A Combinatorial Method for the Evaluation of Yield of Fault-Tolerant Systems-on-Chip

Doru P. Munteanu
Military Technical Academy
G. Cosbuc 81–83
Bucharest 75275, Romania
munteanud@mta.ro

Víctor Suñé, Rosa Rodríguez-Montañés, Juan A. Carrasco [*]
Departament d'Enginyeria Electrònica
Universitat Politècnica de Catalunya
Diagonal 647, plta. 9
08028 Barcelona, Spain
{sunye,rosa,carrasco}@eel.upc.es

Abstract

In this paper we develop a combinatorial method for the evaluation of yield of fault-tolerant systems-on-chip. The method assumes that defects are produced according to a model in which defects are lethal and affect given components of the system following a distribution common to all defects. The distribution of the number of defects is arbitrary. The method is based on the formulation of the yield as 1 minus the probability that a given boolean function with multiple-valued variables has value 1. That probability is computed by analyzing a ROMDD (reduced ordered multiple-value decision diagram) representation of the function. For efficiency reasons, we first build a coded ROBDD (reduced ordered binary decision diagram) representation of the function and then transform that coded ROBDD into the ROMDD required by the method. We present numerical experiments showing that the method is able to cope with quite large systems in moderate CPU times.

1 Introduction

Systems-on-chip are becoming popular. The high densities and areas of those integrated systems make them very susceptible to manufacturing defects. In fact, complex systems-on-chip are likely to have a very small yield if they are not designed with built-in fault-tolerance. Then, there is a need for efficient methodologies for estimating the yield of complex fault-tolerant systems-on-chip. When the fault-tolerant system-on-chip has a regular structure, it is often possible to make "ad-hoc" evaluations (see, for instance, [11, 12, 17, 18]). However, many fault-tolerant designs do not have a regular structure, particularly those using a sophisticated network-on-chip as a communication subsystem among the intellectual property cores (IPs) [3]. Computing the yield of such systems-on-chip is difficult, mainly because the fact that realistic defect distributions have clustering [7, 13, 14, 15, 16, 18] and, thus, introduce dependencies among the failed states of the components of the system (see, for instance, [18, 27]). Simulation is an approach which is not severely limited by the complexity of the system, but tends to be expensive and does not provide strict error control. The aim of this paper is to develop a combinatorial method for the evaluation of the yield of fault-tolerant systems-on-chip with precise error control which can cope with quite complex systems using currently affordable computational resources.

We assume that the fault-tolerant system-on-chip is made up of a set $\{1, 2, \ldots, C\}$ of components and that whether the system is functioning or not is determined from the failed states of the components through a fault-tree function $F(x_1, \ldots, x_C)$, where variable x_i takes the value 1 if and only if component i is failed and the function takes the value 1 if and only if the system is not functioning. No restriction is imposed on $F(x_1, \ldots, x_C)$. It will be assumed that a gate-level description of the function is available.

The production of manufacturing defects will be modeled using the following probabilities:

$$Q_k = P[\text{number of manufacturing defects is } k],\quad k = 0, 1, 2, \ldots,$$

$$P_i = P[\text{a given defect affects component } i \text{ and is lethal}],$$

It will be assumed that all defects will be distributed over the components making up the system and will be lethal following the probabilities P_i, $1 \le i \le C$, independently of the number of defects, of which components affect the remaining defects and of whether those defects are lethal

[*]This work was supported by the "Comisión Internacional de Ciencia y Tecnología" (CICYT) of the Ministry of Science and Technology of Spain under the research grant TAP1999-0443-C05-05.

or not. That model is useful from the designer's point of view, since the distribution of the number of defects Q_k, $k = 0, 1, 2, \dots$ could be easily provided by the manufacturer of the system-on-chip and the probabilities P_i, $1 \le i \le C$ could be estimated from the final layout of the system-on-chip using appropriate tools [19, 21, 31, 32] or from IP layouts and routing estimates [30]. Thus, the methodologies could be used at several design stages. The assumed model is consistent with all compound Poisson yield models [18], which include the widely used negative binomial distribution for the number of defects. The assumed model will not be consistent however with yield models accounting for spatial clustering[1] such as the one proposed in [22].

From a computational point of view, it is convenient to map the previously described model into a model taking into account only lethal manufacturing defects, i.e. defects which effectively make some component of the system to be defective (not to work properly). That model includes the probabilities:

$$Q'_k = P[\text{number of lethal manufacturing defects is } k], \quad k = 0, 1, 2, \dots,$$

$$P'_i = P[\text{a given lethal defect affects component } i] .$$

The reason why the last model is computationally more convenient is basically because, since not all defects will be lethal, the distribution Q'_k, $k = 0, 1, 2, \dots$ will be shifted to lower values of k in relation to the distribution Q_k, $k = 0, 1, 2, \dots$ and, then, if only up to M defects are analyzed (the computational cost of the methods will increase with M), higher accuracy will be obtained if the distribution Q'_k, $k = 0, 1, 2, \dots$ is used instead of Q_k, $k = 0, 1, 2, \dots$. The mapping can be performed using:

$$Q'_k = \sum_{m=k}^{\infty} Q_m \binom{m}{k} P_L^k (1 - P_L)^{m-k} , \qquad (1)$$

$$P'_i = \frac{P_i}{P_L} ,$$

where $P_L = \sum_{i=1}^{C} P_i$ is the probability that any given defect is lethal. As previously commented, the negative binomial distribution is the most widely used distribution for the number of defects affecting a chip. That distribution has the form:

$$Q_k = \frac{\Gamma(\alpha + k)}{k!\Gamma(\alpha)} \frac{(\lambda/\alpha)^k}{(1 + \lambda/\alpha)^{\alpha+k}} , \qquad (2)$$

where λ is the expected number of defects and α is the clustering parameter (the clustering increases for decreasing α). It is known (see [15]) that, when the distribution of the number of defects is negative binomial, the distribution of the number of lethal defects is also negative binomial with the same clustering parameter. More precisely, when the distribution of the number of defects is given by (2), the distribution of the number of lethal defects is:

$$Q'_k = \frac{\Gamma(\alpha + k)}{k!\Gamma(\alpha)} \frac{(\lambda'/\alpha)^k}{(1 + \lambda'/\alpha)^{\alpha+k}} ,$$

with $\lambda' = P_L \lambda$. Similar results hold for all compound Poisson distributions [18].

