JJEE

Volume 3, Number 3, 2017
Pages 193-207

Jordan Journal of Electrical Engineering

ISSN (Print): 2409-9600, ISSN (Online): 2409-9619

A Distributed Multiplexed Mutual Inter-Unit in-Operation Test Method for Mesh-Connected VLSI Multiprocessors

Jamil S. Al-Azzeh

Department of Computer Engineering, Al-Balqa' Applied University, Amman, Jordan
e-mail: jamil.azzeh@bau.edu.jo

Received: July 19, 2017 *Accepted: August 17, 2017*

Abstract— The problem of in-operation fault detection in mesh-connected multicore and many-core VLSI multiprocessors is considered. A novel approach to the multiprocessor test based on the combination of self-test and mutual inter-unit test performed in a multiplexed mode is presented; it allows for an increase in the successful fault detection probability. Formal rules are defined for forming sets of testing and tested neighbors for each processor node of the mesh that is invariant to the location of the node within the mesh and its dimension. In contrast to the mutual inter-unit test mechanism, the same testing neighbor is alternately used to test the two processors in a multiplexed mode; and the test result is formed by applying the majority operator to the individual faulty/healthy tags calculated by all testing neighbors and the current unit itself in the course of its self-test. The formulae for determining the number of testing neighbors for each node depending on the dimension of the mesh are given. The successful fault detection probability is evaluated in the case when the proposed approach is used. The successful fault detection probability versus the individual test unit reliability dependencies is investigated. For all practically significant cases, the proposed approach is shown to provide an increased successful fault detection probability compared to the mutual inter-unit test and self-checking.

Keywords— Built-in self-test, Fault tolerance, Majority operator, Mesh-connected VLSI multiprocessors, Multiplexer, Mutual inter-unit test.

Nomenclature

D	number of dimensions in the multiprocessor mesh
$u_{x_1x_2\ldots x_d}$	a processor node (unit) of a d-dimensional multiprocessor
X	horizontal coordinate of a node in the mesh
Y	vertical coordinate of a node in the mesh
M	number of rows in the multiprocessor mesh
N	number of columns in the multiprocessor mesh
$C_{x_1x_2\ldots x_d}$	set of neighbors tested by processor $u_{x_1x_2\ldots x_d}$
$C^1_{x_1x_2\ldots x_d}$	subset of neighbors tested by processor $u_{x_1x_2\ldots x_d}$ during phase 1
$C^2_{x_1x_2\ldots x_d}$	subset of neighbors tested by processor $u_{x_1x_2\ldots x_d}$ during phase 2
$K_{x_1x_2\ldots x_d}$	set of neighbors testing processor $u_{x_1x_2\ldots x_d}$
$K'_{x_1x_2\ldots x_d}$	set of nodes testing processor $u_{x_1x_2\ldots x_d}$
B_i	i^{th} parallel thread of the testing algorithm
$T^{x_1x_2\ldots x_d}(k)$	k^{th} test signature produced by testing node $u_{x_1x_2\ldots x_d}$
$k^{\max}_{x_1x_2\ldots x_d}$	number of test signatures supported by testing node $u_{x_1x_2\ldots x_d}$
K	test signature counter of processor $u_{x_1x_2\ldots x_d}$
$\tau^{\max}_{i1}$	maximum time needed by tested processor node $u_{x^{i1}_1x^{i1}_2\ldots x^{i1}_d}$ to form a test response

τ_{i1}	test response wait counter of node $u_{x_1^{i1}x_2^{i1}\ldots x_d^{i1}}$
$R^{x_1^{i1}x_2^{i1}\ldots x_d^{i1}}(k)$	test response signature issued by tested processor $u_{x_1^{i1}x_2^{i1}\ldots x_d^{i1}}$ after $T^{x_1x_2\ldots x_d}(k)$ is received
$R^{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}(k-1)$	test response signature issued by tested processor $u_{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}$ after $T^{x_1x_2\ldots x_d}(k-1)$ is received
$R_0^{x_1^{i1}x_2^{i1}\ldots x_d^{i1}}(k)$	reference test response signature expected to be issued by processor $u_{x_1^{i1}x_2^{i1}\ldots x_d^{i1}}$ after receiving $T^{x_1x_2\ldots x_d}(k)$
$R_0^{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}(k-1)$	reference test response signature expected to be issued by processor $u_{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}$ after receiving $T^{x_1x_2\ldots x_d}(k-1)$
$\varphi^{x_1x_2\ldots x_d}_{x_1^{i1}x_2^{i1}\ldots x_d^{i1}}$	partial faulty/healthy flag of processor $u_{x_1^{i1}x_2^{i1}\ldots x_d^{i1}}$ formed by testing processor $u_{x_1x_2\ldots x_d}$ during test phase 1
$\varphi^{x_1x_2\ldots x_d}_{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}$	partial faulty/healthy flag of processor $u_{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}$ formed by testing processor $u_{x_1x_2\ldots x_d}$ during test phase 2
$\varphi_{x_1x_2\ldots x_d}$	faulty/healthy flag of processor $u_{x_1x_2\ldots x_d}$
s_i	$u_{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}$ response to analyze disable flag
$P(t)$	successful fault detection probability provided by the proposed approach
$\pi(t)$	probability that a processor node of the multiprocessor is unambiguously detected as faulty by a separate test unit or a self-test unit
$P_0(t)$	successful fault detection probability provided by the mutual inter-unit test approach
C_j^i	number of combinations of *i* items out of *j*

I. INTRODUCTION

Multicore and many-core VLSI multiprocessors have become a promising trend in the construction of high-performance embedded systems [1], [2]. Ongoing VLSI miniaturization has enabled the integration of up to thousands of processing cores on a single chip [3], [4]. However, the unreliability of multiprocessor components has emerged as one of the fundamental barriers to future scaling [5]. To maintain connectivity and correct long-term operation of such many-core systems, specific fault-tolerance issues must be taken into account when designing the multiprocessor fabric. Pinpointing the location of defective or faulty components is one of these issues.

