Generalized Hopfield Neural Network for Concurrent Testing

Julio Ortega, Alberto Prieto, Member, IEEE, Antonio Lloris, and Francisco J. Pelayo

Abstract—This paper deals with testing of digital circuits, spectral techniques, and neural networks: It shows how an optimization problem that appears in the design of concurrent testable circuits can be expressed by using digital spectral techniques in such a way that makes it possible to compute an acceptable solution by using neural networks.

To build a concurrent testable circuit, it is necessary to use some specific checking circuitry for analyzing the outputs of that circuit while it is in operation. The goal is to check the circuitry that, with less hardware complexity than a given bound, allows us to detect as many errors as possible at the outputs of the circuit under test. This paper provides a measure, the aliasing probability, to evaluate the performances of the checking circuitry. It also shows how, by using spectral techniques based on the Reed–Muller transform, the aliasing probability can be expressed as a function of the Reed–Muller coefficients. In this way, to obtain the checking circuitry means to select a set of Reed–Muller spectral coefficients, with less elements than a given bound, that minimizes the aliasing probability: It is an NP-complete problem as in the majority of combinatorial optimization problems.

It has been suggested [1] that neural networks, in particular the Hopfield neural network, may be used to solve combinatorial optimization problems, as it always moves toward decreasing an energy function that is quadratic in terms of the processor states. To apply the neural networks to design the checking circuit for concurrent testing, the aliasing probability has been used as an energy function, and the Hopfield neural network has been modified to have an associated energy function with any type of polynomial dependence on the processor states. This paper analyzes the so-called generalized Hopfield neural network, considers its application to solve optimization problems with an objective function taking any nonlinear form, and, as an example, uses the generalized Hopfield network in the design of the extra modules of a concurrent testable circuit.

Index Terms—Aliasing probability, combinatorial optimization, concurrent testing, fault tolerance, Hopfield neural networks, NP-completeness, Reed–Muller spectrum, spectral techniques.

NOMENCLATURE

\( m \) number of inputs of a CUT
\( n \) number of outputs of a CUT
\( r \) number of functions in the extraction module
\( y = (y_1, \ldots, y_m) \) inputs of a CUT
\( x = (x_1, \ldots, x_n) \) outputs of a CUT (inputs to the extraction module)
\( \delta x = (\delta x_1, \ldots, \delta x_r) \)
\( e = (e_1, \ldots, e_n) \)
\( F(y) = f_1(y), \ldots, f_n(y) \)
\( H(y) = h_1(y), \ldots, h_r(y) \)
\( G(x) = g_1(x), \ldots, g_r(x) \)
\( p(x, e) \) probability that output \( x \) is affected by the error \( e \)
\( p_A \) aliasing probability of the extraction module
\( p_{Ai} \) aliasing probability of the extraction module function \( g_i \)
\( p'_{Ai} \) aliasing probability modified with constraints

\[ C_2 = \{0, 1\} \]
\[ C_m^2 = C_2 \times C_2 \times \cdots C_2 \]
\[ F_2^m \]
\[ \oplus \]
\[ \odot \]
\[ (x_i^{-1}) \]
\[ V(F_2^m, C_2, \oplus, \odot) \]
\[ T_i(y) (i = 0, \ldots, 2^m - 1) \]
\[ \alpha_i(y) (i = 0, \ldots, 2^m - 1) \]
\[ \phi(i_1, \ldots, i_j) \]

functions that map \( C_2^m \) onto \( C_2 \)
module-2 sum
module-2 product
complement of the binary \( x_i \)
vector space matrix product in \( V(F_2^m, C_2, \oplus, \odot) \)
Reed–Muller function element of the Reed–Muller spectrum
detection potential of the product of spectral coefficients \( \alpha_1, \alpha_2, \ldots, \alpha_j \).

I. INTRODUCTION

Since defects, either permanent or temporary [2], may arise at any time in the life of an integrated circuit, and reliable electronic systems must be available, circuits should not only be tested at the time of manufacturing, but should preferably be checked periodically or even constantly as they are working. The off-line built-in self-test (BIST) techniques [3], [4] make periodical testing feasible, as well as improve access to the different modules in an integrated circuit. Temporary or intermittent defects, however, may not
be detected by using these techniques but require instead a concurrent test. Bearing in mind that as the integrated circuits become smaller and more complex, intermittent defects occur more frequently [5], so the need to develop efficient techniques for concurrent testing also increases [6]–[12]. It is essential for systems that must operate in noisy environments, or which are required to be extra secure, that operation may take place reliably even in the presence of faults and that the systems automatically stop working without causing harmful effects in their environments in an extreme case (fault tolerant systems) [2], [12]–[16]. Moreover, the development of wafer scale integration (WSI) technology has rendered fault tolerance indispensable for any system included in a single wafer as a means of increasing the yield of the manufacturing process [17].

A fault tolerant system, if working correctly, should have the following capabilities [12]; a) to detect the presence of faults; b) to make the defect diagnosis, that is, to locate the defect; c) to reconfigure itself in such a way that the defective part is isolated; and d) to recover from error, that is, to cancel any effects the defect may have caused prior to detection. From item a, the capability of detecting an error when it occurs, or as soon as possible, is therefore a starting point for fault tolerance, and concurrent testing is the way to detect the circuit malfunction immediately.

There are many publications on concurrent error detection, which mostly concentrate on the design of checkers for different codes and for simple functional units with coded inputs and outputs [9], [18]–[24]. However, few methods have been developed in the area of concurrent testing of arbitrary circuits with uncoded inputs and outputs. Among such methods, there is a general one for predicting output parity in a combinational circuit [7], a self-checking general prediction circuit that provides detection of multiple errors [8], and an extended-parity checking technique for detecting an even number of errors [25]. In a more recent paper [10], the circuitry used for BIST off-line testing was used with some additional modules for concurrent testing.

This paper reports on the design of the checking circuitry with hardware complexity below a given bound that allows us to detect the highest number of errors at the outputs of the circuit under test. With the aid of spectral techniques based on the Reed–Muller spectrum, the design of such checking circuitry has been transformed into a combinatorial optimization problem which, as it is shown in this paper, is also an NP-complete problem [26]. In NP-complete problems, only approximate solutions may be found to be within reasonable computing times for any problem size because the computational complexity involved in their calculation grows exponentially with that size. Among the procedures proposed for acceptable solutions to NP-complete optimization problems are the neural networks [27]–[31], especially the Hopfield network. Nevertheless, as it has been claimed [32], the linear summation of inputs to the neurons of the Hopfield neural networks is not able to model the nonlinear interactions among neurons that some optimization problems require. Consequently, a nonlinear neural framework, called the generalized Hopfield network (GHN), has been introduced [32]. Here, the general structure of a GHN is presented, and it is considered its use for solving combinatorial optimization problems, as the design of the extra modules for concurrent testable circuits, where the function to minimize can take any nonlinear form (not only quadratic).

Following this introduction, Section II presents the modules of a concurrent testable circuit according to the scheme proposed here and establishes the equations for designing the extraction module in terms of Reed–Muller spectral coefficients. Section III deals with the description of a generalized Hopfield neural network, provides its use in the design of the extraction module, and shows how the GHN weights and thresholds are calculated to provide an acceptable solution for designing this module. The experimental results obtained for various examples and the performances of the proposed method are given in Section IV, and Section V provides some conclusions.

II. CONCURRENT TESTABLE CIRCUITS

In this section the basic background related to the design of concurrent testable circuits is presented.

A circuit under test (CUT) and the changes required to make it concurrently testable are shown in Fig. 1. The new outputs for the concurrent testable circuit are built by using the outputs \( x = (x_1, \ldots, x_n) \) of the initial CUT and some additional outputs \( \delta x = (\delta x_1, \ldots, \delta x_m) \), which are calculated in such a way that \( x^* = (x, \delta x) \) is a word of a given code C. The code checker circuit (CC) in the figure determines whether output \( (x, \delta x) \) belongs to code C or not; should the output \( x^* \) not belong to code C, the CC circuit will generate an error signal.

