Randomized Algorithms: A System-Level, Poly-Time Analysis of Robust Computation

Cesare Alippi, *Senior Member*, *IEEE*

Abstract—This paper provides a methodology for analyzing the performance degradation of a computation once affected by perturbations. The suggested methodology, by relaxing all assumptions made in the related literature, provides design guidelines for the subsequent implementation of complex computations in physical devices. Implementation issues, such as finite precision representation, fluctuations of the production parameters, and aging effects, can be studied directly at system level, independently from any technological aspect and quantization technique. Only the behavioral description of the computational flow, which is assumed to be Lebesgue measurable and the architecture to be investigated are needed. The suggested analysis is based on the recent theory of Randomized Algorithms, which transforms the computationally intractable problem of robustness investigation in a poly-time algorithm by resorting to probability.

Index Terms—Embedded system design, finite precision error analysis, randomized algorithms, sensitivity analysis, system level design.

1 Introduction

We say that an application is robust when the impact of bounded perturbations on the associated computation provides a graceful degradation in performance with respect to a given figure of merit [1], [2], [3], [4], [5]. Testing the robustness of an application at system level before any implementation/physical aspect is taken into account goes in the direction of a safe and reliable design for embedded systems foreseen in [6]. There, one of the envisioned design challenges is in the introduction of reliability at low cost, obtained without brute force redundancy (e.g., replicated HW, such as triple modular redundancy): Software and hardware must anticipate/soften electronic and nonelectronic failure modes to at least "fail safe," in the sense that a smooth performance loss is expected.

By following this philosophy, we address the robustness analysis issue in a wide sense: Perturbations can affect the computation as discrete events, e.g., as it happens in the stuck-at model or as continuous variables; hence, resembling the soft perturbations of analog devices. De facto, perturbations abstract physical sources of uncertainty affecting the computation.

A robustness analysis approach at system level can be therefore envisaged to estimate, before a specific physical perturbation takes place, the structural loss in performance of the application. This allows the designer to decide whether validating the candidate architectural choice or not and identify the most critical points of the computational flow at the very high level of the design cycle.

• *The author is with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy. E-mail: alippi@elet.polimi.it.*

Manuscript received 17 Dec. 1999; revised 17 Oct. 2001; accepted 6 Nov. 2001.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 111114.

In general, the technical literature addresses the robustness analysis problem by considering a specific function to be implemented on a given architecture and well-defined perturbations.

Based on this set-up, deterministic approaches hide the probabilistic structure of the perturbation on the computation by considering a worst case scenario [5]. Quite often, the obtained information is weak and useless since it derives from bounding the error propagation along the computation.

Probabilistic approaches are somehow more effective and study the influence of perturbations on the computation by evaluating the generation and the propagation of perturbations through the computational flow [7], [8], [9]. To make the mathematics more amenable, such papers limit the analysis to a specific computation (e.g., linear functions [7], neural networks [9], [10], [11], [12], Discrete Cosine Transform [13], and Fast Fourier Transform [14]), consider well-defined perturbations (mostly associated with quantization techniques), assume the function to be regular (e.g., it is required continuity and differentiability) and the small perturbation hypothesis [7], [8], [10], [11].

The small perturbation hypothesis allows the function associated with the computation to be linearized with a Taylor expansion and supports the error propagation rule; the analysis is acceptable only if the function is linear or the feasible perturbations are small enough to grant the validity of the expansion.

Unfortunately, regularity assumptions and the small perturbation hypothesis significantly limit the applicability of the methodology to the large class of strongly nonlinear functions and/or severe perturbations, respectively.

In addition, almost all papers focus the attention on the implementation of a specific function; specificity prevents, most of the time, the extension of the methodology to other classes of functions.

Different from the current literature, this paper provides a general methodology for analyzing and quantifying the robustness of a computation for a *large class* of functions directly at a system level. A system level robustness analysis also allows the designer to study the impact of *any* perturbation of *any* nature influencing the behavioral computation and quantify its effect on the computation output.

Finite precision representations (e.g., fixed/floating point representations, truncation, and rounding), fluctuations in the production process, transient and permanent faults affecting the computation are relevant examples of perturbations: All these implementation-related sources of uncertainties are abstracted by behavioral perturbations. A system level analysis based on the concept of behavioral perturbations hence groups all these complex fully interacting aspects within an effective and homogeneous methodological framework.

The suggested methodology allows removing all the hypotheses addressed in the related literature. In particular,

- The methodology is no more specific to a given application but general and applicable to the wide class of Lebesgue-measurable functions (basically, every mathematical computation related to the engineering field is Lebesgue-measurable),
- Results are independent from any technological and implementation issue,
- The small perturbation assumption is no more required,
- No assumptions are made on perturbations and on their placement within the computation, and
- The continuity/differentiability hypotheses for the function to be implemented are relaxed.

The analysis is therefore extremely general. Of course, generality makes it almost impossible to solve the robustness problem in a closed form for a generic function. In addition, since it is required testing the whole application for robustness we have to explore all the continuous perturbation space (so as to cover *all* the unknown physical perturbations that might, in the future, influence the computation).

It is evident that this problem is intractable from the computational point of view. In fact, even if we should only consider a uniform grid on the k-dimensional perturbation space (k represents the generally large number of points we wish to test for robustness), say g points for each dimension, we should consider g^k points for robustness. This sampling algorithm scales badly with g (the resolution in the grid) and k (the dimension of the perturbation space); this effect is known as the "curse of dimensionality" [15], a complexity theoretic barrier.

The key point for solving the robustness problem in all its aspects is the introduction of the recent theory based on Randomized Algorithms [16], [17], [18]. Randomized algorithms derive from the learning theories [19], [20] and are strictly related to the Montecarlo method; they turn, under weak hypothesis, an intractable problem into a tractable one, which can be tackled with a polynomial complexity. The price we have to pay is that results are valid in probability with accuracy and confidence levels that can be made *arbitrarily* close to zero and 100 percent, respectively.

Wide evidence for the effectiveness of such approaches can be found in the control theory community where great efforts have been devoted to the analysis and design of robust controllers [17], [18], [21], [22], [23], [24] and, indirectly, in the extreme relevance and widespread use of their father, the Montecarlo method.