[1]Spatial clustering refers to the fact that irrespectively of the expected number of defects on the system-on-chip, defects tend to cluster spatially.

2 The method

In the method the yield, Y, is computed analyzing whether the system is functioning or not assuming $0, 1, 2, \dots, M$ lethal defects. Let

$$Y_k = P[\text{system is functioning} \mid \text{there are } k \text{ lethal defects}] .$$

We have

$$Y = \sum_{k=0}^{\infty} Q'_k Y_k .$$

Analyzing up to M defects we can pessimistically estimate Y by

$$Y^M = \sum_{k=0}^{M} Q'_k Y_k ,$$

with error bounded from above by $\sum_{k=M+1}^{\infty} Q'_k = 1 - \sum_{k=0}^{M} Q'_k$. Then, given a suitable error control parameter ε, we can select

$$M = \min \left\{ m \ge 0 \ : \ 1 - \sum_{k=0}^{m} Q'_k \le \varepsilon \right\} ,$$

guaranteeing and absolute error in the yield estimation $\le \varepsilon$.

The yield estimate Y^M can be formalized as the probability that a boolean function of certain independent integer-valued random variables is equal to 1. Assume that the defects are numbered in some arbitrary order. Those random variables are:

$$W = \begin{cases} k, 0 \le k \le M & \text{if there are } k \text{ lethal defects} \\ M + 1 & \text{if there are more than} \\ & \ M \text{ lethal defects} \end{cases}$$

and, for $1 \le k \le M$,

$$V_k = i \quad \text{if the } k\text{th lethal defect affects component } i .$$

Note that the random variable W takes values in $\{0, 1, \ldots, M+1\}$ and each random variable V_k takes values in $\{1, 2, \ldots, C\}$. The random variable W has probability distribution $P[W = k] = Q'_k$, $0 \leq k \leq M$, $P[W = M+1] = 1 - \sum_{k=0}^{M} Q'_k$. The random variables V_k have probability distributions $P[V_k = i] = P'_i$, $1 \leq k \leq M$, $1 \leq i \leq C$.

Let $I_k(x)$ denote the boolean function with integer-valued variable x returning the value 1 if $x = k$ and the value 0 otherwise and let $I_{\geq l}(x)$ denote the boolean function with integer-valued variable x returning the value 1 if $x \geq l$ and the value 0 otherwise. Let the boolean function

$$\begin{aligned} G(w, v_1, v_2, \ldots, v_M) &= I_{M+1}(w) \\ &\vee\ F\left(\bigvee_{l=1}^{M} I_{\geq l}(w) \wedge I_1(v_l), \right. \\ &\qquad \left. \ldots, \bigvee_{l=1}^{M} I_{\geq l}(w) \wedge I_C(v_l)\right) . \end{aligned} \tag{3}$$

Then, we have the following result.

Theorem 1. $Y^M = 1 - P[G(W, V_1, V_2, \ldots, V_M) = 1]$.

Intuitively, the reason why Theorem 1 holds is that $I_{M+1}(W)$ "tells" whether the number of lethal defects is $> M$, $I_{\geq l}(W)$ "tells" whether there is a lth lethal defect, $I_i(V_l)$ "tells" whether the lth lethal defect affects component i and, then, $\bigvee_{l=1}^{M} I_{\geq l}(W) \wedge I_i(V_l)$ "tells" whether component i is affected by some of the first M lethal defects. A formal proof follows.

Proof of Theorem 1 The quantity $1 - Y_k$ is the probability that given there are k lethal defects the system is not functioning. Since, assuming there are k lethal defects, component i is failed if and only if $\bigvee_{l=1}^{k} I_i(V_l) = 1$, we have

$$1 - Y_k = P\left[F\left(\bigvee_{l=1}^{k} I_1(V_l), \ldots, \bigvee_{l=1}^{k} I_C(V_l)\right) = 1\right] . \tag{4}$$

Using the theorem of total probability and the independence of the random variables $W, V_1, \ldots, V_M$:

$$\begin{aligned} &P[G(W, V_1, \ldots, V_M) = 1] \\ &\quad= \sum_{k=0}^{M+1} P[W = k] P[G(W, V_1, \ldots, V_M) = 1 \mid W = k] \\ &\quad= \sum_{k=0}^{M} Q'_k P[G(k, V_1, \ldots, V_M) = 1] \\ &\qquad+ \left(1 - \sum_{k=0}^{M} Q'_k\right) \\ &\qquad\quad P[G(M+1, V_1, \ldots, V_M) = 1] . \end{aligned} \tag{5}$$

But, from the definition of G (3), for $0 \leq k \leq M$:

$$G(k, v_1, \ldots, v_M) = F\left(\bigvee_{l=1}^{k} I_1(v_l), \ldots, \bigvee_{l=1}^{k} I_C(v_l)\right) \tag{6}$$

and

$$G(M+1, v_1, \ldots, v_M) = 1 . \tag{7}$$

Then, using (4)–(7)

$$\begin{aligned} &P[G(W, V_1, \ldots, V_M) = 1] \\ &\quad= \sum_{k=0}^{M} Q'_k (1 - Y_k) + 1 - \sum_{k=0}^{M} Q'_k \\ &\quad= 1 - \sum_{k=0}^{M} Q'_k Y_k = 1 - Y^M . \quad \square \end{aligned}$$