If a dedicated defect detection and isolation mechanism is used, a VLSI multiprocessor can be made healthy despite containing faulty components. If no specific spare replacement scheme is used, the overall performance of the multiprocessor gracefully degrades [6], [7]. Otherwise [8], [9], the multiprocessor performance is retained even in the presence of unhealthy components. In both cases, a multiprocessor with physical defects is logically reconfigured and remains healthy as a whole.

The problem of locating faulty components in VLSI multiprocessors is typically solved based on the use of built-in self-checking or neighbor-checking methods [10]-[12], [13]-[17]. For example, it can be done by allowing a processor node to send probe signals to its neighbors and to mark neighboring units as defective if an acknowledgment is not received within a certain time interval. Self-test methods are a simple and efficient solution for providing in-operation fault detection. However, because test hardware itself is not 100% reliable, relatively low testability is the main problem of the self-test approach. Additionally, self-checking mechanisms may miss faults/defects in some cases or treat healthy units as defective; thus, the probability that a processor node's fault is properly detected is not high enough. The same applies to the neighbor-checking methods in which processor nodes test their peers independently, i.e., with no inter-processor coordination.

A more complex approach, the mutual inter-unit test, is presented in [18], where each processor node is checked by a number of its neighbors (3, 5, 7, or more depending on the number of dimensions in the mesh). The final faulty/healthy decision is made based on the majority operator rule. With this approach, the successful fault detection probability increases by 10% and even more. The main drawback of this approach is that testing units are used inefficiently, e.g., a testing unit does nothing and simply spins half a time period until a response token from its tested neighbor arrives. Thus, the problem is to find a solution to improve the utilization of testing units across the multiprocessor mesh and to make an additional increase in the successful fault detection probability.

In the present paper, we propose an extended version of the mutual inter-unit test technique presented in [18], which we refer to as the multiplexed mutual inter-unit test method. Our main contribution is that we increase the utilization of testing hardware by allowing each test unit of each processor node to check its two neighbors (not necessarily direct neighbors), and split the checking time period into two phases. During the first phase, neighbor A is tested while neighbor B is expected to send a test response. During the second phase, neighbor B is checked while neighbor A is expected to provide a test response. Thus, we eliminate the idle time period when a test unit spins until it receives a response from the tested neighbor; therefore, testing hardware is used more efficiently. With our approach, the number of testing neighbors at each processing node is doubled compared to that in [18], making it possible to significantly increase the successful fault detection probability. One must mention that doubling the number of testing neighbors at each processor node does not lead to hardware doubled because each test unit is used in a time division manner.

In the following sections, we formally state the multiplexed mutual inter-unit test approach for a d-dimensional mesh-connected many-core multiprocessor for concurrently detecting faulty/defective nodes across the mesh. A parallel test algorithm is presented based on the proposed methodology. The successful fault detection probability is evaluated to demonstrate that the proposed approach provides increased testability compared to the mutual inter-unit test technique and self-checking.

II. Multiplexed Mutual Inter-Unit Self-Test Idea

The key idea of the multiplexed mutual inter-unit test is that each processor node of the multiprocessor is occasionally checked by a subset of its neighbors, not necessarily direct neighbors (which we call "testing neighbors"); and is additionally self-checked using the same test algorithm. At the same time, the processor node tests another subset of its neighbors, not necessarily direct neighbors (so-called "tested neighbors"). The final faulty/healthy decision for each processor is made based on the majority operator result obtained from the partial results returned by the testing neighbors and the self-checking hardware.

The set of testing neighbors for each processing node is formed depending on the number of dimensions (d) of the multiprocessor topology. Its cardinality plus 1 (to take into account that each node is self-checked) must be odd, so that the majority operator is applicable for producing the final faulty/healthy decision. If a processor node is assumed to be faulty, its status tag (which can be either faulty or healthy) is immediately transmitted to all of its direct neighbors, so that they stop relaying packets to the faulty node and use alternative paths according to the fault-tolerant routing algorithm implemented. The faulty processor node is said to be isolated from the rest of the mesh in such a case. Depending on the fault-tolerant strategy adopted, faulty nodes may be replaced by spare units, thus retaining the multiprocessor performance. They can also be logically "thrown away" with no further replacement, so that the many-core system performance will gracefully degrade.
The multiplexed mutual inter-unit test mechanism may be considered as an advanced form of neighbor- and self-checking because the test hardware itself is also tested online. For example, if one of the testing processor nodes produces a wrong faulty/healthy decision, then the tested node (which is not actually faulty) will not necessarily be detected as faulty by mistake as the resulting faulty/healthy decision is formed by the majority operator applied to the set of partial fault detection tags.

III. CONSTRUCTING TESTED AND TESTING NEIGHBOR SETS

The formation of testing and tested neighbor sets is one of the main issues in the organization of the multiplexed mutual inter-unit test. In this section, we provide formal rules to define these sets for a many-core multiprocessor with an arbitrary dimension $d \geq 2$.
Let us first consider a 2-dimensional system. Let $U = \{u_{xy}\}$ be the set of processor nodes, where x and y are coordinates (indexes) of a particular node in the mesh in the horizontal and vertical dimensions, respectively, $x = \overline{0,n-1}$, $y = \overline{0,m-1}$, with m and n representing the number of rows and columns of the mesh, respectively. Then, for a given arbitrary $x \in \{0,1,\ldots,n-1\}$ and $y = \{0,1,\ldots,m-1\}$, we can define the set of testing neighbors as follows:

$$K_{xy} = \left\{ \begin{array}{c} u_{x,(y+2)\bmod m}, \\ u_{(x+1)\bmod n,(y+1)\bmod m}, \\ u_{(x+2)\bmod n,y}, \\ u_{x+((1-\text{sign}(x))+(1-\text{sign}(x-1)))n-2,y}, \\ u_{x+(1-\text{sign}(x))n-1,y+(1-\text{sign}(y))m-1}, \\ u_{x,y+((1-\text{sign}(y))+(1-\text{sign}(y-1)))m-2} \end{array} \right\} \quad (1)$$

Taking into account that node u_{xy} is also self-checked, the set of processors that make the final faulty/healthy decision is

$$K'_{xy} = K_{xy} \cup \{u_{xy}\} \quad (2)$$

Fig. 1 illustrates rules (1) and (2) in detail. In this figure, the tiles correspond to the processing nodes; the testing neighbors are shown in gray. For the processing nodes located at the edges or one step next to at least one edge of the mesh, the testing processors are taken at the opposite edges of the mesh (the dashed tiles and arrows illustrate this situation). According to Fig. 1, the proposed approach requires that $m \geq 5$ and $n \geq 5$.

