


Energy-efficient digital processing via Approximate Computing

# Energy-Efficient Digital Processing via Approximate Computing

Daniele Jahier Pagliari and Massimo Poncino and Enrico Macii

**Abstract**

## 1 Introduction

While the most characteristic feature of a Smart System is its capability of *sensing* a set of environmental quantities and *actuating* appropriate actions in response to those signals, it is obvious that a significant part of its functional operations is involved with the *elaboration* of the information carried by the signals [15]. This elaboration is usually done after converting the analog, asynchronous environmental signals into the digital domain.

Part of the smartness of a Smart Systems is therefore expressed in the autonomous and transparent operation based on closed loop control and predictive capabilities, as well as improved signal processing technologies. The former functions are normally carried out by a micro-controller or processor core, whereas the latter ones rely on either a Digital Signal Processor (DSP) or an Application Specific Integrated Circuit (ASIC). Hybrid architectures, that combine one or more general purpose CPUs with one or more hardware *accelerators* are also increasingly popular [12].

Such "processing" dimension, coupled with the energy-autonomous nature of these systems put significant emphasis on their energy efficiency [4, 13]. Measures for reducing energy (and power) consumption vary according to the engineering domain of the component being considered. In the computing subsystem, classical

Daniele Jahier Pagliari
Politecnico di Torino, www.polito.it e-mail: daniele.jahier@polito.it

Massimo Poncino
Politecnico di Torino, www.polito.it e-mail: massimo.poncino@polito.it

Enrico Macii
Politecnico di Torino, www.polito.it e-mail: enrico.macii@polito.it

low-power techniques for processors and digital circuits can be fruitfully exploited [36]. In this chapter, however, we focus on the explicit signal processing task and show how we can effectively leverage an emerging design paradigm called Approximate Computing [21, 51].

Approximate Computing has its foundations in the tradeoff between quality and energy. It is based on the principle that, accepting a controlled degradation in output quality, energy consumption can be reduced significantly, by changing either the implementation of a computing device or its operating conditions. In Smart Systems, inputs to the computing subsystem often consist of physical data sampled by some type of sensor. This information is inherently imprecise, both because of environmental noise and measurement errors, and because of the limited precision of transducers and analog-to-digital converters. Therefore, small approximations in computation may be accepted, as their impact on the final output quality may be negligible with respect to the effect of input imprecisions.

Moreover, some of the outputs in Smart Systems are power actuators, which normally have much longer time constants with respect to electronic elements. As a consequence, an error that manifests only *intermittently* for short amounts of time is automatically "filtered-out" by the actuators.

Both aspects show that the maximum computation accuracy constraint is often overly restrictive in Smart Systems applications. In other words, these applications are *error resilient*, i.e. can tolerate some computational errors without a significant impact on the quality of results. This property makes them the ideal targets for the Approximate Computing methodology. In fact, if the design constraints of the processing subsystem are set correctly, approximations can be made negligible with respect to input errors and/or to the resolution of outputs, effectively reducing energy consumption without impacting the output quality in a significant manner.

## 2 Error-Resilient Computing Paradigms

In modern electronics, energy has become a primary concern, due in particular to the widespread diffusion of mobile, battery-powered devices. Researchers identified *error resilience* as a common characteristic often found in applications performed by such devices, including Smart Systems, that can be exploited to optimize their energy efficiency [8].

### *2.1 Error Resilience*

Error resilience, or tolerance, can be defined as the capability of tolerating some errors without a significant impact on output quality. There are several factors that affect the resilience of an application. In this Section, we try to categorize the most recurring ones (see Figure 1).

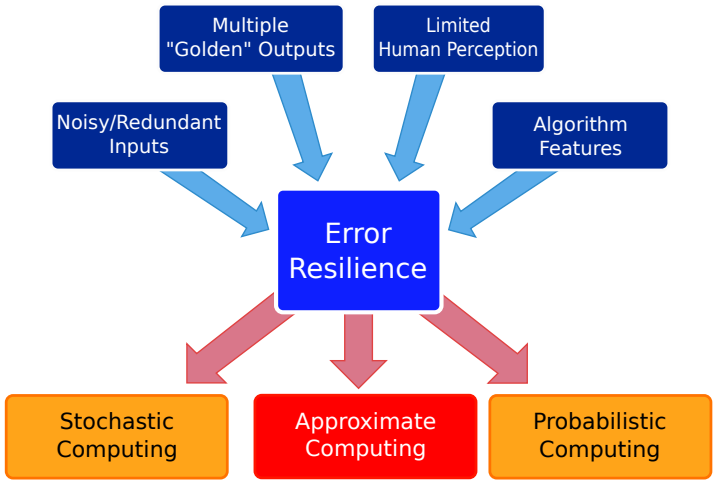


**Fig. 1** Overview of error resilience features and error resilient design paradigms.

**Noisy or redundant input data.** One cause of resilience is the fact that a system deals with inputs affected either by errors or by some form of environmental noise. In this situation, errors in computation can be tolerated, as long as they are negligible with respect to the imprecisions on inputs. The latter, in fact, effectively constitute an upper bound on the accuracy that can be obtained at the outputs. One example of this situation has been already anticipated in Section 1, and corresponds to applications that process data coming from analog sensors, which are inherently affected by environmental noise. Similarly, computing elements inserted in the decoding chain of a communication system also fall in this category, as their inputs are affected by noise and interferences in the channel. Another family of resilient applications are those that process a redundant input data set, as is typical in several machine learning tasks. In this case, some computations can be approximated or even skipped completely, because they do not add information (quality) to the results.

**Absence of a unique golden output.** For many applications, the definition of optimal, "golden" results is informal or fuzzy. This may happen because multiple outcomes are equivalently valuable for the purposes of the system mission. Alternatively, the optimality of the results produced by a computation can be inherently unknown, because of the presence of random, semi-random or heuristic operations. In the latter case, slightly perturbing the outputs of an internal operation might neigher worsen nor improve the quality of the final results. The first property is present in many data mining tasks. As an example, two similar outputs produced by a web browser search engine might be considered equally good for the end-user, and it is very difficult to distinguish the optimal. On the other hand, applications in the domains of optimization and operation research often display the second feature. In

literature, the broad family of Recognition, Mining and Synthesis (RMS) applications, is often reported as a particularly significant example of error resilience [8].

**Limited human perception.** Resilience can also come from the fact that the final results of a computing task are often evaluated by people. Human sense organs have a limited resolution both in terms of spacial and temporal dimensions and in terms of discernible "values". As an example, let us consider the visual system. Two full-HD images that differ only for a few pixels are hardly distinguishable to the naked eye (spatial dimension). Similarly, a single altered frame in a video stream is also practically unnoticeable (temporal dimension). Finally, the two images or videos are still perceived identical, even if differences are much more frequent in space and time, as long as the pixels colors are only slightly altered (values dimension). In summary, approximations that are sufficiently *rare*, in space or time, or sufficiently *small* do not affect the perceived quality. The majority of the applications that belong to this category are found in the domains of multimedia and telecommunication e.g. audio/video compression, imaging tasks, etc. Notice that this feature is partially overlapped with the previous one, since the fact that certain differences are not noticeable by humans can be interpreted as a quality equivalence among multiple outputs.

**Algorithmic features.** Lastly, an application can be resilient because of the inner characteristics of the involved algorithms. In particular, certain computational patterns favor the mitigation or the rejection of errors. A typical example of such patterns is *iterative refinement*, used in many recognition and mining applications as well as for the solution of systems of linear equations [21]. This pattern starts from an inexact initial solution and iteratively improves it. Because of how it is constructed, possible additional error contributions introduced by approximate computations will be reduced as well. Again, there is a partial overlapping between this characteristic and the absence of a unique golden output. For example, pseudo-randomness and heuristic decisions can be also thought of as algorithmic features.

In summary, in a resilient application, the quality of results can be thought of as a *continuous function* of the quality of computations, as opposed to a *boolean* one. Approximate Computing is one of the design paradigms that exploit this new dimension for the optimization of computing systems, mostly in terms of energy and power consumption (Figure 1).

## *2.2 Error Resilient Paradigms*

In literature, Approximate Computing is distinguished from other similar approaches, that also leverage error tolerance, such as Stochastic and Probabilistic Computing [21]. In this section we briefly describe the main characteristics and differences among these three perspectives.

### 2.2.1 Stochastic Computing

Stochastic Computing (SC) was first theorized in the 1960s by two independent researchers in Europe and in the United States [17, 43]. In this paradigm, information stored in streams of bits is interpreted numerically as the probability of occurrence of the logic value 1. As an example, a 10-bit stream with four 1s and six 0s is interpreted as the rational number 4/10. In formal terms, a string of $n$ bits with $n_1$ bits at logic 1 corresponds to the real number:

$$p = \frac{n_1}{n} \tag{1}$$

It can be seen easily that all numbers in this system belong to the interval [0,1], and that their representation is not unique. For example:

$$(0,0,0,1)_{SC} = (0,1,0,0)_{SC} = (0,0,1,0)_{SC} = 0.25_{10} \tag{2}$$

The initial interest for Stochastic Computing was motivated by the fact that some arithmetic operations between bit-streams can be implemented very efficiently in hardware [2]. Moreover, stochastic streams also have good tolerance properties in response to soft (i.e. transient) faults. In fact, if any of the bits of a $n$-bit string changes value because of a transient fault, the error in terms of the corresponding real number is always $1/n$. On the contrary, in binary the impact doubles for each bit going from the Least Significant Bit (LSB) to the Most Significant Bit (MSB).

However, Stochastic Computing also has some major drawbacks. First of all, increasing the precision of a calculation requires an exponential increase in the length of bit streams. This affects both the speed of calculations and the bandwidth required for memory access and communication. Moreover, not all possible representations of a real number yield the most accurate result when used as input of an operation. More specifically, *correlation* between the different inputs of an operation generates errors at the outputs. To cope with this issue, random or pseudo-random architectures (e.g. Linear Feedback Shift Registers) are used to generate the input stochastic streams in the "most uncorrelated" way possible. The problematic aspect is that the probability of correlation errors increases with the number of levels of logic, and in circuits with feedback. Furthermore, some topological aspects of circuits, such as *reconvergent fanouts*, enforce a strong correlation among streams.

The issues related with data correlation and the large bandwidth requirements constrained the application of SC away from the field of general purpose computing. However, this paradigm was used successfully in several domains, such as neural networks, control systems and image processing [2]. Recently, SC has regained interest due to its application to the decoding of Low-Density Parity Check (LDPC) codes. Many of the most efficient algorithms for this task are probabilistic, and therefore error resilient in nature. It has been shown that fast and low cost implementations of such algorithms benefit from a SC representation of data [19].

### 2.2.2 Probabilistic Computing

The trend labeled Probabilistic Computing (PC) originates from the increasing reliability issues of integrated circuits as technology scales [42]. Because of physical phenomena such as thermal noise and aging, standard CMOS devices are increasingly unreliable from one generation to another. Therefore, hardware designers and manufacturers put great effort in

building *reliable* circuits from *unreliable* physical switches via fault tolerant architectures, thermal aware design, etc.