If there is some defect in the CUT, it could produce changes on its outputs, \( x = (x_1, \ldots, x_n) \), for some of its input patterns, \( y = (y_1, \ldots, y_m) \). The changes on the outputs, that will be noted as \( e = (e_1, \ldots, e_n) \) hereafter, are called errors and the input patterns that allow the errors to be present at the outputs are called test patterns for the corresponding defect. Thus, when the output \( x \) is affected by an error \( e \) due to a defect in the CUT, the actual CUT output is \( x \oplus e = (x_1 \oplus e_1, \ldots, x_n \oplus e_n) \) and the CC generates the error signal if it is verified that the \( (x \oplus e, \delta x) \) does not belong to code C.

Because the CC circuit must be designed to identify any possible erroneous outputs of the CUT, it is necessary to have some information about the errors that could be observed at the outputs of the CUT. Such information is given by the so-called error model, which is defined by the set of probabilities.
and the prediction and pattern monitor module corresponds to the circuitry that generates the bits $\bar{e}$. The extraction and prediction modules synthesize the functions $G(x) = (g_1(x), \ldots, g_n(x))$ and $H(y) = (h_1(y), \ldots, h_n(y))$, respectively, verifying that $H(y) = G(x)$, that is, $h_i(y) = g_i(x)$ for $i = 1, \ldots, r$, if $x = F(y)$ and $y = (y_1, \ldots, y_m)$ is a test pattern. The comparision module generates an error signal when $G(F(y)) \neq H(y)$ for a test pattern $y$. This system will be self-checking for a set of faults $FL$ [6], if:

$$\forall \text{ fault } F \in FL, \text{ an input } y, \text{ exists such that } H(y) \neq G(F(y))$$

and will be fault secure for $FL$ [12] if, being $F^*(y)$ the faulty function:

$$\forall F \in FL, \text{ when } F(y) \neq F^*(y) \text{ then } G(F^*(y)) \neq H(y).$$

To ensure (1), each fault of $FL$ must have at least one test pattern, and to ensure (2), the extraction circuit must not mask any error in the outputs of the CUT. The probability that an error in the CUT outputs will not be detected by the extraction module is known as aliasing probability, $p_A$ [33].

An expression to evaluate that probability is given in what follows.

A. Aliasing Probability for Extraction Module

An extraction module with $r$ outputs, $g_i(x)$ ($i = 1, \ldots, r$), may not establish the presence of an error $e = (e_1, \ldots, e_n)$ for an output $x = (x_1, \ldots, x_n)$ of the correct circuit, when it is verified that

$$g_i(x) \oplus g_i(x \oplus e) = 0, \forall i = 1, \ldots, r.$$  

Thus, for any extraction function, $g_i$, it is possible to define a function of the outputs and the errors, $A_i(x, e)$, which will be 1 if the error $e$ is not detected by $g_i(x)$, and 0 otherwise. So, considering (3), it is verified that

$$A_i(x, e) = 1 - g_i(x) \oplus g_i(x \oplus e).$$

The function $A_i$ is the so-called aliasing function for $g_i$. As $A_i(x, e) = 1$ implies that the probability $p(x, e)$ contributes to the aliasing probability for the function $g_i(x)$, the aliasing probability for the $i$th output of the extraction module, $p_{Ai}$, can be expressed by

$$p_{Ai} = \sum_{x} \sum_{e \neq 0} p(x, e) A_i(x, e).$$

Thus, by using the definition of $A_i(x, e)$,

$$p_{Ai} = 1 - \sum_{e \neq 0} p(x, e) (g_i(x) \oplus g_i(x \oplus e)).$$

As the $g_i$ functions comprising the extraction module are independent, the aliasing probability for the extraction module, $p_A$, is obtained from the values of $p_{Ai}$ by using

$$p_A = \prod_{i=1}^{r} p_{Ai}. $$
The aim is to minimize (6), as well as to attempt to minimize the hardware complexity of the circuit implementing the extraction module: The number of different functions, $g_i$, must be as low as possible (i.e., $p_A$ and $r$ must be minimized). Keeping this in mind, the process to obtain an extraction module with an aliasing probability less than $P_A$, with $r$ less than $n$, is specified in the following algorithm.

**Algorithm 1**

1. Do $i = 1$, and fix $P_A$ as the maximum admissible value for $P_A$.
2. While ($p_A > P_A$ and $i \leq n$) do:
   3. Find the function $g_i$ which minimizes (5).
   4. Obtain the value of $p_{Ai}$ corresponding to $g_i$.
   5. Evaluate (6) using $p_{Ai}$, ..., $p_{Ai}$.
   6. Increment $i$.

The nucleus of this algorithm is Step 3. To elaborate a procedure for minimizing $p_{Ai}$, (5) must be expressed in an operative way. To do that, we use the Reed–Muller spectral coefficients [34]–[36], which allow any binary function to be expressed univocally. In this way, Step 3 involves finding the Reed–Muller spectrum of the function $g_i$ which minimizes (5).

The next section deals with the Reed–Muller spectrum.

**B. Reed–Muller Spectrum**

Combinational binary circuits with $m$ inputs, $y_1, \ldots, y_m$, and one output synthesize functions that map elements of $C^m_2 = C_2 \times x_1 \times \cdots \times C_2$, onto the elements of $C_2 = \{0,1\}$. The set of all those functions will be noted as $F^m_2$. It is possible to define the vector space $V(F^m_2, C_2, \oplus, \odot)$, $\oplus$ being the module-2 sum and $\odot$ the module-2 product both over $C_2$, whose vectors are built from the minterm description of the $F^m_2$ functions. This vector space has a base comprising the $2^m$ functions [35], [36]:

$$T_i(y) = y_1^i \odot y_2^j \odot \cdots \odot y_m^j,$$

(7)

where

$$i = i_1 + 2i_2 + 2^2i_3 + \cdots + 2^{m-1}i_m, \quad i_j \in C_2$$

$$y = (y_1, y_2, \cdots, y_m) \in C^m_2, \quad y_j \in C_2$$

$$y_1^0 = 1, \quad y_j^0 = y_j.$$

They are the so-called Reed–Muller functions, and as they define a base for $V(F^m_2, C_2, \oplus, \odot)$, any function belonging to $F^m_2$ may be univocally expressed as

$$f(y) = \bigoplus_{i=0}^{2^m-1} \alpha_i \odot T_i(y) = \bigoplus_{i=0}^{2^m-1} \alpha_i \odot (y_1^i \odot y_2^j \odot \cdots \odot y_m^j)$$

$$\alpha_i \in C_2.$$  

(8)

The expansion of $f \in F^m_2$ provided in (8) gives the Reed–Muller representation with zero polarity of the said function [34], [35]. The coefficients of such an expansion, which shall be named either Reed–Muller coefficient (RMC) or Reed-Muller spectrum, can be determined by a transform defined by the matrix $R(2, m)$ obtained from $m$ times the Kronecker product, $\otimes$, of the matrix $R(2, 1)$ by itself:

$$R(2, m) = R(2, 1) \otimes \ldots \otimes R(2, 1)$$

(9)

where

$$R(2, 1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

This way it is possible to define the RM expansion of zero polarity as a special type of linear transform, similar to the Fourier transform or the Walsh–Hadamard transform. As an example, for binary functions of three variables, $f \in F^3_2$, the transformation matrix is

$$R(2, 3) = R(2, 1) \otimes R(2, 1) \otimes R(2, 1)$$

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}$$

and the functions $T_2(y), T_3(y)$, and $T_4(y)$ are

$$T_2(y) = y_2, \quad T_3(y) = y_1 \oplus y_3, \quad T_4(y) = y_1 \oplus y_2 \oplus y_3.$$

Thus, given a function, $f \in F^3_2$, for example, $f(y_1, y_2, y_3) = y_1 \oplus (y_2 \oplus y_3)$, the RMC's are obtained from the vector defined by the description of the function in terms of its minterms, $[f(y)] = [0 1 1 1 0 1 0 1 1]$, multiplied by $R(2, 3)$:

$$[f(y)].R(2, 3) = [0 1 1 1 0 0 1 1 1]$$

the dot (.) standing for the matrix product defined in $V(F^3_2, C_2, \oplus, \odot)$. Thus, for that function, the Reed–Muller expansion with zero polarity will be

$$f(y) = y_1 \oplus y_2 \oplus y_1y_2 \oplus y_2y_3 \oplus y_1y_2y_3,$$

where the symbol $\oplus$ for the module-2 product has been suppressed. This notation will be used hereafter.