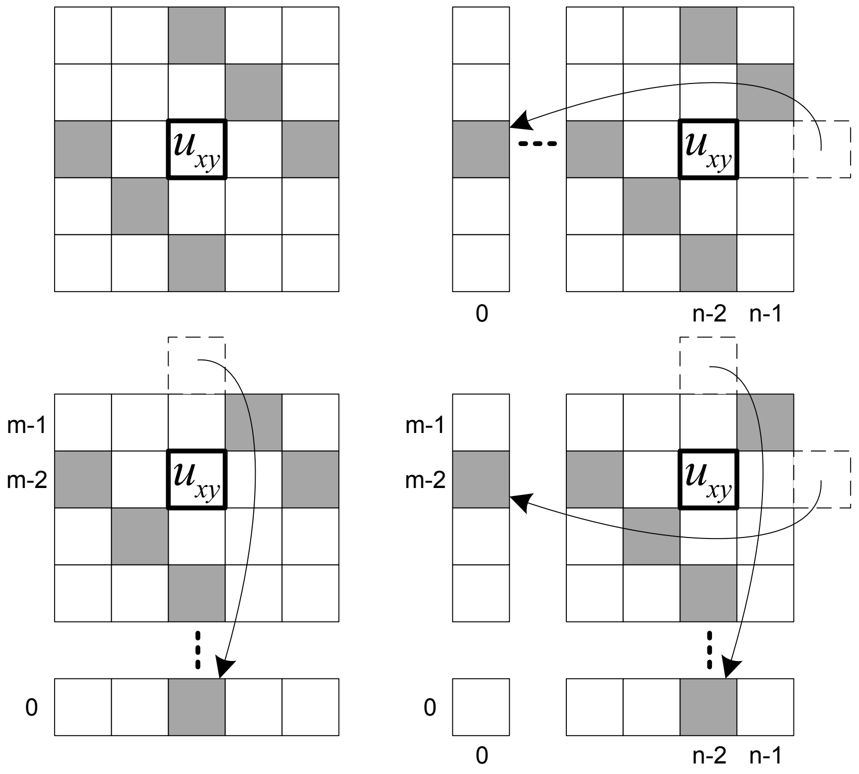


Fig. 1. Formation of testing neighbor sets in a 2-dimensional many-core multiprocessor

The set C_{xy} of processor nodes tested by processor u_{xy}, $x \in \{0,1,\ldots,n-1\}$ and $y = \{0,1,\ldots,m-1\}$, is formally the same as K_{xy}. However, set C_{xy} is split into subsets C_{xy}^1 and C_{xy}^2 which are to be checked during phase 1 and 2, respectively (this is why we call our method the multiplexed test):

$$C_{xy}^1 = \begin{Bmatrix} u_{x,(y+2)\,mod\,m}, \\ u_{(x+1)\,mod\,n,(y+1)\,mod\,m}, \\ u_{(x+2)\,mod\,n,y} \end{Bmatrix} \tag{3}$$

$$C_{xy}^2 = \begin{Bmatrix} u_{x+((1-\text{sign}(x))+(1-\text{sign}(x-1)))n-2,y}, \\ u_{x+(1-\text{sign}(x))n-1,y+(1-\text{sign}(y))m-1}, \\ u_{x,y+((1-\text{sign}(y))+(1-\text{sign}(y-1)))m-2} \end{Bmatrix} \tag{4}$$

It is important to mention that for each node of C^1_{xy}, there is a corresponding node in C^2_{xy}. Thus, a one-to-one correspondence may be defined for each pair of sets C^1_{xy} and C^2_{xy}:

$$\begin{bmatrix} u_{x,(y+2)\,mod\,m} \leftrightarrow u_{x+((1-\mathrm{sign}(x))+(1-\mathrm{sign}(x-1)))n-2,y} \\ u_{(x+1)\,mod\,n,(y+1)\,mod\,m} \leftrightarrow u_{x+(1-\mathrm{sign}(x))n-1,y+(1-\mathrm{sign}(y))m-1} \\ u_{(x+2)\,mod\,n,y} \leftrightarrow u_{x,y+((1-\mathrm{sign}(y))+(1-\mathrm{sign}(y-1)))m-2} \end{bmatrix} \tag{5}$$

Equation (5) in turn sets a mapping of tested neighbor pairs onto the individual test units of each node u_{xy}. Fig. 2 illustrates the construction of a tested neighbor set and shows how correspondence (5) is applied. In this figure, the dots denote separate test units of the current node u_{xy}. The dashed arrows show which neighbor nodes are checked by the corresponding test units; 1's and 2's represent the test phase numbers for these test units. According to Fig. 2, node u_{xy} includes 3 individual test units (3 dots): the uppermost one (upper left corner of tile u_{xy}) will check neighbors $u_{x-2,y}$ and $u_{x,y+2}$; the middle one (the middle of tile u_{xy}) will check nodes $u_{x-1,y-1}$ and $u_{x+1,y+1}$; and the lowermost one (lower right corner of tile u_{xy}) will check processors $u_{x,y-2}$ and $u_{x+2,y}$.