In the method, the probability $P[G(W, V_1, \ldots, V_M) = 1]$ is computed building an ROMDD (reduced ordered multiple-valued decision diagram) of the function $G(w, v_1, \ldots, v_M)$. ROMDDs are a natural extension of the well-known ROBDDs (reduced ordered binary decision diagrams) [5] in which both the variables and the function are allowed to be multiple-valued. A gate-level representation of the function $G(w, v_1, \ldots, v_M)$ can be obtained from a gate-level representation of $F(x_1, \ldots, x_C)$ as shown in Figure 1, where the gate labeled i inside is a "filter" gate returning the value 1 if its integer-valued input has value i and returning the value 0 otherwise and the gate labeled $\geq i$ inside is a "filter" gate returning the value 1 if its integer-valued input has value $\geq i$ and returning the value 0 otherwise. As ROBDDs, ROMDDs are canonical representations which can be built and manipulated in a similar way as ROBDDs. An ROMDD representing a function F, which can take values in the set S_F, of variables x_i, $i = 1, 2, \ldots, n$, which can take values in the sets S_i is a directed acyclic graph with up to $|S_F|$ terminal nodes each labeled with a distinct value of the set S_F. Every non-terminal node is labeled by an input variable x_i and has as many as $|S_i|$ edges, each labeled by a subset of S_i, with subsets associated with different edges being non-intersecting. The ROMDD has a unique non-terminal node without incoming edges, representing the function $F(x_1, \ldots, x_n)$, called the top node. The input variables encountered in every path from the top node to a terminal node form a sequence of non-repeating input variables consistent with an ordering $x_{p(1)}, \ldots, x_{p(n)}$ of the input variables of the function. Every non-terminal node of the ROMDD represents a unique function of the set of input variables which are found in some path from the node to some terminal node. That a ROMDD is a canonical representation means that, given F, the ROMDD only

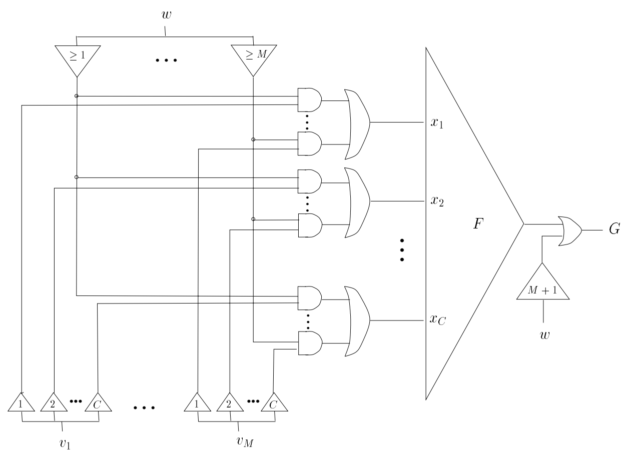

Figure 1. Gate-level description of the function $G(w, v_1, \ldots, v_M)$.

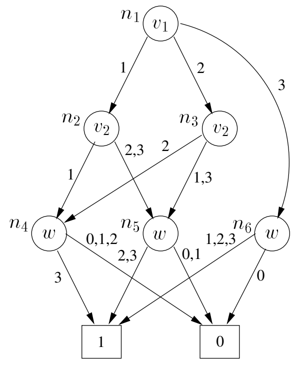

Figure 2. Small ROMDD to illustrate the computation of $P[G(W, V_1, \ldots, V_M) = 1]$.

depends on the selected ordering $x_{p(1)}, \ldots, x_{p(n)}$ for the multiple-valued variables.

Using the fact that the random variables $W, V_1, \ldots, V_M$ are independent and that the function represented by a non-terminal node only depends on the set of variables found on paths from the non-terminal node to terminal nodes, it is possible to compute $P[G(W, V_1, \ldots, V_M) = 1]$ from an ROMDD representation of the function $G(w, v_1, \ldots, v_M)$. This can be achieved by assigning the value 1 to the terminal node labeled "1" and the value 0 to the terminal node labeled "0", making a depth-first, left-most traversal [1] of the ROMDD, and computing the probability that the function represented by a non-terminal node has value 1 when returning from each non-terminal node. Assume that node n has associated with it the variable w, that $M = 4$, and that n has edges to nodes n_1, n_2 and n_3 with subsets of values of w $\{0, 1\}$, $\{3\}$ and $\{2, 4, 5\}$, respectively. Then, denoting by $value(x)$ the "value" variable associated with node x, when returning from node n, $value(n)$ would be computed as $(P[W = 0] + P[W = 1]) \times value(n_1) + P[W = 3] \times value(n_2) + (P[W = 2] + P[W = 4] + P[W = 5]) \times value(n_3)$. At the end of the traversal, the "value" variable of the top node will hold $P[G(W, V_1, \ldots, V_M) = 1]$. We illustrate the computational procedure with the small ROMDD shown in Figure 2 which corresponds to a fault-tolerant system having fault-tree function $F(x_1, x_2, x_3) = x_1x_2 + x_3$ and $M = 2$ under the multiple-valued variable ordering v_1, v_2, w. This implies that the random variable W will take values in the set $\{0, 1, 2, 3\}$ and the random variables V_1 and V_2 will take values in the set $\{1, 2, 3\}$. Using a depth-first, left-most traversal of the ROMDD, $P[G(W, V_1, V_2) = 1] = value(n_1)$ would be computed following the sequence:

$$
\begin{aligned}
&value(n_4) = Q'_3\,,\\
&value(n_5) = Q'_2 + Q'_3\,,\\
&value(n_2) = P'_1 \times value(n_4) + (P'_2 + P'_3) \times value(n_5)\,,\\
&value(n_3) = P'_2 \times value(n_4) + (P'_1 + P'_3) \times value(n_5)\,,\\
&value(n_6) = Q'_1 + Q'_2 + Q'_3\,,\\
&P[G(W, V_1, V_2) = 1] = value(n_1) =\\
&\quad P'_1 \times value(n_2) + P'_2 \times value(n_3) + P'_3 \times value(n_6)\,.
\end{aligned}
$$

Although there are algorithms and packages for ROMDD manipulation [23, 29], there is currently consensus in the ROMDD community that the most efficient way for analyzing multiple-valued functions of multiple-valued variables is by using coded ROBDDs [23, 24]. A coded ROBDD of a multiple-valued function $H(x_1, x_2, \ldots, x_n)$ of multiple-valued variables x_i is the ROBDD of any function $H'(x_{1,1}, \ldots, x_{1,k_1}, x_{2,1}, \ldots, x_{2,k_2}, \ldots, x_{n,1}, \ldots, x_{n,k_n})$ which represents $H(x_1, x_2, \ldots, x_n)$ in terms of groups $x_{i,1}, x_{i,2}, \ldots, x_{i,k_i}$ of binary variables encoding the multiple-valued variables x_i. Formally, denoting by D_i the domain of x_i and by $x_{i,1}(j), \ldots, x_{i,k_i}(j)$ the codeword representing value $j \in D_i$ in the code used for x_i, H' has to satisfy $H'(x_{1,1}(j_1), \ldots, x_{1,k_1}(j_1), x_{2,1}(j_2), \ldots, x_{2,k_2}(j_2), \ldots, x_{n,1}(j_n), \ldots, x_{n,k_n}(j_n)) = H(j_1, \ldots, j_n)$ for every $(j_1, \ldots, j_n) \in D_1 \times D_2 \times \cdots \times D_n$.

