Journal of Engineering and Applied Sciences 12 (Special Issue 5): 7112-7118, 2017
ISSN: 1816-949X

A Distributed Majority-Operator-Based Built-In Mutual Inter-Node Test Method for Mesh-Connected VLSI Multiprocessors

[1]Jamil Al-Azzeh, [2]Evgeny A. Titenko and [2]Igor V. Zotov
[1]Department of Computer Engineering, Al-Balqa' Applied University, Salt, Jordan
[2]South West State University, St. 50 years of October 94, 305040 Kursk, Russia

Abstract: The problem of online processing node fault detection in mesh-connected multicore and many-core VLSI multiprocessors is considered. A novel hardware-level approach to the multiprocessor test based on mutual inter-processor checking is presented which presupposes that a coordinated healthy/faulty decision is made for each processor core by applying the majority operator to the individual healthy/faulty tags calculated by the corresponding set of testing neighbors. Formal rules are defined for forming sets of testing and tested neighbors for each processor node of the mesh which are invariant to the location of the node within the mesh and to its dimension. The formulae to determine the number of testing neighbors for each node depending on the dimension of the mesh are given. A parallel hardware-level algorithm implementing the proposed test method is presented and its possible hardware implementation is discussed. The successful fault detection probability is evaluated in the case when the proposed approach is used, its dependencies on the individual test node reliability are investigated. The proposed approach is shown to provide increased successful fault detection probability compared to the traditional self-checking and neighbor-checking for all practically significant cases.

Key words: Mesh-connected VLSI multiprocessors, fault tolerance, testability, built-in self-test, majority operator, processor

INTRODUCTION

VLSI multiprocessors multicore and many-core (VLSI MPs) are a highly efficient solution for implementing high performance embedded systems, combining fine-grain concurrency with decentralized and logically distributed architecture (Anonymous, 2015a, b; Vangal *et al*., 2008). Increasing complexity of VLSI MPs becomes a problem because the probability that, a processor node or a link in a multiprocessor may appear faulty (defective) grows relatively high as the number of processor nodes increases. In spite that, continuous advances in manufacturing technologies have reduced the defect densities, a relatively low VLSI MP fabrication yield is still an issue (Ciciani, 1998; Kolonis *et al*., 2009).

A VLSI multiprocessor containing defective nodes (and links) can be made healthy as a whole subject to a dedicated defect detection and isolation mechanism is employed. Despite that, the overall performance of the multiprocessor degrades, it may be considered defect-free (Jigang and Srikanthan, 2003; Fukushi and Horiguchi, 2004). If a certain redundancy, e.g., a set of spare nodes is introduced and specific methods are used to make it possible to detect and logically replace defect nodes with spare ones (Takanami, 2001; Roychowdhury *et al*., 1990; Lin *et al*., 2009) 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, it is important that every faulty node is properly detected and isolated to let the rest of the multiprocessor operate, possibly with slightly decreased performance. This problem is typically solved based on the usage of self-checking and neighbor-checking methods, both hardware and software-level (Jafri *et al*., 2014; Rajski *et al*., 2004; Aguilera *et al*., 2000; Nicolaidis and Anghel, 1999; Zhang *et al*., 2014; Bernardi *et al*., 2014; Psarakis *et al*., 2010; Krstic *et al*., 2002; Stroud *et al*., 2004; Raik and Govind, 2012). These techniques allow detecting both manufacturing defects and local faults and they are a suitable solution to provide online core/link fault/defect detection. However, relatively low testability is the main problem of the above approaches because no

Corresponding Author: Jamil Al-Azzeh, Department of Computer Engineering, Al-Balqa' Applied University, Salt, Jordan

coordination between test units of different processor nodes is carried out to pinpoint faulty cores; yet, checking algorithms may miss some faults/defects and sometimes they treat healthy nodes as defective. To alleviate the above problem the mutual inter-node coordinated test can be employed, meaning that each multiprocessor node is checked by some other nodes and the final healthy/faulty decision is made based on a certain formal cooperation rule which takes into account the local decisions made by particular test nodes (Al-Azzeh *et al.*, 2015). This approach is developed in the present study.