To define sets $K_{x_1x_2\ldots x_d}$, $C^1_{x_1x_2\ldots x_d}$,and $C^2_{x_1x_2\ldots x_d}$ for a general case d-dimensional mesh, it is necessary to extend formulae (1)-(5) by adding extra properly indexed elements with all possible combinations of indices. The d-dimensional case formulae are not stated here for complexity reasons.
One can prove that

$$\left|K_{x_1x_2\ldots x_d}\right| = 2\left[d(d-1)+1\right] \tag{6}$$

$$\left|K'_{x_1x_2\ldots x_d}\right| = \left|K_{x_1x_2\ldots x_d}\right| + 1 = 2d(d-1)+3. \tag{7}$$

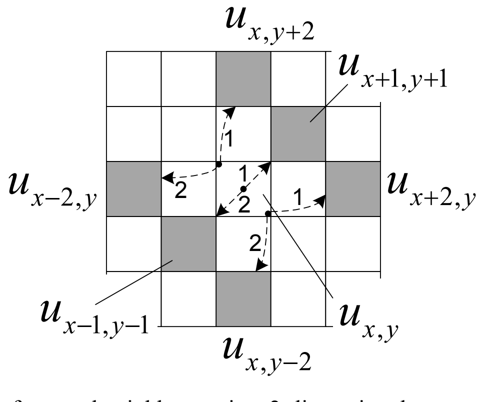


Fig. 2. Formation of a tested neighbor set in a 2-dimensional many-core multiprocessor

According to (7), $\left|K_{x_1x_2\ldots x_d}\right| = 1(\bmod 2)$, each processor node has an odd number of testing processors (including itself),making it possible to apply the majority operator to produce the resulting faulty/healthy tag. The number of testing neighbors in 2-dimensional meshes is minimal: $\left|K'_{xy}\right| = 7$. In a 3-dimensional array, each node has $\left|K'_{xyz}\right| = 15$ testing nodes. This is more than twice that of the mutual inter-unit test method proposed in [18]. Therefore, we assume that testability increases (a proof is given below).

IV. Detailed Description of the Test Procedure

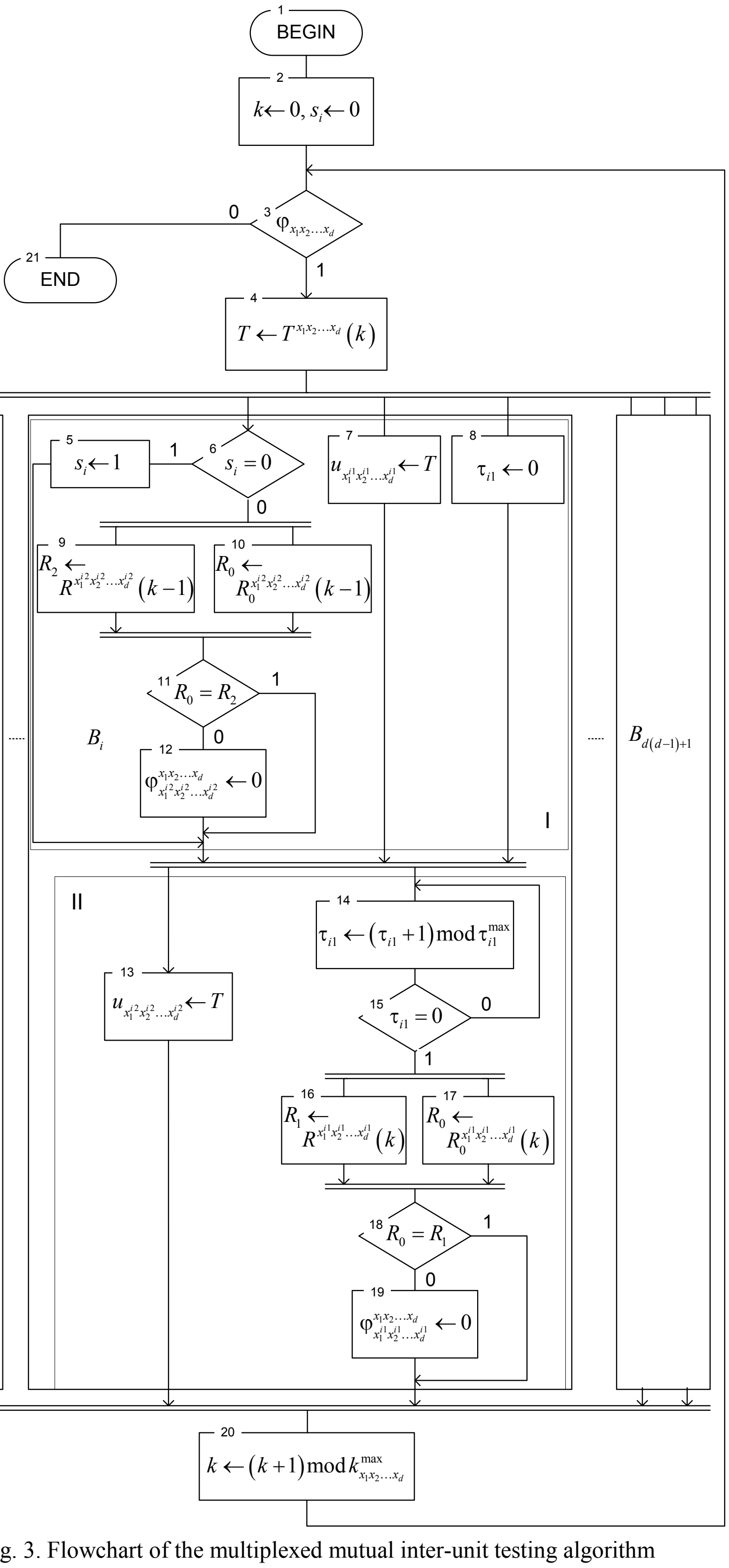
The multiplexed mutual inter-unit test process can be represented in the form of a parallel algorithm with a set of threads $B_1, B_2, \ldots, B_{d(d-1)+1}$, where thread B_i defines a test statement sequence performed by a particular test unit corresponding to tested neighbors $u_{x_1^{i1}x_2^{i1}\ldots x_d^{i1}}$ and $u_{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}$ (see Fig. 3). The algorithm applies to many-core mesh-connected systems with an arbitrary dimension $d \geq 2$.

The algorithm in Fig. 3 includes the main test loop while the current processor node (which is $u_{x_1x_2\ldots x_d}$) is considered healthy by its testing nodes, including itself (being healthy is indicated as $\varphi_{x_1x_2\ldots x_d} = 1$). Each loop is split into 2 phases (numbered as I and II in Fig. 3).