Coded ROBDDs can be used directly in many applications such as formal verification. However, the combinatorial method for yield computation requires the availability of the ROMDD. Given an ordering $x_{p(1)}, \ldots, x_{p(n)}$ of the multiple-valued variables, the ROMDD can be efficiently obtained from a coded ROBDD if the coded ROBDD is obtained using an ordering for the binary variables in which the variables encoding each multiple-valued variable are kept grouped and the groups are ordered according to the ordering $x_{p(1)}, \ldots, x_{p(n)}$. The conversion procedure is based on viewing the coded ROMDD as made up of layers, where each layer contains the nodes with binary variables encoding a given multiple-valued variable. Some of the nodes in each layer are entry nodes (they have incoming arcs from other layers). The procedure builds incrementally the ROMDD by processing bottom-up each layer of the coded ROBDD. Processed entry nodes of the coded ROBDD are associated with nodes of the constructed ROMDD. Let $mapping(n)$ be the node of the ROMDD associated with the entry node n of the coded ROBDD. The bottom layer of the coded ROBDD is processed by creating a copy n'_0 in the ROMDD of the non-terminal node n_0 of the coded ROBDD with value 0 and creating a copy n'_1 in the ROMDD of the non-terminal node n_1 of the coded ROBDD with value 1 and making $mapping(n_0) = n'_0$ and $mapping(n_1) = n'_1$. The remaining layers of the coded ROBDD are processed by processing each entry node n of the layer as follows. For each possible value i of the multiple-valued variable x associated with the layer, it is determined which entry node of a different (down) layer is reached from n when the values of the group of binary variables associated with the layer encoding value i are followed. Let $n_{s(i)}$ be the node of the coded ROBDD reached when the value i is "simulated". If all $mapping(n_{s(i)})$ are equal to some node n' of the ROMDD, then $mapping(n)$ must be made equal to n' and no node has to be added to the ROMDD. Otherwise, the ROMDD must have a node associated with n with multiple-valued variable x. That node must have successor $mapping(n_{s(i)})$ for each value i of x. If there exists in the ROMDD some node n' with multiple-valued variable x and successor $mapping(n_{s(i)})$ for each value of x, then, $mapping(n)$ is set to n' and no node is added to the ROMDD. Otherwise, a node n' with multiple-valued variable x and successor $mapping(n_{s(i)})$ for each value i of x is added to the ROMDD and $mapping(n)$ is set to n'. When not all combinations of values of the groups of binary variables encode values in the domain of the associated multiple-valued variable, the ROMDD built in that way may have nodes which are unreachable from the top node. Such nodes are identified and deleted by making a depth-first, leftmost traversal of the ROMDD starting from the top node. Figure 3 illustrates the processing of a layer of a coded ROBDD associated with a multiple-valued variable x which takes values in the domain $\{1, 2, 3\}$ and in which two binary variables x_1, x_0 have been used to encode variable x using the code $1 = 00$, $2 = 01$ and $3 = 10$.

The coded ROBDD is built by processing an implementation of the function $G(w, v_1, \ldots, v_M)$ in binary logic obtained by encoding the variable w in binary using a minimum number of bits. For the variables v_i, since they have values in the domain $\{1, 2, \ldots, C\}$, a binary code of minimum number of bits encoding $v_i - 1$ is used. Such strategy keeps minimum the number of binary variables and tends to result in coded ROBDDs of minimum size. Filter gates are

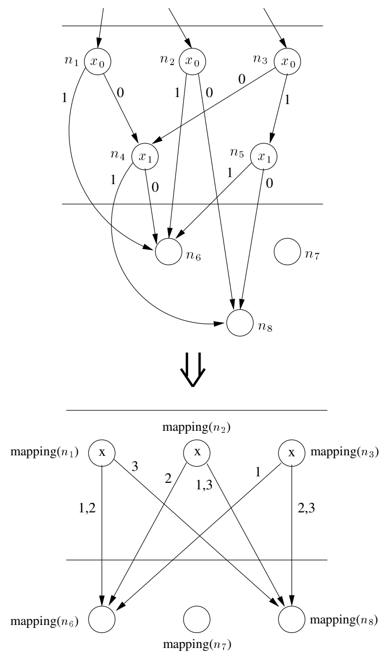

Figure 3. Illustration of the procedure for obtaining the ROMDD from the coded ROBDD

substituted by binary logic expressed in terms of the binary variables $w_{l'}, \ldots w_0$ encoding the multiple-valued variable w and the binary variables $v_i^l \ldots v_i^0$ encoding each multiple-valued variable v_i. Although, with given orderings of the variables, the coded ROBDD and the ROMDD will be independent on the particular implementation of that logic, that implementation may affect the heuristic-based orderings to be described next. This makes convenient to report which logic is used. Calling z_k, $1 \leq k \leq M$, the output of the "filter" gate labeled $\geq k$ having as input w, and calling z_{M+1} the output of the "filter" gate labeled $M+1$ having as input w, the binary logic used for generating z_k, $1 \leq k \leq M+1$ is:

$$z_{M+1} = lit(w_{l'}, M+1) \wedge lit(w_{l'-1}, M+1) \wedge \cdots \wedge lit(w_0, M+1) \, ,$$

$$z_k = z_{k+1} \vee lit(w_{l'}, k) \wedge lit(w_{l'-1}, k) \wedge \cdots \wedge lit(w_0, k) \, , \quad 1 \leq k \leq M \, ,$$

where $lit(w_i, m) = w_i$ if the ith bit of the binary code representing m is 1 and $lit(w_i, m) = \overline{w}_i$ if it is 0, where $\overline{x}$ denotes the complement of the binary variable x. Calling z_i^k the output of the "filter" gate labeled k having as input v_i, the binary logic used for generating z_i^k is:

$$z_i^k = lit(v_i^l, k-1) \wedge lit(v_i^{l-1}, k-1) \wedge \cdots \wedge lit(v_i^0, k-1) \, ,$$

where $lit(v_i^j, m) = v_i^j$ if the jth bit of the binary code representing m is 1 and $lit(v_i^j, m) = \overline{v_i^j}$ if it is 0. For building the coded ROBDD the implementation of the method uses the well-known BDD Library developed at Carnegie-Mellon University [2].