The aim of the manuscript is to expand the VLSI multiprocessor mutual inter-node test method initially presented by the researchers in (Al-Azzeh *et al.*, 2015). In the study, we formally state the mutual inter-node test approach for the d-dimensional VLSI MP architecture which makes it possible to concurrently detect faulty/defective nodes across a mesh-connected VLSI multiprocessor. A parallel inter-node 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 study, we demonstrate that the mutual inter-node test environment provides increased testability compared to the self-checking and neighbor-checking techniques.

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

The set of "testing neighbors" for each processor node is formed depending on the number of dimensions (d) of the VLSI multiprocessor topology and it should satisfy the odd cardinality requirement to make the majority operator applicable to produce the final healthy/faulty decision. 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-node test is carried out simultaneously in all the processor cores across the mesh so that, a faulty signal is simultaneously transferred to the physical neighbors of the corresponding faulty processor node which makes it possible to efficiently isolate (or replace) faulty/defective nodes in a timely manner.

The mutual inter-node test mechanism may be considered as an advanced form of neighbor-checking because the operability of the test hardware itself is implicitly tested. For example, if one of the testing processor nodes produces a wrong healthy/faulty decision, then the tested node (which is in fact healthy) 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 individual fault detection signals. This means that the mutual inter-node test mechanism's testability is better compared to the traditional self-checking and neighbor-checking techniques. A more formal demonstration is shown at the end of the study.

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-node test. In this study, we provide formal rules to define these sets for a VLSI MP of arbitrary dimension d≥2.

Let us first consider a 2-dimensional multiprocessor. Let U = $\{u_{xy}\}$ be the set of its processor nodes (cores) where x and y are coordinates (indexes) of a particular node in the mesh in the horizontal and vertical dimensions, respectively, $x=\overline{0,n\text{-}1}$, $y=\overline{0,m\text{-}1}$ 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 node u_{xy}, respectively. Then for given arbitrary x∈ {0, 1, ..., n-1} and y = {0, 1, ..., m-1}, we can formulate the following rules (Eq. 1 and 2):

$$C_{xy}=\left\{u_{(x+1)\bmod n,\,y},u_{(x+1)\bmod n,\,(y+1)\bmod m},u_{x,\,(y+1)\bmod m}\right\}\qquad(1)$$

$$K_{xy}=\left\{\begin{array}{l}u_{x(y+(1-\mathrm{sign}(y))m-1)},\;u_{(x+(1-\mathrm{sign}(x))n-1),(y+(1-\mathrm{sign}(y))m-1)},\\ u_{(x+(1-\mathrm{sign}(x))n-1),\,y}\end{array}\right\}\qquad(2)$$

Equation 1 and 2 take into account the fact that leftmost, rightmost, topmost and bottommost processor nodes have fewer physical neighbors than those located in the other parts of the mesh. For example, a topmost node has no neighbor above, that's why, its tested neighbor set should include the bottommost node in the same column. The same applies to a leftmost node that has no neighbor on its left; its testing neighbor set should include the rightmost node in the same row. Figure 1 illustrates the rule 1 and 2 in detail.

