Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 04, No. 02, August 2018.

IMPROVED TESTABILITY METHOD FOR MESH-CONNECTED VLSI MULTIPROCESSORS

Jamil Al-Azzeh

(Received: 04-Feb.-2018, Revised: 27-Mar.-2018 and 16-Apr.-2018, Accepted: 25-Apr.-2018)

ABSTRACT

The problem of in-operation embedded hardware-level fault detection in mesh-connected VLSI multiprocessors is considered. A new approach to the multiprocessor test based on the mutual inter-unit checking is presented, which allows increasing the successful fault detection probability. Formal rules are defined for forming sets of testing and tested neighbors for each unit which are invariant to the location of the unit within the topological structure of the multiprocessor and its dimension. The final test result for each processor unit is formed by applying the majority operator to the individual faulty/healthy tags calculated by all testing neighbors. The formulae to determine the number of testing neighbors for each unit depending on the dimension of the multiprocessor are given. The successful fault detection probability is evaluated in the case when the proposed approach is used; the successful fault detection probability vs. multiprocessor dimension and the successful fault detection probability vs. the individual test unit reliability dependencies are investigated. The proposed approach is shown to provide increased successful fault detection probability compared to the self-test for all practically significant cases.

KEYWORDS

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

NOMENCLATURE

D	The number of dimensions in the multiprocessor mesh
$u_{x_1x_2\ldots x_d}$	A unit of a d-dimensional multiprocessor
x	The horizontal coordinate of a unit in the mesh
y	The vertical coordinate of a unit in the mesh
z	The "depth" coordinate of a unit in the 3D mesh
m	The number of rows in the multiprocessor mesh
n	The number of columns in the multiprocessor mesh
p	The "depth" of the 3D multiprocessor mesh
$C_{x_1x_2\ldots x_d}$	The set of neighbors tested by processor unit $u_{x_1x_2\ldots x_d}$
$K_{x_1x_2\ldots x_d}$	The set of neighbors testing processor unit $u_{x_1x_2\ldots x_d}$
$\varphi_{x_1x_2\ldots x_d}$	The faulty/non-faulty flag of processor unit $u_{x_1x_2\ldots x_d}$
$\varphi_{x_1x_2\ldots x_d}^{x_1'x_2'\ldots x_d'}$	The partial faulty/non-faulty flag of a tested processor unit $u_{x_1'x_2'\ldots x_d'}$ formed by testing processor unit $u_{x_1x_2\ldots x_d}$
$u_{x_1^ix_2^i\ldots x_d^i}$	An i^{th} tested neighbor of processor unit $u_{x_1x_2\ldots x_d}$

J. Al-Azzeh is with Department of Computer Engineering, Al-Balqa Applied University, Al-Salt, Jordan. Email: jamil.azzeh@bau.edu.jo

B_i	An i^{th} parallel thread of the mutual inter-unit test algorithm
$T^{x_1x_2\ldots x_d}(k)$	A k^{th} test signature produced by testing unit $u_{x_1x_2\ldots x_d}$
$k^{\max}_{x_1x_2\ldots x_d}$	The number of test signatures supported by testing unit $u_{x_1x_2\ldots x_d}$
$R^{x_1^i x_2^i \ldots x_d^i}(k)$	The test response signature issued by a tested unit $u_{x_1^i x_2^i \ldots x_d^i}$ after $T^{x_1x_2\ldots x_d}(k)$ is received
$R_0^{x_1^i x_2^i \ldots x_d^i}(k)$	The reference test response signature expected to be issued by a tested unit $u_{x_1^i x_2^i \ldots x_d^i}$ after receiving test signature $T^{x_1x_2\ldots x_d}(k)$
$\tau_0^{\max}$	The interval of time between two adjacent test loops
τ_0	Next test loop wait counter
$\tau_i^{\max}$	The maximum time needed to form the test response by an i^{th} tested unit $u_{x_1^i x_2^i \ldots x_d^i}$
τ_i	Test response wait counter
$\pi^+(t)$	The probability that a processor unit is properly detected as faulty by a separate test unit
$P^+(t)$	The probability that a processor unit is properly detected as faulty by a set of testing neighbors
C_j^i	The number of combinations of i elements out of j
#	The majority operator

1. INTRODUCTION

Many modern computer applications require the use of high performance VLSI-based embedded systems to attain desired efficiency, low cost and high flexibility [1]-[22]. VLSI multiprocessors (VLSI MPs), also known as multicore processors, represent a highly efficient solution for implementing such high performance embedded systems, combining fine-grain concurrency with decentralized and logically distributed architecture [23]. Increasing complexity of VLSI MPs becomes a problem, because the probability that a processor unit or a link in a multiprocessor may appear faulty (defective) grows relatively high as the number of processor units increases [24]-[25].

A VLSI multiprocessor containing defective units (and links) can be made healthy as a whole subject to dedicated defect detection and isolation mechanism is employed [26]-[27]. If a certain redundancy, e.g., a set of spare units is introduced and specific methods are used to make it possible to detect and logically replace defect units with spare ones, then the VLSI MP may be treated as defect-free and retains its performance at the same time. In both cases, a multiprocessor with physical defects is logically reconfigured and VLSI MP fabrication yield loss is reduced as a result.

For successful VLSI MP logical reconfiguration [26], it is important that every faulty node is properly detected and isolated to let the rest of the multiprocessor operate [27], possibly, with slightly decreased performance [28]. This problem is typically solved based on the usage of self-checking or self-test methods [29]-[41]. Self-test technique allows detecting both manufacturing defects and local faults and is a suitable solution to provide fast fault/defect detection that does not require powering off and un-mounting the multiprocessor for repair. However, relatively low testability is the main problem of the

self-test approach, because test hardware itself is not 100% reliable; yet, self-checking algorithms may miss many faults/defects and sometimes treat healthy units as defective. To significantly alleviate the above problem, mutual inter-unit test can be employed, meaning that each multiprocessor unit is checked by some other units and the final faulty/non-faulty decision is made based on a certain formal rule, which takes into account the local decisions made by particular test units [39]-[41]. This approach is developed in the present paper.

The aim of the manuscript is to expand the VLSI multiprocessor mutual inter-unit test method initially presented in [39]-[41]. In the following sections, we formally state the mutual inter-unit test approach for the d-dimensional VLSI MP architecture, which makes it possible to concurrently detect faulty/defective units across a mesh-connected VLSI multiprocessor. A parallel inter-unit test algorithm is presented based on the proposed formal approach and dedicated test hardware implementing the above algorithm is diagrammed and briefly discussed. At the end of the paper, we demonstrate that the mutual inter-unit test environment provides increased testability compared to the self-checking technique.