It is well-known that the size of the ROBDD of a boolean function of binary variables depends on the ordering of the binary variables. Similarly, the size of the ROMDD of a multiple-valued function of multiple-valued variables depends on the ordering of the multiple-valued variables. The variables are most often sorted using heuristics and an abundant literature is available about heuristics for ordering the variables of boolean functions of binary variables using gate-level representations of the functions [4, 6, 8, 9, 10, 20, 25, 26]. Those heuristics can be classified into static and dynamic depending on whether the ordering is computed before the ROBDD is built or the ordering may be changed during the ROBDD construction. Three heuristics which are relatively simple to implement and which have good performance are the **topology** heuristic described in [26], the **weight** heuristic described in [25] and the **H4** heuristic described in [4]. In the **topology** heuristic, input variables are sorted as found in a depth-first, left-most traversal of the gate description. In the **weight** heuristic, a weight 1 is assigned to the inputs, and, processing the gate description bottom-up, a weight equal to the sum of the weights of the fan-in nodes is assigned to the non-input nodes. Then, nodes in the fan-in of each non-input node are reordered in order of increasing weight, respecting the original ordering in case of a tie, and input variables are sorted as found in a depth-first, left-most traversal of the gate description with reordered fan-in. In the **H4** heuristic, input variables are sorted as found in a depth-first, left-most traversal of the gate description with nodes in the fan-in of a non-input node dynamically sorted when the non-input node is first visited using the following two criteria, in that order: first, nodes having minimum number of non-visited inputs in its dependency cone; second, nodes with minimum sum of indices of visited inputs in its dependency cone (the index of a visited input is the order assigned to the input). As in the case of the **weight** heuristic, in case of a tie, the original ordering of the fan-in of a non-input node is preserved. We will experiment with the following orderings for the variables $w, v_1, \ldots, v_M$:

wv: $w, v_1, \ldots, v_M$.

wvr: $w, v_M, \ldots, v_1$.

vw: $v_1, \ldots, v_M, w$.

vrw: $v_M, \ldots, v_1, w$.

t: ordering which results when the heuristic **topology** is applied to the gate-level description of $G(w, v_1, \ldots, v_M)$ in binary logic and the multiple-valued variables are sorted in increasing order of the average indices over the groups of binary variables encoding each multiple-valued variables.

w: same as t but using the heuristic **weight** for sorting the binary variables.

h: same as t but using the heuristic **H4** for sorting the binary variables.

The size of the coded ROBDD is affected by the ordering of the group of binary variables encoding each multiple-valued variable. Then, it is convenient to use an ordering for those groups of binary variables yielding ROBDDs of as small size as possible. We will experiment with the following orderings for the groups of binary variables encoding each multiple-valued variable:

ml: most to least significant bit.

lm: least to most significant bit.

t: ordering which results when the binary variables are sorted in increasing ordering of the indices given by the **topological** heuristic.

w: same as t but using the **weight** heuristic.

h: same as h but using the **H4** heuristic.

We allow the use of orderings ml and lm for the groups of binary variables in combination with any ordering for the multiple-valued variables. However, we will only allow the use of an ordering t for the groups of binary variables in combination with the ordering t for the multiple-valued variables, the use of an ordering w for the group of binary variables in combination with the ordering w for the multiple-valued variables, and the use of the ordering h for the groups of binary variables in combination with the ordering h for the multiple-valued variables.

3 Benchmarks description

In this section we describe the benchmarks which will be used to evaluate the performance of the combinatorial method for evaluating the yield. The benchmarks are two scalable examples which instantiate systems-on-chip of increasing numbers of components. The first scalable example, called MSn, is the system-on-chip with the architecture illustrated in Figure 4. The system includes a cluster of two "master" Intellectual Property cores IPM and n clusters including two "slave" Intellectual Property cores IPS. Those Intellectual Property cores are interconnected using communication modules CM and CS and two buses. Buses are assumed to be not affected by manufacturing defects. This

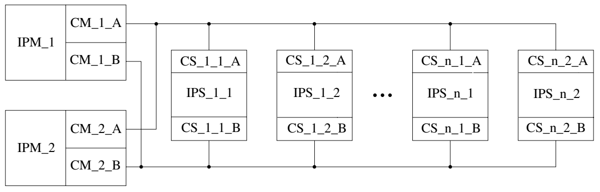

Figure 4. Architecture of system-on-chip MSn.

implies that the system can be conceptualized as made up of only IPMs, IPSs and communication modules. The system is operational if at least an unfailed IPM can communicate with at least an unfailed IPS of each cluster using unfailed communication modules. The communication between the IPM and each IPS has to be direct, i.e. it can only involve a bus and two communication modules. Manufacturing defects are assumed to follow a negative binomial distribution with clustering parameter $\alpha = 3$; for the expected number of defects two values will be assumed: $\lambda = 2$ and $\lambda = 4$. Furthermore, the probabilities P_i will be taken so that $P_L = \sum_{i=1}^{C} P_i = 0.5$ (and, then, λ' has the values 1 and 2) and, calling, P_{IPM} the P_i probability of an IPM, P_{IPS} the P_i probability of an IPS, and P_{C} the P_i probability of a communication module, the following relationships are satisfied: $P_{\text{IPS}}/P_{\text{IPM}} = 0.5$, $P_{\text{C}}/P_{\text{IPM}} = 0.1$.