As the Reed–Muller transform verifies the theorem of Good [35], [36], it is possible to define a fast procedure [35], [36], similar to the fast Fourier transform or the fast Walsh transform, to determine the RMC's of a function $f \in F^m_2$. In [34] and [35], a complete list of properties of the Reed–Muller spectral coefficients is provided.

From (8), any extraction module function, $g_i$, of a CUT with $n$ outputs, $(x_1, \cdots, x_n)$, can be expressed in terms of Reed–Muller coefficients as follows:

$$g_i(x) = \bigoplus_{j=0}^{N-1} \alpha_j T_j(x)$$

(10)

where

$$x = (x_1, \cdots, x_n), j = j_1 + 2j_2 + \cdots + 2^{n-1}j_n, j_k \in \{0,1\}$$

$$T_j(x) = x_1^{j_1} x_2^{j_2} \cdots x_n^{j_n},$$

$$\alpha_j \in \{0,1\}$$ is the $j$th spectral coefficient, and $N = 2^n$. 


If an error in the outputs of the CUT, \( e = (e_1, \ldots, e_n) \), occurs then

\[
g_i(x \oplus e) = \bigoplus_{j=0}^{N-1} \alpha_j T_j(x \oplus e)
\]

(11)

being

\[
T_j(x \oplus e) = (x_1 \oplus e_1)^{j_1}(x_2 \oplus e_2)^{j_2} \cdots (x_n \oplus e_n)^{j_n}.
\]

In addition, using the definition of the Reed-Muller functions (7),

1) \( T_j(x) = \begin{cases} 1, & \text{if } j_k \leq x_k \forall k = 1, \ldots, n \\ 0, & \text{otherwise.} \end{cases} \)

(12)

2) \( T_j(x \oplus e) = \begin{cases} 1, & \text{if } j_k \leq x_k \oplus e_k \forall k = 1, \ldots, N \\ 0, & \text{otherwise.} \end{cases} \)

is verified that

\[
g_i(x) \oplus g_i(x \oplus e) = \bigoplus_{j \in S(x, e)} \alpha_j (T_j(x) \oplus T_j(x \oplus e)) = \bigoplus_{j \in S(x, e)} \alpha_j
\]

(13)

where the set \( S(x, e) \) contains all the indices that verify

a) \( j_k \leq x_k \forall k = 1, \ldots, n \)

or

b) \( j_k \leq x_k \oplus e_k \forall k = 1, \ldots, n, \)

but not conditions a and b at the same time. (14)

By substituting (13) into (5), the expression of \( p_{A_i} \) is obtained in terms of the Reed-Muller coefficients of \( g_i \)

\[
p_{A_i} = 1 - \sum_{x \in S(x, e)} p(x, e) \bigoplus_{j \in S(x, e)} \alpha_j.
\]

(15)

As an example, let \( x = (0, 1, 1) \) and \( e = (1, 0, 1) \). The \( j \) indices that verify condition a in (14) have the form \((0, d, d)\), where \( d \) is a “don’t care.” The values of index \( j \) that verify condition b in (14) have the form \((d, d, 0)\) since \( x \oplus e = (1, 1, 0) \), thus, those that verify only one of two conditions are \((0, d, 1)\) and \((1, d, 0)\); that is \( S((0, 1, 1), (1, 0, 1)) = \{(0, 0, 1), (0, 1, 1), (1, 0, 0), (1, 1, 0)\} \).

This way, to obtain the function \( g_i \) with the lower value for its aliasing probability means to select the values of the Reed-Muller coefficients that minimize (15). These coefficients allow one to define the required \( g_i \). As it is proved in the following theorem, the RMC selection problem is an NP-complete problem.

**Theorem 1:** To compute the Reed-Muller coefficients, \( \alpha_j (j = 0, \ldots, N - 1) \), that minimize (15) is an NP-complete problem.

**Proof:** To minimize \( p_{A_i} \) implies computing the values for the aliasing functions of (4), \( A_i(x, e) \), for any pair \((x, e)\).

Supposing that such values are known, to obtain the values for the Reed-Muller coefficients is equivalent to solving

\[
\bigcup_{x, e \in \Phi} \left[ (1 - A_i(x, e)) \bigoplus_{j \in S(x, e)} \alpha_j \right] = 1
\]

which is an NP-complete problem [37]-[39].

III. **GENERALIZED HOPFIELD NETWORK**

It has been shown that neural networks, particularly the Hopfield network, could provide good solutions to NP-complete problems, such as the traveling salesman problem [29]. Although these solutions may not be the best solution for a given problem, they can be computed rapidly and be very useful to solve certain difficult problems where the time to obtain a solution is limited. Our proposal is to solve the extraction module design problem by using neural networks, thus this section gives a brief description of the Hopfield neural network and its convergence properties and describes an extension of that network, the so-called generalized Hopfield network.

The original model of a Hopfield neural network uses two-states threshold neurons, each neuron \( i \) having an output \( V_i \) with two possible values \( V_i^0 \) and \( V_i^1 \), which usually are taken as 0 and 1, respectively, and inputs from the other neurons of the network and from external inputs, \( I_i \). The total input to the neuron \( i \) is given by

\[
\sum_{j \neq i} T_{ij} V_j + I_i,
\]

(16)

where the element \( T_{ij} \) is the interconnection weight from neuron \( j \) to neuron \( i \).

The threshold associated to neuron \( i \) is \( U_i \), and its output changes according to a threshold rule

\[
V_i \rightarrow V_i^0 \text{ if } \sum_{j \neq i} T_{ij} V_j + I_i < U_i
\]

\[
V_i \rightarrow V_i^1 \text{ if } \sum_{j \neq i} T_{ij} V_j + I_i > U_i
\]

(17)

that is sampled at random times. As the sampling times of each neuron are independent of the times at which other neurons are interrogated, the network functions asynchronously. It is possible to associate an appropriate energy function to the network that is always decreased by any state change produced by the algorithm. The energy function is

\[
E = -\frac{1}{2} \sum_{i,j \neq i} T_{ij} V_i V_j - \sum_i (U_i - V_i)
\]

(18)

and its change, \( \delta E \), due to changing in the state of the neuron \( i, \delta V_i \), is given by

\[
\delta E = - \sum_{j \neq i} T_{ij} V_j \delta V_i - (I_i - U_i) \delta V_i
\]

(19)
where $T_{ij} = T_{ji}$ and $T_{ii} = 0$ have been supposed. According to (17), $\delta V_i$ is positive only when the expression into the bracket is positive. As any change in $E$ under the algorithm is negative and $E$ is bounded because there are a finite number of possible states, the network moves in the direction of the decreasing energy function and finds a state that is a local minimum for $E$. This network has been used in several practical applications, including associative memory and optimization [1], [28]–[31]. Its application to optimization is nevertheless restricted to problems whose cost function can be identified with a quadratic energy function. In this section, we use a generalized Hopfield neural network to solve optimization problems with an associated energy function that depends not only on the state of a single neuron but also on the state of groups of two or more neurons.