Probabilistic Computing reverses this idea, accepting to work with *unreliable* circuits. To this end, a probabilistic model is formulated, in which each switching device is associated with a probability of correctness $p < 1$, and computing architectures are built on top of this model. The rationale that motivates PC is that accepting a small drop in $p$ can provide a large reduction in energy consumption, hence enabling very efficient computation.

Physical devices exhibiting this type of accuracy vs energy tradeoff have been first theorized and then manufactured, giving birth to the logic family known as Probabilistic CMOS (PCMOS). In parallel, Probabilistic Boolean Logic (PBL) has been adopted as an abstract model to support designs based on PCMOS [6]. Probabilistic Computing devices have been used to design architectures for various error resilient applications, including decision systems, pattern recognition, and cryptography, with promising results in terms of power efficiency [42].

### 2.2.3 Approximate Computing

Approximate Computing (AC), which is the main focus of this chapter, is allegedly the most successful error resilient design paradigm to this day [21]. It differentiates from Probabilistic Computing because the traditional deterministic logic model for switching devices is maintained; AC methods do not assume any probabilistic nature in the underlying physical devices that constitute the computing system. Similarly, AC is also a separate body of work with respect to Stochastic Computing. In fact, in most instances, information is processed and transmitted in standard binary representations (unsigned, two‘s complement, etc).

Approximate Computing focuses on building imprecise or inexact circuits with *deterministic* architectures (e.g. based on standard CMOS transistors). Statistical considerations, for example on input data, are extensively used, but in general, a reliable behavior of the hardware at the physical level is assumed. Hence, it is sometimes referred to also as Inexact Computing [42]. The main goal of AC is, as mentioned, power and energy reduction, but there are also techniques that exploit approximation to achieve performance increase and silicon area optimization..

The success of this approach is partly due to the fact that it relies on existing models and design techniques, modified appropriately to target the error resilient application domain, rather than completely changing the perspective. This makes it more accessible to hardware and software engineers accustomed to standard design

flows. As an example, many embodiments of Approximate Computing are inspired by techniques from the hardware and software reliability literature [23, 47, 7], while others make use of modified versions of standard Electronic Design Automation (EDA) algorithms [53, 48, 41].

In an attempt to build a topology of Approximate Computing techniques, a first order classification can be done to split them in two groups:

- The first family includes techniques that modify the *architecture* of an existing hardware or software design, introducing approximations in the computation, in such a way that the impact of these modifications on the output quality is statistically negligible, while the power and energy consumption are significantly reduced (e.g. [35]).
- The second group includes techniques in which power reduction is achieved by modifying the *operating conditions* of the target design rather than its structure. Structural modifications can still be present, but in this case they are a way to mitigate the quality loss rather than to directly reduce power (e.g. [23]).

A second classification of Approximate Computing literature can be done according to the level of abstraction. This is the organization that will be followed in the rest of this chapter to describe the details of some of the most successful techniques. In particular, we can identify: (i) Transistor/Gate-Level Hardware Techniques, (ii) Algorithm/Architecture-Level Hardware Techniques, (iii) Processor-Level Hardware Techniques, (iv) Cross-Level Hardware Techniques and (v) Software Techniques. In this discussion, we will mostly focus on hardware approaches, since they are currently the most diffused and effective ones. Software AC will be briefly described in Section 4.

## 3 Error/Quality Metrics

Before describing the details of the different Approximate Computing techniques, it is important to briefly discuss the matter of quality evaluation. AC is essentially a tradeoff between quality and some other metric, usually power or energy. To seek for optimality, both axes of the design space must be quantifiable. Well-estabilished models are available in literature for the power and energy consumption of digital circuits [36]. In this section, we focus on metrics to evaluate quality, and in particular to those suitable for hardware systems. Since quality is a function of the errors introduced by approximate computations, quality and error metrics are often used equivalently. In general, a quality metric can be obtained by taking the inverse of the corresponding error metric.

### 3.1 Error Rate and Error Significance

Quality is mostly a domain-specific metric, and there is certainly not a universal function valid for the optimization of any Approximate Computing target. However, the overall quality of a computing system is often the result of two concurring factors, which are commonly referred to as Error Rate and Error Significance (or Magnitude) [21].

Error Rate (ER) is the *rate* or *frequency* of occurrence of errors. It is most commonly measured as the number of erroneous outputs over the number of total outputs considered, both erroneous ($n_e$) and correct ($n_c$):

$$ER = \frac{n_e}{(n_e + n_c)} \tag{3}$$

Informally, ER can be considered as the *average probability* of occurrence of an error over the time interval considered for its evaluation [8].

Error Significance (ES) defines instead the *severity* of a single error. Since the notion of severity is strictly related to the mission of a system, the number of ES functions found in literature is much larger than for ER. In this context, we mention some of the most common ones. The simplest ES metric is the absolute value of the numerical difference ($D$) between the correct ($v_c$) and erroneous ($v_e$) values of a given output at a given time instant [48, 26]:

$$\|D\| = \|v_c - v_e\| \tag{4}$$

A commonly found alternative definition uses the squared difference $D^2$ instead. The Hamming Distance ($HD$), defined as the number of bits that differ between the binary representations of the correct and erroneous values, is often used for strategies targeting arithmetic circuits [26]:

$$HD = \sum_{bit(i)=1} (v_c \oplus v_e) \tag{5}$$

where $\oplus$ is the bitwise XOR operator. Finally, the *relative* difference ($RD$) normalizes the error with respect to the correspondent correct value:

$$RD = \left| \frac{v_c - v_e}{v_c} \right| \tag{6}$$

### 3.2 Composite Metrics and the Fail Small or Fail Rare Concept

In most Approximate Computing literature, some form of composite metric is used that encompasses the information of both ER and ES [29, 33, 49]. The simplest of such metrics is Rate-Significance ($RS$), defined as the product between the rate and the maximum significance over a set of vectors

[49]:

$$RS = ER \cdot \max(ES) \tag{7}$$

ER and ES are in turn defined according to one of Equations 3 and 4-6. In *RS*, the contributions of ER and ES can also be

assigned a different importance by means of appropriate weighting. A slightly more refined metric is the Mean Error Distance (*MED*), defined as the mean of $\|D\|$ (see Equation 4) over the considered set of $N$ output vectors [33]:

$$MED = \frac{1}{N}\sum_{i=1}^{N}\|v_{c,i} - v_{e,i}\| \tag{8}$$

If errors occurring in any time instant only depend on the value of the correct output in the same instant, *MED* can be equivalently defined as:

$$MED = \sum_{j=1}^{M}\|v_{c,j} - v_{e,j}\| \cdot \mathbb{P}(v_{c,j}) = \mathbb{E}\left[\|v_{c,j} - v_{e,j}\|\right] \tag{9}$$

where $M$ is the number of possible *values* assumed by the correct output and $\mathbb{P}$ and $\mathbb{E}$ stand for probability and expected value respectively [1]. The Normalized Error Distance (*NED*) corresponds to the *MED* normalized to the maximum possible error on a given output [33]:

$$NED = \frac{MED}{\max(\|D\|)} \tag{10}$$

With respect to the unnormalized version, this metric is better to compare the impact of errors on buses with different widths or using different data representations. Very similar to *MED* is the Mean Squared Error (*MSE*) [29]:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(v_{c,i} - v_{e,i})^2 \tag{11}$$

And again, under the assumption that errors only depend on "present" output values:

$$MSE = \mathbb{E}\left[(v_{c,j} - v_{e,j})^2\right] \tag{12}$$

Up to this point, we mentioned mainly general purpose metrics. However, many of the most used metrics in AC are domain-specific. The most popular in digital signal processing and communication applications is the Signal-to-Noise Ratio (SNR), defined as [1, 29, 23, 47]:

$$SNR = \frac{\sigma_s^2}{\sigma_w^2} \tag{13}$$

where $\sigma_s^2$ is the power of the *signal* component, i.e. of the set of error-free values carrying useful information and $\sigma_w^2$ is the power of the *noise* component, i.e. of the

[1] However, in many AC architectures errors are either affected by previous history of inputs and output values, or by external conditions, hence this definition is not valid

combination of all types of disturbances affecting the measured values. Therefore, when *SNR* is used, errors due to approximate computations are modeled as a *noise source*. One of the advantages of this metric is that it allows to combine approximations with disturbances caused by other factors, such as limited accuracy of analog components, external radiations and crosstalk in a communication systems, etc. In that case, assuming that computation inexactnesses and other sources of noise are uncorrelated, the definition of SNR becomes [47]:

$$SNR = \frac{\sigma_s^2}{\sigma_{noise}^2 + \sigma_{appr}^2} \tag{14}$$

This quantity is is often expressed in decibel (dB). Notice that SNR is strictly related to MSE, as the "noise" due to approximations is defined as:

$$N_{appr} = v_c - v_e \tag{15}$$

and hence its average power $\sigma_{appr}^2$ is exactly expressed by Equation 11. A further variant is the Peak Signal to Noise Ratio (PSNR), which considers the ratio between the maximum value assumed by the correct data ($\max(\|v_c\|)$ and the noise variance. Lastly, many literature works use higher level application-specific models of error/quality [8, 41, 38]. Since these metrics are so specific, they are too many to be enumerated here.

In general, because of the nature of most error resilient applications, combined metrics that consider both the rate and the significance of errors are often the most effective ones. For these applications, the relation between computational errors and measured quality is perfectly summarized by the rule of thumb of *"Fail Small or Fail Rare"* [8]: the quality of outputs produced by an inexact computing system acting over a set of inputs for a given period of time is acceptable if approximations are either small in significance or rare in time. This is an essential guideline to the design of Approximate Computing systems.

## 4 Approximate Computing Techniques

A schematic view of the main Approximate Computing techniques organized by abstraction level is shown in Figure 2. In this section we provide a brief overview of each group of approaches, aimed at conveying the general principles, and not intended as a complete survey. We focus mostly on solutions suitable for the optimization of DSP tasks commonly found in Smart Systems, and only mention the remaining ones. However, we reference the most important publications, in case the reader is interested. Solutions are examined with a bottom-up order, because low level techniques often constitute the building blocks of higher level ones.