During the first phase, a test signature $T^{x_1x_2\ldots x_d}(k)$ is sent to all the tested neighbor nodes of $C^1_{x_1x_2\ldots x_d}$ (see statements 4 and 7, where $T \leftarrow T^{x_1x_2\ldots x_d}(k)$ and $u_{x_1^{i1}x_2^{i1}\ldots x_d^{i1}} \leftarrow T$ show the test signature transmission via internal buffer *T*). At the same time, response signatures from all the tested nodes of $C^2_{x_1x_2\ldots x_d}$ are received and compared to the expected response (see statements 9-12). The expected response $R_0^{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}(k-1)$ is stored in internal buffer R_0, while the one provided by the tested neighbor is buffered in internal register R_2. If $R_0 \neq R_2$ (condition 11). Statement 12 is executed; and tag $\varphi^{x_1x_2\ldots x_d}_{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}$ becomes zero. This means that the current node $u_{x_1x_2\ldots x_d}$ assumes neighbor $u_{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}$ to be faulty. Otherwise, statement 12 is skipped; and neighbor $u_{x_1^{i2}x_2^{i2}\ldots x_d^{i2}}$ is still assumed to be healthy.

During the second phase (which follows the barrier in the middle of branch B_i), the test signature $T^{x_1x_2\ldots x_d}(k)$ is transferred to the tested processor nodes of $C^2_{x_1x_2\ldots x_d}$ (see statement 13, where $u_{x_1^{i2}x_2^{i2}\ldots x_d^{i2}} \leftarrow T$ show the test signature transmission from internal buffer *T*). Concurrently, response signatures are transmitted from the tested nodes of $C^1_{x_1x_2\ldots x_d}$ and then analyzed by the current processor node (see statements 16-19). The expected response $R_0^{x_1^{i1}x_2^{i1}\ldots x_d^{i1}}(k)$ is transferred to internal buffer R_0, while the response issued by the tested neighbor is latched in internal register R_1. If $R_0 \neq R_1$ (condition 18), then statement 19 follows; and tag $\varphi^{x_1x_2\ldots x_d}_{x_1^{i1}x_2^{i1}\ldots x_d^{i1}}$ becomes clear. This means that the node

$u_{x_1 x_2 \dots x_d}$ assumes neighbor $u_{x_1^{i1} x_2^{i1} \dots x_d^{i1}}$ to be faulty. Otherwise, statement 19 is not executed; and neighbor $u_{x_1^{i1} x_2^{i1} \dots x_d^{i1}}$ is still supposed to be healthy.



1 BEGIN
2 $k \leftarrow 0, s_i \leftarrow 0$
3 $\varphi_{x_1 x_2 \dots x_d}$
0
1
21 END
4 $T \leftarrow T^{x_1 x_2 \dots x_d}(k)$
5 $s_i \leftarrow 1$
6 $s_i = 0$
7 $u_{x_1^{i1} x_2^{i1} \dots x_d^{i1}} \leftarrow T$
8 $\tau_{i1} \leftarrow 0$
9 $R_2 \leftarrow R^{x_1^{i2} x_2^{i2} \dots x_d^{i2}}(k-1)$
10 $R_0 \leftarrow R_0^{x_1^{i2} x_2^{i2} \dots x_d^{i2}}(k-1)$
11 $R_0 = R_2$
12 $\varphi^{x_1 x_2 \dots x_d}_{x_1^{i2} x_2^{i2} \dots x_d^{i2}} \leftarrow 0$
B_1
B_i
$B_{d(d-1)+1}$
I
II
13 $u_{x_1^{i2} x_2^{i2} \dots x_d^{i2}} \leftarrow T$
14 $\tau_{i1} \leftarrow (\tau_{i1}+1) \bmod \tau_{i1}^{\max}$
15 $\tau_{i1} = 0$
16 $R_1 \leftarrow R^{x_1^{i1} x_2^{i1} \dots x_d^{i1}}(k)$
17 $R_0 \leftarrow R_0^{x_1^{i1} x_2^{i1} \dots x_d^{i1}}(k)$
18 $R_0 = R_1$
19 $\varphi^{x_1 x_2 \dots x_d}_{x_1^{i1} x_2^{i1} \dots x_d^{i1}} \leftarrow 0$
20 $k \leftarrow (k+1) \bmod k^{\max}_{x_1 x_2 \dots x_d}$

Fig. 3. Flowchart of the multiplexed mutual inter-unit testing algorithm

Note that statements 8, 14, and 15 are necessary to count the time needed to form test response signatures by the tested processors (only phase 2 needs such a timer; owing to the use of barriers, it automatically applies to phase 1).
In the algorithm shown in Fig. 3, much work is done in parallel making it possible to concurrently test processor nodes across the entire mesh. All the conditions and statements of the algorithm are simple enough to be implemented in hardware, which additionally contributes to the inter-unit test environment performance.

V. Comparing the Proposed Approach to the Mutual Inter-Unit Test and Self-Checking

The proposed approach is a good alternative to existing in-operation testing methods based on self-checking or neighbor-checking. It provides better multiprocessor testability, which we will demonstrate in the present section.
Let $\pi(t)$ be the probability that a processor node of the multiprocessor is unambiguously detected as faulty by an individual test or a self-test unit. All possible situations when a node is unambiguously detected as faulty by its tested neighbors are considered. Let us first assume that $d=2$. In this case $\left|K'_{xy}\right|=7$, i.e., each node is checked by 7 test units. The probability that all these 7 units assume the current node to be faulty is:

$$P_7^7(t)=\pi(t)^7=\pi(t)^7\left[1-\pi(t)\right]^0 \tag{8}$$

If there are 6 units out of 7 assuming the current node to be faulty, then the fault detection probability will be:

$$P_7^6(t)=C_7^6\pi(t)^6\left[1-\pi(t)\right]^1 \tag{9}$$

where $C_7^6=\dfrac{7!}{6!(7-6)!}$ stands for the number of possible selections of 6 testing neighbors out of 7 testing units. It correctly reports that the current node is faulty. In general, it is known that $C_N^M=\dfrac{N!}{M!(N-M)!}$.
In the same way, we can deduce formulae for the cases when there are 5 and 4 units out of 7 indicating that the current node is faulty:

$$P_7^5(t)=C_7^5\pi(t)^5\left[1-\pi(t)\right]^2 \tag{10}$$

$$P_7^4(t)=C_7^4\pi(t)^4\left[1-\pi(t)\right]^3 \tag{11}$$

If there are three or less units reporting that the current node is faulty, then we consider it healthy according to the majority operator rule.
The sum of (8)-(11) gives us the following formula:

$$P(t)\big|_{d=2}=\sum_{i=4}^{7}P_7^i(t)=\sum_{i=4}^{7}C_7^i\pi(t)^i\left[1-\pi(t)\right]^{7-i} \tag{12}$$

Formula (12) makes it possible to evaluate the successful fault detection probability in a 2-dimensional mesh multiprocessor.