The second scalable example is the system-on-chip ESEN $n \times m$ with the architecture described in Figure 5 for the case $n = 8, m = 2$. The system includes $(n \times m)/2$ Intellectual Property cores IPA and $(n \times m)/2$ Intellectual Property cores IPB interconnected by a ESEN multiexchange interconnection network with n inputs [28], through $m \times 1$ concentrators (C) in case $m > 1$, in which each switching element (SE) of the first and last stage have a redundant copy. The system is operational if $(n \times m)/2 - 1$ unfailed IPAs and $(n \times m)/2 - 1$ unfailed IPBs can communicate through the interconnection network. It is assumed that links are not affected by manufacturing defects. Thus, the system can be conceptualized as made up of only IPAs, IPBs, SEs and, in case $m > 1$, Cs. As in the first scalable example, manufacturing defects are modeled using a negative binomial distribution with clustering parameter $\alpha = 3$ and for the expected number of defects two values will be assumed: $\lambda = 2$ and $\lambda = 4$. Furthermore, the probabilities P_i will be taken so that $P_L = \sum_{i=1}^{C} P_i = 0.5$ (and, then, λ' has the values 1 and 2) and, calling, P_{IPA} the P_i probability of an IPA, P_{IPB} the P_i probability of an IPB, P_{SE} the P_i probability of a SE, and P_{C} the P_i probability of a C, the following relationships are satisfied: $P_{\text{IPB}}/P_{\text{IPA}} = 0.5$, $P_{\text{SE}}/P_{\text{IPA}} = 0.05$ and $P_{\text{C}}/P_{\text{IPA}} = 0.02$.

Table 1 gives the number of components C of the benchmarks which will be used to evaluate the performance of the combinatorial methods and the number of gates of the gate-level descriptions of the corresponding fault-tree functions used in the experiments.

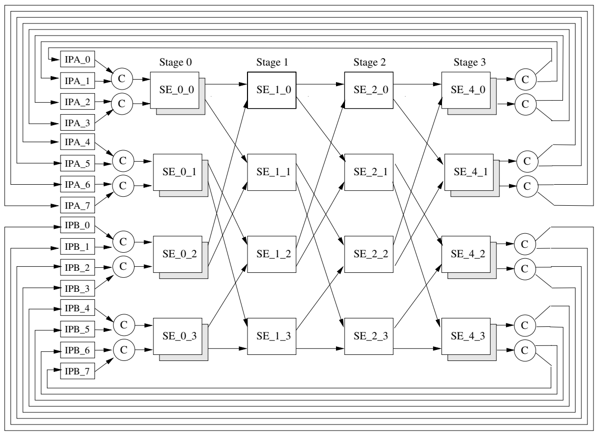

Figure 5. Architecture of system-on-chip ESEN8x2.

Table 1. Number of components (C) of the benchmarks and number of gates of the gate-level descriptions of the corresponding fault-tree functions.

benchmark	C	gates
MS2	18	27
MS4	30	51
MS6	42	75
MS8	54	99
MS10	66	123
ESEN4x1	14	13
ESEN4x2	26	26
ESEN4x4	34	74
ESEN8x1	32	73
ESEN8x2	56	122
ESEN8x4	72	314

4 Results

All experiments reported in this section were performed in a workstation with a Sun-Blade-1000 processor and 4 GB of memory. We will examine first how the ordering of the multiple-valued variables $w, v_1, \ldots, v_M$ affects the size of the ROMDD. After that, we will examine how the ordering of the binary variables within each group of binary variables encoding a multiple-valued variable affects the size of the coded ROBBD which is built to derive from it the ROMDD. We will run the method with an error requirement $\varepsilon = 2 \times 10^{-3}$. Table 2 gives the sizes (number of nodes) of the ROMDD for all benchmarks under the orderings of the multiple-valued variables wv, wvr, vw, vrw, t, w, and h defined in Section 2. The heuristic weight (w) is consistently the one which yields better results. The ordering wvr ($W, V_M, \ldots, V_1$) gives ROMDDs of exactly the same size as w, but it fails in one case in which the method succeeds under the ordering w. Table 3 gives the sizes (number of nodes) of the coded ROBDDs from which the ROMDDs are obtained for the ordering w for the multiple-valued variables and the orderings ml, lm and w for the groups of binary variables considered in Section 2 to be used in conjunction with the ordering w for the multiple-valued variables. The heuristic ml seems to be the best one: it gives better results in all cases except for MS4, in which the other two heuristics perform slightly better. It is interesting to note that the differences among the three heuristics are small and that the heuristics lm and w give exactly the same results in all cases. Based on our experiments, it seems that the best heuristics are w for the multiple-valued variables and ml for the groups of binary variables encoding each multiple-valued variable. We will asses more depthly the performance of the method for those heuristics. Table 4 gives, for the benchmarks in which the method succeeded, the CPU times, peak number of ROBDD nodes (maximum sum of the nodes of the ROBDDs which had to be held simultaneously in memory when processing the generalized fault-tree), size of the coded ROBDD and size of the ROMDD. Several comments are in order. First, CPU times are reasonable in all cases, since in the worst case (ESEN8x2, $\lambda = 2$) the CPU time is about 18 minutes. Second, the peak number of ROBDD nodes can be or not much larger than the size of the final coded ROBDD. In practice, the application of the method is limited by that peak, since it is that peak which determines the peak memory consumption of the method. Third, the size of the coded ROBDD is always about 10 times the size of the ROMDD. With that factor, even an efficient implementation of ROMDDs is likely to consume more memory than the coded ROBDD, which has a much simpler structure. Thus, the approach of working with coded ROBDDs and translate the final coded ROBDD to the ROMDD required to perform the yield computations seems to be a good approach. This is consistent with the conclusion reached by researchers in the ROMDD community that coded ROBDDs is probably the most efficient way of handling ROMDDs [24]. Putting all results together, it seems that the method can efficiently compute the yield of systems with up to about 60 components when the average number of lethal defects is moderate ($\lambda = 2$) and up to about 30 components when the average number of lethal defects is large ($\lambda = 4$). The number of components which the method can handle depends, of course, on the value of the truncation parameter M. That parameter had value 6 for the examples with $\lambda = 2$ and value 10 for the examples with $\lambda = 4$.

5 Conclusions

Systems-on-chip have reached a complexity degree that make them very susceptible to manufacturing defects so that reasonable yields can only be achieved with the use of fault-tolerant techniques. That application of fault-tolerance calls for efficient methodologies for evaluation of yield of fault-tolerant systems-on-chip. Such evaluation is difficult because realistic models for manufacturing defects production have clustering and, thus, introduce dependencies among the failed states of the components making up the system. In this paper, we have developed a combinatorial method for the evaluation of yield of fault-tolerant systems-on-chip supporting realistic models with clustering for manufacturing defects production. The method builds a ROMDD of a boolean function with multiple-valued variables which allows to compute with a predefined accuracy the yield. The ROMDD is built automatically from a gate-level description of the fault-tree specifying the structure function of the system. The computational complexity of the method increases with the expected number of lethal defects in the fault-tolerant system. We have shown, however, that the method is able to deal using currently affordable computational resources with systems having tens of components. In the future, we are planning to extend the method to allow the evaluation of the operational reliability of a fault-tolerant system-on-chip taking into account manufacturing defects.