2. THE MUTUAL INTER-UNIT TEST APPROACH

The key idea of the mutual inter-unit test is that each processor unit of the multiprocessor is periodically checked by a subset of its physical neighbors (so called "testing neighbors") and, at the same time, this processor unit tests another subset of its physical neighbors (so called "tested neighbors") and the final faulty/non-faulty decision for each processor unit is made based on the majority operator result obtained from the partial results returned by the testing neighbors.

The set of "testing neighbors" for each processor unit is formed depending on the number of dimensions (d) of the VLSI multiprocessor topology and should satisfy the odd cardinality requirement to make the majority operator applicable to produce the final test result. The same applies to the formation of the set of "tested neighbors", except that the cardinality of the set may not be odd. The process of mutual inter-unit test is carried out simultaneously in all the units across the mesh, so that a faulty signal is simultaneously transferred to the physical neighbors of the corresponding faulty processor unit, which makes it possible to efficiently isolate (or replace) faulty/defective units in a timely manner.

The mutual inter-unit test mechanism may be considered as an advanced form of self-checking, because the operability of the test hardware itself is also tested. For example, if one of the testing processor units produces a wrong faulty/non-faulty decision, then the tested unit (which is in fact non-faulty) will not be necessarily detected as faulty by mistake as the resulting faulty signal is formed by the majority operator applied to a set of partial fault detection signals. This means that the mutual inter-unit test mechanism testability is better compared to the self-test technique. A more formal demonstration is shown at the end of the paper.

3. THE FORMATION OF TESTING AND TESTED NEIGHBOR SETS

The formation of testing and tested neighbor sets is one of the key problems in the organization of mutual inter-unit test. In this section, we provide formal rules to define these sets for a VLSI MP of arbitrary dimension $d \geq 2$.

Let us first consider a 2-dimensional multiprocessor. Let $U = \{u_{xy}\}$ be the set of its processor units, where x and y are coordinates (indices) of a particular unit in the mesh in the horizontal and vertical dimensions, respectively, $x = \overline{0, n-1}$, $y = \overline{0, m-1}$, with m and n standing for the number of rows and columns of the mesh, respectively. Let C_{xy} and K_{xy} designate the sets of tested and testing neighbors of processor unit u_{xy}, respectively. Then, for given arbitrary $x \in \{0,1, \dots n-1\}$ and $y = \{0,1, \dots m-1\}$, we can formulate the following rules:

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

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

Equations (1) and (2) above take into account the fact that leftmost, rightmost, topmost and bottommost processor units have fewer physical neighbors than those located in the other parts of the mesh. For example, a topmost unit has no neighbor above, that's why its tested neighbor set should include the bottommost unit in the same column. The same applies to a leftmost unit that has no neighbor on its left; its testing neighbor set should include the rightmost unit in the same row. Figure 1 illustrates rules (1) and (2) in detail. Note that rules (1) and (2) give just one of several possible testing/tested neighbor allocations around each unit, providing the minimal number of neighbors involved.

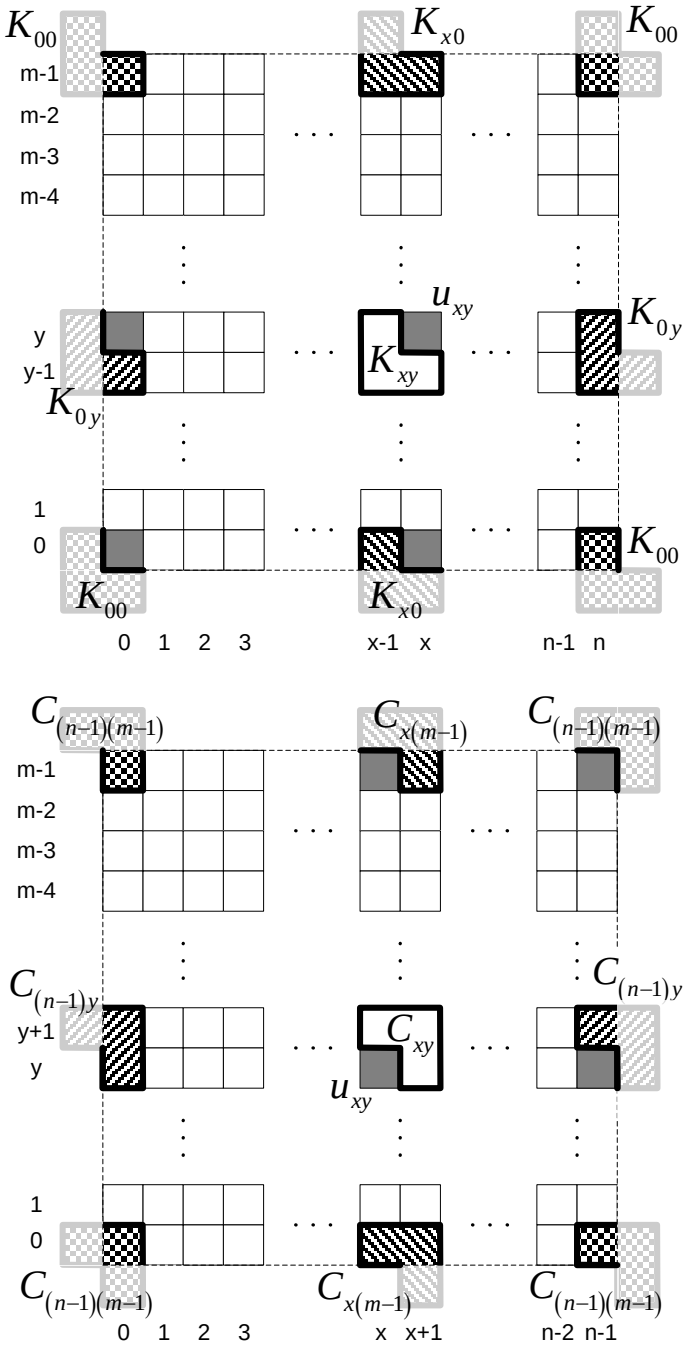

Figure 1. The formation of tested and testing neighbor sets in a 2-dimensional mesh multiprocessor.