Given a set of $N$ binary elements $V_1, \ldots, V_N$ with $V_i \in \{0, 1\}$, a generalized optimization problem involves determining the values for the $V_i$’s which minimize the energy function:

$$E = E_0 - \sum_{i=1}^{N} B_i V_i - \sum_{i=1}^{N} \sum_{j=1}^{N} B_{ij} V_i V_j$$
$$- \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} B_{ijk} V_i V_j V_k - \cdots$$ (20)

Because there is a finite number of binary combinations for the $V_i$’s, the existence of minima in the energy is guaranteed. The change on the energy function due to a change in the value of the element $V_i$ is given by

$$\frac{\delta E}{\delta V_i} = - \left( T_i + \sum_{j=1}^{N} T_{ij} V_j + \frac{1}{2} \sum_{j=1}^{N} \sum_{k=1}^{N} T_{ijk} V_j V_k + \cdots \right),$$ (21)

where $T_{i\ldots k} = (n!)/B_{i\ldots k}$, and it has been supposed that the coefficients $T$ with repeated indices are zero and all the coefficients $T$ with the same, but permuted, set of indices are equal.

Thus, to decrease the energy function ($\delta E < 0$),

$$- \left( T_i + \sum_{j=1}^{N} T_{ij} V_j + \frac{1}{2} \sum_{j=1}^{N} \sum_{k=1}^{N} T_{ijk} V_j V_k + \cdots \right) \delta V_i < 0$$ (22)

must be verified, and so the activation law for the neurons is

$$V_i \to 0 \text{ if } -T_i \geq \left( \sum_{j=1}^{N} T_{ij} V_j + \frac{1}{2} \sum_{j=1}^{N} \sum_{k=1}^{N} T_{ijk} V_j V_k + \cdots \right)$$

where $T_i = I_i - U_i$, $U_i$, and $I_i$ being the threshold and the external input of the neuronal element $i$, respectively. As can be seen from (23), each neuron not only has inputs from other neurons but also from products of the outputs of neurons. Thus, the coefficient $T_{i\ldots j}$ represents a weight from the product of the neurons $j_1, \ldots, j_s$ to the neuron $i$. If all the coefficients $T_{i\ldots j\ldots k}$ with $s > 2$ are zero, the energy function (20) has a quadratic dependence on the $V$’s and corresponds to a Hopfield neural network.

Fig. 4 shows the structure of a network of analog processors that model the neurons. Now, each neuron $i$ has an analog output, called $V_i$, as before, which corresponds to an amplifier implementing a sigmoid monotonic input-output relation, $V_i = \gamma(v_i)$, and changes according to the resistance-capacitance charging equation:

$$C_i \left( \frac{dV_i}{dt} \right) = I_i + T_i V_i + \sum_{j} T_{ij} V_j$$
$$+ \frac{1}{2!} \sum_{j} \sum_{k} T_{ijk} V_j V_k + \cdots$$ (24)

where

$$T_i = - \frac{1}{R_i} - \sum_{j} T_{ij} - \frac{1}{2!} \sum_{j} \sum_{k} T_{ijk} \cdots$$ (25)

and it has been supposed that the weights are invariants to permutations in the indices. The network implements the feedback products of outputs of neurons, $V_1, V_2, \ldots$, by using threshold gates as AND gates—the function AND is linearly separable. The network has been called the generalized Hopfield network because the inputs to each neuron of the network not only come from the outputs of other neurons but also from the product of several outputs.

The energy function for this GHN is given by

$$E = E_0 - \int_{I_0}^{I_{V_i}} T_i \gamma^{-1}(V) dV - \sum_{i=1}^{N} I_i V_i$$
$$- \frac{1}{2!} \sum_{i=1}^{N} \sum_{j=1}^{N} T_{ij} V_j$$
$$- \frac{1}{3!} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} T_{ijk} V_j V_k - \cdots$$ (26)

and its time derivative, considering a symmetrical $T$ against permutations in their indices, is given by

$$\frac{dE}{dt} = - \sum_{i} \frac{dV_i}{dt} \left( I_i + T_i V_i + \sum_{j} T_{ij} V_j + \frac{1}{2!} \sum_{j} \sum_{k} T_{ijk} V_j V_k + \cdots \right)$$ (27)
where $W_j = \binom{2^n}{(S-x)j}$ is the cardinal of $S(x,e)$, and $\mu_k$ is a function that represents the $k$th combination of the $S$ indices, $k$, of the elements $\alpha_k \in S(x,e)$ taken $j$ by $j$.

Thus,

$$p_{A_i} = 1 - \sum_{x \in \mathbb{S}} \sum_{e \neq 0} p(x,e) \left( \sum_{j \in S(x,e)} (-2)^{j-1} \left( \frac{W_j}{\sum_{s=1}^{W_j} \alpha_{\mu_{\alpha_s}(1)} \cdots \alpha_{\mu_{\alpha_s}(j)}} \right) \right) \quad (30)$$

There are only arithmetic sums and products in (30), which may be transformed by regrouping the terms and defining

$$\phi(\mu_{\alpha_s}(1), \cdots, \mu_{\alpha_s}(j)) = \sum_{(x,e) \in R(\mu_{\alpha_s}(1) \cdots \mu_{\alpha_s}(j))} p(x,e) \quad (31)$$

so that

$$p_{A_i} = 1 - \sum_{j=1}^{2^n} \sum_{s=1}^{W_j} (-2)^{j-1} \phi(\mu_{\alpha_s}(1), \cdots, \mu_{\alpha_s}(j)) \quad (32)$$

where $W_j = \binom{2^n}{j}$ and $R(\mu_{\alpha_s}(1), \cdots, \mu_{\alpha_s}(j))$ is the set of all the couples $(x,e)$ with $e \neq 0$ whose probabilities $p(x,e)$ are multiplied in (15) by the EXOR sum of the spectral coefficients $\alpha_{\mu_{\alpha_s}(1)} \cdots \alpha_{\mu_{\alpha_s}(j)}$. The coefficients $\phi(\mu_{\alpha_s}(1), \cdots, \mu_{\alpha_s}(j))$ will be called the detection potential of the spectral coefficients $\alpha_{\mu_{\alpha_s}(1)} \cdots \alpha_{\mu_{\alpha_s}(j)}$. As can be seen, the value of any detection potential depends on the error model that describes the effects of the defects in the CUT.

Variation in aliasing probability owing to a change in the state of the neuron $j$ (spectral coefficient $\alpha_j$) is expressed as

$$\delta p_{A_i} = - \delta \alpha_j \left( \phi(j) + \sum_{k=1}^{2^n} \sum_{s=1}^{W_j} ((-2)^k \phi(\mu_{\alpha_s}(1), \cdots, \mu_{\alpha_s}(k))) \right. \quad (33)$$

To decrease $p_{A_i}$, $\delta p_{A_i} < 0$ must occur, and, therefore,

$$\alpha_j \rightarrow 1 \quad \text{if } \phi(j) > \sum_{k=1}^{2^n} \sum_{s=1}^{W_j} ((-2)^k \phi(\mu_{\alpha_s}(1), \cdots, \mu_{\alpha_s}(k)))$$

$$\alpha_j \rightarrow 0 \quad \text{if } \phi(j) \leq \sum_{k=1}^{2^n} \sum_{s=1}^{W_j} ((-2)^k \phi(\mu_{\alpha_s}(1), \cdots, \mu_{\alpha_s}(k))) \quad (34)$$

Expression (34) defines the activation law of a neuron, and (32) matches the energy function of a GHN, with as many neurons as coefficients could appear in the Reed–Muller spectrum of the function $g_i$, synthesized by the extraction module. In that GHN:

1. The energy corresponds to the aliasing probability $p_{A_i}$
2. The state of the $j$th neuron is the value of the spectral coefficient $\alpha_j$,

A. Synthesis of Extraction Module Using a GHN

Consideration will now be given to the use of a GHN in the search for the Reed–Muller spectral coefficients that minimize (15). In such an expression, the module-2 sum, $\bigoplus_{j \in \mathbb{S}(x,e)} \alpha_j$, appears. It is possible to transform that module-2 sum into an arithmetic sum by using the formula

$$\bigoplus_{j \in \mathbb{S}(x,e)} \alpha_j = \sum_{j \in \mathbb{S}(x,e)} (-2)^{j-1} \left( \sum_{s=1}^{W_j} \alpha_{\mu_{\alpha_s}(1)} \cdots \alpha_{\mu_{\alpha_s}(j)} \right) \quad (29)$$
3. The threshold of the \( j \)th neuron is the detection potential of the coefficient \( \alpha_j, \phi(j) \). Moreover, the weight corresponding to the input \( \alpha_{\mu_j(1)} \cdots \alpha_{\mu_k(j)} \) on the \( j \)th neuron will be \((-2)^t \phi(\mu_1(1) \cdots \mu_k(j))\).

