional Networks”, British Machine Vision Conference, 2014.

[83] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, A. Yakovlev, “Energy-Efficient Approximate Multipliers using Bit Significance-

Driven Logic Compression", accepted at *Design, Automation, and Test in Europe 2017 (DATE) conference.*

[84] P. Meinerzhagen, C. Roth and A. Burg, "Towards generic low-power area-efficient standard cell based memory architectures," 2010 53rd IEEE International Midwest Symposium on Circuits and Systems, Seattle, WA, 2010, pp. 129-132.

[85] O. Andersson, B. Mohammadi, P. Meinerzhagen, A. Burg and J. N. Rodrigues, "Ultra Low Voltage Synthesizable Memories: A Trade-Off Discussion in 65 nm CMOS," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 63, no. 6, pp. 806-817, June 2016.

[86] P. Meinerzhagen, S. M. Y. Sherazi, A. Burg and J. N. Rodrigues, "Benchmarking of Standard-Cell Based Memories in the Sub- Vt Domain in 65-nm CMOS Technology," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 1, no. 2, pp. 173-182, June 2011.

[87] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, A.Burg, "Power, Area, and Performance Optimization of Standard Cell Memory Arrays through Controlled Placement", ACM TOADES, 2016.

[88] J. Emer, V. Sze, Y-H. Chen, "Tutorial on Hardware Architectures for Deep Neural Networks," ,[Online], Available: http://eyeriss.mit.edu/tutorial.html, 2016.

[89] Y. H. Chen, T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127-138, Jan. 2017.