The structure of the paper is as follows: Section 2 formulates in more detail the problem at a behavioral level and relates behavioral entities with physical ones. Section 3 introduces the concept of robust computation and characterizes the robustness analysis by relating the loss in performance with behavioral perturbations. Section 4, after a brief introduction of the theory behind Randomized Algorithms, applies them to estimate, with a poly-time algorithm, the robustness degree of the computation. Experiments are given in Section 5, where Randomized Algorithms have been used to test the robustness of a time multiplexed digital multiply-add-accumulate solution for computing the scalar product. The simplicity of the chosen architecture is only apparent, since truncation operators associated with a digital implementation transforms the function in a highly nonlinear one.

2 Behavioral and Physical Computations

The section introduces the concepts of behavioral computation, behavioral architecture, and behavioral perturbations as abstractions of their physical counterparts. Behavioral entities constitute the system level framework onto which the subsequent robustness analysis is developed. More in detail, at the system level, we need

- a behavioral description of the computational flow associated with the application,
- a target Reference Behavioral Architecture (RBA) executing the behavioral computation, and
- a set of behavioral perturbations modeling uncontrollable uncertainties associated with the application and affecting the RBA.

A behavioral description for the computation requires the characterization, at behavioral level, of all the operators involved in the computation and their features by means of an appropriate description language. A behavioral architecture is an abstract architecture executing the behavioral computation and, hence, it is characterized by the way the computation is carried out. The behavioral architecture abstracts a physical architecture; its structure is suggested by the designer or provided by automatic design tools.

Once a specific architecture has been selected among a set S of equivalent solutions supporting the execution of the computation, it becomes the target RBA.

As a simple example, we consider the behavioral computation associated with the linear function $y = a_1x_1 + a_2x_2$ with real and scalar operands. The behavioral computation is characterized by the addition and multiplication operators; the description is completed by labelling operands as real, scalar, and bounded.

From the behavioral computation we deduce the set S of equivalent behavioral architectures supporting the execution of the behavioral computation. As we mentioned S is composed of all feasible architectures able to support the scalar product execution. The potentially large cardinality of S is reduced by the designer who prunes, manually or

```
A1:                          A2:
s1 = a1x1; s2 = a2x2;        y = 0;
y = s1 + s2;                 for (i = 1; i <= 2; i++) y = y + aixi;
```

Fig. 1. Two behavioral architectures associated with the $y = a_1x_1 + a_2x_2$ function.

with the support of automatic tools, the architecture space. For instance, in a digital realization, pruning is based on the available macrocell libraries and on rough estimates for silicon area and power consumption.

In this example, we identify within S the two relevant behavioral architectures *A1* and *A2* described by the C-like code in Fig. 1.

Note that *A1* resembles a parallel implementation in which each abstract operator is mapped one-to-one to a physical operator. Conversely, *A2* refers to a multiply-add-accumulate time multiplexed architecture. If the designer—or some automated procedure—selects *A2* to be the candidate architecture for solving the application, and hence to be studied for robustness, then *A2* becomes the RBA.

The third element to be defined is the behavioral perturbation, an abstraction for uncertainties affecting the computation in all its aspects. Such perturbations can be classified according to their nature in application-dependent perturbations and architecture-dependent perturbations:

- Application-dependent perturbations address all uncertainties affecting the application itself. Basically, in this class we have noise associated with measured quantities [25] and fluctuations affecting somehow the application parameters [26], [27], [28]. Once such source of uncertainty has been characterized, e.g., by associating a probabilistic description to perturbations, we have fully described the behavioral computational flow. Application-dependent perturbations are uncontrollable, intrinsic with the application but independent from any architectural or implementation issue. Knowledge about the nature of application-dependent perturbations can be exploited by automatic design tools to restrict the design search space, e.g., by dimensioning some components [7] as it happens in those applications whose parameters have been identified (linear models [27], neural networks [29], [30], [31], adaptive wavelets [32]).
- Architecture-dependent perturbations influence the computational flow of the RBA, which becomes a Perturbed Behavioral Architecture (PBA). PBA is an abstraction of a physical architecture as architecture-dependent perturbations are abstractions of physical perturbations. Examples of physical perturbations are introduced by the word-length dimensioning of variables associated with the computational flow [7], [8], quantization operators, look-up tables and approximated nonlinear computation [33], [34] on the digital side. In analog realizations, we encounter fluctuations due to the production process and "electronic" noise [35], transient and permanent faults, and small deviations from the working point due to aging effects. Also, the particular choice for the physical platform implementing the application, would it be a dedicated hardware or software or both, is a sort of perturbation affecting the computation (think of perturbations caused by fixed or floating point representations). Such aspects are also related to Hardware/Software codesign (e.g., see [36]) since a proper word-length dimensioning in a perturbed environment has an immediate positive impact on computational complexity and power consumption [7].

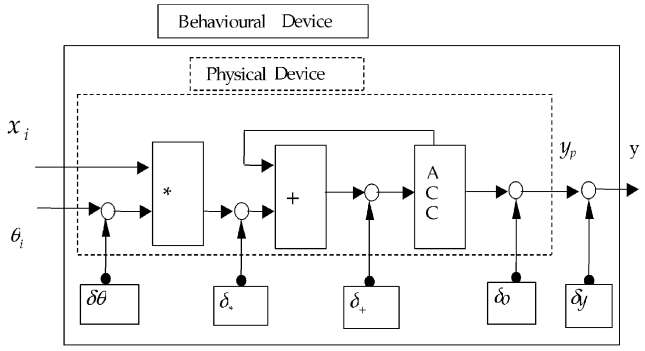

Fig. 2. The multiply-accumulate digital architecture with perturbations affecting the computation.

The above sources of uncertainty transform the behavioral computation from correct to quasicorrect [4] since the output provided by a physical device does not generally coincide with the ideal value provided by an error-free application and an error-free device. In addition, application and architecture dependent perturbations have an indistinguishable impact on the computation output, being impossible to separate the different sources of "noise": Application and architectural perturbations are therefore intrinsically and intimately related. The close relationship between the two perturbations makes difficult the perturbation versus final performance analysis. This effect amplifies whenever the function associated with the application solution is nonlinear and discontinuous as it happens in the *max* or the *sum of steps* functions considered in several applications [31], [33], [34].

Behavioral perturbations tackle physical errors by grouping them within a unique framework which abstracts their complex nature. The consequence is that a physical perturbation is only a realization of a behavioral perturbation and, as such, does not need to be studied separately from the others. The price we have to pay is that we lose specificity, i.e., by considering a particular physical perturbation (e.g., induced by truncation), we could improve the robustness estimate being tailored to the specific error phenomenon.