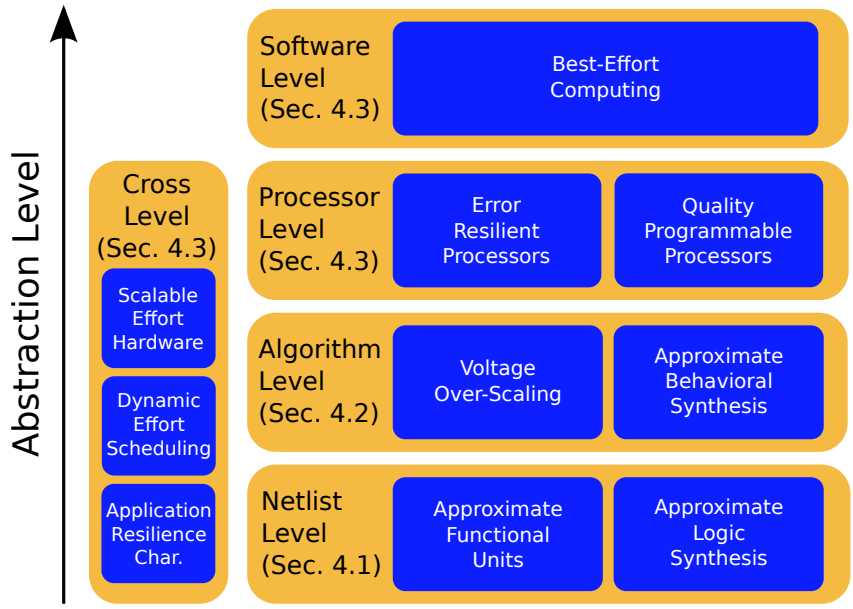


**Fig. 2** Overview of the main Approximate Computing techniques organized by abstraction level.

## *4.1 Transistor/Gate-Level Techniques*

In Figure 2 we use the term Netlist-Level techniques to identify all those approaches that operate on the transistor or gate-level netlist of circuits. Two main trends are identified in literature at this abstraction. Some researchers have developed approximate functional units, building inexact versions of the most common circuits found in digital datapaths. Others have focused on automation aspects, developing tools for the logic synthesis of approximate circuits.

### 4.1.1 Approximate Functional Units

The design of approximate versions of Functional Units (FU) is one of the most popular applications of AC. Efforts are mainly directed toward the most common datapath elements, namely *adders* and *multipliers*. A comprehensive survey of these techniques can be found in [21].

For what concerns approximate adders, two main categories of approaches can be identified. The first group modifies the structure of a single-bit Full-Adder (FA) and then generates multiple-bit adders with the modified FA. The second also starts from accurate architectures, but considers them from a coarser point of view, changing the way in which their building elements are connected.

In the category, one of the earliest designs is the so-called *Lower-Part-OR Adder* (LOA) [37], in which some FAs are replaced by a simple OR gate (see Figure 3). In [20, 55] the authors propose a similar idea, but in this case they generate approximate FAs with transistor-level simplifications. These solutions directly reduce area and power (by affecting leakage, internal capacitances and switching activity) and also decrease the delay of the FA, improving its performance, or alternatively enabling a more aggressive voltage scaling, to further reduce power consumption. Both goals are pursued while minimizing the truth table differences between the approximate versions and the original FA. Approximate FAs are used in the LSBs of a multi-bit adder, while MSBs are summed accurately (e.g. with a Carry Lookahead Adder) to preserve quality, as shown in Figure 3.

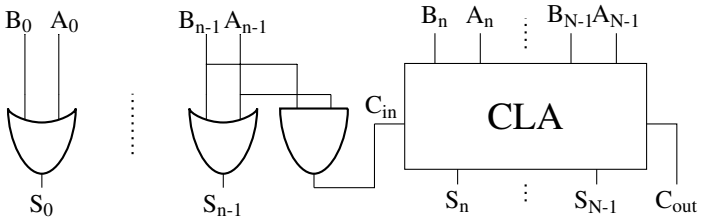


**Fig. 3** Lower-Part-OR Adder block diagram

The second group of approximate adders is mostly constituted by designs that *break the carry propagation chain*. Being the critical path, breaking it enables delay reduction at the expense of possible errors in the sum output. As mentioned, this can be exploited for voltage scaling. Moreover, the carry chain is also responsible for most switching activity in an adder, due to its frequent glitches. Therefore, reaching signal stability earlier also has a direct impact on power. Adders based on this principle can be found in [58, 57, 60, 59, 35, 25]. The authors of [54] use the same idea to design a *Variable Latency Speculative Adder*, able to identify errors due to the broken carry chain and compute the correct result, at the expense of additional latency. In [26] an *Accuracy Configurable Adder* is proposed, in which multiple error configurations can be set at runtime. The configurations that produce larger errors have a smaller delay, and hence allow to scale the voltage more.

Approximate multipliers have a comparatively small literature. Some solutions construct multipliers based on approximate adders [25, 35], while others modify existing array multiplier architectures removing or simplifying the FAs involved in MSBs calculations [37, 31]. A different approach is proposed in [30], where a multi-bit unsigned multiplier is constructed starting from an approximate elementary cell that performs 2-bit multiplication. It is shown that, by changing a single entry in the truth table of the 2x2 multiplier, hardware complexity can be reduced of almost 50% (see Figure 4). Moreover, the erroneous condition can be easily detected and

compensated. Thus, the architecture also supports an accurate mode of operation, used only for quality-critical multiplications, at the expense of additional power. Finally, [34] proposes an advanced design for an accuracy configurable multiplier, based on input pre-processing.

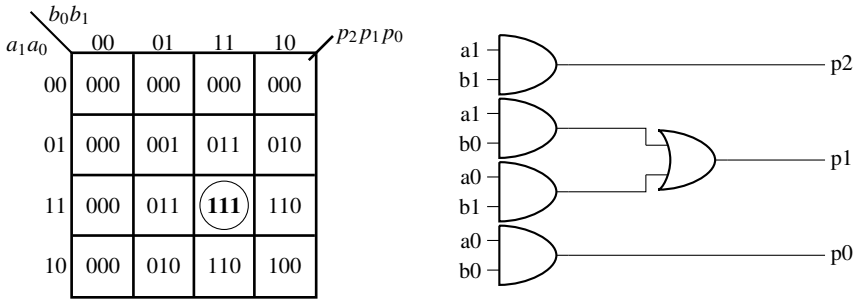


**Fig. 4** Modified 2x2 multiplication Karnaugh map and corresponding netlist.

### 4.1.2 Approximate Logic Synthesis

Approximate Logic Synthesis (ALS) deals with the generalization of the concepts introduced for the approximation of functional units, and applies them to *any* gate-level circuit. The rationale is that automation is indispensable to extend the approach to larger and more complex designs. Some of the first efforts in this direction are found in [48, 49], where methods for ALS are proposed targeting two-level and multi-level circuits respectively. These solutions tackle the ALS problem directly, using ad-hoc algorithms either to reduce the number of literals [48] or to simplify some nets as if they were redundant [49], based on a given quality constraint.

A fundamental work on ALS is the tool developed in [53], named Systematic methodology for Automatic Logic Synthesis of Approximate circuits (SALSA). SALSA introduces several novelties with respect to previous approaches, including the fact that ALS is transformed into a traditional synthesis problem, allowing to fully leverage the optimization capabilities of commercial tools. Moreover, it also unlinks the synthesis procedure from a specific quality metric, allowing the designer to set their preferred metric as an input, and making the tool much more flexible. The basic idea of SALSA is to construct the so called Quality Constraint Circuit (QCC), shown in Figure 5. The triangular block contains a used-defined logic-level implementation of the quality function, and its single-bit output $Q$ is at 1 if and only if the desired quality constraint is respected. SALSA considers the input conditions for which $Q$ is independent from a certain bit in the approximate circuit outputs

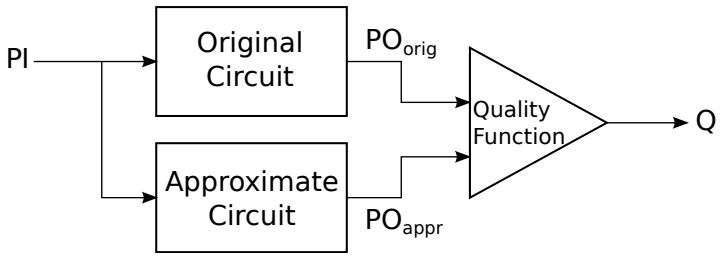


**Fig. 5** Quality Constraint Circuit (QCC) in SALSA.

$PO_{appr,i}$, i.e. the Observability Don't Cares (ODCs) of $Q$ with respect to $PO_{appr,i}$. Since those are the input combinations for which changing the value of $PO_{appr,i}$ does not violate the quality constraint, they are used to simplify the approximate circuit, reducing area, power, and delay. Both the individuation of ODCs and the corresponding simplifications are performed with commercial tools for classic logic synthesis. The process is repeated iteratively on all bits of $PO_{appr}$, progressively updating the approximate circuit in the QCC. Several optimizations are proposed to reduce the computational complexity of the algorithm. Recently, the SALSA methodology has been extended to consider also sequential circuits [45].

## *4.2 Algorithm/Architecture-Level Techniques*

Algorithm/Architecture-Level techniques for approximate hardware can be conceptually defined as those that concurrently optimize a signal processing algorithm and the hardware architecture that implements it. The considered targets can be as simple as a Multiply and ACcumulate (MAC) or as complex as a complete Fast Fourier Transform (FFT). Therefore, there is not a well defined distinction between this family of techniques and the previously mentioned Netlist-Level ones. Architecture-Level methods are often the most effective ones for the DSP applications found in Smart Systems.

### 4.2.1 Voltage Over Scaling

A large portion of Algorithm-Level techniques for Approximate Computing is based on the Voltage Over Scaling (VOS) principle [29]. In a sequential device, at a given clock frequency, the critical voltage $V_{dd,crit}$ can be defined as the smallest supply voltage that guarantees correct operation, i.e. absence of timing violations in the worst case operating conditions. VOS-based approaches let the device operate at voltages $V_{dd} < V_{dd,crit}$, exploiting the fact that timing errors only occur in corre-

spondence of certain combinations of inputs. Therefore, in the majority of cases [2], the circuit still produces the correct result. The clear advantage is that consumption is reduced significantly, due to the super-linear relation between supply voltage and both dynamic and leakage power [36].

The first embodiment of VOS to produce approximate circuits was originally proposed in [23], and is named Algorithmic Noise Tolerance (ANT). Its main idea is to let an arithmetic or DSP system work in VOS conditions, then leverage an error-free Error Control (EC) block to detect the occurrence of a timing violation and to *mitigate* its effects on the output quality (see Figure 7). Different types of ANT, in particular for what concerns the implementation of the EC block, are proposed in [24, 46, 47] and others. This technique is one of the most suitable for DSP operations, and is analyzed in details in Section 5.

VOS is considered from a different perspective in Significance Driven Design (SDD) [3, 40, 28, 22]. Instead of reducing errors due to VOS, this family of techniques tries to confine their occurrence to *non-significant* computations, i.e. computations that do not impact drastically the output quality. This goal is achieved acting both on the algorithm and on the corresponding hardware architecture. As an example, [3] applies SDD to a hardware implementation of the Discrete Cosine Transform (DCT). Some DCT coefficients are slightly altered (algorithmic modification) in order to improve hardware sharing (architectural modification). This sharing allows to compute high-energy DCT components, which are the most relevant to determine the visual characteristics of the output image, with a reduced delay, while long combinational paths are related to less significant components. Therefore, in VOS conditions, timing errors will only affect the latter.