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

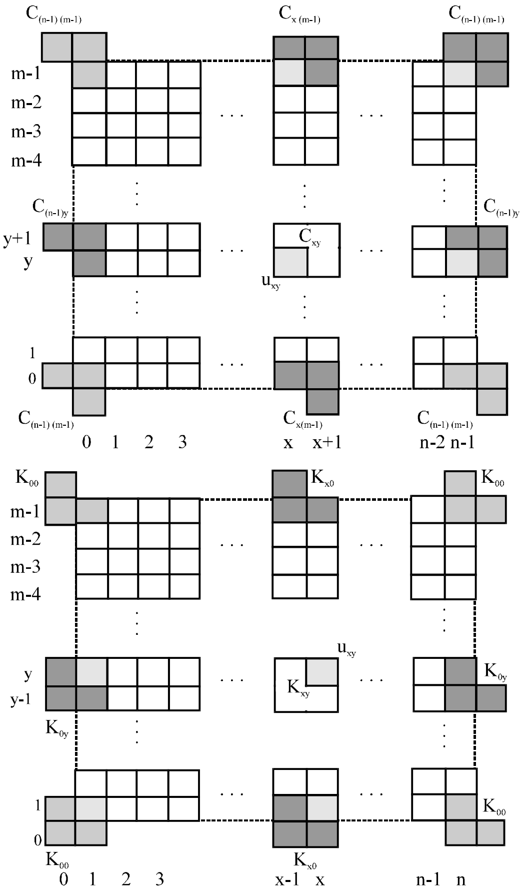

Fig. 1: The formation of tested and testing neighbor sets in a 2-dimensional mesh multiprocessor

$$\varphi_{xy} = \#\begin{pmatrix} \varphi_{xy}^{(x+(1-\mathrm{sign}(x))n-1),y}, \\ \varphi_{xy}^{(x+(1-\mathrm{sign}(x))n-1),(y+(1-\mathrm{sign}(y))m-1)}, \varphi_{xy}^{x(y+(1-\mathrm{sign}(y))m-1)} \end{pmatrix} \quad (3)$$

where # denotes the majority operator, $\varphi_{xy}{}^{x'y'} = 1$, if node u_{xy} "is considered" healthy by node $u_{x'y'}$ and $\varphi_{xy}^{x'y'} = 0$, otherwise where x', y' are the placeholders standing for the corresponding upper indices in Eq. 3. According to Eq. 3, node 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:

$$C_{xyz} = \begin{Bmatrix} u_{(x+1)\mathrm{mod}\,n,y,z}, u_{x(y+1)\mathrm{mod}\,m,z}, u_{x,y(z+1)\mathrm{mod}\,p}, \\ u_{(x+1)\mathrm{mod}\,n(y+1)\mathrm{mod}\,m,z}, \\ u_{(x+1)\mathrm{mod}\,n,y(z+1)\mathrm{mod}\,p}, u_{x(y+1)\mathrm{mod}\,m(z+1)\mathrm{mod}\,p}, \\ u_{(x+1)\mathrm{mod}\,n(y+1)\mathrm{mod}\,m(z+1)\mathrm{mod}\,p} \end{Bmatrix} \quad (4)$$

$$K_{xyz} = \begin{Bmatrix} u_{(x+(1-\mathrm{sign}(x)n-1),y,z)}, u_{x(y+(1-\mathrm{sign}(y))m-1),z}, \\ u_{x,y(z+(1-\mathrm{sign}(z))p-1)}, \\ u_{(x+(1-\mathrm{sign}(x))n-1)(y+(1-\mathrm{sign}(y))m-1),z}, \\ u_{(x+(1-\mathrm{sign}(x))n-1),y(z+(1-\mathrm{sign}(z))p-1)}, \\ u_{x(y+(1-\mathrm{sign}(y))m-1)(z+(1-\mathrm{sign}(z))p-1)}, \\ u_{(x+(1-\mathrm{sign}(x))n-1)(y+(1-\mathrm{sign}(y))m-1)(z+(1-\mathrm{sign}(z))p-1)} \end{Bmatrix} \quad (5)$$

$$\varphi_{xyz} = \#\begin{Bmatrix} \varphi_{xy}^{(x+(1-\mathrm{sign}(x))n-1),y,z} \varphi_{xy}^{x(y+(1-\mathrm{sign}(y))m-1),z} \\ \varphi_{xy}^{x,y(z+(1-\mathrm{sign}(z))p-1)} \\ \varphi_{xy}^{(x+(1-\mathrm{sign}(x))n-1)(y+(1-\mathrm{sign}(y))m-1),z} \\ \varphi_{xy}^{(x+(1-\mathrm{sign}(x))n-1),y(z+(1-\mathrm{sign}(z))p-1)} \\ \varphi_{xy}^{x(y+(1-\mathrm{sign}(y))m-1)(z+(1-\mathrm{sign}(z))p-1)} \\ \varphi_{xy}^{(x+(1-\mathrm{sign}(x))n-1)(y+(1-\mathrm{sign}(y))m-1)(z+(1-\mathrm{sign}(z))p-1)} \end{Bmatrix} \quad (6)$$