If set K_{xy} is defined for each processor unit u_{xy} , then the faulty/non-faulty decision may be made according to the following rule:

$$\varphi_{xy} = \#\left(\begin{array}{c} \varphi_{xy}^{\left(x+(1-\mathrm{sign}(x))n-1\right),y}, \\ \varphi_{xy}^{\left(x+(1-\mathrm{sign}(x))n-1\right),\left(y+(1-\mathrm{sign}(y))m-1\right)}, \\ \varphi_{xy}^{x,\left(y+(1-\mathrm{sign}(y))m-1\right)} \end{array} \right) \tag{3}$$

where # denotes the majority operator, $\varphi_{xy}^{x'y'} = 1$, if unit u_{xy} 'is considered' non-faulty by unit $u_{x',y'}$ and $\varphi_{xy}^{x'y'} = 0$ otherwise, where x', y' are the placeholders standing for the corresponding upper indices in Equation (3). According to (3), unit u_{xy} is treated as faulty and needs to be isolated from the mesh, if $\varphi_{xy} = 0$.

The rules (1)-(3) may be easily extended to mesh topologies of higher dimensions. For example, for a 3-dimensional multiprocessor they could be formulated as follows:

$$C_{xyz} = \left\{ \begin{array}{c} u_{(x+1)\bmod n,y,z}, u_{x,(y+1)\bmod m,z}, \\ u_{x,y,(z+1)\bmod p}, u_{(x+1)\bmod n,(y+1)\bmod m,z}, \\ u_{(x+1)\bmod n,y,(z+1)\bmod p}, \\ u_{x,(y+1)\bmod m,(z+1)\bmod p}, \\ u_{(x+1)\bmod n,(y+1)\bmod m,(z+1)\bmod p} \end{array} \right\} \tag{4}$$

$$K_{xyz} = \left\{ \begin{array}{c} u_{\left(x+(1-\mathrm{sign}(x))n-1\right),y,z}, \\ u_{x,\left(y+(1-\mathrm{sign}(y))m-1\right),z}, \\ u_{x,y,\left(z+(1-\mathrm{sign}(z))p-1\right)}, \\ u_{\left(x+(1-\mathrm{sign}(x))n-1\right),\left(y+(1-\mathrm{sign}(y))m-1\right),z}, \\ u_{\left(x+(1-\mathrm{sign}(x))n-1\right),y,\left(z+(1-\mathrm{sign}(z))p-1\right)}, \\ u_{x,\left(y+(1-\mathrm{sign}(y))m-1\right),\left(z+(1-\mathrm{sign}(z))p-1\right)}, \\ u_{\left(x+(1-\mathrm{sign}(x))n-1\right),\left(y+(1-\mathrm{sign}(y))m-1\right),\left(z+(1-\mathrm{sign}(z))p-1\right)} \end{array} \right\} \tag{5}$$

$$\varphi_{xyz} = \#\left\{ \begin{array}{c} \varphi_{xy}^{\left(x+(1-\mathrm{sign}(x))n-1\right),y,z}, \\ \varphi_{xy}^{x,\left(y+(1-\mathrm{sign}(y))m-1\right),z}, \\ \varphi_{xy}^{x,y,\left(z+(1-\mathrm{sign}(z))p-1\right)}, \\ \varphi_{xy}^{\left(x+(1-\mathrm{sign}(x))n-1\right),\left(y+(1-\mathrm{sign}(y))m-1\right),z}, \\ \varphi_{xy}^{\left(x+(1-\mathrm{sign}(x))n-1\right),y,\left(z+(1-\mathrm{sign}(z))p-1\right)}, \\ \varphi_{xy}^{x,\left(y+(1-\mathrm{sign}(y))m-1\right),\left(z+(1-\mathrm{sign}(z))p-1\right)}, \\ \varphi_{xy}^{\left(x+(1-\mathrm{sign}(x))n-1\right),\left(y+(1-\mathrm{sign}(y))m-1\right),\left(z+(1-\mathrm{sign}(z))p-1\right)} \end{array} \right\} \tag{6}$$

where m, n and p are the sizes of the mesh in X, Y and Z dimensions, respectively. To define sets $C_{x_1x_2\ldots x_d}$ and $K_{x_1x_2\ldots x_d}$ for a general cased-dimension mesh, it is necessary to extend Equations (4)-(6) by adding extra properly indexed elements (u and φ) and all possible combinations. The d-dimensional case equations are not stated here for evident reasons. One can prove that:

$$\left|C_{x_1x_2\ldots x_d}\right| = \left|K_{x_1x_2\ldots x_d}\right| = d\left(d-1\right)+1 \tag{7}$$

Thus, $\left|K_{x_1x_2\ldots x_d}\right| = 1(mod2)$; i.e., each processor unit has an odd number of testing neighbors that makes it possible to apply the majority operator to produce the resulting faulty/non-faulty flag. According to (7), the number of testing neighbors in 2-dimensional meshes is minimal: $\left|K_{xy}\right| = 3$. In a 3-dimensional array, each unit has $\left|K_{xyz}\right| = 7$ testing neighbors.

4. THE MUTUAL INTER-UNIT TEST PROCEDURE

The process of mutual inter-unit test may be represented as a parallel algorithm including a set of threads $B_1, B_2, \ldots, B_{d(d-1)+1}$, where thread B_i defines a test statement sequence corresponding to tested neighbor $u_{x_1^i x_2^i \ldots x_d^i}$ (see Figure 2). The algorithm applies to a VLSI MP of any dimension $d \geq 2$.