To shed light on the close relationships linking behavioral perturbations a given RBA and the physical counterparts, we focus the attention on a simple case, the digital multiply-add-accumulator architecture of Fig. 2. The analysis can be immediately extended to a more complex architecture. The RBA is subject to four different behavioral perturbations abstracting the physical errors:

- An independent perturbation $\delta\theta$ affecting each coefficient of the filter. $\delta\theta$ could be associated with a finite precision representation for the coefficients,

- A perturbation δ_* modeling the source of error introduced by the multiplication operator (e.g., the output of the multiplier is truncated to the same number of bits of the operands),
- A perturbation δ_+ affecting the output of the addition operator (e.g., a right shift of the register containing the operand, a truncation of the output, etc),
- A perturbation δo (e.g., a truncation) to keep the output defined on a fixed number of bits.

Obviously, in the case of a parallel RBA behavioral perturbations would be inserted in different points of the computational flow.

It must be noted that the presence of a perturbation in a point of the computation must be intended as an equivalent error affecting that specific point. The perturbation takes care of all sources of errors introduced in the computation between two subsequent perturbation-affected points. No assumptions are made, for instance, on the specific realization of the multiplier, which will be characterized later on in lower levels of the design cycle. We only require that the physical error generated by the physical device implementing that part of the computation belongs to the perturbation domain of the behavioral perturbation.

Therefore, we consider only mutually independent perturbations, each of which represents an effective equivalent error contribution in a specific point. The effect of errors coming from previous computational modules and propagated up to the envisaged one will combine with the local error in a nonlinear way depending on the structure of the computation and the chosen architecture. In fact, for the Fig. 2 architecture, we have $y = y_p + \delta y$ with

$$\delta y = \delta y(x) = \delta y(\delta\theta, \delta o(x), \delta_+(x), \delta_*(x), x)$$

depending in a nonlinear and complex way on inputs and propagated errors. A behavioral perturbation hides such a complex unknown relationship by simply considering perturbations defined within a feasible perturbation space.

From the specific example, we derive the general procedure for associating a behavioral element with its physical counterpart:

- Behavioral and physical architectures are structurally the same. In fact, if we select a physical architecture immediately, we have associated its abstracting RBA and viceversa.
- A behavioral perturbation must be introduced in a point of the behavioral computation whenever the physical computation introduces in the same point an independent source of uncertainty (e.g., caused by truncation). We denote by Δ the perturbation vector containing all behavioral perturbations associated with a PBA and by k the dimension of the perturbation space, namely the number of points of the RBA to be simultaneously tested for robustness.

The designer, having in mind the final physical architecture, will therefore identify the number and the placement of perturbations in the RBA. The domain of the behavioral perturbations derives from a priori hints (see the experimental section) and experience; an automatic procedure driven by HW/SW codesign strategies could be successfully considered to characterize such an information.

In the case of Fig. 2 the perturbation vector is $\Delta = [\delta\theta_1, \cdots, \delta\theta_n, \delta_*, \delta_+, \delta_o]$, where n represents the number of coefficients of the linear filter. Of course, we could assume no errors at the multiplier and at the adder outputs; in such a case the behavioral perturbation vector to be considered is $\Delta = [\delta\theta_1, \cdots, \delta\theta_n, \delta_o]$.

3 Perturbations, Loss in Performance and Robust Computation

Denote by $y = f(x)$, $y \in Y \subset \Re^1$, and $x \in X \subset \Re^d$ the mathematical description of the behavioral computation (the methodology also supports the extension to consider $y \in \Re^m$). Let $D \subset \Re^k$ be the k-dimensional compact set of the behavioral perturbations and $\Delta \in D$ a generic architecture-dependent perturbation affecting the RBA. The PBA implementing the $y = f(x, \Delta)$ function is therefore a perturbed realization of the RBA limited in performance by Δ. In the perturbed space all PBAs belong to a neighbourhood of RBA described by x and Δ spanning X and D, respectively. If $Vol(D)$ is the volume of the perturbation domain D we have that

$$\lim_{Vol(D)\to 0} PBA = RBA \quad \forall x \in X \tag{2.1}$$

even if nothing can be said a priori about the statistical nature of the neighborhood or the convergence properties of the limit.

Surely, the accuracy of the PBA in tracking the RBA depends on the specific application and its robustness, the magnitude of the perturbation, and the architectural structure of the RBA. In fact, if the application is intrinsically robust the perturbation will induce a weaker effect on final performance.

To compute the performance loss introduced by the PBA, we consider a generic discrepancy function $u(\Delta)$ which is assumed to be Lebesgue-measurable with respect to D.

A common discrepancy function requires the evaluation of the average

$$u = u(\Delta) = \int (y(x) - y(x, \Delta))^2 ddp_{xy} dxy, \tag{2.2}$$

where ddp_{xy} is the joint distribution density function defined over (X,Y). In general, the evaluation of (2.2) is complex and a closed form does not exist. Nevertheless, we can approximate (2.2) with its empirical risk evaluated over the available $N_x(x, y)$ pairs

$$u = u(\Delta) = \frac{1}{N_x} \sum_{i=1}^{N_x} (y(x_i) - y(x_i, \Delta))^2. \tag{2.3}$$

Refer to [26], [27], [37] for the minimum N_x granting a good approximation of (2.2).

Another popular figure of merit used to quantify the impact of a perturbation on a signal is the Noise to Signal ratio *NSR*

$$u(\Delta) = NSR = \frac{\frac{1}{N_x} \sum_{i=1}^{N_x} \big(y(x_i) - y_p(x_i, \Delta)\big)^2}{\frac{1}{N_x} \sum_{i=1}^{N_x} \big(y(x_i) - \mu_y\big)^2}, \tag{2.4}$$

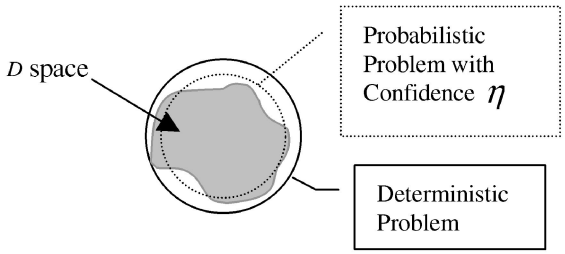

Fig. 3. The D domain is shadowed, all points outside the dashed circled do not satisfy the $u(\Delta) \leq \bar{\gamma}$ bound; the set of D for which $u(\Delta) \leq \bar{\gamma}$ has a measure according to Lebesgue $1 - \eta$.

where μ_y is the mean value of the output. Since the denominator is constant for a given application (2.4) formally coincides with (2.3).