If the detection potentials for the grouping of more than two spectral coefficients, \((-2)^t \phi(\mu_1(1) \cdots \mu_k(t))\) \( t > 1 \), are negligible, then the network reduces to a Hopfield neural network. Fig. 5 shows the GHN whose energy function can be arranged to model the aliasing probability of a CUT with two outputs.

**B. Simplification of GHN**

No constraint has so far been placed on the hardware complexity of the solution, \( g_j \), obtained. Solutions with a high-cost hardware implementation are of little use, although they obtain a very small aliasing probability. The number of nonzero spectral coefficients appearing in the expression of a solution may be used as a definition of its hardware complexity:

\[
C = \sum_{j=1}^{2^n} \alpha_j. \tag{35}
\]

It is necessary to ensure that the solution obtained has a low enough hardware complexity \( C \). So, the problem is to search for the function that minimizes (15), among the set of functions verifying that \( C \leq C_0 \). The standard method to solve this type of problem, namely, to find the minimum of a function in the presence of constraints, is by introducing the constraints via Lagrange multipliers [29]. Here, two Lagrange multipliers must be used such that the new function to be minimized is \( p'_{Ai} \), defined as \( p'_{Ai} = p_{Ai} + \Omega \), where

\[
\Omega = k_1(C - C_0) + k_2(C - C_0)^2. \tag{36}
\]

Note that \( p'_{Ai} \) is no longer a probability because it could be less than 0 and higher than 1.

As can be seen from (36), \( \Omega \) depends on three parameters: \( C_0, k_1, \) and \( k_2 \). The parameter \( C_0 \) is the maximum acceptable hardware complexity for any extraction function expressed in terms of the number of nonzero coefficients in its Reed–Muller expansion. The parameters \( k_1 \) and \( k_2 \) represents the influence of the linear and quadratic terms of \( \Omega \), respectively. Whereas the linear term in (36) reduces the value of \( p'_{Ai} \) for a hardware complexity \( C \) verifying \( C < C_0 \) to drive the network to solution with less hardware, the quadratic term makes \( p'_{Ai} \) higher than \( p_{Ai} \) for \( C > C_0 \), and prevents an extraordinary reduction in \( p'_{Ai} \) for \( C < C_0 \) which would give the trivial case, \( C = 0 \), as the solution. By choosing \( k_2 = -k_1(1 - C_0) \), it is verified that \( \Omega > 0 \) for \( C > C_0 \) and \( C = 0 \), and that \( \Omega \leq 0 \) for \( 1 \leq C \leq C_0 \).

The variation of \( \Omega \) when the coefficient \( \alpha_j \) is changed, is given by

\[
\delta \Omega = \delta \alpha_j \left( k_1 + k_2 (C - C_0) \right) = \delta \alpha_j \left[ \left( k_1 - 2k_2C_0 \right) + 2k_2 \sum_{i=1}^{2^n} \alpha_i \right]. \tag{37}
\]

Note that \( \delta p'_{Ai} - \delta p_{Ai} + \delta \Omega \), the detection potentials that determine the weights and thresholds of the GHN, must be updated as shown below:

\[
\phi'(j) = \phi(j) - (k_1 - 2k_2C_0)
\]

\[
\phi'(j, i) = \phi(j, i) + k_2. \tag{38}
\]

Since there will not be more than \( C_0 \) nonzero coefficients in the solution, the potentials \( \phi(i_1, \cdots, i_{C_0}, j) \) will not influence the value of \( p'_{Ai} \) for the solution and, therefore, they can be set to zero. It was stated in Theorem 1 that the problem of selecting the set of Reed–Muller coefficients that minimize \( p_{Ai} \) is an NP-complete problem. By using a GHN circuit, the time needed to find a solution is reduced thanks to the parallelism that it shows, but as a result, the GHN needed has a hardware complexity that grows exponentially with the size of the problem. Thus, to solve the large-sized instances of NP-complete problems, it is necessary to use heuristics. The simplification done in the weights can be considered as such heuristics. In this way, the solution obtained would not be the best solution but could be good enough, and the hardware complexity of the required GHN is thus reduced considerably as compared with the \( 2^N \) weights that would be needed if no potential were initially annulled. The number of weights required now is

\[
NW(C_0) = \sum_{i=1}^{C_0} \binom{2^n}{i}. \tag{39}
\]
C. Example of Synthesis

This subsection gives a detailed description of a simple example to illustrate the construction procedure and working of the network. Some of the results obtained on applying the procedure to more complex cases are also provided in Section IV. Given a circuit with two outputs, the aim is to find a function for the extraction module, $g_1$:

$$g_1(x_1, x_2) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_1 x_2. \quad (40)$$

Therefore

$$g_1(x_1 \oplus e_1, x_2 \oplus e_2) = \alpha_0 + \alpha_1 (x_1 \oplus e_1) + \alpha_2 (x_2 \oplus e_2) + \alpha_3 (x_1 \oplus e_1) (x_2 \oplus e_2).$$

$$g_i(x) \oplus g_i(x \oplus e) = (\alpha_1 e_1 + \alpha_2 e_2 + \alpha_3 e_1 e_2) + \alpha_3 e_2 x_1 + \alpha_3 e_1 x_2$$

and the expression for the aliasing probability will be

$$p_{AI} = 1 - \sum_x \sum_{e \neq 0} p(x, e) [\alpha_1 e_1 + \alpha_2 e_2 + \alpha_3 e_1 e_2] + \alpha_3 e_2 x_1 + \alpha_3 e_1 x_2].$$

Thus, by substituting the values of $x$ and $e$, and grouping some terms

$$p_{AI} = 1 - [p(00, 01) \alpha_1 + p(01, 01) \alpha_1 + p(10, 01) \alpha_1 + p(11, 01) \alpha_1 + p(00, 11) \alpha_1 + p(01, 11) \alpha_1 + p(10, 11) \alpha_1 + p(11, 11) \alpha_1]$$

$$= 1 - [(\phi(1) + \phi(2) \alpha_2 + \phi(3) \alpha_3) + 2(\phi(1, 2) \alpha_1 \alpha_2 + \phi(1, 3) \alpha_1 \alpha_3 + \phi(2, 3) \alpha_2 \alpha_3) + 4\phi(1, 2, 3) \alpha_1 \alpha_2 \alpha_3]. \quad (41)$$

which is the personalization of (32) for a CUT with two outputs, that is, the extraction module has two inputs. The detection potentials appearing in (41) are defined by the following expressions:

$$\phi(1) = p(00, 01) + p(01, 01) + p(10, 11) + p(11, 11)$$
$$\phi(2) = p(00, 01) + p(10, 11) + p(11, 11)$$
$$\phi(3) = p(00, 01) + p(01, 11) + p(10, 11) + p(11, 11)$$
$$\phi(1, 2) = p(00, 01) + p(10, 11) + p(11, 11)$$
$$\phi(1, 3) = p(00, 01) + p(10, 11) + p(11, 11)$$
$$\phi(2, 3) = p(00, 11) + p(01, 11) + p(10, 11) + p(11, 11)$$
$$\phi(1, 2, 3) = p(00, 11) + p(11, 11)$$

Once the expression for $p_{AI}$ has been found, the changes in the state of the three neurons in the network occur according to the following expressions:

1. $\delta \alpha_1 > 0$, if $\phi(1) > [2(\phi(1, 2) \alpha_2 + \phi(1, 3) \alpha_3) - 4\phi(1, 2, 3) \alpha_1 \alpha_2 \alpha_3]. \delta \alpha_1 < 0$, otherwise.
2. $\delta \alpha_2 > 0$, if $\phi(2) > [2(\phi(1, 2) \alpha_1 + \phi(2, 3) \alpha_3) - 4\phi(1, 2, 3) \alpha_1 \alpha_2 \alpha_3]. \delta \alpha_2 < 0$, otherwise.
3. $\delta \alpha_3 > 0$, if $\phi(3) > [2(\phi(1, 3) \alpha_1 + \phi(2, 3) \alpha_2) - 4\phi(1, 2, 3) \alpha_1 \alpha_2 \alpha_3]. \delta \alpha_3 < 0$, otherwise.