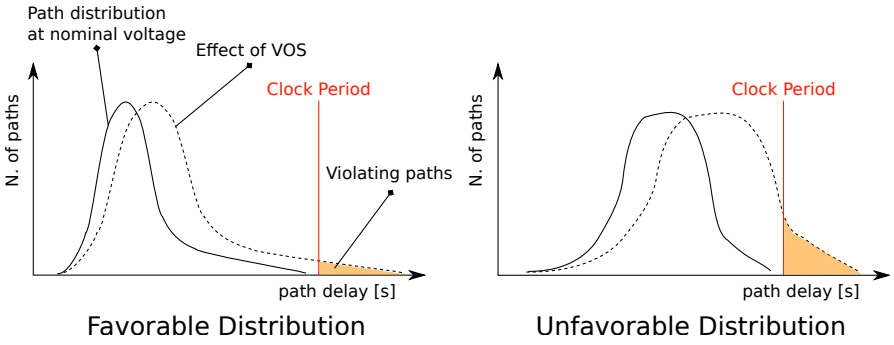


**Fig. 6** Impact of slack distribution on VOS-induced errors.

An important aspect of all VOS designs is that the probability of timing errors must be as small as possible, to limit the impact on quality. These errors are induced by combinations of inputs that activate long combinational paths in the architec-

[2] The precise rate of timing errors depends on circuit topology and on the characteristics of input data. Moreover, it can be modified with optimization techniques (e.g. gate sizing).

ture. Therefore, the distribution of path lengths influences significantly the behavior of a system in VOS conditions; the ideal condition is to have few long paths and many short ones, as shown in Figure 6. Two different works consider the problem of easing timing degradation of circuits in VOS [27, 39]. The former proposes a *VOS-friendly* gate sizing algorithm, that changes the path distribution to produce a gradual degradation of quality and enable aggressive over-scaling. In [39] the focus is on architectural modifications of some *meta-functions*, i.e. recurring operations common to different error resilient applications, aimed at changing their path distribution to reduce the error rate and significance at a given voltage.

Other literature involves VOS as key feature for power reduction. In [16, 14, 18] the authors proposed techniques and devices (such as the so-called Razor flip-flops) that aim at detecting and avoiding VOS-induced errors. Error avoidance is accomplished allocating additional clock cycles to let the longest combinational paths stabilize, for instance via clock gating or pipeline stalling. These techniques are very popular, but since they *avoid* errors rather than accepting them, they do not strictly belong to the Approximate Computing domain.

### 4.2.2 Approximate Behavioral Synthesis

Recent works have addressed the approximate synthesis of designs described at the behavioral level, i.e. starting from a functional model of the system behavior (typically graph-based) rather than from a net of logic gates. The main effort is documented in [41], where the authors propose a tool named Automated Behavioral Approximate CircUit Synthesis (ABACUS), that produces multiple approximate versions of a circuit starting from an original accurate model in behavioral HDL. ABACUS operates by transforming the HDL in an Abstract Syntax Tree (AST) model, then modifying the AST with a series of *operators*, each of which introduces some approximations while preserving syntactic correctness. Operators include LSBs-truncation, variable to constant substitution, loop unrolling, etc. Moreover, ABACUS can also use models of approximate functional units like those presented in Section 4.1.1. Approximated ASTs are then transformed back to HDL and processed with a standard synthesis and simulation flow to evaluate their cost (in terms of area and power) and their accuracy. To avoid an exponential number of synthesis and simulation phases, ABACUS does not apply all approximating operands to all feasible locations in the AST. Instead, in each iteration it generates a number of approximated ASTs choosing operands and locations at random, then greedily selects the *best* solution as the starting point for subsequent iterations. Optimality is expressed by a fitness function that weights cost and accuracy components. The entire process is then repeated for a fixed number of iterations. A hard threshold on accuracy is also set, so that solutions that affect quality too much are not synthesized.

### *4.3 Other Techniques*

In this section we present Approximate Computing techniques that are less interesting for the context of DSP in Smart Systems, but nevertheless need to be mentioned to provide a complete view of the state of the art (see Figure 2).

**Processor-Level Techniques.** In the domain of programmable processors, AC solutions are mostly based on the acceptance of errors due to process variability, aging, temperature, external radiation, etc. Rather than hardening the entire processor against these errors, only some cores or units are made reliable with fault tolerance mechanisms. Those units manage the control flow and execute critical operations, while error resilient computations are offloaded to unreliable hardware. The total area and power costs related to hardening are thus significantly reduced [32, 56]. A different approach is found in [52], where the authors propose a vector processor in which the "quality-level" of each operation can be set at the software level, thanks to a specific Instruction Set Architecture. Different quality/power levels are obtained in hardware combining reduction of operands bit-widths with power and clock gating.

**Cross-Level Techniques.** Another branch of research has focused on methodological aspects related to AC, such as the automatic assessment of error resilience, and the combination (or *synergy*) of techniques at multiple abstraction levels. The first objective has produced the so-called *Application Resilience Characterization* (ARC) framework [8], able to individuate the error resilient computational kernels in an application and evaluate the impact of different AC techniques (e.g. approximate functional units, VOS meta-functions, etc.) on them. The ARC framework is used in [9] to generate *Scalable Effort Hardware* architectures, that combine AC techniques at different abstraction levels to obtain greater power and area savings. A further improvement is found in [10], where a technique called *Dynamic Effort Scheduling* is proposed to set the global quality level at runtime, acting on knobs that affect the different AC techniques involved.

**Software Techniques.** Most literature on AC at the software level exploits the previously mentioned *iterative refinement* (also found as *iterative convergence*) computational pattern. A technique called *Best Effort Computing* has been devised for these algorithms in [38, 5], based on principles similar to those of best effort communication in the networking stack. An algorithm is divided in *guaranteed computations*, which are fundamental for the final result, and *optional computations*, which can tolerate errors without a significant impact on output quality. The relaxed correctness requirement on the latter group is exploited in the Operating System to perform different types of optimizations, aimed both at performance improvement and cost containment. For example, optional computations can be skipped entirely, to avoid exceeding the power budget, or synchronization between threads can be relaxed to improve performance.

# 5 Algorithmic Noise Tolerance for DSP

Many Smart Systems applications involve some form of Digital Signal Processing (DSP). Depending on the complexity of the task, designers can choose different types of computing devices to implement it. In some cases, a programmable processor is mandatory, while in others the entire application can be implemented with ASIC hardware. Even in the first setting, however, recent trends in industry indicate that computing platforms are becoming increasingly *heterogeneous*, combining general purpose processors with domain-specific hardware *accelerators* [12]. The need for accelerators is driven by the double objective of improving performance while reducing energy consumption; being less flexible, dedicated hardware can be designed to be faster and more efficient than general purpose processors. Moreover, for large sells volumes, heterogeneity also reduces costs, as a given performance level is reached with simpler hardware, thanks to specialization.

In summary, complex Smart Systems are likely to include some form of specialized hardware module, which is often in charge of the most computationally intensive and energy consuming processing tasks. As mentioned in Section 1, DSP applications involved in Smart Systems frequently exhibit error resilience. Consequently, Approximate Computing configures as a potential source for further energy-driven optimization. Smart Systems specialized ASICs are often more complex than a single arithmetic module; examples of frequently found elements include FIR and IIR filters, FFT or DCT processors, etc. Multiple of such elements are normally integrated in a single IC. The most suitable Approximate Computing techniques for this kind of complexity are those working at the Algorithm/Architecture Level (see Section 4.2), because their high-level view allows to optimize the hardware system in its entirety, rather than each functional unit separately. In particular, *Algorithmic Noise Tolerance* (ANT) for the mitigation of errors due to *Voltage Over Scaling* (VOS) is currently the most mature and well-defined approach for approximating computation in specialized hardware modules of this nature [23].

## *5.1 ANT Overview*

The most general scheme of an ANT-based architecture is shown in Figure 7. The gray box corresponds to the hardware that performs the main functionality (e.g. a digital filter) and is referred to as *Main DSP* (MDSP) block. Its primary inputs and outputs are labeled $X$ and $Y_M$ respectively. The MDSP operates in VOS conditions ($V_{dd} < V_{dd,crit}$), therefore it consumes significantly less dynamic ($P_{dyn}$) and leakage ($P_{leak}$) power than in nominal conditions, according to the well known power dependencies for CMOS circuits:

$$P_{dyn} \sim f_{clk} C_L V_{dd}^2 \tag{16}$$

$$P_{leak} \sim I_{leak} V_{dd} \tag{17}$$

where $f_{clk}$ is the clock frequency, $C_L$ is the total load capacitance and $I_{leak}$ is the leakage current. However, the logic in the MDSP is slowed down by the reduced supply voltage; in fact, the delay of combinational CMOS gates depends on voltage according to the following equation:

$$\tau_d = \frac{V_{dd}}{\beta(V_{dd} - V_{th})^{\alpha}}, \ \alpha > 1 \tag{18}$$

In ANT, while $V_{dd}$ is reduced below the critical value, the clock frequency ($f_{clk}$) used to drive sequential memory elements such as Flip-Flops is not modified, in order to preserve the same performance (latency or throughput) as in nominal conditions. Consequently, the MDSP is subject to input-dependent timing errors; when a long combinational path is activated by a certain input combination, the logic can fail to produce the correct, stable value in time before the following clock edge. To cope with these errors, the MDSP is coupled with an *Error Control* (EC) block, delimited by the dashed line in Figure 7. The purpose of the EC block is to detect the occurrence of an error due to a timing violation, and limit its effects providing a suitable *approximation* of the correct result at the global outputs of the ANT architecture ($Y$). The combination of MDSP and EC block is sometimes referred to as *soft DSP* [23].

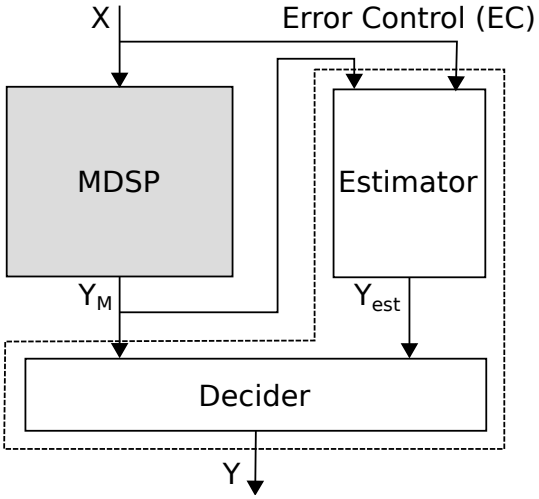


**Fig. 7** General scheme of an ANT architecture.

Thanks to the fact that the EC block does not need to *correct* an error, as in a fault tolerant system, but only to approximate the corresponding correct value, its hardware complexity is limited. This conveniently reduces the overheads in terms of additional silicon area and power; clearly, in order to be useful, the complete ANT system in VOS must consume less than the MDSP in nominal supply voltage

conditions. Moreover, relaxing the objective of complete error correction also allows to make the EC block timing compliant in VOS. This is a necessary requirement, because it guarantees that the error mitigation performed in the EC block is not affected by errors itself.