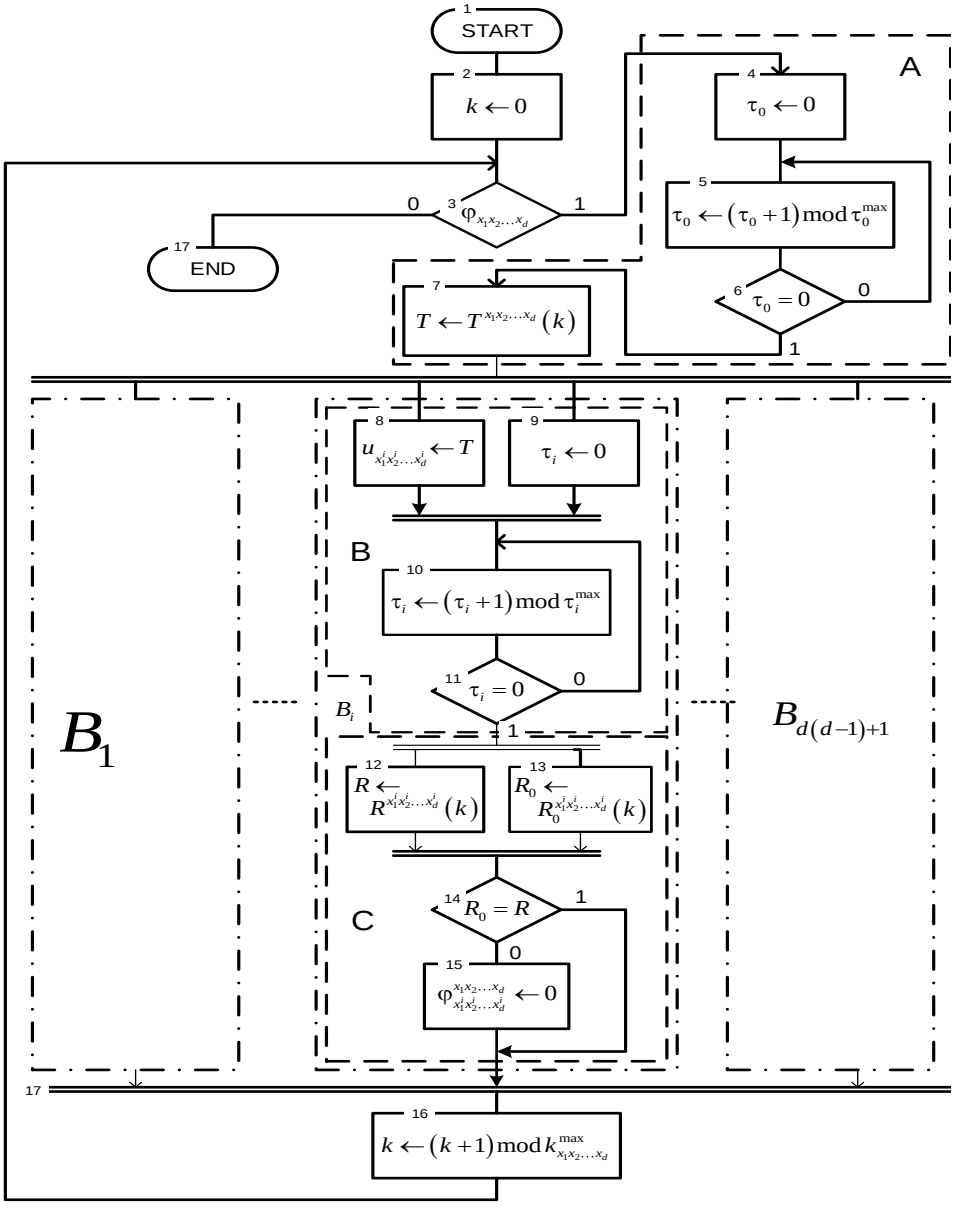

Figure 2. Flow-chart of the mutual inter-unit test algorithm.

The algorithm in Figure 2 includes the main test loop which is executed while the corresponding Processor unit (which is meant to be $u_{x_1x_2\ldots x_d}$) is considered healthy by its testing neighbors $K_{x_1x_2\ldots x_d}$ (see condition 3). As another loop begins, a new test signature $T^{x_1x_2\ldots x_d}(k)$ is formed (see statement 7), which is simultaneously transferred to tested neighbors $C_{x_1x_2\ldots x_d}=\left\{u_{x_1^i x_2^i\ldots x_d^i}\right\}$ (see statement 8). After all the tested neighbors have returned corresponding response signatures $\left\{R^{u_{x_1^i x_2^i\ldots x_d^i}}(k)\right\}$ (see statement 12), a decision is made by the processor $u_{x_1x_2\ldots x_d}$ whether a particular tested neighbor $u_{x_1^i x_2^i\ldots x_d^i}$ is faulty or healthy (see condition 14 and statement 15).

The algorithm in Figure 2 consists of three main sections: A, B and C (see the dash lines). Section A is necessary to spin τ_0^{max} clock ticks until the next test loop begins and another test signature $T^{x_1x_2\ldots x_d}(k)$ gets ready for transfer. Section B is responsible for transferring the test signature to tested unit $u_{x_1^i x_2^i\ldots x_d^i}$ and performs counting τ_i^{max} clock ticks until a response from the tested processor is supposed to arrive. Section C first controls the arrival of test response $R^{u_{x_1^i x_2^i\ldots x_d^i}}(k)$ from unit $u_{x_1^i x_2^i\ldots x_d^i}$ and then generates reference test response $R^{x_1x_2\ldots x_d}(k)$ to compare it to $R^{x_1^i x_2^i\ldots x_d^i}(k)$. If the above are equal, then tested neighbor $u_{x_1^i x_2^i\ldots x_d^i}$ is assumed to be healthy; otherwise, it is considered faulty and the partial faulty/non-faulty decision flag $\varphi_{x_1^i x_2^i\ldots x_d^i}^{x_1x_2\ldots x_d}$ is reset to zero. This flag is then used in Equations like (3) and (6) (depending on the value of d) to produce the final decision flag $\varphi_{x_1^i x_2^i\ldots x_d^i}$. The meaning of the symbols used in the flowchart of Figure 2 is presented in Table 1.

Table 1. The meaning of the symbols used in Figure 2.

Symbol	Meaning
$k_{x_1x_2\ldots x_d}^{\max}$	The number of test signatures supported by processor unit $u_{x_1x_2\ldots x_d}$
$k,\ k=\overline{0,k_{x_1x_2\ldots x_d}^{\max}-1}$	Test signature counter of processor unit $u_{x_1x_2\ldots x_d}$
$\tau_0^{\max}$	The interval (in clock ticks) between two adjacent test loops
τ_0	Next test loop wait counter
$\tau_i^{\max},\ i=\overline{1,d(d-1)+1}$	The maximum time needed to form a test response by tested processor unit $u_{x_1^i x_2^i\ldots x_d^i}$
$\tau_i,\ i=\overline{1,d(d-1)+1}$	Test response wait counter
$T^{x_1x_2\ldots x_d}(k)$	k^{th} test signature supported by processor unit $u_{x_1x_2\ldots x_d}$
$R^{x_1^i x_2^i\ldots x_d^i}(k)$	Test response signature issued by tested processor unit $u_{x_1^i x_2^i\ldots x_d^i}$ after $T^{x_1x_2\ldots x_d}(k)$ is received
$R_0^{x_1^i x_2^i\ldots x_d^i}(k)$	The reference test response signature expected to be issued by processor unit $u_{x_1^i x_2^i\ldots x_d^i}$ after receiving $T^{x_1x_2\ldots x_d}(k)$
T, R, R_0	Temporarily used variables
$\leftarrow$	The value assignment/transfer operator

In the algorithm diagrammed in Figure 2, much work is done in parallel, which makes it possible to concurrently test processor units across the entire multiprocessor structure. All the conditions and statements of the algorithm are simple enough to be implemented in hardware, which additionally contributes to the mutual inter-unit test environment performance. Yet, the test response signature mechanism used in the presented algorithm allows configuring test actions performed by tested neighbors taking into account the test complexity/duration trade-off.