where m, n and p are the sizes of the mesh in the X, Y and Z dimensions, respectively. To define sets $C_{x_1,x_2,\ldots,x_d}$ and $K_{x_1,x_2,\ldots,x_d}$ for a general case d-dimension mesh, it is necessary to extend Eq. 4-6 by adding extra properly indexed elements (u and φ) in all possible combinations. The d-dimensional case formulae are not stated here for complexity reasons. One can prove that:

$$\left|C_{x_1,x_2,\ldots,x_d}\right| = \left|K_{x_1,x_2,\ldots,x_d}\right| = d(d\text{-}1)+1 \quad (7)$$

Thus, $\left|K_{x_1,x_2,\ldots,x_d}\right| = 1(\mathrm{mod}\,2)$, i.e., each processor node has an odd number of testing neighbors that makes it possible to apply the majority operator to produce the resulting healthy/faulty flag. According to Eq. 7, the number of testing neighbors in 2-dimensional meshes is minimal: $|K_{xy}| = 3$. In a 3-dimensional array, each node has $|K_{xyz}| = 7$ testing neighbors.

MATERIALS AND METHODS

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

The algorithm in Fig. 2 includes the main test loop which executes while the corresponding processor node (which is meant to be $u_{x_1,x_2,\ldots,x_d}$) is considered healthy by its testing neighbors $K_{x_1,x_2,\ldots,x_d}$ (condition 3). As another

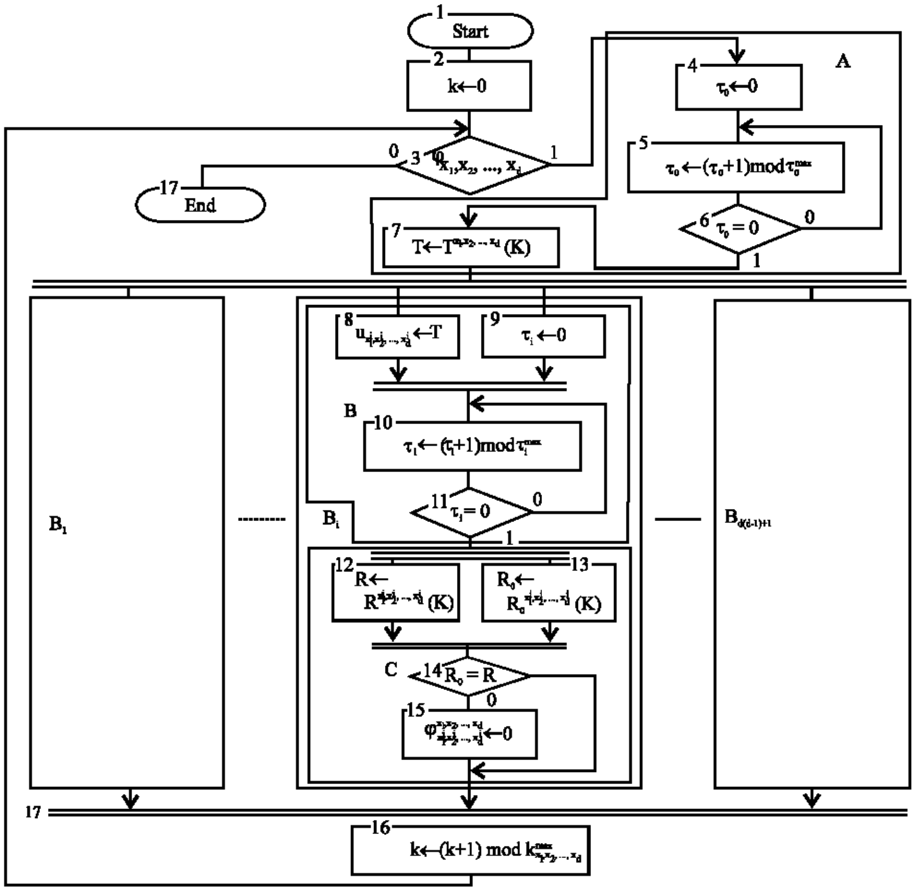
1 Start
2 k←0
3 $\varphi_{x_1,x_2,...,x_d}$
0
1
17 End
A
4 τ_0←0
5 τ_0←$(\tau_0+1)\bmod\tau_0^{max}$
6 $\tau_0=0$
0
1
7 T←$T^{x_1,x_2,...,x_d}$(K)
8 $u_{x_1^i,x_2^i,...,x_d^i}$←T
9 τ_i←0
B
10 τ_i←$(\tau_i+1)\bmod\tau_i^{max}$
11 $\tau_i=0$
0
1
B_1
B_i
$B_{d(d-1)+1}$
12 R← $R^{x_1^i,x_2^i,...,x_d^i}$(K)
13 R_0← $R_0^{x_1^i,x_2^i,...,x_d^i}$(K)
C
14 $R_0=R$
0
15 $\varphi^{x_1,x_2,...,x_d}_{x_1^i,x_2^i,...,x_d^i}$←0
17
16 k←(k+1) mod $k^{max}_{x_1,x_2,...,x_d}$