Let us now assume that $d=3$. In this case $\left|K'_{xy}\right|=15$, each node is checked by 15 test units. The fault detection probability formulae can be deduced in the same way as (8)-(11):

$$P_{15}^{15}(t)=\pi(t)^{15}=\pi(t)^{15}\left[1-\pi(t)\right]^{0} \tag{13}$$

$$P_{15}^{14}(t)=C_{15}^{14}\pi(t)^{14}\left[1-\pi(t)\right]^{1} \tag{14}$$

$$P_{15}^{13}(t)=C_{15}^{13}\pi(t)^{13}\left[1-\pi(t)\right]^{2} \tag{15}$$

$$P_{15}^{12}(t)=C_{15}^{12}\pi(t)^{12}\left[1-\pi(t)\right]^{3} \tag{16}$$

$$P_{15}^{11}(t)=C_{15}^{11}\pi(t)^{11}\left[1-\pi(t)\right]^{4} \tag{17}$$

$$P_{15}^{10}(t)=C_{15}^{10}\pi(t)^{10}\left[1-\pi(t)\right]^{5} \tag{18}$$

$$P_{15}^{9}(t)=C_{15}^{9}\pi(t)^{9}\left[1-\pi(t)\right]^{6} \tag{19}$$

$$P_{15}^{8}(t)=C_{15}^{8}\pi(t)^{8}\left[1-\pi(t)\right]^{7} \tag{20}$$

Note that if there are 7 or less units reporting that the current node is faulty, then we again consider it healthy according to the majority operator rule.

The sum of equations (13)-(20) gives us the following formula:

$$P(t)\big|_{d=3}=\sum_{i=8}^{15}P_{15}^{i}(t)=\sum_{i=8}^{15}C_{15}^{i}\pi(t)^{i}\left[1-\pi(t)\right]^{15-i} \tag{21}$$

Formula (21) makes it possible to evaluate the successful fault detection probability in a 3-dimensional mesh multiprocessor.

In the same fashion, taking into account (7), we can deduce the following generic equation for a given arbitrary $d\geq 2$:

$$P(t)=\sum_{i=\left\lceil {2d(d-1)+3}/{2}\right\rceil}^{2d(d-1)+3}C_{2d(d-1)+3}^{i}\pi(t)^{i}\left[1-\pi(t)\right]^{2d(d-1)-i+3} \tag{22}$$

Using (22), it is possible to evaluate the probability $P(t)$ that an arbitrary processor node is properly detected as faulty by itself and its tested neighbors in a multiprocessor of any dimension.

Fig. 4 shows the $P(t)$ versus $\pi(t)$ graphs for a fixed number of multiprocessor dimensions $d\in\{2,3,4,5,6\}$, is obtained based on (22).

According to Fig. 4, it is evident that:

- the probability $P(t)$ grows significantly as the multiprocessor dimension d increases because the number of neighbors testing each processor node is proportional to d^2 ($P(t)$ grows faster than that in the mutual inter-unit test method [18] as a result of the presence of a multiplier "2" in (7));
- the probability $P(t)$tends to "1" much faster than the probability $\pi(t)$ does; and the greater the value of $\pi(t)$, the faster $P(t)$ tends to "1". In a 2-dimensional multiprocessor, $P(t)$ is equal to 0.87 when $\pi(t)$ is still as low as 0.7, i.e., $P(t)/\pi(t)\approx 1.24$; in a 3-dimensional system, $P(t)$ tends to 0.95 while $\pi(t)$ remains as low as 0.7. The probability that a faulty node is properly detected becomes 1.36 times higher.

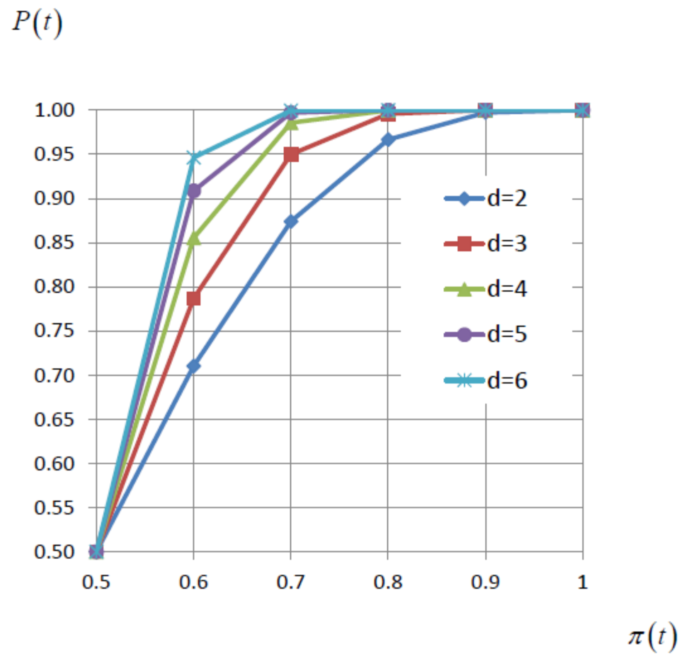


Fig. 4. $P(t)$ versus $\pi(t)$ graphs obtained for a fixed $d\in\{2,3,4,5,6\}$

One must mention that the proposed approach works well only if$\pi(t)>0.5$. If $\pi(t)=0.5$, then $P(t)=0.5$. Therefore, the multiplexed mutual inter-unit test has no effect. If $\pi(t)<0.5$, then $P(t)$degrades with respect to $\pi(t)$. Thus, it is assumed that $\pi(t)>0.5$. We suppose that a standalone test unit is capable of properly detecting faults in more than half of the cases. Based on (8), it is possible to compare the proposed approach to the mutual inter-unit test [18]. Let $P_0(t)$ be the successful fault detection probability provided by the mutual inter-unit test. Let us investigate the $P(t)/P_0(t)$ versus $\pi(t)$ dependencies for a fixed $d\in\{2,3,4,5,6,7,8,9\}$. The $P(t)/P_0(t)$ versus $\pi(t)$ graphs are shown in Fig. 5.