5. THE MUTUAL INTER-UNIT TEST HARDWARE

The mutual inter-unit test algorithm of Figure 2 may be directly presented as a hardware-level implementation. The logic diagram of the embedded test hardware constructed according to the above algorithm is shown in Figure 3.

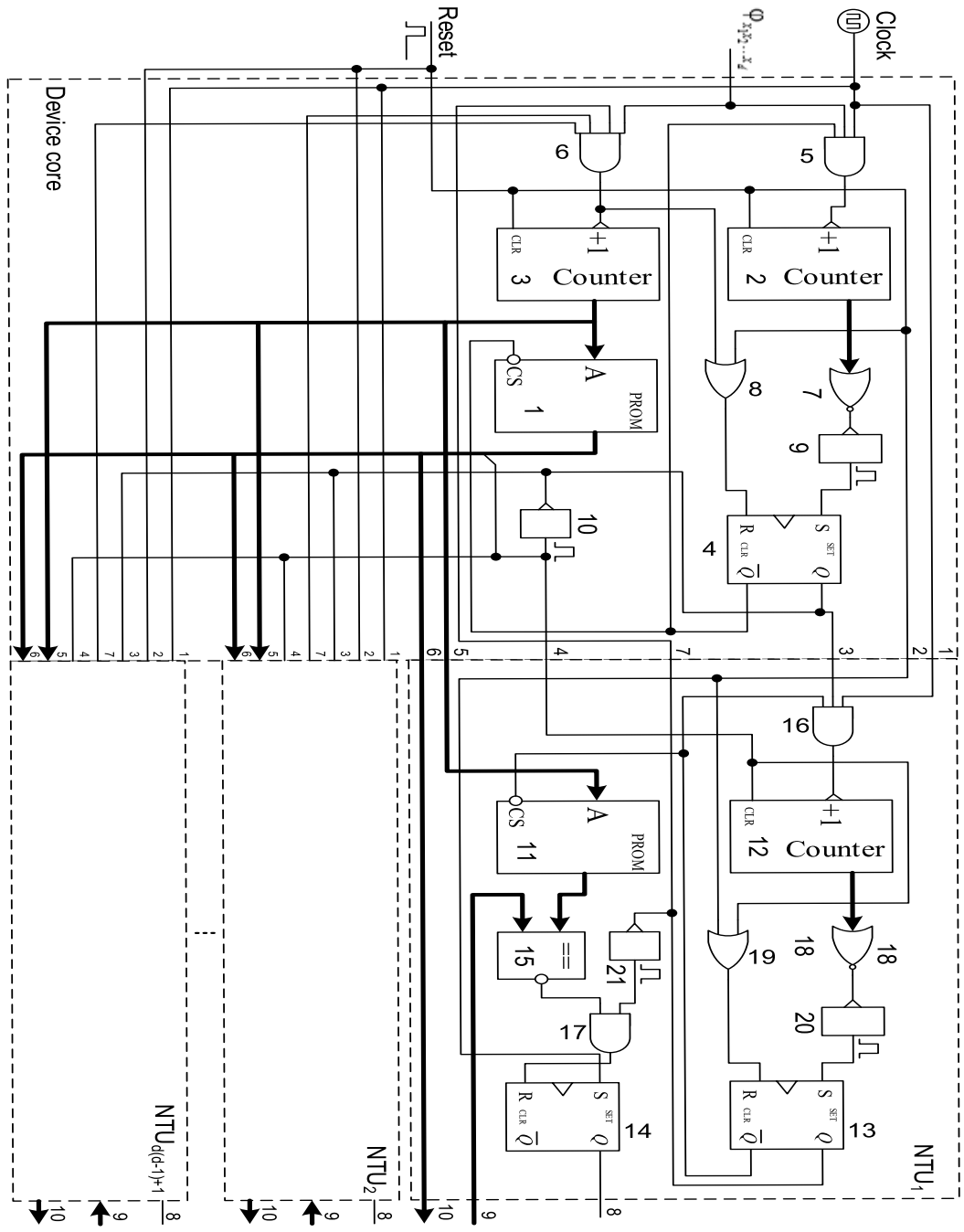

Figure 3. Logic diagram of the embedded test hardware implementing the mutual inter-unit test algorithm.

The device of Figure 3 is supposed to be a part of each processor unit; it consists of the device core and $d(d-1)+1$ neighbor test units (NTUs). The device core executes the initial and final sequential threads of the mutual inter-unit test algorithm, while NTUi implements thread $B_i, i = \overline{1, d(d-1)+1}$ (see Figure 2).Taking into account the fact that the NTUs are identical, only NTU1 is detailed in Figure 3. The adopted input/output numbering scheme helps understand the connections between the device core and the NTUs. The functions of the units and logic gates shown in Figure 3 are detailed in Table 2.

Table 2. The functions of the units and gates presented in Figure 3

Unit or gate	Function
Memory unit 1	Stores the test signatures issued by the processor unit
Circular binary counter 2	Counts the clock pulses that arrived between two adjacent test loops performed by the processor unit
Circular binary counter 3	Points to the next test signature in memory 1 to be issued by the processor unit
Flip-flop 4	Indicates whether counter 2 has zeroed or not
AND gate 5	Stops clock pulses from arriving at counter 2
AND gate 6	Stops clock pulses from arriving at counter 3
NORgate 7 together with univibrator9	Detect whether counter 2 has re-entered the zero state
OR gate 8	Necessary to OR the pulses clearing flip-flop 4
Univibrator 10	Produces a pulse which forces the NTUs to start operation
Memory unit 11	Stores the ref. response signatures for the tested neighbors of the current processor
Circular binary counter 12	Counts the clock pulses that arrived until the corresponding tested neighbor sends a response signature
Flip-flop 13	Indicates whether counter 12 has zeroed or not
Flip-flop 14	Indicates whether the corresponding tested neigh. is currently healthy or faulty
Comparator 15	Compares the test response sent by the tested neighbor to the corresponding reference test response read from memory 11
AND gate 16	Stops clock pulses from arriving at counter 12
AND gate 17	Stops reset pulses from arriving at counter 14
NOR gate 18 combined with univibrator 20	Detect whether counter 12 has re-entered the zero state
OR gate 19	Necessary to OR the pulses clearing flip-flop 13
Univibrator 21	Produces a pulse to clear flip-flop 14

6. Comparing the Mutual Inter-unit Test Approach to Self-checking

The mutual inter-unit test approach is a good alternative to the self-checking technique, providing better multiprocessor testability, which is demonstrated in the present section.