Thus, defining

$$G_1 = -(\phi(1) - [2(\phi(1, 2) \alpha_2 + \phi(1, 3) \alpha_3) - 4\phi(1, 2, 3) \alpha_1 \alpha_2 \alpha_3])$$
$$G_2 = -(\phi(2) - [2(\phi(1, 2) \alpha_1 + \phi(2, 3) \alpha_3) - 4\phi(1, 2, 3) \alpha_1 \alpha_2 \alpha_3])$$
$$G_3 = -(\phi(3) - [2(\phi(1, 3) \alpha_1 + \phi(2, 3) \alpha_2) - 4\phi(1, 2, 3) \alpha_1 \alpha_2 \alpha_3])$$

a $\delta \alpha_i > 0$ will occur if $G_i < 0$, and a $\delta \alpha_i < 0$ if $G_i > 0$.

As an example, considering an error model defined by the following specific values for error probabilities $p(x, e)$:

$$p(00, 01) = p(01, 01) = 0.1, \quad p(10, 10) = p(11, 10) = 0.1, \quad p(00, 11) = p(01, 11) = p(10, 11) = p(11, 11) = 0.05$$

we have

$$G_1 = -(0.6 - 0.4 \alpha_2 - 0.6 \alpha_3 + 0.4 \alpha_2 \alpha_3)$$
$$G_2 = -(0.6 - 0.4 \alpha_1 - 0.6 \alpha_3 + 0.4 \alpha_1 \alpha_3)$$
$$G_3 = -(0.5 - 0.6 \alpha_1 - 0.6 \alpha_2 + 0.4 \alpha_1 \alpha_2),$$

and the evolution of the network will be as follows, if it begins with the initial state $\alpha = (\alpha_1, \alpha_2, \alpha_3) = (0, 0, 0)$:

1. For this situation $G_2 = -0.6$, therefore, $\delta \alpha_2 = 1$ and then $\alpha = (0, 1, 0)$.  
2. Now $G_3 = 0.1$ and $\delta \alpha_3 < 0$ should now occur. As $\alpha_3 = 0$, the state is maintained.  
3. The value for $G_1$ is $-0.2$, therefore, $\delta \alpha_1 > 0$ and thus $\alpha = (1, 1, 0)$.

Therefore, $G_1 < 0, G_2 < 0, and G_3 > 0$, and the values for $\alpha_1, \alpha_2, and \alpha_3 do not change. This means that the solution is $g_1 = x_1 \oplus x_2$, which is indeed the best solution since the distributions of errors used as the starting point indicated a greater probability for the error patterns with a single erroneous bit. With $\alpha_1 = 1, \alpha_2 = 1, and \alpha_3 = 0$, the value for the aliasing probability is $p_{AI} = 0.2$, which corresponds to the probability of an erroneous bit in the output. If lower aliasing probability than $p_{AI} = 0.2$ is required, then the procedure is repeated with all the error probabilities detected by the function $g_1$ set to zero and normalized. In this case, the new error model to be used is

$$p(00, 11) = p(01, 11) = p(10, 11) = p(11, 11) = 0.25,$$

and the values for the $G_i$ are

$$G_1 = -(1 - 2\alpha_2 - \alpha_3 + 2\alpha_2 \alpha_3)$$
$$G_2 = -(1 - 2\alpha_1 - \alpha_3 + 2\alpha_1 \alpha_3)$$
$$G_3 = -(0.5 - \alpha_1 - \alpha_2 + 2\alpha_1 \alpha_2).$$
TABLE II

<table>
<thead>
<tr>
<th>Error Model</th>
<th>$p(x_i, x_j, x_k, x_l)$</th>
<th>$p(x_i, x_j, x_k)$</th>
<th>$p(x_i, x_j)$</th>
<th>$p(x_i, x_j, x_k, x_l, x_n)$</th>
<th>$p(x_i, x_j, x_k, x_l, x_n, x_m)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>EX2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>EX3</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>EX4</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

As before, the selected starting point is $\alpha = (0, 0, 0)$. As $G_1 < 0$, $\alpha_1$ may become $\alpha_1 = 1$. Now, $G_2 > 0$, $G_3 > 0$ if $G_1 < 0$, since $\alpha_2 = 0, \alpha_3 = 0$, and $\alpha_1 = 1$, no changes may occur. A state of equilibrium has been reached, and the extraction module is therefore defined by

$$g_1 = x_1 \oplus x_2, \quad g_2 = x_1.$$ 

As $PA_2 = 0, PA_1 = PA_2 = 0$, which means that all errors cause a change in the outputs of the extraction module. If we wish to impose a limitation on the complexity of the solution obtained, $C_0$, the term

$$\Omega = k_1[(\alpha_1 + \alpha_2 + \alpha_3) - C_0] + k_2[(\alpha_1 + \alpha_2 + \alpha_3) - C_0]^2$$

should be added to (41) for the aliasing probability, $p'_{Ai} = p_{Ai} + \Omega$.

Since

$$\delta\Omega = \delta\Omega_1 = k_1 + 2k_2[(\alpha_1 + \alpha_2 + \alpha_3) - C_0]$$

is verified, the functions $G_1, G_2$, and $G_3$ will be

$$G_1 = -[(\phi(1) - [k_1 - 2k_2C_0]) - 2(\phi(1, 2) + k_2)\alpha_2$$
$$+ 2(\phi(1, 3) + k_2)\alpha_3 + 2k_2\alpha_1 - 4(\phi(1, 2, 3)\alpha_2\alpha_3)]$$

$$G_2 = -[(\phi(2) - [k_1 - 2k_2C_0]) - 2(\phi(1, 2) + k_2)\alpha_1$$
$$+ 2(\phi(2, 3) + k_2)\alpha_3 + 2k_2\alpha_2 - 4(\phi(1, 2, 3)\alpha_1\alpha_3)]$$

$$G_3 = -[(\phi(3) - [k_1 - 2k_2C_0]) - 2(\phi(1, 3) + k_2)\alpha_1$$
$$+ 2(\phi(2, 3) + k_2)\alpha_2 + 2k_2\alpha_3 - 4(\phi(1, 2, 3)\alpha_1\alpha_2)]$$

(45)

If $C_0$ is fixed to 2 and $k_1 = k_2$, then $\Omega$ is below 0 when $C < 3$ and $\Omega$ is above 0 when $C > 3$ and $C = 0$. Thus, for the distribution of error probabilities considered,

$$G_1 = -[(0.6 + 3k_1) - (0.4 + 2k_1)\alpha_2 - (0.6 + 2k_1)\alpha_3 - 2k_1\alpha_1 + 0.4\alpha_2\alpha_3]$$

$$G_2 = -[(0.6 + 3k_1) - (0.4 + 2k_1)\alpha_1 - (0.6 + 2k_1)\alpha_3 - 2k_1\alpha_2 + 0.4\alpha_1\alpha_3]$$

$$G_3 = -[(0.5 + 3k_1) - (0.6 + 2k_1)\alpha_1 - (0.6 + 2k_1)\alpha_2 - 2k_1\alpha_3 + 0.4\alpha_1\alpha_2].$$

If, for example, $k_2 = 0.1$ is set, the solution obtained is $g_1 = x_1 \oplus x_2$, as in the previous case.