The graphs of Fig. 5 demonstrate that the proposed approach provides better testability than the mutual inter-unit test method in low-dimensional multiprocessors ($2\le d\le 3$). $P(t)/P_0(t)$ reaches its maximum at $0.6\le\pi(t)\le 0.7$. The maximum successful fault detection probability increase takes place when $\pi(t)=0.7$ and $d=2$, with a value of approximately 11.5%. In a 3-dimensional multiprocessor, $P(t)/P_0(t)$ reaches its maximum at $\pi(t)=0.6$ and exhibits a value of 10.9%. We must mention that the proposed approach becomes less efficient as the multiprocessor dimension d and the probability $\pi(t)$ increase. For example, if $d=4$ and $\pi(t)=0.7$, the successful fault detection probability growth becomes as low as 5%; and if $\pi(t)=0.8$, it decreases to just 0.68%. When $d \geq 5$, the proposed approach retains its efficiency in a narrow range of $\pi(t)$. For $\pi(t) \geq 0.7$, a maximum growth of only 2.5% is attained.

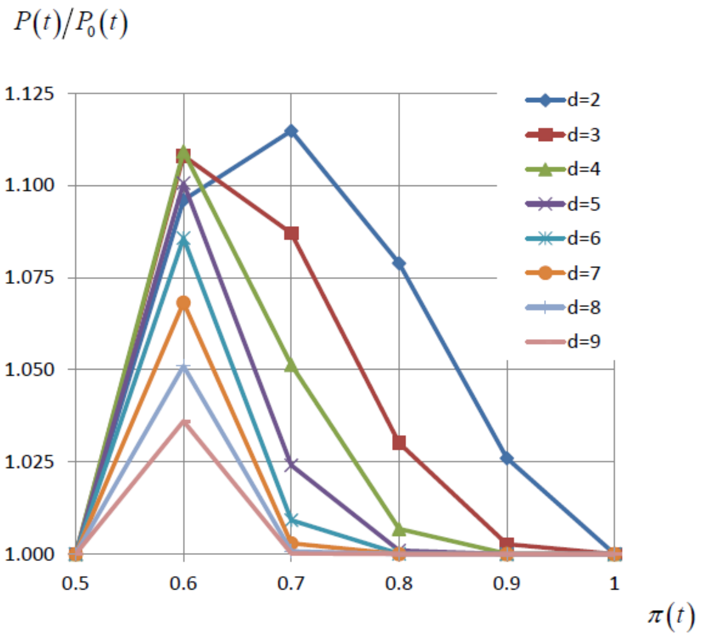


Fig. 5. $P(t)/P_0(t)$ versus $\pi(t)$ graphs for a fixed $d \in \{2, 3, 4, 5, 6, 7, 8, 9\}$

Using (8), one can compare the proposed approach to traditional self-checking by assuming that each node periodically performs a self-test with no communication with its peers. Under the assumption that a neighbor-test and a self-test are based on the same procedure, it is possible to allow $\pi(t)$ to be the successful fault detection probability provided by an individual test and a self-test unit of a processor node. Let us investigate the $P(t)/\pi(t)$ versus $\pi(t)$ dependencies for a fixed $d \in \{2,3,4,5,6,7,8,9\}$. The $P(t)/\pi(t)$ versus $\pi(t)$ relationships are shown in Fig. 6.

From the graph shown in Fig. 6, we can see that the proposed approach provides the lowest fault detection probability increase in the case of a 2-dimensional multiprocessor: given $\pi(t)=0.9$, then $P(t)/\pi(t) \approx 1.108$; if $\pi(t)=0.8$, then $P(t)/\pi(t) \approx 1.245$; if $\pi(t)=0.7$, then $P(t)/\pi(t) \approx 1.357$; if $\pi(t)=0.6$, then $P(t)/\pi(t) \approx 1.184$; if $\pi(t)=0.55$, then $P(t)/\pi(t) \approx 1.108$. The graph in Fig. 6 also shows that for a high $\pi(t)$ (i.e., $\pi(t) \geq 0.9$), $P(t)/\pi(t)$ becomes almost constant regardless of the value of d given. The closer $\pi(t)$ is to 1, the lower the increase in the successful fault detection probability. For $\pi(t)=0.9$ and

$d > 2$, we obtain $P(t)/\pi(t) \approx 1.11$. Thus, we have an 11% testability growth. The proposed approach appears to be the most efficient when $\pi(t) \approx 0.6 \div 0.7$.

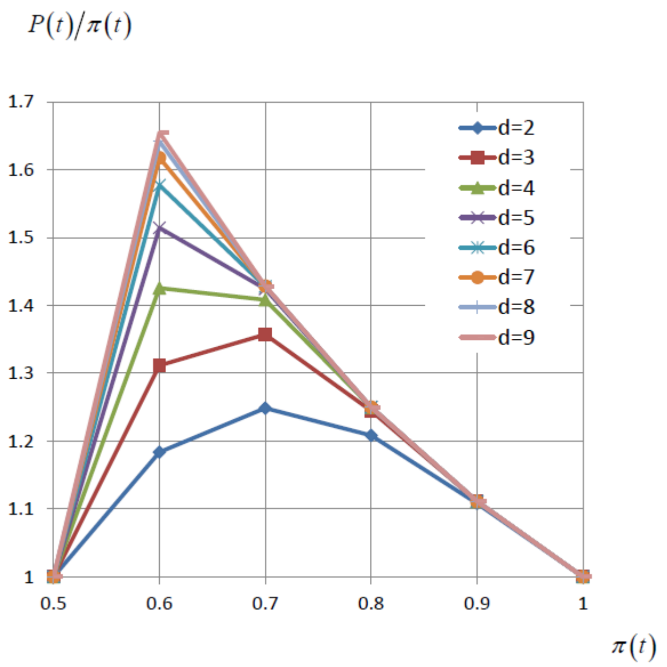


Fig. 6. $P(t)/\pi(t)$ versus $\pi(t)$ graphs for a fixed $d \in \{2, 3, 4, 5, 6, 7, 8, 9\}$