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *Data structures and algorithms*, Addison-Wesley, 1983.

[2] The BDD Library. Available at `http : //www − 2.cs.cmu.edu/ modelcheck/bdd.html`.

[3] L. Benini and G. De Micheli, "Networks on Chip: A New SoC Paradigm," *IEEE Computer*, 2002, vol. 35, no. 1, pp. 70–78.

[4] M. Bouissou, F. Bruyère and A. Rauzy, "BDD Based Fault-Tree Processing: A Comparison of Variable Ordering Heuristics," *Proc. European Safety and Reliability Association Conference (ESREC'97)*, 1997, C. Guedes Soares, ed., vol. 3, pp. 2045–2052.

[5] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, 1986, vol. C-35, no. 8, pp. 677–691.

[6] K. M. Butler, D. E. Ross, R. Kapurand M. Ray Mercer, "Heuristics to Compute Variable Orderings for Efficient Manipulation of Ordered Binary Decision Diagrams," *Proc. 28th ACM/IEEE Design Automation Conference*, 1991, pp. 417–420.

[7] J. Cunningham, "The Use and Evaluation of Yield Models in Integrated Circuit Manufacturing," *IEEE Trans. on SemiconductorManufacturing*, 1990, vol. 3, no. 2, pp. 60–71.

[8] M. Fujita, H. Fujisawa and N. Kawato, "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams," *Proc. IEEE Int. Conf. on Computer Aided Design (ICCAD'88)*, 1988, pp. 2–5.

[9] M. Fujita, Y. Matsunaga and T. Kakuda, "On Variable Ordering of Binary Decision Diagrams for the Application of Multi-level Logic Synthesis," *Proc. IEEE European Conference on Design Automation (EDAC'91)*, 1991, pp. 50–54.

[10] M. Fujita, H. Fujisawa and Y. Matsunaga, "Variable Ordering Algorithms for Ordered Binary Decision Diagrams and Their Evaluation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 1, January 1993, pp. 6–12.

Table 2. Size (number of nodes) of the ROMDDs used in the method for the evaluation of yield for $\varepsilon = 2 \times 10^{-3}$ for the heuristics for the ordering of the multiple-value variables wv, wvr, vw, vrw, t, w, and h (— indicates that the method failed due to excessive memory requirements).

benchmark	wv	wvr	vw	vrw	t	w	h
MS2, $\lambda = 2$	3,202	2,034	2,035	73,405	3,202	2,034	3,202
MS4, $\lambda = 2$	28,392	22,760	22,761	882,505	28,392	22,760	28,392
MS6, $\lambda = 2$	119,260	103,228	103,229	3,989,917	119,260	103,228	119,260
MS8, $\lambda = 2$	344,320	309,136	309,137	—	344,320	309,136	344,320
MS10, $\lambda = 2$	797,908	731,748	731,749	—	797,908	731,748	797,908
MS2, $\lambda = 4$	25,038	7,534	7,535	—	25,038	7,534	25,038
MS4, $\lambda = 4$	1,345,390	—	—	—	1,345,350	635,530	1,345,350
ESEN4x1, $\lambda = 2$	5,090	3,046	3,047	190,059	5.090	3,046	5,090
ESEN4x2, $\lambda = 2$	11,031	6,995	6,996	486,205	11,031	6,995	11,031
ESEN4x4, $\lambda = 2$	29,391	19,547	19,548	1,469,685	29,391	19,547	29,391
ESEN8x1, $\lambda = 2$	169,764	134,512	134,513	—	169,764	134,512	169,764
ESEN8x2, $\lambda = 2$	373,117	303,657	303,658	—	373,117	303,657	373,117
ESEN4x1, $\lambda = 4$	38,594	11,666	11,667	—	38,594	11,666	38,594
ESEN4x2, $\lambda = 4$	97,671	30,783	30,784	—	67,671	30,783	97,671
ESEN4x4, $\lambda = 4$	296,175	96,231	96,232	—	—	96,231	—

Table 3. Size (number of nodes) of the coded ROBDDs used in the method for the evaluation of yield for $\varepsilon = 2 \times 10^{-3}$ for the heuristic w for the ordering of the multiple-value variables and the heuristic ml, lm and w for the ordering of the groups of binary variables.

benchmark	ml	lm	w
MS2, $\lambda = 2$	24,237	28,418	28,418
MS4, $\lambda = 2$	243,254	236,915	236,915
MS6, $\lambda = 2$	1,120,255	1,290,274	1,290,274
MS8, $\lambda = 2$	3,154,056	3,283,401	3,283,401
MS10, $\lambda = 2$	7,954,261	10,019,092	10,019,092
MS2, $\lambda = 4$	361,428	439,700	439,700
MS4, $\lambda = 4$	11,885,214	11,492,704	11,492,704
ESEN4x1, $\lambda = 2$	19,338	20,721	20,721
ESEN4x2, $\lambda = 2$	54,705	65,208	65,208
ESEN4x4, $\lambda = 2$	184,332	283,338	283,338
ESEN8x1, $\lambda = 2$	904,777	972,506	972,506
ESEN8x2, $\lambda = 2$	2,244,340	2,796,165	2,796,165
ESEN4x1, $\lambda = 4$	105,511	109,692	109,692
ESEN4x2, $\lambda = 4$	378,686	414,939	414,939
ESEN4x4, $\lambda = 4$	1,513,441	2,117,587	2,117,587

Table 4. Performance of the method for the evaluation of yield for $\varepsilon = 2 \times 10^{-3}$ with the heuristic w for ordering the multiple-valued variables and the ordering ml for the groups of binary variables.