The designer will select, for each application, the most suitable loss function $\boldsymbol{u}$.

The selected PBA is acceptable if the loss in performance induced by perturbations is such that

$$<< u(\Delta) \text{ is small} >> \text{ for } \forall \Delta \in D. \tag{2.5}$$

For instance, the ratio of (2.4) multiplied by 100 represents the relative percentage loss in performance of the device. The device is robust and solves our application if (2.4) is below some tolerated loss in performance. This observation directly leads to the concept of robust computation.

3.1 The Deterministic Definition of Robust Computation

We say that a computation is robust at level $\bar{\gamma}$ for a given D when $\bar{\gamma}$ is the minimum positive value, granting that $u(\Delta) \leq \bar{\gamma}$ is satisfied $\forall \Delta \in D$.

The computation is robust if $\bar{\gamma}$ is below some tolerable loss in performance directly coming from the accuracy requirement of the application. Note that the interval $[0, \bar{\gamma}]$ contains all the possible losses in performance associated with a given PBA when the architecture-dependent perturbations span D.

We can finally suggest a constructive algorithm for comparing the robustness of different architectures: Architecture C_1 is more robust than architecture C_2 if $\bar{\gamma}_1 < \bar{\gamma}_2$.

The designer can consider different RBAs and test several PBAs: The PBAs solving the application have $\bar{\gamma}$s below the acceptable performance loss.

The deterministic problem cannot be solved in a closed form for a generic function; in addition a point by point investigation would require an infinite number of tests for a continuous D. To solve such a computationally intractable problem we weaken the deterministic problem by formulating a dual probabilistic one.

3.2 The Probabilistic Definition of Robust Computation

We say that a computation is robust at level $\bar{\gamma}$ for a given D. when $\bar{\gamma}$ is the minimum positive value granting that $u(\Delta) \leq \bar{\gamma}$ is satisfied $\forall \Delta \in D$ at least with probability η:

$$\Pr(u(\Delta) \leq \bar{\gamma}) \geq \eta, \quad \forall \Delta \in D. \tag{2.6}$$

EXTRACT $N \geq \frac{\ln \frac{2}{\delta}}{2\varepsilon^2}$ POINTS FROM D;

GENERATE THE FUNCTION $p_N = p_N(\gamma)$;

SELECT THE MINIMUM γ_m SO THAT $\hat{p}_N(\gamma) = 1, \forall \gamma_> \gamma_m$;

γ_m IS THE ESTIMATE OF $\bar{\gamma}$;

Fig. 4. The procedure for computing $\bar{\gamma}$

Differently from the deterministic problem the probabilistic one tolerates the existence of perturbations Δs not satisfying the $u(\Delta) \leq \bar{\gamma}$ inequality.

Denote by $\Delta_{in\bar{\gamma}}$ a generic point satisfying the $u(\Delta) \leq \bar{\gamma}$ inequality. We have that η represents the ratio between the volume of perturbation points satisfying the $u(\Delta) \leq \bar{\gamma}$ and the total volume of the perturbation space

$$\frac{Vol(\Delta_{in\bar{\gamma}})}{Vol(\Delta)} = \eta, \quad Vol(\Delta_{in\bar{\gamma}}) = \int_{D_{in\bar{\gamma}}} d\Delta. \tag{2.7}$$

In other words, we tolerate the fact that some perturbations might not satisfy the bound; the Lebesgue measure of such a set is $1 - \eta$. This situation has been geometrically depicted in Fig. 3.

From the geometrical interpretation of the probabilistic robust computation we have that in probability at least 100η percent of perturbations Δ will generate a loss in performance below $\bar{\gamma}$. Of course, $\bar{\gamma}$ depends on the size of D, the robustness of the application, and the position in which the perturbations have been injected in the computational flow.

It is relevant to outline that when $\eta = 1$ all points in probability satisfy the inequality; anyway, there could exist a not empty set of points Ω not satisfying it; the Lebesgue measure of such set is anyway null (and, hence, null the probability of extracting them).

When the function is continuous with respect to X and D, so is the $u(\Delta)$-transformed space. Under this assumption we have that the points not satisfying the requirement, if any, lie close to the ones which satisfy it [38] and, hence, the estimate for $\bar{\gamma}$ is reliable even if Ω is not null.

In the following, we will require η to be 1 to grant, at least in probability, that all points satisfy the inequality. In this case, the probabilistic robust computation and the probabilistic one coincide with a probability of one and the obtained $\bar{\gamma}$ becomes an effective estimate for the robustness degree of the computation when executed by the PBA.

Of course, we would like $\bar{\gamma}$ to be as small as possible since this implies a small loss in performance (the output of the PBA would float in a small neighborhood of the RBA).

$\bar{\gamma}$ characterizes the robustness degree of the PBA and allows the designer to solve two architectural problems at the very high levels of the design flow:

1. Identify a set of RBAs, the associated PBAs, and test their robustness.
2. Given RBA and its PBA, test the robustness of the computation by considering increasing sizes for D

FOR EACH APPLICATION/ARCHITECTURE EXTRACT $N \geq \frac{\ln\frac{2}{\delta}}{2\varepsilon^2}$ POINTS FROM D

GENERATE THE FUNCTIONS $p_N = p_N(\gamma)$

FOR EACH FUNCTION $p_N = p_N(\gamma)$ SELECT THE MINIMUM $\bar{\gamma}$ SO THAT

$\hat{p}_N(\gamma) = 1, \forall\, \gamma_> \bar{\gamma}$ IS SATISFIED;

SELECT THE APPLICATION/ARCHITECTURE ASSOCIATED WITH THE SMALLEST AMONG $\bar{\gamma}$ S

Fig. 5. The procedure for selecting the best architecture/application within a given set.

(i.e., we consider perturbations having increased magnitude).

During the design phase the designer will try to enlarge the size of D as much as possible still keeping the performance loss $\bar{\gamma}$ below the acceptable level. The size of D has an immediate impact on subsequent design phases since it can be directly related to the dimensioning of the module implementing the computation. In fact, if the final implementation is digital then a large D implies that we can reduce the number of bits to represent the perturbation-affected variables or that the module implementing the computation can be implemented with a lower resolution. Independently from the way the computational module is implemented its effective physical error must belong to D to grant a performance loss below $\bar{\gamma}$. Conversely, if the realization is analog, we can identify the accuracy required by the components (tolerable deviations from their nominal value). In general, any physical perturbation belonging to D, however it has been generated, will induce a performance loss below $\bar{\gamma}$ and, hence, acceptable by the application.