IV. EXPERIMENTAL RESULTS

This section gives some of the results obtained by the simulation of the GHN's for various error distributions that could be observed at the CUT outputs. The errors have been modeled using an independent 2-ary channel [33] for each output, $p(e_i = 1/x_i) i = 1, \cdots, n$. The GHN software simulator has been implemented with two options:

1. At any given time, only one neuron in the network may evaluate its inputs and changes consequently [Mono-neuron Asynchronous Evolution, MmA Evolution].
2. A random number of neurons may be evaluating their inputs at any given time (Multi-neuron Asynchronous Evolution, MIE Evolution). In this way, a physically implemented GHN is modeled more realistically.

To show the efficiency of the GHN, some examples of CUT's with three outputs were used, and four error models were considered (see Table II for their characteristics).

In the experiments carried out, a value was taken for $C_0$ equal to the number of outputs from the CUT, $C_0 = n$. Moreover, the values for $k_1$ and $k_2$ were adjusted so that $\Omega(C)$ vanishes if $C = n$ and $C = 1$, and, therefore, $k_2 = k_1/(n - 1) = k_1/2$. Thus, the number of weights for each of the neurons is reduced so that now only 63 weights
are needed compared with 128 for $n = 3$. Fig. 6(a) and (b), respectively, represent $\Omega(C)$ for some values of $k_2$, keeping $C_0$ as a constant, and for some values of $C_0$, keeping $k_2$ as a constant.

Figs. 7–9 show the behavior of an algorithm that simulates the MIA Evolution of a GHN for two examples of error models. The values of $p_{AI}$ for each of the possible solutions are indicated in Fig. 7, whereas Fig. 8 and 9 show the $p_{AI}'$ ones. As may be seen in Fig. 7, if $k_2 = 0$ ($\Omega = 0$), the network cannot escape from a local minimum that does not correspond to the optimal extraction function for the considered error model. This problem can be solved by setting a constraint for the complexity, $C_0$, and $k_2 > 0$: When the different nonzero values for $k_2$ are introduced, some local minima for $p_{AI}$ are no longer the local minima for $p_{AI}'$. Moreover, to constrain the complexity of the solution allows one to make zero all the potentials with more than $C_0 + 1$ indices. In Fig. 8, $k_2 = 0.2$ and, in Fig. 9, $k_2 = 0.3$. With a positive value for $k_2$, two solutions with the same values for $p_{AI}$ [Fig. 7(b)] but different ones for $C$ can be distinguished, by using $p_{AI}'$ to select the least costly hardware [compare Fig. 8(b) with Fig. 7(b)]. In addition, the value of $k_2$ shall be selected carefully: If $k_2$ is too high, a solution with a nonminimal value for $p_{AI}$ could be obtained, as can be seen in Fig. 9(a) where the $x_2 \oplus x_3$ is the minimum for $p_{AI}'$ but not for $p_{AI}$.

The results obtained for each distribution of probability and some values of $k_2$ are shown in Tables III and IV for a CUT with three outputs and for MIA evolution and MIA evolution, respectively. These tables also provide a comparison of the solutions obtained with the exact solutions (the ones with less $p_{AI}$ for each error model). Such solutions were determined by evaluating the aliasing probabilities for each of the possible solutions: This is feasible in this case, as the space of the solutions is not too big, and allows us to verify the accuracy of the proposed method. As can be seen in Tables III and IV, for each error model, there is a value for $k_2$ that allows the network to reach a minimum of $p_{AI}$ with the desired complexity. The value for the parameter $k_2$
Fig. 9. Values of \( p'_{A} \) corresponding to the \( p_{A} \) values of Fig. 5 with \( C_{8} = 3 \) and \( k_{2} = 0.3 \). The errors of the CUT are modeled with (a) EM1 and (b) EM2.

has been determined empirically by doing simulations starting with \( k_{2} = 0.0 \) and increasing it by 0.05 in each simulation until a solution is obtained such that its value for \( p'_{A} \) is higher than the value corresponding to the solution obtained in the previous simulation, assuming that both solutions have less spectral coefficients than \( C_{8} \). Thus, the value of \( k_{2} \) selected and the solution computed for the considered error model are those corresponding to the simulation previous to the last one.

From Tables III and IV, the number of iterations for the network to obtain this solution, with the correspondent \( k_{2} \), is less than the number of spectral Reed–Muller coefficients and, also, a network with MIA evolution converges faster than a network with MnA evolution.

The simulator has been implemented in the programming language C on a Sun-3 workstation (4 MIPS), and the executable program has occupied 24 Kbytes. Table V shows the computing time for each iteration in CUT’s with a different number of outputs and values of the higher hardware complexity allowed, \( C_{8} \). If the number of outputs of a CUT is too high to be solved by that software simulator due to the size of the memory needed and the amount of CPU time required, it is even possible to use the simulator by partitioning the set of outputs of the CUT in several subsets with less outputs and arranging the error models adequately. Of course, the extractor module synthesized this way would have at least one

<table>
<thead>
<tr>
<th>Error Model</th>
<th>Initial Status ( {a_{n} \ldots a_{1}} )</th>
<th>Solution with ( \text{min.} , p_{A} ) ( {a_{n} \ldots a_{1}} )</th>
<th>Solution Obtained ( {b_{n} \ldots b_{1}} )</th>
<th>( p_{A} )</th>
<th>( k_{2} )</th>
<th>Iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM1</td>
<td>( {111111} ) (1101000) ( p_{A} = 0.1 )</td>
<td>( {1111001} ) (1110000)</td>
<td>( {1100100} )</td>
<td>0.300</td>
<td>0.0</td>
<td>3</td>
</tr>
<tr>
<td>DM2</td>
<td>( {111111} ) (1001000) ( p_{A} = 0.1 )</td>
<td>( {1101100} ) (1100100)</td>
<td>( {1100100} )</td>
<td>0.335</td>
<td>0.3</td>
<td>4</td>
</tr>
<tr>
<td>DM3</td>
<td>( {111111} ) (1001000) ( p_{A} = 0.24 )</td>
<td>( {1100111} ) (1010111)</td>
<td>( {1110011} )</td>
<td>0.370</td>
<td>0.3</td>
<td>4</td>
</tr>
<tr>
<td>DM4</td>
<td>( {111111} ) (1101000) ( p_{A} = 0.124 )</td>
<td>( {1111001} ) (1011110)</td>
<td>( {1100100} )</td>
<td>0.305</td>
<td>0.3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error Model</th>
<th>Initial Status ( {a_{n} \ldots a_{1}} )</th>
<th>Solution with ( \text{min.} , p_{A} ) ( {a_{n} \ldots a_{1}} )</th>
<th>Solution Obtained ( {b_{n} \ldots b_{1}} )</th>
<th>( p_{A} )</th>
<th>( k_{2} )</th>
<th>Iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM1</td>
<td>( {111111} ) (1101000) ( p_{A} = 0.1 )</td>
<td>( {1100100} ) (1100100)</td>
<td>( {1110000} )</td>
<td>0.100</td>
<td>0.0</td>
<td>3</td>
</tr>
<tr>
<td>EM2</td>
<td>( {111111} ) (1001000) ( p_{A} = 0.1 )</td>
<td>( {1100100} ) (1100100)</td>
<td>( {1110000} )</td>
<td>0.100</td>
<td>0.0</td>
<td>3</td>
</tr>
<tr>
<td>EM3</td>
<td>( {111111} ) (1001000) ( p_{A} = 0.24 )</td>
<td>( {1100111} ) (1010111)</td>
<td>( {1110011} )</td>
<td>0.370</td>
<td>0.3</td>
<td>4</td>
</tr>
<tr>
<td>EM4</td>
<td>( {111111} ) (1101000) ( p_{A} = 0.124 )</td>
<td>( {1111001} ) (1011110)</td>
<td>( {1100100} )</td>
<td>0.124</td>
<td>0.3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n#</th>
<th>( C_{8} )</th>
<th>NW</th>
<th>CTT(s)</th>
<th>CTT/NW(10^{-3} , s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>21</td>
<td>0.0996</td>
<td>2.362</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>42</td>
<td>0.0936</td>
<td>2.229</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>49</td>
<td>0.1348</td>
<td>2.750</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>150</td>
<td>0.2697</td>
<td>1.798</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>210</td>
<td>1.0840</td>
<td>5.160</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>465</td>
<td>1.1280</td>
<td>2.426</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>775</td>
<td>2.0100</td>
<td>2.590</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1323</td>
<td>2.6610</td>
<td>2.010</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2583</td>
<td>37.934</td>
<td>14.69</td>
</tr>
</tbody>
</table>
function for each subset of outputs. This strategy allows one to reduce the complexity of the problem by circuit partitioning, which is a solution usually adopted for coping with problems involved in designing and testing complex digital circuits [41]-[43].