Note that a decreasing $\pi(t)$ makes $P(t)/\pi(t)$ grow; furthermore, the higher the value of *d*, the faster the growth. For example, if $\pi(t) = 0.8$ and $d > 2$, then $P(t)/\pi(t) > 1.24$. Thus, the proposed approach is 24% better than self-checking. For $\pi(t) = 0.7$ and $d > 3$, we obtain $P(t)/\pi(t) > 1.4$, indicating a 40% increase. When $\pi(t) = 0.6$ and $d > 5$, we have $P(t)/\pi(t) > 1.51$. Thus, our approach provides a 51% testability growth.

In most cases, we can presume that the multiplexed mutual inter-unit test approach guarantees that the successful fault detection probability will increase by 11-25% compared to the self-checking approach. Maximum growth is attained when $\pi(t) = 0.7$. Note that the mutual inter-unit test [18] has a growth of 8-12% under the same assumptions. It appears to be1.37-2.083 times less efficient.

VI. Conclusion

In the present paper, we proposed the multiplexed mutual inter-unit test approach to improve the testability of mesh-connected multicore and many-core multiprocessors by increasing the successful fault detection probability with respect to the mutual inter-unit test and traditional self-checking. We showed that our approach is applicable to multiprocessors of arbitrary dimensions; its effectiveness was demonstrated to be maximized when the number of dimensions was equal to 2 or 3, which matches the technological limitations of modern VLSI multiprocessors. The multiplexed mutual inter-unit test technique allows for online hardware-level testing of all the processor nodes across the mesh in parallel, thus significantly contributing to the performance of the test environment.

REFERENCES

[1] Mellanox Corporation (2016). TILE-Gx8036™ Overview [Online]. Available: http://www.mellanox.com/related-docs/prod_multi_core/PB_TILE-Gx36.pdf

[2] Azul Systems (2015). Vega 3 Processor [Online]. Available: http://www.azulsystems.com/products/vega/processor.

[3] S. Borkar, "Thousand core chips: a technology perspective," *Proceedings of 44th ACM/IEEE Design Automation Conf*erence, pp. 746-749, 2007.

[4] The Parallela Project (2014). Epiphany-V: A 1024-core 64-bit RISC processor [Online]. Available: https://www.parallella.org/2016/10/05/epiphany-v-a-1024-core-64-bit-risc-processor.

[5] R. Marculescu, U. Ogras, L. Peh, N. Jerger and Y. Hoskote, "Outstanding research problems in NOC design: system, microarchitecture, and circuit perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, pp. 3-21, 2009.

[6] M. Fukushi and S. Horiguchi, "Reconfiguration algorithm for degradable processor arrays based on row andcolumn rerouting," *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 496-504, 2004.

[7] J. Wu and S. Thambipillai, "An improved reconfiguration algorithm for degradable VLSI/WSI arrays," *Journal of Systems Architecture,* vol. 49, no 1-2, pp.23-31, 2003.

[8] V. Roychowdhury, J. Bruck and T. Kailath, "Efficient algorithms for reconfiguration in VLSI/WSI arrays," *IEEE Transactions on Computers,* vol. 39, no.4, pp.480-489, 1990.

[9] I. Takanami, "Built-in self-reconfiguring systems for fault tolerant mesh-connected processor arrays by direct spare replacement," *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 134-142, 2001.

[10] M. Aguilera, W. Chen and S. Toueg, "Failure detection andconsensus in the crash-recovery model," *Distributed Computing,* vol. 13, no. 2, pp.99-125, 2000.

[11] S. Jafri, S. Piestrak, O. Sentieys and S. Pillement, "Design of the coarse-grained reconfigurable architecture DART with on-line error detection," *Microprocessors and Microsystems,* vol. 38, no. 2, pp.124-136, 2014.

[12] J. Rajski, J. Tyszer, M. Kassab and N. Mukherjee, "Embedded deterministic test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 23, no. 5, 776-792, 2004.

[13] S. Lin, W. Shen, C. Hsu, C. Chao and A. Wu, "Fault-tolerant router with built-in self-test/self-diagnosis and fault-isolation circuits for 2D mesh based chip multiprocessor systems," *Proceedings of International Symposium on VLSI Design, Automation and Test*, pp. 72-75, 2009.

[14] Z. Zhang, D. Refauvelet, A. Greiner, M. Benabdenbi and F. Pecheux, "On-the-field test and configuration infrastructure for 2-D-mesh NoCs in shared-memory many-core architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 6, pp.1364-1376, 2014.

[15] P. Bernardi, L. Ciganda, E. Sanchez and M. Reorda, "MIHST: a hardware technique for embedded microprocessor functional on-line self-test," *IEEE Transactions on Computers,* vol. 63, no.11, pp. 2760-2771. 2014.

[16] G. Miorandi, A. Celin, M. Favalli and D. Bertozzi, "A built-in self-testing framework for asynchronous bundled-data NoC switches resilient to delay variations," *Proceedings of IEEE International Symposium on Networks-on-Chip (NOCS)*, pp.1-8, 2016.

[17] L. Huang, J. Wang, M. Ebrahimi, M. Daneshtalab, X. Zhang, G. Li and A. Jantsch, "Non-blocking testing for network-on-chip," *IEEE Transactions on Computers,* vol. 65, no. 3, pp. 679-692, 2016.

[18] J. Al-Azzeh, M. Leonov, D. Skopin, E. Titenko and I. Zotov, "The organization of built-in hardware-level mutual self-test in mesh-connected VLSI multiprocessors," *International Journal on Information Technology*, vol. 3, no. 2, pp. 29-33, 2015.