The EC block is composed of two main elements, an *Estimator* and a *Decider*. The Estimator is the component that produces an approximation of the correct MDSP output, called $Y_{est}$. In general, the estimation is obtained using information on MDSP inputs and outputs, although the most popular ANT implementations only use either one or the other. The Decider, instead, is the module in charge of selecting the output of the MDSP or that of the Estimator, depending on the possible occurrence of a timing error. Notice that both MDSP and Estimator outputs must be *latched* before being fed to the Decider. Otherwise, the decision logic might end up in the critical path, becoming prone to delay errors, and rendering the entire ANT architecture useless. Latching, however, increases of one clock cycle the latency of the system, with respect to the original MDSP.

### 5.1.1 Decision Scheme

ANT is based on two important assumptions on the nature of timing errors due to VOS. Firstly, timing violations must be *rare in time*, so that in most of the cases the MDSP produces the correct output, and that value can be used as global output of the system. This allows to maintain an average output quality which is comparable to that of the MDSP in nominal voltage conditions. The validity of this assumption depends on the selected VOS voltage, on the input data sequence, and on the distribution of timing paths in the MDSP. However, techniques such as slack redistribution can be leveraged to enforce it [27]. Secondly, the errors produced in correspondence of a timing violation must be *large in magnitude*, so that they are easily detectable by the Decider. This condition is verified for the large majority of DSP hardware systems. In fact, these systems are composed of a sequence of elementary functional units such as adders and multipliers, for which the critical timing paths are relative to the computation of the MSBs of result. Therefore, when an error occurs, it most likely affects drastically the output.

The fact that errors are large in magnitude allows to select between MDSP and Estimator outputs using as discriminating factor the absolute value difference between the two. The Estimator is designed to provide a good approximation of the correct output. Therefore, if at a given instant its output is significantly different from that of the MDSP, it is safe to assume that the latter is subject to a VOS-induced error, and vice versa. In summary, the decision scheme is the following:

$$y[n] = \begin{cases} y_M[n], & \text{if } |y_M[n] - y_{est}[n]| \leq T_h \\ y_{est}[n], & \text{if } |y_M[n] - y_{est}[n]| > T_h \end{cases} \tag{19}$$

where, $y[n]$, $y_M[n]$ and $y_{est}[n]$ indicate the values of signals $Y$, $Y_M$ and $Y_{est}$ at time $n$ respectively [3]. It is important to highlight that this decision scheme effectively mitigates also errors not caused by VOS (e.g. Deep Sub-Micron noise, external radiation, aging, etc.), whose effects on $Y_M$ are significant, as long as the EC block is not affected by the same errors.

### 5.1.2 Quality Metric and Problem Formulation

The main targets of ANT are DSP systems, such as digital filters or FFT accelerators. These systems often process analog data produced by sensors and converted to the digital domain, or received from a communication channel. Both types of input are affected by noise. Therefore, in nominal supply voltage conditions (i.e. in absence of VOS errors) the output of the MDSP can be expressed as:

$$y_{M,0}[n] = s[n] + w[n] \tag{20}$$

where $s[n]$ is the signal component, and $w[n]$ is a generic noise component, that may be due to different physical sources. A particularly suitable metric for assessing the processing quality of this type of systems is the Signal to Noise Ratio (SNR), defined in Equation 13. The ANT strategy saves power accepting to introduce errors due to timing violations, which are only partially corrected by the EC block. The residual errors on $Y$ can be modeled as an additional source of noise, as explained in Section 3. The global output quality of the ANT system can thus be measured (in decibel) with the following expression:

$$SNR = 10\log\left(\frac{\sigma_s^2}{\sigma_w^2 + \sigma_{err}^2}\right) \tag{21}$$

where $\sigma_{err}^2$ (assumed uncorrelated with $\sigma_w^2$) is the power of the additional errors introduced by ANT:

$$\sigma_{err}^2 = \mathbb{E}[(y_{M,0}[n] - y[n])^2] \tag{22}$$

$y[n]$ is defined as in Equation 19, so it takes into account the reduction of VOS-induced errors thanks to the EC block. Clearly, the SNR is not the only possible quality metric for ANT, but given the nature of the considered MDSPs, it is largely the most used. A typical goal is to design the EC block so that errors due to VOS timing violations are negligible with respect to external noise ($\sigma_{err}^2 << \sigma_w^2$).

In general, the problem of designing an optimal ANT architecture for a given MDSP system can be defined as: *finding the VOS voltage and feasible EC block implementation that minimize total power consumption, including EC block overheads, under a minimum quality constraint, and so that the entire ANT system consumes less than the single MDSP in nominal conditions.* In formal terms:

[3] In general, we use capital letters to refer to a signal, and lowercase ones to indicate its value at a given instant in time.

$$\begin{cases} \min[P_{ant}(V_{vos}, \mathbf{k})] \\ Q(V_{vos}, \mathbf{k}) > Q_{des} \\ P_{ant}(V_{vos}, \mathbf{k}) < P_{mdsp}(V_{nom}) \end{cases} \tag{23}$$

In Equations 23, $Q$ represents a generic quality metric (e.g. *SNR*) and $Q_{des}$ is the minimum acceptable quality for the target application. $V_{vos}$ and $V_{nom}$ are respectively the VOS and nominal (i.e. error free) supply voltages. $P_{mdsp}$ is the power of the MDSP, and only depends on voltage, since the MDSP hardware is fixed. On the contrary, the power of the entire ANT system ($P_{ant}$) depends both on the VOS operating point and on the generic vector of parameters $\mathbf{k}$ which determines the implementation of the EC block:

$$P_{ant}(V_{vos}, \mathbf{k}) = P_{mdsp}(V_{vos}) + P_{ec}(V_{vos}, \mathbf{k}) \tag{24}$$

Depending on how the EC block is designed, $\mathbf{k}$ corresponds to a different set of parameters, as explained in the following.

Two main families of ANT architectures have been proposed in literature: *Prediction-Based* ANT [23, 24] and *Reduced Precision Redundancy* (RPR) ANT [46, 47]. Hybrid approaches have also been studied in [47]. In the next sections, we briefly describe the features of the existing types of ANT, which mainly differ in the implementation of the Estimator. Then, we concentrate on RPR ANT and provide a complete example of its application.

## *5.2 Prediction-Based ANT*

In Prediction-Based ANT the Estimator is a linear forward predictor, that provides an approximation ($Y_p$) of the correct output based on the recent history of the system. In particular, the estimate is based on a linear combination of the last $N_p$ outputs produced by the MDSP:

$$y_p[n] = \sum_{k=1}^{N_p} h_p[k] y[n-k] \tag{25}$$

The weights $h_p[k]$ are chosen to minimize the Mean Squared Error (MSE) of the predictor $\mathbb{E}[e_p^2[n]]$ in absence of VOS-induced timing violations (i.e. when $e_p[n] = \|y_{M,0}[n] - y_p[n]\|$). The way in which these coefficients are obtained is discussed in detail in [44].

The hardware that implements an Estimator with the transfer function of Equation 25 is shown in Figure 8, where the boxes labeled with a "D" are delay elements, i.e. Flip-Flops. The figure also reports one of the possible decision schemes for this architecture, which follows the same principle of Equation 19. The Decider computes internally the prediction error and compares it to a threshold $T_h$. Given that prediction coefficients are computed to minimize the approximation error in absence of VOS, if $e_p[n]$ is large it is assumed that a timing violation has occurred.

In that case, the predictor output is used as approximation of the correct value. If instead the error is less than $T_h$, the MDSP output is selected. Other more complex decision schemes have been proposed in [24]; they are not reported here for brevity.

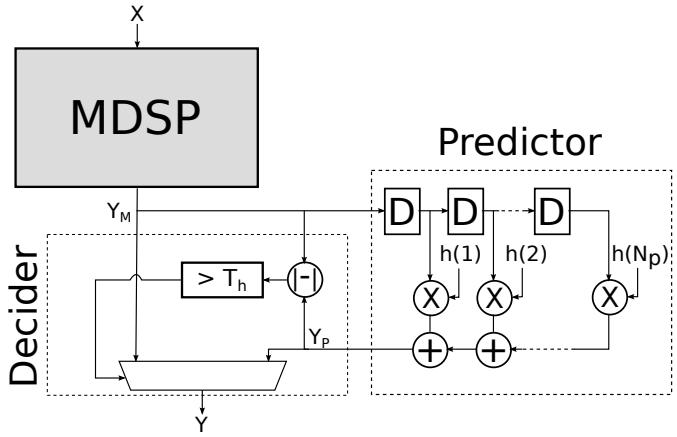


**Fig. 8** Prediction-Based ANT architecture.

The overheads in terms of cost and power of the Estimator in Prediction-Based ANT depend on the length of the prediction window $N_p$. The simplest version uses a single past value of the MDSP output; in this case, it is referred to as *Difference-Based* Estimator, since the value that is compared with $T_h$ is simply the (possibly weighted) difference between the current and previous values of $Y_M$. Clearly, a longer prediction window allows to obtain a more accurate estimate.

Setting the threshold $T_h$ for the Decider is one of the main design issues of this type of architecture. In [24] the authors propose to use a multiple of the standard deviation of the prediction error ($T_h = k\sigma_{e_p}$). However, notice that this does not guarantee that the Decider always selects the MDSP output when the system is free of VOS-induced errors. If the MDSP output varies suddenly, causing a large error in the predictor, the Decider may assume a delay error has occurred when in reality it has not. In general, Prediction-Based ANT systems have good detection capabilities when the outputs of the MDSP exhibit a strong *temporal correlation*, that is when subsequent values of $Y_M$ are correlated with each other.

Another limitation of this scheme is that the predictor output is biased by errors occurred in the previous $N_p$ instants. In this time window, if another error occurs, detection and output estimation capabilities of the ANT architecture become much less accurate. In particular, performance degrade significantly when *error bursts* happen at the MDSP output.

In summary, despite its simplicity and relatively small overhead, Prediction-based ANT is limited to few MDSP, such as narrowband low-pass filters, that exhibit strong output correlation. Moreover, it does not allow to aggressively scale the sup-

ply voltage $V_{vos}$, since its effectiveness is limited to situations in which delay errors do not accumulate in the prediction window.

### *5.3 Reduced-Precision Redundancy ANT*

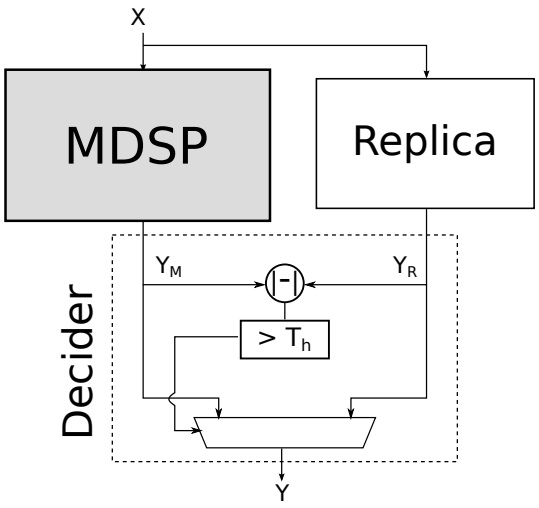


**Fig. 9** Reduced Precision Redundancy ANT architecture.

RPR ANT uses an alternative scheme for the Error Control block; the entire architecture is depicted in Figure 9. In this case, the Estimator is a *low-complexity replica* of the MDSP, obtained reducing the precision of internal operations. The replica receives the same inputs as the MDSP at every clock cycle, and produces a corresponding approximation of the main block results. Thanks to the smaller and thus faster hardware operands involved, the replica can be designed to be timing compliant in VOS conditions. Its internal logic is designed so that approximate results differ from accurate ones in the LSBs; therefore, the approximation error has a small magnitude. This allows the Decider to select between MDSP and replica outputs based on the absolute value of their difference, as explained for the Prediction-Based architecture.