Fig. 2: Flow-chart of the mutual inter-node test algorithm

loop begins, a new test signature $T^{x_1,x_2,..,x_d}(k)$ is formed (statement 7) which is simultaneously transferred to tested neighbors $C_{x_1,x_2,..,x_d}=\{u_{x_1^i,x_2^i,..,x_d^i}\}$ (statement 8). After all the tested neighbors have returned corresponding response signatures $\{R^{x_1^i,x_2^i,..,x_d^i}(k)\}$ (statement 12) a decision is made by the processor $u_{x1,x2,..,xd}$ whether a particular tested neighbor $u_{x_1^i,x_2^i,..,x_d^i}$ is faulty or healthy (condition 14 and statement 15).

The algorithm in Fig. 2 consists of 3 main sections, A-C (the dash lines). Section A is necessary to spin $\tau_i{}^{max}{}_i$ clock ticks until the next test loop begins and another test signature $T^{x_1,x_2,..,x_d}(k)$ gets ready for transfer. Section B is responsible for transferring the test signature to tested node $u_{x_1^i,x_2^i,..,x_d^i}$ 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 $R^{x_1^i,x_2^i,..,x_d^i}(k)$ from node $u_{x_1^i,x_2^i,..,x_d^i}$ and then generates reference test response $R^{x_1,x_2,..,x_d}(k)$ to comp are it to $R^{x_1^i,x_2^i,..,x_d^i}(k)$ If the above are equal, then tested neighbor $u_{x_1^i,x_2^i,..,x_d^i}$ is assumed to be healthy, otherwise it is considered faulty and the partial faulty/non-faulty decision flag $\varphi^{x_1,x_2,..,x_d}_{x_1^i,x_2^i,..,x_d^i}$ is reset to zero. This flag is then used in Eq. 3-6 (depending on the value of d) to produce the final decision flag $\varphi_{x_1^i,x_2^i,..,x_d^i}$.

The meaning of the symbols used in the flow-chart of Fig. 2 is presented in Table 1. In the algorithm diagrammed in Fig. 2, much work is done in parallel which makes 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 mutual inter-node 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 tradeoff.

The mutual inter-node test hardware: The mutual inter-node test algorithm of Fig. 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 Fig. 3.