Let $\pi^+(t)$ be the probability that a processor unit of the multiprocessor is properly detected as faulty by a separate test unit (NTU). Taking into account that there are $C_j^i = \frac{j!}{i!(j-i)!}$ possible combinations of testing neighbors correctly reporting that the current processor is faulty, the following Equation may be deduced:

$$P^+(t) = \sum_{i=\left\lceil d(d-1)+1/2 \right\rceil}^{d(d-1)+1} C^i_{d(d-1)+1} \pi^+(t)^i \left[1-\pi^+(t)\right]^{d(d-1)-i+1} \quad (8)$$

Equation (8) gives the probability $P^+(t)$ that a processor unit is properly detected as faulty subject to the mutual inter-unit test approach employed.

To evaluate the effect provided by the mutual inter-unit test, we assume that $\pi^+(t)$ equals the probability of successful self-test (we presuppose that each processor has a built-in NTU or similar hardware to check its state) and then calculate $P^+(t)/\pi^+(t)$ depending on d and $\pi^+(t)$ with fixed $\pi^+(t)$ and d, respectively (see Figure 4).

(a) $P^+(t)/\pi^+(t)$ (b) $P^+(t)/\pi^+(t)$

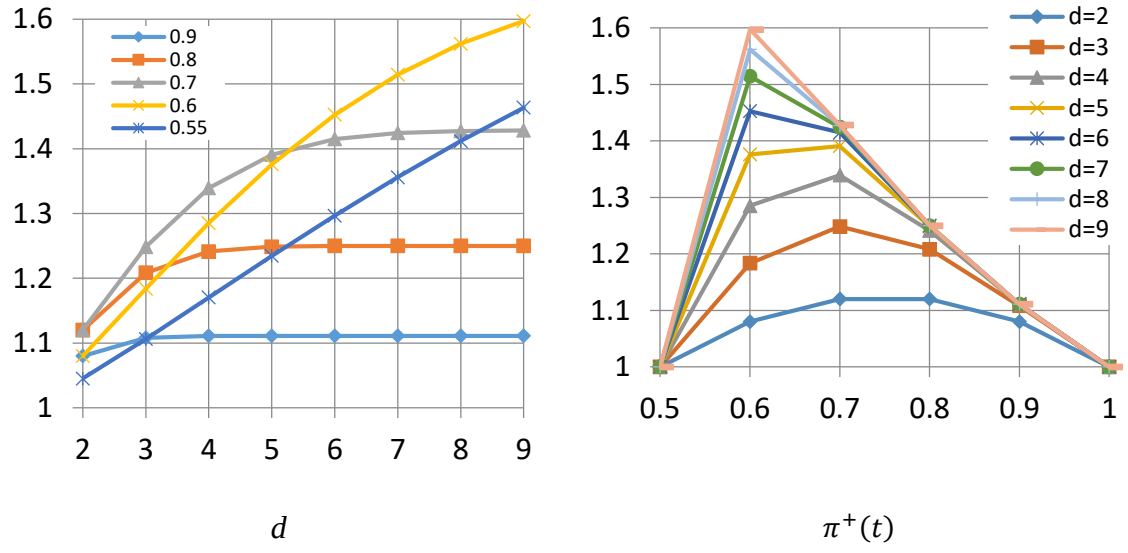

Figure 4. (a) $P^+(t)/\pi^+(t)$ versus d and (b) $P^+(t)/\pi^+(t)$ versus $\pi^+(t)$ graphs for fixed $\pi^+(t)$ and d, respectively.

The graphs in Figure 4 demonstrate that the mutual inter-unit test approach is effective as long as $\pi^+(t) \in [0.6; 0.9]$. If $\pi^+(t) \to 1$ or $\pi^+(t) \to 0.5$, then $P^+(t)/\pi^+(t) \to 1$, thus the effectiveness gracefully degrades. Our approach provides minimal effect for 2-dimensional multiprocessors (8% to 12% better than self-checking with $\pi^+(t) \in [0.6; 0.9]$). If more dimensions are added, then with $\pi^+(t) \in [0.6; 0.8]$, it is possible to get 20% or more effectiveness growth. Note that $\pi^+(t) = 0.6$ is approximately the point of maximum effectiveness as the number of dimensions increases.

7. The Area Cost of the Test Hardware

In our investigation, we have evaluated the area cost added by the test hardware. We assumed different VLSI MP dimensions d and calculated the area occupied by the logic gates in the test unit hardware. We fixed the number of test instructions supported by each processor unit k_{max}.

The graphs shown in Figure 5 demonstrate the VLSI area penalty *versus* VLSI MP dimension *d*. For low-dimension multiprocessors, the extra area remains as low as *O*(10000) logic gates, which is much lower than the area occupied by the processor core. For example, in a 3-dimensional multiprocessor, less than 18000 extra gates need to be added to each processor unit to support up to 32 test routines at each unit. If we add more test scenarios, then the VLSI area penalty will increase drastically. For example, a 3-dimensional VLSI MP needs over 260000 extra logic gates at each unit to support up to 512 different test routines.