If the entire output images of MDSP and replica, i.e. the sets of output values produced by any sequence of inputs is known, the decision threshold $T_h$ can be computed as:

$$T_h = \max_{\forall input} \|y_{M,0}[n] - y_R[n]\| \tag{26}$$

where $y_{M,0}[n]$ and $y_R[n]$ are the error-free MDSP outputs and the Replica outputs at time $n$ respectively. Equation 26 ensures that in absence of VOS-induced errors, the MDSP output is always selected by the Decider. However, the entire images of $Y_M$ and $Y_R$ are easy to compute only for few simple MDSPs. Therefore, statistical thresholds similar to those proposed for Prediction-Based ANT might be necessary also in this case.

In RPR ANT the tradeoff between quality and EC block overheads is explored varying the number of bits ($B_r$) used for the internal operands of the replica (which corresponds to the generic parameter **k** in Equation 23). A larger replica consumes more power, both switching and leakage, and has a larger silicon cost. However, it produces a more accurate approximation of the MDSP output, and therefore it allows to maintain a higher quality at the global output $Y$. As explained in Section 5.1.2, the goal is typically to make errors due to VOS negligible with respect to external noise.

The advantage of RPR with respect to Prediction-based schemes is its higher flexibility. In fact, since the output estimate is computed based only on the present value of inputs, this architecture behaves in an equivalent manner regardless of the correlation between subsequent outputs in the MDSP [4]. Also, the error mitigation capabilities of RPR ANT are not affected by recent errors, except for cases in which MDSP and replica have internal feedback networks. The drawback of RPR is that, especially for complex MDSPs, replicas have larger overheads with respect to linear predictors.

## *5.4 Hybrid ANT*

Prediction-based ANT performance degrade when two timing errors occur too close in time, effectively limiting the minimum $V_{vos}$ at which the system can be operated. On the other hand, RPR ANT has a larger power overhead. An hybrid scheme, proposed in Figure 10 can be used to overcome these limitations, exploiting the best features of both architectures.

In hybrid ANT, the EC block is equipped with two different Estimators: a linear predictor and a reduced-precision replica, designed as in Sections 5.2 and 5.3 respectively. An additional Finite State Machine (FSM) selects between two operating modes. In normal conditions, the approximation of the correct output is obtained with the linear predictor ($Y_{est} = Y_P$) whereas the replica is turned off via power/clock gating, eliminating most of its overheads. When an error is sensed by the Decider ($\|y_{est}[n] - y_M[n]\| > T_h$) the FSM turns on the replica and switches the leftmost multiplexer of Figure 10, so that $Y_{est}$ becomes equal to $Y_R$. This preserves the error detection and mitigation capabilities of the EC block, which is not affected by the accumulation of errors in the Predictor. The system remains in "RPR-mode" until no errors are detected for a given number of clock cycles. Then, it returns to normal

[4] Different sequences of data may cause a different VOS-error rate, which results in a different quality, but they have no influence on the error mitigation capabilities of the architecture.

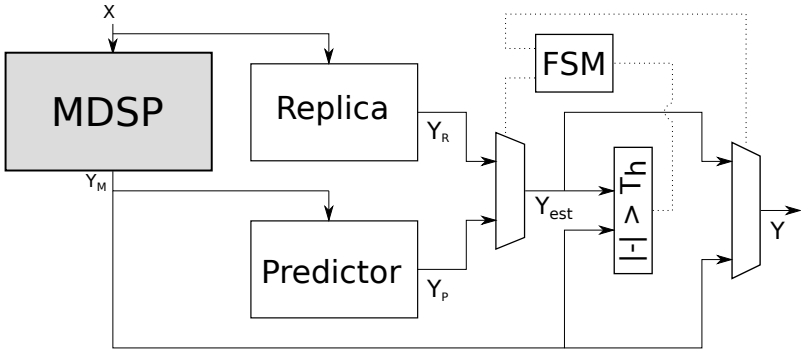


**Fig. 10** Hybrid ANT architecture.

operation. In [47] the authors show that hybrid ANT outperforms both RPR and Prediction-Based architectures in terms of output quality, for the particular case of a digital filter MDSP. The major drawback of this scheme is clearly the larger silicon area required.

## 6 A Case Study: RPR ANT applied to a Finite Impulse Response Filter

To conclude the discussion on Approximate Computing for digital processing in Smart Systems, this section provides a detailed example of application of ANT, which as explained in Section 5, is one of the most promising approaches in this context. With a similar rationale, the hardware architecture that we optimize is a digital filter, due to its wide range of possible uses in Smart Systems applications (e.g. communication, multimedia, control, etc.) [44]. We select a 16-th order lowpass Finite Impulse Response (FIR) filter, with a cutoff frequency of $w_c = 0.1\pi rad/s$.

The starting point for our optimization is the Verilog code that models the FIR filter at the Register-Transfer Level (RTL). The particular implementation considered in this section uses the so-called Direct Form architecture, and its input ($U$) and output ($Y$) ports are 12-bit and 24-bit wide respectively. A simplified block diagram of the FIR filter is shown in Figure 11. Its transfer function is:

$$y[n] = \sum_{i=0}^{15} u[n-i] \cdot w[i] \tag{27}$$

We adopt Reduced-Precision Redundancy ANT as an optimization technique, because of its higher flexibility with respect to prediction-based approaches. In particular, we have anticipated that prediction-based ANT is effective only if the filter has a narrowband transfer function. RPR, on the contrary, can be exploited indepen-

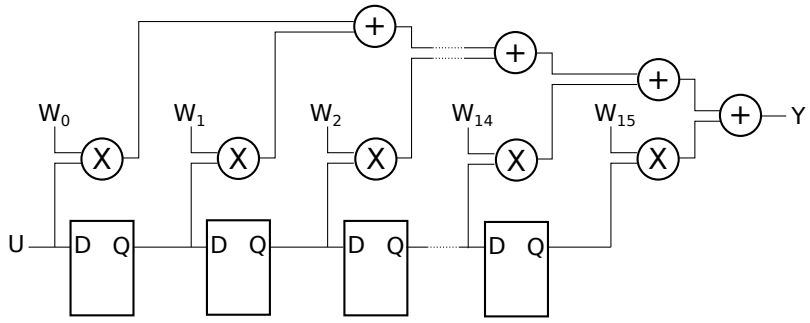


**Fig. 11** Direct Form FIR filter block diagram.

dently from the parameters of the FIR (tap-length and coefficient values), although the resulting performance may vary.

The objective in RPR ANT is to find the minimum VOS supply voltage and number of bits in the replica operands that satisfy a given quality constraint (as in Equations 23). Previous approaches [46, 47], have provided a solid theoretical foundation for this type of optimization. However, their search for the optimal parameters is based on simplified and often unrealistic assumptions.

The most important limitation is related to the method of evaluation of VOS-induced errors and of their effects on the output quality. Specifically, the variance of errors introduced by ANT, expressed by Equation 22, is a fundamental element for the computation of the SNR and many other quality metrics (e.g. MSE). This quantity depends both on the accuracy of approximations provided by the EC block, and on the rate at which the replica output is selected, over the entire time window considered.

In [46] the rate of VOS-induced errors is evaluated as the percentage of static timing paths in the MDSP that have a negative slack at a given $V_{vos}$. Since path activation depends on the sequence of input vectors, this method implicitly assumes that *all pairs of inputs* are equally probable. However, in most applications, the probability of occurrence of input vectors is not uniformly distributed (e.g. in arithmetic cores often the operands have a Gaussian distribution with zero mean) let alone that of input pairs. The assumption of uniformly probable path activation biases also the estimation of error significance, since every possible VOS-induced error is considered equally likely. This, in turn, affects the evaluation of which errors are detected and mitigated by the EC block, and of the corresponding approximation accuracy. All these aspects influence significantly Equation 22, and might produce misleading results (both overly optimistic or pessimistic) on the number of replica operand bits required to ensure a minimum quality at a given $V_{vos}$.

Another important aspect that is partially neglected by literature is the effect of voltage scaling on the replica. We have mentioned that the replica must be timing compliant in VOS, in order to let the ANT system function correctly. However, this is not trivially achieved, and depends strongly on the topology of the circuit. It does

not suffice to assume that the reduction in critical delay of the replica is proportional to the number of removed bits in its operands. Timing compliance must be checked accurately, and can also be enforced, for instance with gate re-sizing. The latter technique, however, may increase the power overhead and reduce the total savings of the ANT system. All these details cannot be taken into account by optimization methods such as the one proposed in [46], which use abstract high level models of MDSP and replica.

To cope with these issues, we have developed a tool for the automatic generation of RPR ANT architectures on top of existing MDSPs [**?**]. Our tool is based on *timing simulations* with input stimuli taken from the real application domain of the MDSP, which are used to evaluate the rate and impact of VOS-induced errors accurately. These simulations leverage a set of *standard-cell library characterizations*, that model the target technology (usually CMOS) at different voltage points. This approach allows to consider non-uniform input/input-pairs distributions as well as secondary technological effects that have an impact on timing violations.

Moreover, the *replica implementation is obtained automatically from the MDSP netlist*, removing some inputs and propagating the simplifications in the internal logic. To make it timing compliant, the replica is then re-synthesized with state of the art commercial tools, changing gate sizing if needed. After synthesis, the power overheads are checked, and if they produce a negative power-saving for the entire ANT system, the configuration is discarded. This procedure is repeated in a clever way, minimizing the number of synthesis and simulations required, until the optimal configuration in terms of VOS supply voltage and replica bits is found.

Using this tool, we have studied the application of RPR ANT to the FIR filter from an engineering perspective, gathering detailed results that were not obtainable with the methods originally proposed by literature. In the following sections, we first describe the experimental conditions, then we present some results that confirm the improvements achieved by our tool in terms of accuracy. In particular, we focus on the effects of input distributions and timing correlation on VOS-induced errors. To conclude, we show how the tool can be used to explore the power versus quality tradeoff.