Table 1: The meaning of the symbols used in Fig. 2

Symbols	Meanings
$k^{max}_{x_1,x_2,..,x_d}$	The number of test signatures supported by processor node $u_{x_1,x_2,...,x_d}$
$k, k = \overline{0, k^{max}_{x_1,x_2,...,x_d} -1}$	Test signature counter of processor node $u_{x_1,x_2,...,x_d}$
$\tau^{max}{}_0$	The interval (in clock ticks) between two adjacent test loops
τ_0	Next test loop wait counter
$\tau^{max}_i, i = \overline{1, d(d\text{-}1)+1}$	The maximum time needed to form a test response by tested processor node $u_{x^i_1,x^i_2,...,x^i_d}$
$\tau_i, i = \overline{1, d(d\text{-}1)+1}$	Test response wait counter
$T^{x_1,x_2,...,x_d}(k)$	kth test signature supported by processor node $u_{x_1,x_2,...,x_d}$
$R^{x^i_1,x^i_2,...,x^i_d}(k)$	Test response signature issued by tested processor node $u_{x^i_1,x^i_2,...,x^i_d}$ after $T^{x_1,x_2,...,x_d}(k)$ is received
$R_0^{x^i_1,x^i_2,...,x^i_d}(k)$	The reference test response signature expected to be issued by processor node $u_{x^i_1,x^i_2,...,x^i_d}$ after receiving $T^{x_1,x_2,...,x_d}(k)$
T, R, R_0	Temporarily used variables
←	The value assignment/transfer operator

Table 2: The functions of the nodes and gates presented in Fig. 3

Nodes or gates	Functions
Memory node 1	Stores the test signatures issued by the processor node
Circular binary counter 2	Counts the clock pulses arrived between two adjacent test loops performed by the processor node
Circular binary counter 3	Points to the next test signature in memory 1 to be issued by the processor node
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
NOR gate 7 together with univibrator 9	Detect whether counter 2 has reentered 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 node 11	Stores the reference response signatures for the tested neighbors of the current processor
Circular binary counter 12	Counts the clock pulses 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 neighbor 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 reentered 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

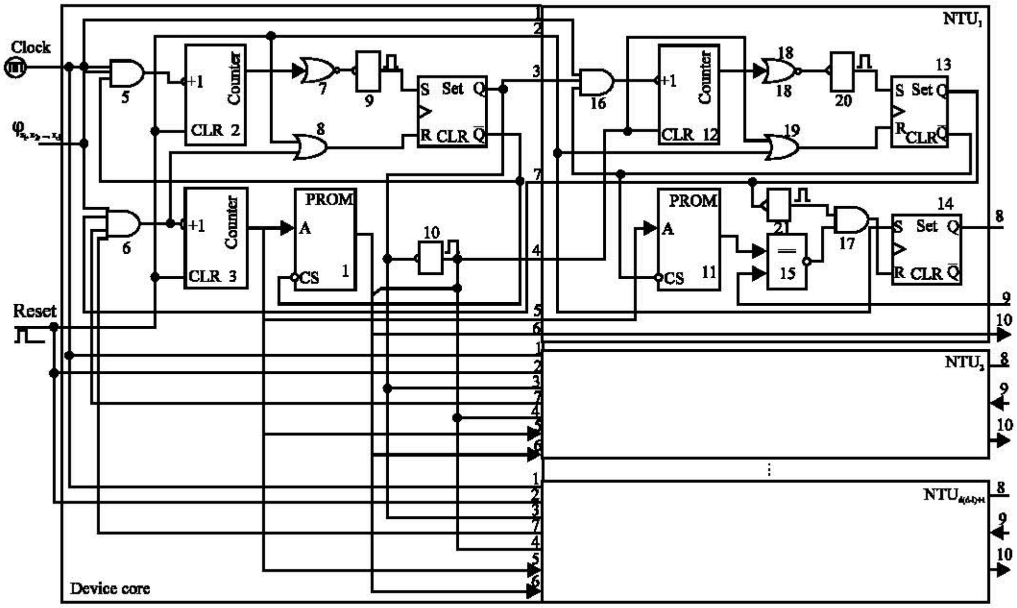

Fig. 3: Logic diagram of the embedded test hardware implementing the algorithm of Fig. 2

The device of Fig. 3 is supposed to be a part of each processor node; it consists of the device core and d(d-1)+1 Neighbor Test Units (NTU). The device core executes the initial and final sequential threads of the mutual inter-node test algorithm while NTU$_i$ implements thread $B_i, i = \overline{1, d(d\text{-}1)+1}$ (Fig. 2). Taking into account the fact that the NTUs are identical, only NTU_1 is detailed in Fig. 3. The adopted input/output numbering scheme helps understand the connections between the device core and the NTUs. The functions of the nodes and logic gates shown in Fig. 3 are detailed in Table 2.