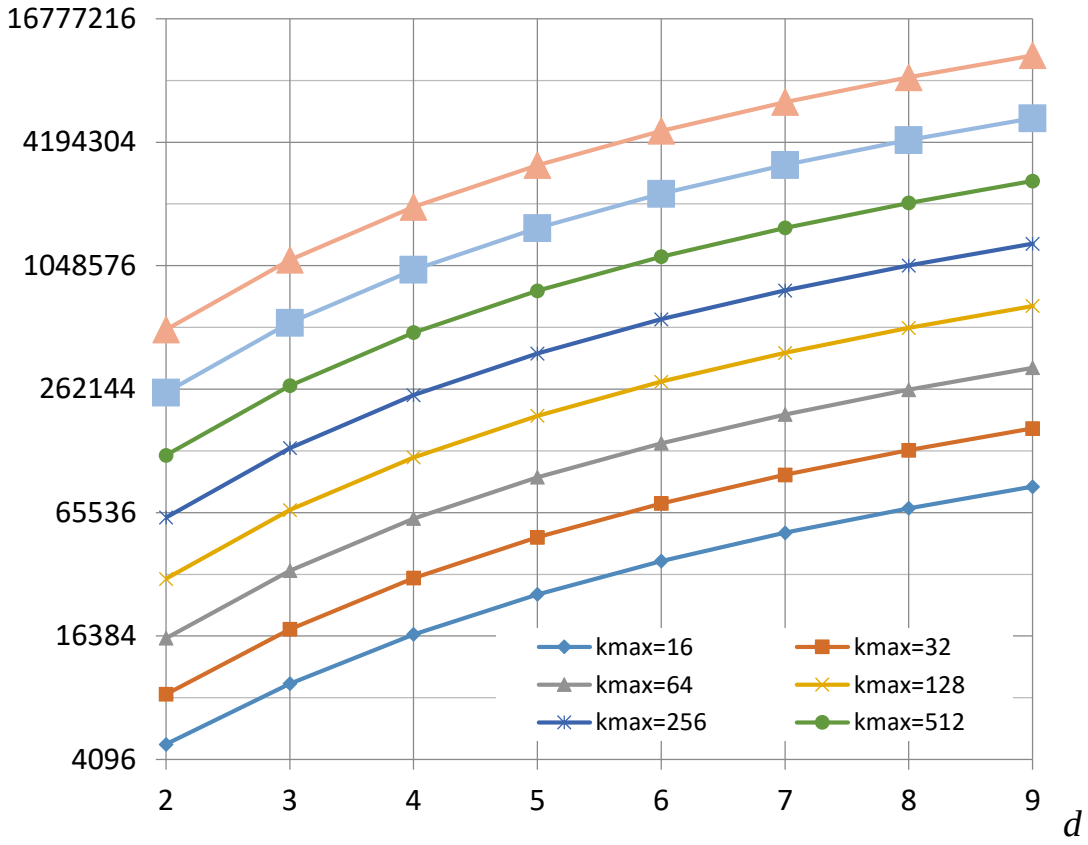

Figure 5. VLSI area (*C*) *versus d* graphs for a fixed number of test instructions supported by a particular unit (k_{max}).

As a consequence, a limited number of test instructions supported are a good option to decrease the VLSI area penalty of the test hardware. Note that each test instruction provides a specific test procedure, thus the selection of a set of test routines becomes a key issue to cover up possible fault patterns.

8. Conclusion

In the present paper, we have presented a new approach, the mutual inter-unit test mechanism, which makes it possible to improve testability of mesh-connected VLSI multiprocessors by increasing the successful fault detection probability with respect to traditional self-checking. We have shown that our approach is applicable to multiprocessors of arbitrary dimension; yet, its effectiveness grows higher as the number of dimensions increases which is important for future generation VLSI MPs. The mutual inter-unit test technique allows hardware-level testing of all the processor units across the mesh in parallel, thus significantly contributing to the test environment performance.

References

[1] Jie-qiong Chen and Guo-qiang Mao, "Capacity of Cooperative Vehicular Networks with Infrastructure Support: Multi-user Case," IEEE Trans. on Vehicular Technology, vol. 67, no. 2, pp. 1546-1560, 2018.

[2] X. D. Song and X. Wang, "Extended AODV Routing Method Based on Distributed Minimum Transmission (DMT) for WSN," Int. Jou. of Electronics and Comm., vol. 69, no. 1, pp. 371-381, 2015.

[3] S. Zhou, J. Chen and S. Liu, "New Mixed Adaptive Detection Algorithm for Moving Target with Big Data," Journal of Vibroengineering, vol. 18, no. 7, pp. 4705-4719, 2016.

[4] H. L. Niu and S. Liu, "Novel Positioning Service Computing Method for WSN," Wireless Personal Communications Journal, vol. 92, no. 4, pp. 1747-1769, 2017.

[5] Z. Ma, "A Novel Compressive Sensing Method Based on SVD Sparse Random Measurement Matrix in Wireless Sensor Network," Engineering Computations, vol. 33, no. 8, pp. 2448-2462, 2016.

[6] D. Zhang, S. Zhou and Ya-meng Tang, "A Low Duty Cycle Efficient MAC Protocol Based on Self-adaption and Predictive Strategy," Mobile Networks & Applications Jour., DOI: 10.1007/s11036-017-0878-x, 2017.

[7] S. Liu and T. Zhang, "Novel Unequal Clustering Routing Protocol Considering Energy Balancing Based on Network Partition & Distance for Mobile Education," Journal of Network and Computer Applications, vol. 88, no. 15, pp. 1-9, DOI: 10.1016/j.jnca.2017.03.025, 2017.

[8] K. Zheng and D. Zhao, "Novel Quick Start (QS) Method for Optimization of TCP," Wireless Networks, vol. 22, no. 1, pp. 211-222, 2016.

[9] D. Zhang, X. J. Kang and J. Wang, "A Novel Image De-noising Method Based on Spherical Coordinates System," EURASIP Journal on Advances in Signal Processing, no. 110, pp. 1-10, DOI: 10.1186/1687-6180-2012-110, 2012.

[10] X. Wang and X. Song, "New Clustering Routing Method Based on PECE for WSN," EURASIP Journal on Wireless Communications and Networking, no. 162, pp. 1-13, DOI: 10.1186/s13638-015-0399-x, 2015.

[11] X. Song and X. Wang, "New Agent-based Proactive Migration Method and System for Big Data Environment (BDE)," Engineering Computations, vol. 32, no. 8, pp. 2443-2466, 2015.

[12] H. L. Niu and S. Liu, "Novel PEECR-based Clustering Routing Approach," Soft Computing, vol. 21, no. 24, pp. 7313-7323, 2017.

[13] Y. Liang, "A Kind of Novel Method of Service-aware Computing for Uncertain Mobile Applications," Mathematical and Computer Modelling, vol. 57, no. 3-4, pp. 344-356, 2013.

[14] C. P. Zhao, "A New Medium Access Control Protocol Based on Perceived Data Reliability and Spatial Correlation in Wireless Sensor Network," Comp. & Elect. Engineering, vol. 38, no. 3, pp. 694-702, 2012.

[15] W. B. Li, "Novel Fusion Computing Method for Bio-Medical Image of WSN Based on Spherical Coordinate," Journal of Vibroengineering, vol. 18, no. 1, pp. 522-538, 2016.

[16] Z. Ma, "Shadow Detection of Moving Objects Based on Multisource Information in Internet of Things," Journal of Experimental & Theoretical Artificial Intelligence, vol. 29, no. 3, pp. 649-661, 2017.