The circuit implementations of the GHN's used to obtain the results of Tables III and IV have also been simulated electrically with SPICE. As an example, Fig. 10 shows the neural network evolution to the right solution for error model EM2. It has been obtained from a SPICE simulation of the GHN corresponding to EM2 with resistances in the range of 1 kΩ, capacities of 1 pF, and parameters $C_0 = 3$ and $k_2 = 0.2$. Each square of Fig. 10 represents the evolution of one of the seven neurons of the network. The network power supply is switched on in $t = 1.2$ ns and before $t = 2.0$ ns a steady state is reached. This state is $(\alpha_1, \ldots, \alpha_7) = (1, 0, 0, 0, 0, 0)$, and so the extraction function obtained is $g_1 = x_1 \oplus x_2$. As can be seen from Tables III and IV, this is the extraction function with less aliasing probability for EM2.

V. CONCLUSIONS

This paper has shown how a generalized Hopfield neural network may be used for designing the checking circuitry of a concurrent testable circuit. The goal is to find the extraction module of the checking circuitry that could be capable of detecting as many errors as possible with less hardware complexity than a given bound. By describing the extraction modules with spectral techniques, it is possible to map the problem in a neural network. Thus, the equation that allows us to evaluate the error detection abilities of an extraction module has been expressed in terms of Reed–Muller spectral coefficients. The obtained equation determines the weights and threshold of the generalized Hopfield network (GHN) used to find an acceptable solution.

In this way, we have to deal with an optimization problem which, as it has been proved, is NP-complete, and it is necessary to reduce the complexity of the GHN needed for problems with moderate sizes. The network simplification has been achieved by defining a constraint on the hardware complexity for the acceptable solutions and by including that constraint in the aliasing probability function to be minimized via two Lagrange multipliers. Thus, control parameters appear in the definition of the neural network needed to implement the aliasing probability, $p_A$, for the error model considered. This constraint can be used to aid the network to escape from local minima of the aliasing probability and to simplify the complexity of the neural network. The capability of the method proposed has been checked experimentally, with software simulations of GHN's and binary neurons, and with SPICE, by doing electrical simulations of the analog circuitry that could implement a GHN. Thus, the efficiency and robustness of the GHN's as a tool for searching extraction modules with high enough errors detection capabilities and, consequently, for designing concurrent testable circuits, have been shown.

ACKNOWLEDGMENT

The authors would like to thank the referees for their useful comments and suggestions.

REFERENCES

Julio Ortega received the B.Sc degree in electronic physics in 1985, the M.Sc. degree in electronics in 1986, and the Ph.D. degree in 1990, all from the University of Granada, Spain. His Ph.D. dissertation has received the Award of Ph.D. Dissertations of the University of Granada.

In March 1988, he was at the Open University (U.K.) as an invited researcher. Currently, he is an Associate Professor in the Departamento de Electrónica y Tecnología de Computadores at the University of Granada. His research interest lies in the fields of artificial neural networks, VLSI design and testing of integrated circuits, and multiple-valued logic.

Alberto Prieto (M’74) received a B.Sc. degree in electronic physics in 1968 from the Complutense University, Madrid, and the Ph.D. degree from the University of Granada, Spain, in 1976. From 1969 to 1970 he was at the Centro de Investigaciones Técnicas de Galipuzcoa and the E.T.S.I. Industriales of San Sebastián. From 1971 to 1984 he was Director of the Computer Centre, and from 1985 to 1990 he was Dean of computer science and technology studies of the University of Granada. He is currently a Full Professor in the Departamento de Electrónica y Tecnología de Computadores at the University of Granada.

In 1975 he was granted a research scholarship at the Department of Informatics, University of Rennes. In July 1987, he was at the Open University, U.K., as guest researcher. In June to August 1991, he was at the TIFR Laboratory, Institute National Polytechnique (INPG) of Grenoble, as guest researcher. During 1991-1992, jointly with Prof. P. Treleaven, he was main researcher of a Spanish-British Integrated Action on Artificial Neural Networks, held between the University of Granada and the University College of London. He is a main researcher of Spanish-German Join Research Action between the University of Granada and the Dortmund University and of Spanish-French Join Research Action between the University of Granada and the TIFR Laboratory of the INPG of Grenoble. Prof. Prieto was the Organization Chairman of the International Workshop on Artificial Neural Systems (IANN’91) (Granada, Sept. 1991), which was organized and sponsored by the Spanish Chapter of the Computer Society of IEEE, and is the General Chairman of the IANN’93, organized in cooperation with the IFIP WG 10.6 and the IEEE Neural Network Council. His research interests are in the areas of artificial neural networks, multiple-valued circuits, microprocessor-based systems, and VLSI design and testing. Prof. Prieto is a nominated member of the IFIP WG10.6 (Neural Computer Systems). His is FIP-Spain Editorial Board, AEIIA, AECET, IEEE and ACM. His Ph.D. dissertation has received two awards: the Award of Ph.D. Dissertations of the University of Granada and the Cienta Foundation National Award.
Antonio Lloris Ruiz received the Ldo.Sc. degree in 1969 and the Ph.D. degree in 1972, both from the Universidad Complutense, Madrid.

From 1969 to 1972 he was at the Centro de Investigaciones Técnicas de Guipuzcoa, Spain, as a researcher, and, as a lecturer, at the Escuela Técnica Superior de Ingenieros Industriales de San Sebastián. He was a lecturer at the University of Málaga from 1973 to 1976, an Associate Professor at the University of Granada from 1976 to 1989, and a Full Professor at the University of Murcia, where he was the Director of the Escuela Técnica Superior de Ingenieros Industriales. Currently, he is Full Professor at the Departamento de Electrónica y Tecnología de Computadores of the University of Granada, Spain. Dr. Lloris is an honorary Research Fellow at the Open University (England) where he was a visiting reader in 1986 and 1987. He was main researcher of a Spanish-British Joint Research Action between the University of Granada and the Open University. His research interests include multiple-valued logic, VLSI design and testing of integrated circuits, and artificial neural networks.

Francisco J. Peláez received the B.Sc. degree in electrical and electronic physics in 1982, the M.Sc. degree in electronics in 1983, and the Ph.D. degree in 1989, all from the University of Granada, Spain.

In September 1983, he entered the Departamento de Electrónica y Tecnología de Computadores at the University of Granada, where he is currently an Associate Professor. In August 1991, he was at the University College of London as an invited researcher. His research interest lies in the fields of artificial neural networks and VLSI design and testing of analog/digital integrated circuits.