4 A Robust Computation Analysis by Randomized Algorithms

In this section, we suggest a methodology for testing the robustness of a computation by computing the robustness degree $\bar{\gamma}$.

Denote by $p_\gamma = \Pr\{u(\Delta) \leq \gamma\}$ the probability that the performance loss is satisfied for each $\Delta \in D$ given—but generic-nonnegative performance loss γ. Unfortunately, p_γ is unknown and its characterization would require the exploration of the whole perturbation space, a computationally intractable problem. We resort then to randomized algorithms, which transform the intractable problem in a tractable one by suitably sampling the perturbation space.

TABLE 1
D and the Corresponding Number of Bits

Perturbation	$\delta 6$	ε_*	ε_+	δo
Extreme interval α	0.25	0.125	0.25	0.0625
Number of bits equivalent to α	2	3	2	4

4.1 Randomized Algorithms

Let ***u*** be a function measurable according to Lebesgue with respect to the perturbation space D. Extract from D a set of N independent and identically distributed samples Δ_i according to the associated probability density function. Generate then the triplets

$$\{\Delta_i, u(\Delta_i), I(\Delta_i)\}, i = 1, N, \tag{4.1}$$

where $I(\Delta_i)$ is the indicator function

$$I(\Delta_i) = \begin{cases} 1 & \text{if} \quad \mathrm{u}(\Delta_\mathrm{i}) \leq \gamma \\ 0 & \text{if} \quad \mathrm{u}(\Delta_\mathrm{i}) > \gamma. \end{cases} \tag{4.2}$$

and γ is a given nonnegative value.

The unknown probability $p_\gamma = \Pr\{u(\Delta) \leq \gamma\}$ can be approximated with the frequency

$$\hat{p}_N = \frac{1}{N}\sum_{i=1}^{N} I(\Delta_i). \tag{4.3}$$

It is reasonable to expect that by increasing the number of samples $\hat{p}_N$ tends to p_γ within an accuracy degree ε. Since $\hat{p}_N$ is a random variable depending on the particular realization of the N samples the $|\hat{p}_N - p_\gamma| \leq \varepsilon$ inequality is

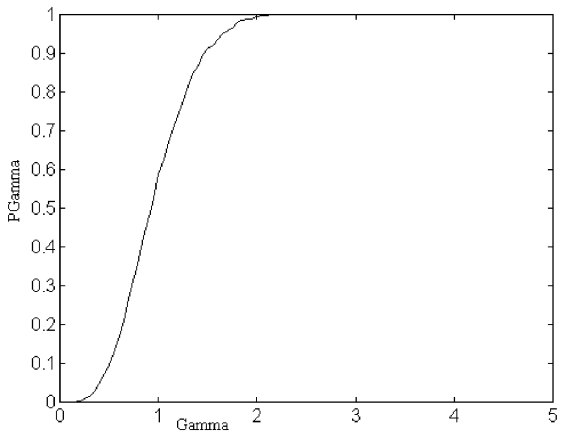

Fig. 6. $\hat{p}_N = \hat{p}_N(\gamma)$, accuracy 5 percent, confidence 99 percent.

TABLE 2
The Different **u** Associated with Different N_x and N Sampling

Run	1	2	3	4
Mean(u)	0.961	0.975	0.960	0.973
Max(u)	2.141	2.462	2.415	2.259

a random variable as well. By introducing a confidence degree $1-\delta$, we shall require that

$$\Pr\{|p_\gamma - \hat{p}_N| \leq \varepsilon\} \geq 1-\delta, \quad \forall \gamma \geq 0, \forall \delta, \varepsilon \in [0-1]. \quad (4.4)$$

The number of samples N granting (4.4) to hold $\forall \gamma \geq 0, \forall \delta, \varepsilon \in [0-1]$ is bounded by the Chernoff inequality [9] :

$$N \geq \frac{\ln \frac{2}{\delta}}{2\varepsilon^2}. \quad (4.5)$$

For instance, by considering a 5 percent in accuracy and 99 percent in confidence we have to extract 1,060 samples from D; with such N we can approximate p_γ with $\hat{p}_N$ introducing the maximum error 0.05

$$(\hat{p}_N - 0.05 \leq p_\gamma \leq \hat{p}_N + 0.05);$$

the inequality holds at least with probability 0.99.

Other bounds can be considered instead of the Chernoff as suggested by Bernoulli and Bienayme [18]. Nevertheless, the Chernoff bound improves upon the others and, therefore, should be preferred if we wish to keep minimal the number of samples to be extracted. The Chernoff bound grants that

- N is independent from the dimension of D (and, hence, it does not depend on the number of perturbations k we are considering),
- N is linear in $\ln \frac{1}{\delta}$ and $\frac{1}{\varepsilon^2}$ (and, hence, it is polynomial in the accuracy and confidence degrees).

The probabilistic problem for a robust computation at level γ can be solved with Randomized Algorithms and, hence, with a polynomial complexity in the accuracy and the confidence degrees independently from the number of points to be tested for robustness. In fact, from (4.4) and (4.5) it follows that

$$\Pr\{|p_\gamma - \hat{p}_N| \leq \varepsilon\} \geq 1-\delta \equiv$$
$$\Pr\left\{\left|\Pr(u(\Delta) \leq \gamma) - \frac{1}{N}\sum_i I(\Delta_i)\right| \leq \varepsilon\right\} \geq 1-\delta. \quad (4.6)$$

From (4.6), if the accuracy ε and $1-\delta$ are small enough, we can confuse p_γ and $\hat{p}_N$ with high accuracy and the probabilistic robust computation problem requiring $p_\gamma \geq \eta$ becomes equivalent to $\hat{p}_N \geq \eta$. We assume ε and $1-\delta$ to be small enough in subsequent derivations.

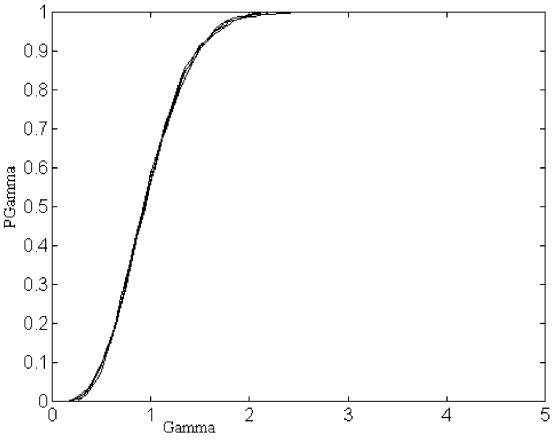

Fig. 7. $\hat{p}_N = \hat{p}_N(\gamma)$, accuracy 5 percent, confidence 99 percent, four runs, $N_x = 40, N = 1060$.