## *6.1 Experimental conditions*

We performed experiments on the FIR filter targeting a 45nm CMOS standard cell technology. The tool has been instructed to find the optimal VOS voltage in the range between 1.1V and 0.55V, with a step of 0.05V. The clock frequency was set to 250MHz. This value allows to meet timing constraints exactly (with positive slack $\sim 100ps$) for the single MDSP at $V_{dd} = 1.1V$, which we considered as the *nominal* voltage condition. For the synthesis of MDSP and EC block we used Synopsys Design Compiler F-2011.09, while for timing and power analysis we used Synopsys PrimeTime Suite F-2011.12, and for behavioral and timing simulations Mentor Modelsim SE 6.4a. As quality metric, we used the Signal-to-Noise Ratio (SNR),

defined as in Equation 21 for all experiments. We measured the total power savings and area overheads of the RPR ANT architecture with the following equations:

$$P_{sav} = \left(1 - \frac{P_{rpr}(V_{vos}, B_r)}{P_{mdsp}(V_{nom})}\right) \cdot 100\% \tag{28}$$

$$A_{ovr} = \left(\frac{A_{rpr}(V_{vos}, B_r)}{A_{mdsp}} - 1\right) \cdot 100\% \tag{29}$$

where $V_{nom}$, $V_{vos}$, $P_{mdsp}$ and $P_{rpr}$ are defined as in Section 5, $A_{mdsp}$ and $A_{rpr}$ represent the silicon area of the MDSP and replica respectively, and $B_r$ is the number of bits in the replica inputs, which in turn determines the widths of internal operands.

## *6.2 Impact of input distribution and correlation*

The first set of experiments aims at highlighting the strong dependence of the distribution and temporal sequence of inputs on the rate of VOS-induced errors, and consequently on the performance of RPR ANT. For this purpose, we ran our optimization tool on the FIR filter four times, with identical settings, but using four different sets of input vectors (S1-S4) in timing simulations. The first two sets contain uniformly distributed vectors. In S1, vectors are in a random sequence, while in S2 they are *sorted* numerically, in order to enforce a strong temporal correlation in the MSBs (with a large probability, only the LSBs vary between two consecutive vectors). S3 and S4 instead represent more realistic inputs for a digital filter. They contain two full swing sinusoids, respectively at $w_{s3} = 0.01\pi rad/s$, that is one tenth of the cutoff frequency, and $w_{s4} = w_c = 0.1\pi rad/s$. In all cases, we suppose that input data is noiseless; we measure quality in terms of SNR, but considering only VOS-induced errors, and we arbitrarily set the minimum quality requirement to 25dB. Moreover, we limit our analysis to those VOS voltages that produce an Error Rate in the MDSP (computed as number of timing violations over total number of vectors) greater than 0% and smaller than 20%.

Figure 12 reports the maximum power savings obtained at each $V_{vos}$ point with the mentioned constraints. The graph clearly shows the impact of different input vectors on the effectiveness of RPR ANT. Two phenomena are particularly important. First of all, at a given $V_{vos}$ the error rates are significantly different depending on the input set. As a numerical example, at $V_{vos} = 0.7V$, the error rates in S2 and S4 are 0.05% and 12% respectively. Consequently, depending on the input set, the replica output is selected more or less frequently, and hence a different value of $B_r$ is required to achieve the desired quality. At $V_{vos} = 0.7V$, the errors for S2 are so rare that a replica that only considers the 2 MSBs of the filter input for its computations is sufficient; in the case of S4, instead, 6 bits are required to achieve 25dB of SNR. At the end, the noticeable product of these phenomena is that the power savings in the two cases differ by almost 40%. Obviously, the input sets used in this section constitute an extreme example. However, they serve as a mean to demon-

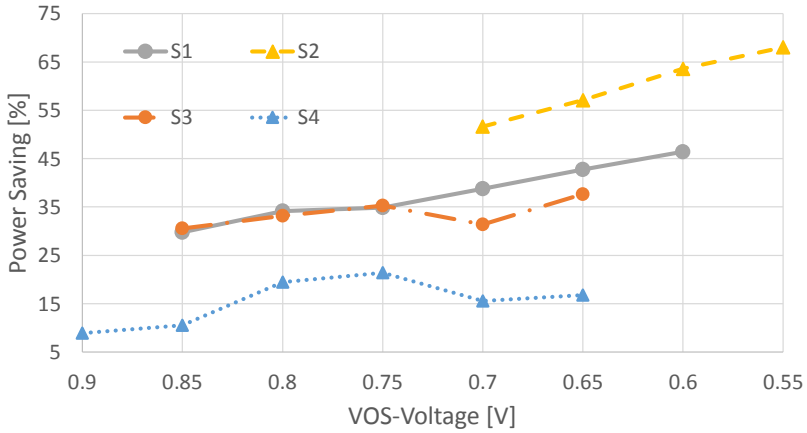


**Fig. 12** RPR ANT Power Saving vs. $V_{vos}$ for different input sets.

strate that simplified assumptions on VOS-induced errors as those in [47] may lead to unrealistic results.

Notice that Figure 12 also indirectly shows one of the features that make ANT effective. We mentioned that the positive slack of the MDSP in nominal conditions is only 100ps, meaning that the clock frequency is appropriate (not too small), according to a classic worst-case design paradigm. However, we discovered that for voltages between $1.05V$ and $0.9V$, none of the input sets produces a timing violation in the MDSP, i.e. the error rate remains 0%. This means that the critical timing path of the FIR, which corresponds to the cascade of carry chains in all adders and multipliers (see Figure 11), is never excited by the provided stimuli. This is an evident reason why relaxing the 100% accuracy constraint, working in VOS, and limiting the impact of the rare errors with ANT is a very effective way to reduce power.

### *6.3 Power versus quality tradeoff*

In this section, we demonstrate a more realistic use of our tool for the optimization of the FIR filter, showing how it can be leveraged to explore the quality versus power tradeoff. To do so, we select a popular application of hardware FIRs, which has been also considered in [47], that is the low-pass filtering at the receiver of a QPSK communication system. In particular, we instruct our optimization tool to perform simulations with a set of inputs generated from a MATLAB model of the IEEE 802.11g WiFi standard, which includes QPSK among the possible modulations. We assume that the inputs to the FIR are affected by Additive White Gaussian Noise (AWGN) due to channel interferences. As explained in [47], a minimum SNR at the filter output of 21.5dB ensures a bit error rate of $10^{-7}$. Therefore, we add AWGN to the input set, so that the output SNR of the filter is approximately at such value. Then, we let

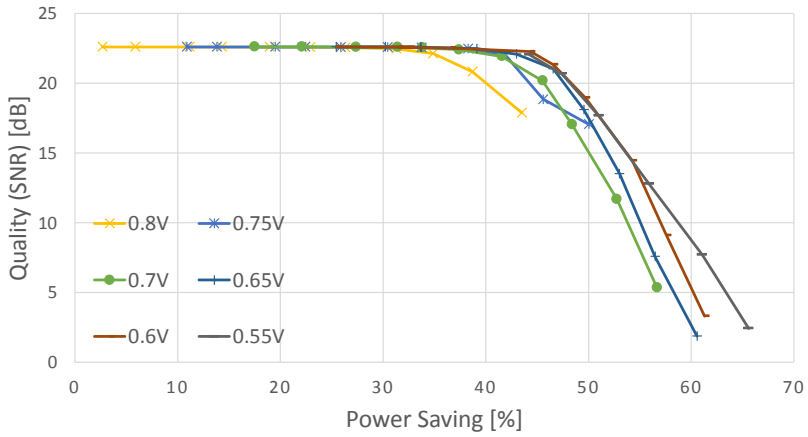


**Fig. 13** RPR ANT Quality vs. Power Saving tradeoff for a 16th order FIR filter.

the tool generate all feasible replica configurations at each $V_{vos}$, without imposing a quality constraint [5]. In this context, feasible means that the ANT architecture has positive power savings, and that the replica is timing compliant. The results on a power saving versus quality plane are reported in Figure 13, in which each curve corresponds to a value of the VOS supply voltage, and points on it correspond to different replica input widths ($B_r$).

Two trends can be identified in the graph. On the left side, quality is determined by the channel AWGN noise, and saturates to a horizontal line. Therefore, all versions of the ANT architecture have the same final quality, independently from $B_r$. On the right side of the graph, instead, the errors due to VOS become dominant with respect to channel noise, and the curves assume a typical *Pareto* shape: implementations with a smaller $B_r$ have smaller overheads, and hence higher total power savings, but also a worse quality. A quality threshold can be visualized as a horizontal line on the graph. The optimal RPR architecture for that quality is found as the rightmost point among all those above the line.

If a designer wants to maintain a quality of 21.5dB at the output of the FIR filter, in order to respect the constraints imposed by the 802.11g standard, RPR allows to save up to 44.96% of *total* power. This is achieved setting the voltage to $V_{vos} = 0.60V$, and inserting in the EC block a replica that considers the 6 MSBs of the 12-bit input for its internal computations. The total area overhead of the EC block, including replica and Decider is 88.39%. Notice that this significant power saving is obtained *without any effective impact on quality*. In fact, quality was already bounded by the noise on the channel.

Interestingly, Figure 13 shows that a lower $V_{vos}$ does not always correspond to a larger power saving, for a given quality constraint. This happens because of the

[5] In its normal operating mode, the tool only synthesizes the replica configuration with minimum $B_r$ to satisfy a quality constraint at each $V_{vos}$.

gate re-sizing performed by our tool in order to make the replica timing compliant. In fact, at lower $V_{vos}$, faster gates, which are also larger and more consuming, are needed to meet timing, and the resulting design often consumes more despite the reduced supply voltage. This trend shows that accurate timing and power evaluations for the replica, which were not performed in [47], are mandatory to correctly estimate the effectiveness of RPR. Notice that the same phenomenon is visible also in Figure 12, since power saving does not increase monotonically with the reduction of $V_{vos}$ for some input sets.

## 7 Conclusions

In this chapter, we have surveyed the most popular design solutions based on the Approximate Computing paradigm. We have shown how, despite its relatively recent formalization, AC has stimulated interesting research at all abstraction levels, from single transistors to complete systems and to software. The growing interest on this subject is demonstrated by the fact that most cited works have been published in the last two or three years. Therefore, it is easy to foresee that in the near future, even more effort from will be devoted to the progress of this field of research by academia and industry.

We envision Approximate Computing as an effective way to reduce power and energy consumption in Smart Systems, providing as motivation the fact that applications performed by these devices are often error resilient. As an example, we have applied a popular AC technique to the optimization of a digital filter, which is one of the most commonly found hardware modules in Digital Signal Processing applications. We have considered the situation in which the filter is part of the receiver of a digital communication system, affected by noise on the channel, and we have shown how, in this setting, almost 50% power saving can be achieved without an effective impact on output quality. Moreover, higher power savings can be obtained if a partial quality reduction is accepted.

## References

1. Abdallah, R., Shanbhag, N.: Minimum-energy operation via error resiliency. IEEE Embedded Systems Letters **2**(4), 115–118 (2010).
2. Alaghi, A., Hayes, J.P.: Survey of stochastic computing. ACM Trans. Embed. Comput. Syst. **12**(2s), 92:1–92:19 (2013).
3. Banerjee, N., Karakonstantis, G., Roy, K.: Process variation tolerant low power dct architecture. In: Design, Automation Test in Europe Conference Exhibition, DATE '07, pp. 1–6 (2007).
4. Bombieri, N., Drogoudis, D., Gangemi, G., Gillon, R., Macii, E., Poncino, M., Rinaudo, S., Stefanni, F., Trachanis, D., van Helvoort, M.: Smac: Smart systems co-design. In: Euromicro Conference on Digital System Design (DSD), pp. 253–259 (2013).