benchmark	CPU time (s)	ROBDD peak	ROBDD	ROMDD	yield
MS2, $\lambda = 2$	0.98	30,987	24,237	2,034	0.944
MS4, $\lambda = 2$	6.23	427,130	243,154	22,760	0.965
MS6, $\lambda = 2$	66.4	2,564,600	1,120,255	103,228	0.975
MS8, $\lambda = 2$	262.1	7,518,549	3,154,056	309,136	0.980
MS10, $\lambda = 2$	862.2	20,344,432	7,954,261	731,748	0.984
MS2, $\lambda = 4$	3.59	124,067	116,960	7,534	0.830
MS4, $\lambda = 4$	827.7	14,175,238	11,885,214	635,530	0.885
ESEN4x1, $\lambda = 2$	0.86	37,231	19,338	3,046	0.910
ESEN4x2, $\lambda = 2$	2.72	200,272	54,705	6,995	0.848
ESEN4x4, $\lambda = 2$	14.64	368,815	184,332	19,547	0.829
ESEN8x1, $\lambda = 2$	172.85	6,544,206	904,777	134,512	0.881
ESEN8x2, $\lambda = 2$	1060.7	29,926,091	2,244,340	303,657	0.835
ESEN4x1, $\lambda = 4$	3.47	143,633	105,511	11,666	0.756
ESEN4x2, $\lambda = 4$	18.34	757,529	378,686	30,783	0.642
ESEN4x4, $\lambda = 4$	108.52	3,027,309	1,513,441	96,231	0.605

[11] I. Koren and D. K. Pradhan, "Yield and performance enhancement through redundancy in VLSI and WSI multiprocessor systems," *Proceedings of the IEEE*, vol. 74, no. 5, May 1986, pp. 699-711.

[12] I. Koren and D. K. Pradhan, "Modeling the effect of redundancy on yield and performance of VLSI systems," *IEEE Trans. on Computers*, vol. C-36, no. 3, March 1987, pp. 344-355.

[13] I. Koren, Z. Koren and D. K. Pradhan, "Designing Interconnection Buses in VLSI and WSI for Maximum Yield and Minimum Delay," *IEEE J. of Solid-State Circuits*, 1988, vol. 23, no. 3, pp. 859–865.

[14] I. Koren and C. H. Stapper, "Yield Models for Defect-Tolerant VLSI Circuits: A Review," *Defect and Fault Tolerance in VLSI Systems. vol. I* (Koren I., ed.), Plenum, 1989, pp. 1–21.

[15] I. Koren, Z. Koren and C. A. Stapper, "A Unified Negative-Binomial Distribution for Yield Analysis of Defect-Tolerant Circuits," *IEEE Trans. on Computers*, 1993, vol. 42, no. 6, pp. 724–734.

[16] I. Koren, Z. Koren and C. Stapper, "A Statistical Study of Defect Maps of Large Area VLSI IC's," *IEEE Trans. on Very Large Scale Integration Systems*, 1994, vol. 2, no. 2, pp. 249–256.

[17] I. Koren and Z. Koren, "Analysis of a Hybrid Defect-Tolerance Scheme for High-Density Memory ICs," *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems*, 1997, pp. 38–42.

[18] I. Koren and Z. Koren, "Defect Tolerance in VLSI Circuits: Techniques and Yield Analysis," *Proceedings of the IEEE*, 1999, vol. 86, no. 9, pp. 1819–1838.

[19] T. M. Mak, D. Bhattacharya, C. Prunty, B. Roeder, N. Ramadan, J. Ferguson, and Y. Jianlin, "Cache RAM inductive fault analysis with fab defect modeling," *Proc. IEEE Int. Test Conference*, 1998, pp. 862–871.

[20] S. Malik, A. R. Wang, R. K. Brayton, A. Sangiovanni-Vincentelli, "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment," *Proc. IEEE Int. Conf. on Computer-Aided Design (ICCAD'88)*, 1988, pp. 6–9.

[21] C. Metra, S. Di Francescantonio, T. M. Mak, and B. Ricco, "Evaluation of clock distribution networks' most likely faults and produced defects," *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems*, 2001, pp. 357–365.

[22] F. J. Meyer and D. K. Pradhan, "Modeling Defect Spatial Distribution," *IEEE Trans. on Computers*, vol. 38, no. 4, April 1989, pp. 538–546.

[23] D. M. Miller and R. Drechsler, "Implementing a Multiple-Valued Decision Diagram Package," *Proc. 28th IEEE Int. Symp. on Multiple-Valued Logic*, 1998, pp. 52–57.

[24] Miller, D. M., private communication collecting the conclusions of the *30th IEEE Int. Symp. on Multiple-Valued Logic*, 2000.

[25] S. Minato, N. Ishiura and S. Yajima, "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation," *Proc. 27th ACM/IEEE Design Automation Conference*, 1990, pp. 52–57.

[26] M. Nikolskaïa, A. Rauzy and D. J. Sherman, "Almana: A BDD Minimization Tool Integrating Heuristic and Rewriting Methods," *Proc. Int. Conf. on Formal Methods in Computer Aided Design (FMCAD)*, 1998, pp. 100–114.

[27] D. Nikolos, H. T. Vergos, "On the Yield of VLSI Processors with On-Chip CPU Cache," *IEEE Trans. on Computers*, 1999, vol. 48, no. 10, pp. 1138–1144.

[28] S. Rai and Y. C. Oh, "Tighter Bounds on Full Access Probability in Fault-Tolerant Multistage Interconnection Networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 10, no. 3, March 1999, pp. 328–335.

[29] A. Srinivasan, T. Kam, S. Malik and R. K. Brayton, "Algorithms for Discrete Function Manipulation," *Proc. IEEE Int. Conf. on Computer-Aided Design (ICCAD-90)*, 1990, pp. 92–95.

[30] A. Venkataraman and I. Koren, "Determination of yield bounds prior to routing," *Proc. IEEE Int. Symp. on Defect and Fault-Tolerance in VLSI Systems*, 1999, pp. 4–13.

[31] I. A. Wagner and I. Koren, "An Interactive VLSI CAD Tool for Yield Estimation," *IEEE Trans. on Semiconductor Manufacturing*, vol. 8, no. 2, May 1995, pp. 130–138.

[32] J. Yu and F. J. Ferguson, "Maximum likelihood estimation for failure analysis," *IEEE Trans. on Semiconductor Manufacturing*, vol. 11, no. 4, November 1998, pp. 681–691.