[17] Z. Ma, "A Novel Compressive Sensing Method Based on SVD Sparse Random Measurement Matrix in Wireless Sensor Network," Engineering Computations, vol. 33, no. 8, pp. 2448-2462, 2016.

[18] S. Liu and T. Zhang, "Novel Unequal Clustering Routing Protocol Considering Energy Balancing Based on Network Partition & Distance for Mobile Education," Journal of Network and Computer Applications, vol. 88, no. 15, pp. 1-9, DOI: 10.1016/j.jnca.2017.03.025, 2017.

[19] G. Li and K. Zheng, "An Energy-balanced Routing Method Based on Forward-aware Factor for Wireless Sensor Network," IEEE Transactions on Industrial Informatics, vol. 10, no. 1, pp. 766-773, 2014.

[20] H. L. Niu and S. Liu, "Novel PEECR-based Clustering Routing Approach," Soft Computing, vol. 21, no. 24, pp. 7313-7323, DOI: 10.1007/s00500-016-2270-3, 2017.

[21] H. L. Niu and S. Liu, "Novel Positioning Service Computing Method for WSN," Wireless Personal Communications, vol. 92, no. 4, pp. 1747-1769, DOI: 10.1007/s11277-016-3632-y, 2017.

[22] S. Zhou, J. Chen and S. Liu, "New Mixed Adaptive Detection Algorithm for Moving Target with Big Data," Journal of Vibroengineering, vol. 18, no. 7, pp. 4705-4719, 2016.

[23] Th. Rauber and G. Runger, Parallel Programming for Multicore and Cluster Systems, Springer 2013, XIII, 516 p.

[24] Z. Wang, VLSI, InTech, 464 p, 2010.

[25] S. Furber, "Living with Failure: Lessons from Nature?," Proc. of the 11th IEEE European Test Symposium (ETS '06), pp. 4-8, May 2006.

[26] E. Kolonis, M. Nicolaidis, D. Gizopoulos, M. Psarakis, J. Collet and P. Zajac, "Enhanced Self-configurability and Yield in Multicore Grids," Proc. of the 15th IEEE Int. On-line Testing Symposium (IOLTS), pp. 75-80, Jun. 2009.

[27] G. Jiang, W. Jigang and J. Sun, "Efficient Reconfiguration Algorithm for Three-dimensional VLSI Arrays," Proc. of the 26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, pp. 261-265, 2012.

[28] W. Jigang, T. Srikanthan, G. Jiang and K. Wang, "Constructing Sub-arrays with Short Interconnects from Degradable VLSI Arrays," IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 4, pp. 929-938, April 2014.

[29] S. M. A. H. Jafri, S. J. 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.

[30] P. Bernardi, L. M .Ciganda, E. Sanchez and M. Sonza Reorda, "MIHST: A Hardware Technique for Embedded Microprocessor Functional On-line Self-Test," IEEE Transactions on Computers, vol. 63, no. 11, pp. 2760-2771, Nov. 2014.

[31] S. R. Das, "Self-testing of Core-based Embedded Systems with Built-in Hardware," IEEE Proceedings: Circuits, Devices and Systems, vol. 152, no. 5, pp. 539-546, 7 Oct. 2005.

[32] 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 2-D Mesh-Based Chip Multiprocessor Systems," Proc. of the IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT '09), pp. 72–75, Apr. 2009.

[33] C. Stroud, J. Sunwoo, S. Garimella and J. Harris, "Built-in Self-test for System-on-Chip: A Case Study," Proc. of the IEEE International Test Conf. (ITC), pp. 837-846, 2004.

[34] 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, June 2014.

[35] R. Ahlswede and H. Aydinian, "On Diagnosability of Large Multiprocessor Networks," Discrete Applied Mathematics, vol. 156, no. 18, pp. 3464-3474, Nov. 2008.

[36] 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," Proc. of the 10th IEEE/ACM International Symposium on Networks-on-Chip (NOCS 2016), pp.1-8, Aug. 31-Sep. 2, 2016.

[37] 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. 13, no. 9, pp. 679 - 692, Sep. 2014.

[38] E. Cota, F. Kastensmidt, M. Cassel and M. Herve, "A High-fault Coverage Approach for the Test of Data, Control and Handshake Interconnects in Mesh Networks-on-Chip," IEEE Transactions on Computers, vol. 57, no. 9, pp. 1202-1215, 2008.

[39] J. S. Al-Azzeh, M. E. Leonov, D. E. Skopin, E. A. Titenko and I. V. 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.

[40] J. S. Al-Azzeh, “Fault-tolerant routing in mesh-connected multicomputers based on majority-operator-produced transfer direction identifiers”, Jordan Journal of Electrical Engineering, vol.3, no. 2, pp. 102-111, 2017.

[41] J. S. Al-Azzeh, “A Distributed Multiplexed Mutual Inter-Unit in-Operation Test Method for Mesh-Connected VLSI Multiprocessors”, Jordan Journal of Electrical Engineering, vol.3, no. 3, pp. 193-207, 2017.

ملخص البحث:

يتناول هذا البحث مسألة كشف الأخطاء على مستوى المعدات في اثناء التشغيل في المعالجات المتعددة ذات التكامل على نطاقٍ واسعٍ جداً المتصلة على شكل شبكة. فقد تم تقديم طريقة جديدة لاختبار المعالجات المتعددة مبنية على الفحص المتبادل بين الوحدات؛ الأمر الذي يسمح بزيادة احتمالية الكشف الناجح عن الأخطاء. كما تم تحديد قواعد لتشكيل مجموعات فاحصة وأخرى مفحوصة من "الجيران" لكل وحدة من الوحدات، تكون ثابتة بالنسبة لموقع الوحدة ضمن البنية التركيبية للمعالج المتعدد وعدد أبعاده. وتتشكل النتيجة النهائية للاختبار لكل وحدة معالج عن طريق تطبيق عامل الأغلبية على كل واحدة من البطاقات التي تفيد بوجود خطأ أو عدم وجود خطأ والتي تم حسابها من جانب جميع الوحدات المجاورة الفاحصة.

من جهة أخرى، يعرض البحث المعادلات الخاصة بتحديد عدد الوحدات المجاورة الفاحصة لكل وحدة معالج اعتماداً على عدد أبعاد المعالج المتعدد. كذلك تم تقييم احتمالية الكشف الناجح عن الأخطاء على موثوقية وحدات الفحص منفردة. وقد أظهرت الطريقة المقترحة أنها توفر المزيد من احتمالية الكشف الناجح عن الأخطاء إذا قورنت بالاختبار الذاتي لجميع الحالات ذات الأهمية العملية.