RESULTS AND DISCUSSION

Comparing the mutual inter-node test approach to self-checking and neighbor-checking: The mutual inter-node test approach is a good alternative to the self-checking and neighbor-checking techniques providing better multiprocessor testability which is demonstrated in the present study.

Let π(t) be the probability that a processor node of the multiprocessor is properly detected as faulty by a separate test node (NTU). Taking into account that there are C^i_j = j!/i!(j-i)! possible combinations of testing neighbors correctly reporting that the current processor is faulty, the following formula can be deduced:

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

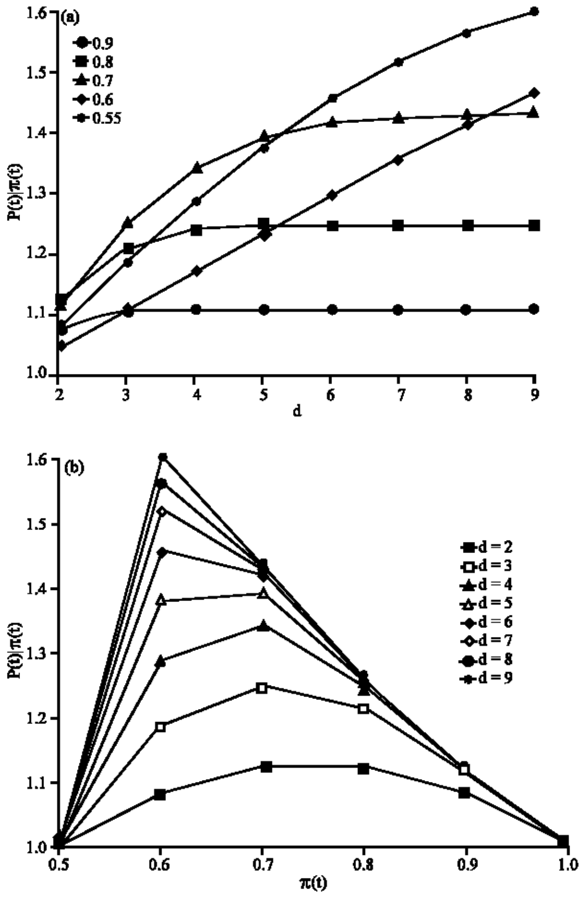

Fig. 4: a) P(t)|π(t) versus d and b) P(t)|π(t) versus π(t) graphs for fixed π(t) and d, respectively

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

To evaluate the effect provided by the mutual inter-node test, we assume that π(t) equals to the probability of successful self-test or neighbor-test (we presuppose that each processor has a built-in NTU or similar hardware to check its or its neighbor's state) and then calculate P(t)/π(t) depending on d and d with fixed π(t) and d, respectively (Fig. 4).

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

CONCLUSION

In the present study, we have presented a new approach, the mutual inter-node 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 and neighbor-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-node test technique allows hardware-level testing of all the processor nodes across the mesh in parallel, thus, significantly contributing to the test environment performance.

REFERENCES

Aguilera, M.K., W. Chen and S. Toueg, 2000. Failure detection and consensus in the crash-recovery model. Distrib. Comput., 13: 99-125.

Al-Azzeh, J.S., M.E. Leonov, D.E. Skopin, E.A. Titenko and I.V. Zotov, 2015. The organization of built-in hardware-level mutual self-test in mesh-connected VLSI multiprocessors. Intl. J. Inf. Technol., 3: 29-33.

Anonymous, 2015a. TILE-Gx72 multicore processor: 72-core processor for intelligent networking and video processing. Tilera, San Jose, California, USA.

Anonymous, 2015b. Vega 3 processor. Azul Systems, Inc, Mountain View, California, USA.

Bernardi, P., L.M. Ciganda, E. Sanchez and M.S. Reorda, 2014. MIHST: A hardware technique for embedded microprocessor functional on-line self-test. IEEE. Trans. Comput., 63: 2760-2771.