4.2 Problem 1: Testing the Robustness of a Behavioral Computation (Application)

To test the robustness of an application with respect to a set of points within the computation we have first to characterize D. D can be provided by some automatic design tool, deduced from a priori information about the entity of physical perturbations, or directly suggested by the designer. We can then determine the upper bound $\bar{\gamma}$ for γ granting that $\forall \Delta \in D$ the performance loss is smaller than $\bar{\gamma}$ with probability one. The obtained $\bar{\gamma}$ represents the maximum loss in performance of the application and, hence, it measures the robustness of the application with respect to the test points. The procedure based on Randomized Algorithms is given in Fig. 4.

4.3 Problem 2: Testing the Robustness of a Given Architecture

This problem is a subcase of the previous one and aims at estimating the robustness of a RBA once subject to a set of perturbations. We remind (Section 2) that behavioral entities are abstractions of physical entities; the problem is therefore equivalent to test the robustness of a given physical architecture by considering its abstract counterpart.

The first step requires associating a behavioral perturbation in the RBA to each *independent* source of perturbation we expect in the computational flow (again we *do not* have to consider the propagation of errors along the computation). The algorithm to be considered is again that of Fig. 4.

4.4 Problem 3: Selecting the Best Architecture/Most Robust Application within a Set of Architectures/Applications

Given a set of architectures solving a given application we wish to determine the best one, defined as the PBA having the minimum value for $\bar{\gamma}$. Selection of the most robust application among a set of equivalent applications is a similar problem which can be solved with the same

TABLE 3
Three PBAs

Perturbation	$\delta\theta$	δ_*	δ_+	δo	Mean(u)	Max(u)
Experiment 1	3	3	3	3	0.272	0.620
Experiment 2	2	2	1	2	1.395	3.351
Experiment 3	1	2	2	2	3.517	8.541

The difference is in the size of D.

methodology. In both cases, we have to consider the algorithm of Fig. 4 suitably extended to extract the minimum among the different solutions as suggested in Fig. 5.

The presented procedures solve different aspects of the robustness problem. It is the designer task to identify the correct procedure on the basis of the application requirements. For instance, if our goal is to identify the computation robustness of a computational flow that needs to be executed on given architecture we have to consider Problem 2. Conversely, if we wish to select the most robust architecture among a set of computationally equivalent architectures, we have to consider Problem 3.

5 Experimental Results

In this section, we explain how the suggested procedures can be used to solve a real application. For its interest and simplicity, we focus the attention on the scalar product computation by considering as RBA the multiply-add-accumulate architecture of Fig. 2. The linear filter we are considering has fixed coefficients $[1.23\ -2.5\ 3.4\ 2.2\ -1.34]$ and receives uniformly distributed inputs in the $[-5,5]$ interval.

As a first example, we wish to test the robustness of a behavioral architecture (Problem 2) in order to dimension the associated physical one.

To this end we assume that each component of the behavioral perturbation vector is uniformly distributed in the $[-\alpha,\alpha]$ interval. The choice of D has an immediate impact on the associated physical architecture. In fact, since the distribution is uniform and the interval symmetric, we have a straight relationship between α and the number of bits q we consider to represent the decimal part of the perturbed variable.

In fact, if truncation is envisioned as a finite precision representation technique, we have that $\alpha = 2^{-q}$; a similar relationship holds also for rounding [11].

Table 1 explains the relationships between the domain of the behavioral perturbations characterized by α affecting the RBA and the dimensioning, at a bit level, of the variables involved in the computation, i.e., the associated physical architecture.

With respect to the table, the fact that α associated with δo is smaller than that of δ_* implies that we are considering a higher precision to represent the output of the computation than the one considered for the output of the multiplier.

To test the robustness of the computation running on the RBA we considered a 5 percent in accuracy and 99 percent in confidence; from the Chernoff bound we need to uniformly extract from D at least 1,060 samples.

According to the algorithm given in Fig. 4 we generated the $\hat{p}_N = \hat{p}_N(\gamma)$ function of Fig. 6. We note that the minimum $\bar{\gamma}$ so that $\hat{p}_N(\bar{\gamma}) = 1$ is 2.141. Again, this means that at least with probability 0.99 we can assert that $u(\Delta) < 2.2, \forall \Delta \in D$. If we can tolerate such loss in performance then we have validated the PBA and, indirectly, the associated physical architecture. Conversely, if the loss in performance cannot be tolerated we either have to consider a different RBA or PBA, e.g., by varying the number of bits and, hence, the size of the perturbation domain.

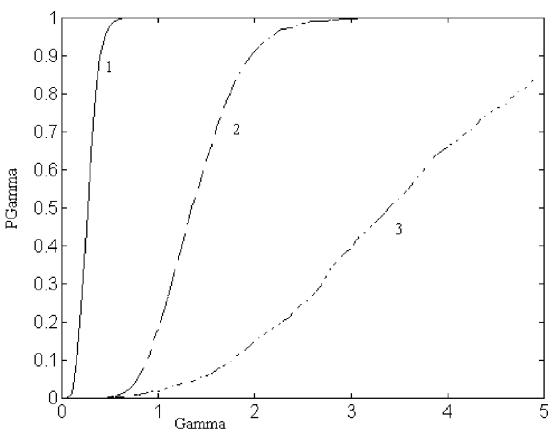

Fig. 8. $\hat{p}_N = \hat{p}_N(\gamma)$. D as in Table 3.

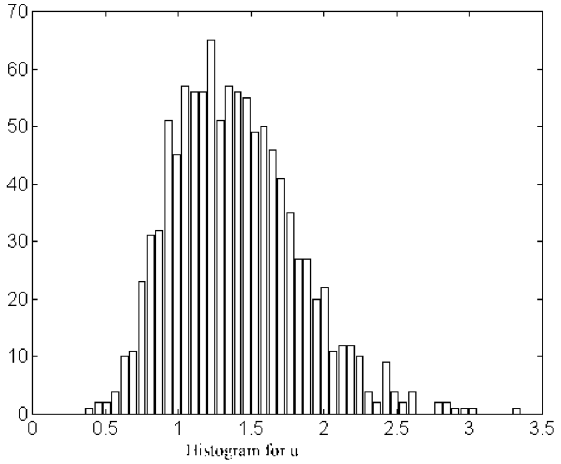

Fig. 9. The histogram of the performance loss for the Experiment 2, Table 3.