5. Chakradhar, S., Raghunathan, A.: Best-effort computing: Re-thinking parallel software and hardware. In: 47th ACM/IEEE Design Automation Conference (DAC), pp. 865–870 (2010)
6. Chakrapani, L.N.B., Palem, K.V.: A probabilistic boolean logic for energy efficient circuit and system design. In: Proceedings of the 2010 Asia and South Pacific Design Automation Conference, ASPDAC '10, pp. 628–635. IEEE Press, Piscataway, NJ, USA (2010).
7. Chen, J., Hu, J.: Energy-efficient digital signal processing via voltage-overscaling-based residue number system. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, **21**(7), 1322–1332 (2013).
8. Chippa, V., Chakradhar, S., Roy, K., Raghunathan, A.: Analysis and characterization of inherent application resilience for approximate computing. In: 50th ACM / EDAC / IEEE Design Automation Conference (DAC), pp. 1–9 (2013)
9. Chippa, V., Mohapatra, D., Raghunathan, A., Roy, K., Chakradhar, S.: Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency. In: 47th ACM/IEEE Design Automation Conference (DAC), pp. 555–560 (2010)
10. Chippa, V., Raghunathan, A., Roy, K., Chakradhar, S.: Dynamic effort scaling: Managing the quality-efficiency tradeoff. In: 48th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 603–608 (2011)
11. Chippa, V., Venkataramani, S., Chakradhar, S., Roy, K., Raghunathan, A.: Approximate computing: An integrated hardware approach. In: 2013 Asilomar Conference on Signals, Systems and Computers, pp. 111–117 (2013).
12. Cong, J., Sarkar, V., Reinman, G., Bui, A.: Customizable domain-specific computing. IEEE Transactions on Design and Test of Computers, **28**(2), 6–15 (2011).
13. Crepaldi, M., Grosso, M., Sassone, A., Gallinaro, S., Rinaudo, S., Poncino, M., Macii, E., Demarchi, D.: A top-down constraint-driven methodology for smart system design. IEEE Circuits and Systems Magazine, **14**(1), 37–57 (2014).
14. Das, S., Tokunaga, C., Pant, S., Ma, W.H., Kalaiselvan, S., Lai, K., Bull, D., Blaauw, D.: Razorii: In situ error detection and correction for pvt and ser tolerance. IEEE Journal of Solid-State Circuits, **44**(1), 32–48 (2009).
15. 2014 MultiAnnual Strategic Research and Innovation Agenda (MASRIA) for the ECSEL Joint Undertaking.
16. Ernst, D., Das, S., Lee, S., Blaauw, D., Austin, T., Mudge, T., Kim, N.S., Flautner, K.: Razor: circuit-level correction of timing errors for low-power operation. IEEE Micro, **24**(6), 10–20 (2004).
17. Gaines, B.R.: Stochastic computing. In: Proceedings of the Spring Joint Computer Conference, AFIPS '67, pp. 149–156. (1967).
18. Ghosh, S., Bhunia, S., Roy, K.: Crista: A new paradigm for low-power, variation-tolerant, and adaptive circuit synthesis using critical path isolation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, **26**(11), 1947–1956 (2007).
19. Gross, W., Gaudet, V., Milner, A.: Stochastic implementation of ldpc decoders. In: Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers. pp. 713–717 (2005).
20. Gupta, V., Mohapatra, D., Raghunathan, A., Roy, K.: Low-power digital signal processing using approximate adders. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, **32**(1), 124–137 (2013).
21. Han, J., Orshansky, M.: Approximate computing: An emerging paradigm for energy-efficient design. In: 18th IEEE European Test Symposium (ETS), pp. 1–6 (2013).
22. He, K., Gerstlauer, A., Orshansky, M.: Controlled timing-error acceptance for low energy idct design. In: Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1–6 (2011).
23. Hegde, R., Shanbhag, N.: Energy-efficient signal processing via algorithmic noise-tolerance. In: International Symposium on Low Power Electronics and Design (ISLPED), pp. 30–35 (1999)
24. Hegde, R., Shanbhag, N.: Soft digital signal processing. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, **9**(6), 813–823 (2001).

25. Huang, J., Lach, J., Robins, G.: A methodology for energy-quality tradeoff using imprecise hardware. In: 49th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 504–509 (2012)
26. Kahng, A., Kang, S.: Accuracy-configurable adder for approximate arithmetic designs. In: 49th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 820–825 (2012)
27. Kahng, A., Kang, S., Kumar, R., Sartori, J.: Slack redistribution for graceful degradation under voltage overscaling. In: 15th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 825–831 (2010).
28. Karakonstantis, G., Mohapatra, D., Roy, K.: System level dsp synthesis using voltage over-scaling, unequal error protection and adaptive quality tuning. In: IEEE Workshop on Signal Processing Systems (SiPS), pp. 133–138 (2009).
29. Karakonstantis, G., Roy, K.: Voltage over-scaling: A cross-layer design perspective for energy efficient systems. In: 20th European Conference on Circuit Theory and Design (ECCTD), pp. 548–551 (2011).
30. Kulkarni, P., Gupta, P., Ercegovac, M.: Trading accuracy for power with an underdesigned multiplier architecture. In: 24th International Conference on VLSI Design (VLSI Design), pp. 346–351 (2011).
31. Kyaw, K.Y., Goh, W.L., Yeo, K.S.: Low-power high-speed multiplier for error-tolerant application. In: IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC), pp. 1–4 (2010).
32. Leem, L., Cho, H., Bau, J., Jacobson, Q., Mitra, S.: Ersa: Error resilient system architecture for probabilistic applications. In: Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1560–1565 (2010).
33. Liang, J., Han, J., Lombardi, F.: New metrics for the reliability of approximate and probabilistic adders. IEEE Transactions on Computers, **62**(9), 1760–1771 (2013).
34. Liu, C., Han, J., Lombardi, F.: A low-power, high-performance approximate multiplier with configurable partial error recovery. In: Proceedings of the Conference on Design, Automation & Test in Europe (DATE), pp. 95:1–95:4. (2014).
35. Lu, S.L.: Speeding up processing with approximation circuits. Computer **37**(3), 67–73 (2004).
36. Macii, E.: Ultra low-power electronics and design. Springer US (2004)
37. Mahdiani, H., Ahmadi, A., Fakhraie, S., Lucas, C.: Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications. IEEE Transactions on Circuits and Systems I: Regular Papers, **57**(4), 850–862 (2010).
38. Meng, J., Chakradhar, S., Raghunathan, A.: Best-effort parallel execution framework for recognition and mining applications. In: IEEE International Symposium on Parallel Distributed Processing, pp. 1–12 (2009).
39. Mohapatra, D., Chippa, V., Raghunathan, A., Roy, K.: Design of voltage-scalable meta-functions for approximate computing. In: Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1–6 (2011).
40. Mohapatra, D., Karakonstantis, G., Roy, K.: Significance driven computation: A voltage-scalable, variation-aware, quality-tuning motion estimator. In: Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), pp. 195–200 (2009).
41. Nepal, K., Li, Y., Bahar, R.I., Reda, S.: Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In: Proceedings of the Conference on Design, Automation & Test in Europe (DATE), pp. 361:1–361:6 (2014)
42. Palem, K., Lingamneni, A.: What to do about the end of moores law probably. In: Proceedings of the 49th Design Automation Conference (DAC), pp. 924–929 (2012).
43. Poppelbaum, W.J., Afuso, C., Esch, J.W.: Stochastic computing elements and systems. In: Proceedings of the Joint Computer Conference (AFIPS), pp. 635–644 (1967).
44. Proakis, J.G., Manolakis, D.G.: Digital Signal Processing (3rd Ed.): Principles, Algorithms, and Applications. Prentice-Hall (1996)
45. Ranjan, A., Raha, A., Venkataramani, S., Roy, K., Raghunathan, A.: Aslan: Synthesis of approximate sequential circuits. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1–6 (2014).

46. Shim, B., Shanbhag, N.: Performance analysis of algorithmic noise-tolerance techniques. In: Proceedings of the 2003 International Symposium on Circuits and Systems (ISCAS), vol. 4, pp. IV–113–IV–116 (2003).
47. Shim, B., Sridhara, S., Shanbhag, N.: Reliable low-power digital signal processing via reduced precision redundancy. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, **12**(5), 497–510 (2004).
48. Shin, D., Gupta, S.: Approximate logic synthesis for error tolerant applications. In: Design, Automation Test in Europe Conference Exhibition (DATE), pp. 957–960 (2010).
49. Shin, D., Gupta, S.: A new circuit simplification method for error tolerant applications. In: Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1–6 (2011).
50. Tanenbaum, A.: Computer Networks, 4th edn. Prentice Hall Professional Technical Reference (2002)
51. Venkataramani, S., Chakradhar, S., Roy, K., Raghunathan, A.: Approximate computing for efficient information processing. In: 12th IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia), pp. 9–10 (2014).
52. Venkataramani, S., Chippa, V.K., Chakradhar, S.T., Roy, K., Raghunathan, A.: Quality programmable vector processors for approximate computing. In: Proceedings of the 46th IEEE/ACM International Symposium on Microarchitecture (MICRO-46) pp. 1–12 (2013).
53. Venkataramani, S., Sabne, A., Kozhikkottu, V., Roy, K., Raghunathan, A.: Salsa: Systematic logic synthesis of approximate circuits. In: 49th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 796–801 (2012)
54. Verma, A., Brisk, P., Ienne, P.: Variable latency speculative addition: A new paradigm for arithmetic circuit design. In: Design, Automation and Test in Europe (DATE), pp. 1250–1255 (2008).
55. Yang, Z., Jain, A., Liang, J., Han, J., Lombardi, F.: Approximate xor/xnor-based adders for inexact computing. In: 13th IEEE Conference on Nanotechnology (IEEE-NANO), pp. 690–693 (2013).
56. Yetim, Y., Martonosi, M., Malik, S.: Extracting useful computation from error-prone processors for streaming applications. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE), pp. 202–207 (2013).
57. Zhu, N., Goh, W.L., Wang, G., Yeo, K.S.: Enhanced low-power high-speed adder for error-tolerant application. In: International SoC Design Conference (ISOCC), pp. 323–327 (2010).
58. Zhu, N., Goh, W.L., Yeo, K.S.: An enhanced low-power high-speed adder for error-tolerant application. In: Proceedings of the 12th International Symposium on Integrated Circuits (ISIC), pp. 69–72 (2009)
59. Zhu, N., Goh, W.L., Yeo, K.S.: Ultra low-power high-speed flexible probabilistic adder for error-tolerant applications. In: International SoC Design Conference (ISOCC), pp. 393–396 (2011).
60. Zhu, N., Goh, W.L., Zhang, W., Yeo, K.S., Kong, Z.H.: Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, **18**(8), 1225–1229 (2010).