Ciciani, B., 1998. Manufacturing Yield Evaluation of VLSI-WSI Systems. IEEE Computer Society Press, Los Alamitos, California, USA.,.

Fukushi, M. and S. Horiguchi, 2004. Reconfiguration algorithm for degradable processor arrays based on row and column rerouting. Proceedings of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'04), October 10-13, 2004, IEEE, Cannes, France, pp: 496-504.

Jafri, S.M., S.J. Piestrak, O. Sentieys and S. Pillement, 2014. Design of the coarse-grained reconfigurable architecture DART with on-line error detection. Microprocess. Microsyst., 38: 124-136.

Jigang, W. and T. Srikanthan, 2003. An improved reconfiguration algorithm for degradable VLSI/WSI arrays. J. Syst. Archit., 49: 23-31.

Kolonis, E., M. Nicolaidis, D. Gizopoulos, M. Psarakis and J.H. Collet *et al*., 2009. Enhanced self-configurability and yield in multicore grids. Proceedings of the 15th IEEE International Symposium on On-Line Testing (IOLTS'09), June 24-26, 2009, IEEE, Sesimbra, Lisbon, Portugal, ISBN:978-1-4244-4596-7, pp: 75-80.

Krstic, A., W.C. Lai, K.T. Cheng, L. Chen and S. Dey, 2002. Embedded software-based self-test for programmable core-based designs. IEEE. Des. Test Comput., 19: 18-27.

Lin, S.Y., W.C. Shen, C.C. Hsu, C.H. Chao and A.Y. Wu, 2009. Fault-tolerant router with built-in self-test-self-diagnosis and fault-isolation circuits for 2D-mesh based chip multiprocessor systems. Proceedings of the International Symposium on VLSI Design, Automation and Test (VLSI-DAT'09), April 28-30, 2009, IEEE, Hsinchu, Taiwan, ISBN:978-1-4244-2781-9, pp: 72-75.

Nicolaidis, M. and L. Anghel, 1999. Concurrent checking for VLSI. Microelectron. Eng., 49: 139-156.

Psarakis, M., D. Gizopoulos, E. Sanchez and M.S. Reorda, 2010. Microprocessor software-based self-testing. IEEE. Des. Test Comput., 27: 4-19.

Raik, J. and V. Govind, 2012. Low-area boundary BIST architecture for mesh-like network-on-chip. Proceedings of the IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS'12), April 18-20, 2012, IEEE, Tallinn, Estonia, ISBN:978-1-4673-1187-8, pp: 95-100.

Rajski, J., J. Tyszer, M. Kassab and N. Mukherjee, 2004. Embedded deterministic test. IEEE. Trans. Comput. Aided Des. Integr. Circ. Syst., 23: 776-792.

Roychowdhury, V.P., J. Bruck and T. Kailath, 1990. Efficient algorithms for reconfiguration in VLSI/WSI arrays. IEEE. Trans. Comput., 39: 480-489.

Stroud, C., J. Sunwoo, S. Garimella and J. Harris, 2004. Built-in self-test for system-on-chip: A case study. Proceedings of the International Conference on Test (ITC'04), October 26-28, 2004, IEEE, Charlotte, North Carolina, USA., pp: 837-846.

Takanami, I., 2001. Built-in self-reconfiguring systems for fault tolerant mesh-connected processor arrays by direct spare replacement. Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, October 24-26, 2001, IEEE, San Francisco, California, USA., pp: 134-142.

Vangal, S.R., J. Howard, G. Ruhl, S. Dighe and H. Wilson *et al*., 2008. An 80-tile sub-100-w teraflops processor in 65-nm cmos. Solid State Circuits IEEE. J., 43: 29-41.

Zhang, Z., D. Refauvelet, A. Greiner, M. Benabdenbi and F. Pecheux, 2014. On-the-field test and configuration infrastructure for 2-D-mesh NoCs in shared-memory many-core architectures. IEEE. Trans. Very Large Scale Integr. Syst., 22: 1364-1376.