TABLE 4
Accuracy, Confidence and N for Experiments 2 and 3

Experiment	Accuracy ε	Confidence 1-δ	Samples N
Experiment 1	5%	99%	1060
Experiment 2	5%	99.99%	1981
Experiment 3	1%	99.9%	38005

Since the Chernoff bound requires N samples to be extracted from D, it is interesting to study the sensitivity of $\hat{p}_N = \hat{p}_N(\gamma)$ with respect to different extractions of $N = 1,060$ samples.

Four runs have been considered with results provided in Table 2. The average of $\boldsymbol{u}$ with respect to the perturbation samples are almost identical as well the maximum values: At least for the chosen RBA, the procedure is robust with respect to the different samplings; differences are obviously associated with the particular realization of N samples and the considered accuracy.

The respective $\hat{p}_N = \hat{p}_N(\gamma)$ curves are given in Fig. 7. We note that such curves are nicely compact to evidence a small dependency in the specific realization of the N sampling.

As a second relevant example, we considered the best architecture selection problem (problem 3). We compared three physical architectures mapped onto three different PBAs. PBAs differ in the size of D, namely in the number of bits q we will consider to represent the decimal parts of variables. The dimension of D is given in Table 3.

We kept the same experimental setup in terms of accuracy and confidence. The resulting $\hat{p}_N = \hat{p}_N(\gamma)$ functions are given in Fig. 8; a number on the function characterizes the experiment. We note that, when high precision is considered (e.g., Experiment 1) the curve increases rapidly and $\bar{\gamma}$ is very small. In such a case, we can assert that the computation is extremely robust and all perturbations will, with high probability, induce a small loss in performance.

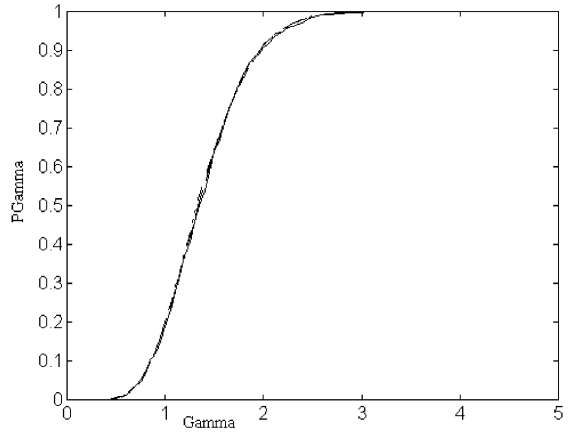

Fig. 10. $\hat{p}_N = \hat{p}_N(\gamma)$. Experiments 2 and 3. Accuracy, confidence, and N as suggested in Table 4.

Conversely, if quantization becomes severe, e.g., as it happens in Experiment 3, $\bar{\gamma}$ is very large (in the plot outside the definition interval). Since the best architecture is the one characterized by the smallest among the $\bar{\gamma}$s (as suggested in the procedure given in Fig. 5) we should select the first architecture as the most robust and performing one. Note that the methodology has not been developed for selecting the best architecture but for testing the architecture loss in performances in polynomial time. Architecture selection among a set of given architectures is a nice consequence.

With respect to the second experiment, we evaluated the frequency of the performance loss in correspondence with the different perturbations. The frequency is given in Fig. 9. A priori hypothesis about the error distribution as done in the literature is therefore unacceptable: The frequency strongly depends on inputs, PBA, and D. We observe that the probability of selecting a perturbation inducing a very high loss performance is low. As a further experiment, we tested the sensitivity of the given application and architecture with respect to different ε and δ degrees (we required in the methodology ε and δ to be sufficiently small). We considered three experiments with ε and δ as given in Table 4.

Results are given in Fig. 10. We observe that the methodology is almost insensitive to ε and δ variations in this application. This information grants that the obtained results hold with high accuracy ($\varepsilon = 0.001$) and confidence ($1 - \delta = 0.999$). Conversely, in this PBA, we obtain good reasonable robustness estimates also with a limited number of samples. The saved simulation time can be used by the designer to test other architectures, still keeping under control the time to market.

6 Conclusions

This paper provides a methodology for analyzing the impact of perturbations on performance for a generic Lebesgue-measurable computation (basically all functions involved in signal/image processing are Lebesgue-measurable). The associated robustness problem, whose solution is computationally intractable, can be nicely addressed with a poly-time procedure based on Randomized Algorithms. Behavioral perturbations have been considered to keep the analysis technology and finite precision representation independent and characterize the robustness of an application/architecture. The suggested methodology allows selecting the best architectural solution among a set of possible candidates as well as quantifying the impact of perturbations affecting a given architecture at the very high level of the design cycle.

ACKNOWLEDGMENTS

The author would like to thank Professor Roberto Tempo, CNR Turin, Italy, for the pleasant and fruitful discussions, which led to this paper, Professor Mariagiovanna Sami, Politecnico di Milano, Italy for her suggestions, and the reviewers whose sharp comments and hints significantly helped to improve the manuscript.

REFERENCES

[1] C.V. Stewart, "Bias in Robust Estimation Caused by Discontinuities and Multiple Structures," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 8, Aug. 1997.
[2] S. Dutt and F.T. Assaad, "Mantissa-Preserving Operations and Robust Algorithm-Based Fault Tolerance for Matrix Computations," *IEEE Trans. Computers,* vol. 45, no. 4, Apr. 1996.
[3] Y.G. Saab, "A Fast and Robust Network Bisection Algorithm," *IEEE Trans. Computers,* vol. 44, no. 7, July 1995.
[4] C. Alippi, "Some Guidelines to Enhance Application-Level Robustness in Linear Computation," *Proc. IEEE Midwestern Symp. Circuits and Systems (MWSCAS),* Aug. 2000.
[5] C. Alippi and F. Balordi, "A Perturbation Size-Independent Analysis of Robustness in Feedforward Neural Networks by Randomized Algorithms," *Proc. Int'l. Conf. Modelling, Control, and Automation (CIMCA),* July 2001.
[6] P. Koopman, "Embedded Systems Design Issues (the Rest of the Story)," *Proc. IEEE Int'l Conf. Computer Design (ICCD),* 1996.
[7] C. Alippi and L. Briozzo, "Accuracy vs. Precision in Digital VLSI Architectures for Signal Processing," *IEEE Trans. Computers,* vol. 47, no. 4, Apr. 1998.
[8] S. Piché, "The Selection of Weights Accuracies for Madalines," *IEEE Trans. Neural Networks,* vol. 6, no. 2, Mar. 1995.
[9] G. Dundar and K. Rose, "The Effects of Quantization on Multilayer Neural Networks," *IEEE Trans. Neural Networks,* vol. 6, no. 6, Nov. 1995.
[10] M. Stevenson, R. Winter, and B. Widrow, "Sensitivity of Feedforward Neural Networks to Weights Errors," *IEEE Trans. Neural Networks,* vol. 1, no. 1, Mar. 1990.
[11] J. Holt and J. Hwang, "Finite Precision Error Analysis of Neural Network Hardware Implementations," *IEEE Trans. Computers,* vol. 42, no. 3, Mar. 1993.
[12] C. Alippi, V. Piuri, and M. Sami, "Sensitivity to Errors in Artificial Neural Networks: A Behavioural Approach," *IEEE Trans. Circuits and Systems—Part 1,* vol. 42, no. 6, June 1995.
[13] F. Zhou and P. Kornerup, "High Speed DCT/IDCT Using a Pipelined CORDIC Algorithm," *Proc. 12th Symp. Computer Arithmetic (ARITH-12),* 1995.
[14] M. Wosnitza, M. Cavadini, M. Thaler, and G. Troster, "A High Precision 1024-Point FFT Processor for 2D Convolution," *Proc. IEEE Int'l Solid-State Circuits Conf.,* 1998.
[15] E.D. Sontag, "VC Dimension of Neural Networks," *Neural Networks and Machine Learning,* 1998.
[16] M. Vidyasagar, "An Overview of Computational Learning Theory and Its Applications to Neural Network Training" *Identification, Adaptation, Learning, NATO ASI Series F,* vol. 153, pp. 400-422, 1996.
[17] M. Vidyasagar, "Statistical Learning Theory and Randomized Algorithms for Control," *IEEE Control Systems,* pp. 69-85, Dec. 1998.
[18] R. Tempo and F. Dabbene, "Probabilistic Robustness Analysis and Design of Uncertain Systems," *Progress in Systems and Control Theory,* vol. 25, pp. 263-282, 1999.
[19] Š. Raudys, "On Dimensionality, Sample Size, and Classification Error of Nonparametric Linear Classification Algorithms," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 6, June 1997.
[20] M. Vidyasagar, *A Theory of Learning and Generalization with Applications to Neural Networks and Control Systems.* Berlin: Springer-Verlag, 1996.
[21] E. Bai, R. Tempo, and M. Fu, "Worst-Case Properties of the Uniform Distribution and Randomized Algorithms for Robustness Analysis," *Proc. IEEE American Control Conf.,* pp. 861-865, 1997.
[22] G. Calafiore, F. Dabbene, and R. Tempo, "Uniform Sample Generation of Vectors in l_p Balls for Probabilistic Robustness Analysis," *Recent Advances in Control,* Apr. 1999.
[23] X. Chen and K. Zhou, "On the Probabilistic Characterization of Model Uncertainty and Robustness," *Proc. IEEE 36th Conf. Decision and Control,* pp. 3816-3821, 1997.
[24] P. Djavdan, H. Tulleken, M. Voetter, H. Verbruggen, and G. Olsder, "Probabilistic Robust Controller Design," *Proc. IEEE 28th Conf. Decision and Control,* pp. 2144-2172, 1989.
[25] W.K. Pratt, *Digital Image Processing.* Wiley Interscience, 1978.
[26] C. Alippi, "FPE-Based Criteria to Dimension Feedforward Neural Networks" *IEEE Trans. Circuits and Systems—Part 1,* vol. 46, no. 8, Aug. 1999.
[27] L. Ljung, *System Identification, Theory for the User.* Prentice-Hall, 1987.
[28] S. Geman, E. Bienenstock, and R. Doursat, "Neural Networks and the Bias/Variance Dilemma," *Neural Computation,* vol. 4, pp. 1-58, 1992.
[29] C. Alippi, S. Ferrari, V. Piuri, M. Sami, and F. Scotti, "New Trends in Intelligent System Design for Embedded and Measurement Applications" *IEEE I&M Magazine,* vol. 2, no. 2, June 1999.
[30] J. Hertz, A. Krogh, and R.G. Palmer, Introduction to the Theory of Neural Computation. Addison-Wesley, 1991.
[31] M.H. Hassoun, *Fundametals of Artificial Neural Networks.* MIT Press, 1995.
[32] Y. Mallet, D. Coomans, J. Kautsky, and O. De Vel, "Classification Using Adaptive Wavelets for Feature Extraction" *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 10, Oct. 1997.
[33] H. Dawid and H. Meyr, "The Differential CORDIC Algorithm: Constant Scale Factor Redundant Implementation without Correcting Iterations," *IEEE Trans. Computers,* vol. 45, no. 3, Mar. 1996.
[34] Y.H. Hu and H.H.M. Chern, "A Novel Implementation of CORDIC Algorithm Using Backward Angle Recoding (BAR)," *IEEE Trans. Computers,* vol. 45, no. 12, Dec. 1996.
[35] M.J. Buckingham, *Noise in Electronic Devices and Systems.* Chichester: Horwood, 1983.
[36] *Hardware/Software Co-Design, Nato ASI Series, Series E: Applied Sciences,* vol. 310, G. De Micheli and M. Sami, eds., Kluwer Academic, 1996.
[37] S. Amari, "Statistical and Information-Geometrical Aspects of Neural Learning," *Computational Intelligence: A Dynamic Perspective,* pp. 71-82, 1995.
[38] J. Dugundji, *Topology.* Boston, Mass.: Allyn and Bacon, 1966.
[39] H. Chernoff, "A Measue of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations," *Annals Math. Statistics,* vol. 23, pp. 493-507, 1952.

Cesare Alippi received the Dr Ing degree in electronic engineering summa cum laude in 1990 and the PhD in computer engineering in 1995, both from Politecnico di Milano, Milano, Italy. He has done research work in computer science at the University College London and the Massachussetts Institute of Technology. Currently, he is an associate professor in information processing systems at the Politecnico di Milano and a research fellow with the Italian National Research Council. His interests include neural networks (learning theories, implementation issues, and applications), genetic algorithms, signal and image processing, and VLIW architectures. His research results have been published in more than 70 technical papers in international journals and conference proceedings. He is a senior member of the IEEE and a member of the IEEE Computer Society.