Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 05, No. 01, April 2019.

DISTRIBUTED MUTUAL INTER-UNIT TEST METHOD FOR *D*-DIMENSIONAL MESH-CONNECTED MULTIPROCESSORS WITH ROUND-ROBIN COLLISION RESOLUTION

Jamil Al-Azzeh

(Received: 16-Oct.-2018, Revised: 23-Nov.-2018, Accepted: 8-Dec.-2018)

ABSTRACT

A collision-free extension to the mutual inter-unit test methodology for d-dimensional VLSI multiprocessors is proposed to guarantee that any processor core is tested only by its neighboring node at a time and no special care needs to be taken to choose those moments when test actions should start. Collision resolution hardware based on the round-robin arbitration routine is discussed in detail. A parallel collision-resolution-aware mutual inter-unit test algorithm is formulated and diagrammed. The proposed approach has been shown to improve the testability of mesh-connected multiprocessors by increasing the probability of successful fault detection as compared with the distributed self-checking methodology. Further, the new approach drastically reduces extra connectivity in the multiprocessor with respect to known mutual inter-unit test methods and leads to more easily manufactured multiprocessor fabric. For example, in a 4-dimensional system, we need 55% less extra connections with our approach.

KEYWORDS

Multiprocessors, VLSI, Mesh topology, Reliability, Testability, Self-test, Mutual inter-unit test.

1. INTRODUCTION

Continuing VLSI miniaturization has enabled the production of high-performance multicore and many-core single-chip multiprocessors comprising up to thousands of processor cores [4], [7]. However, the unreliability of such multiprocessor components has emerged as one of the crucial limitations to future scaling [12]-[13]. To maintain the correct operations of these multiprocessor systems with unhealthy components, specific fault-tolerance issues must be addressed when designing the multiprocessor [1], [3], [11], [15], [23], [27], [34] and [38]-[39]. Detecting the location of faulty components is one of these issues [2], [17] and [36].

A VLSI multiprocessor containing faulty components can be considered healthy if a dedicated fault detection and isolation mechanism is deployed [6], [9], [16], [21], [24], [31]-[32], [37]. With no specific mechanism of spare replacements, the multiprocessor maintains its operation, but its performance gradually degrades [19]-[20]. If a spare replacement mechanism is assumed [14], [18], the multiprocessor's performance is retained in the presence of the unhealthy components.

The detection of faulty components in VLSI multiprocessors is typically solved by built-in distributed self-checking or neighbor-checking methods [5], [10], [26], [28], [30], [40]. It is important that fault detection is done in-operation, which means that no long-term interruption of the multiprocessor is required to pinpoint an unhealthy component. Distributed self-test methods are an efficient, yet simple, solution to fault detection [8], [22], [25], [29], [33], [35]. However, these methods are characterized by relatively low testability: they may miss faulty components in some cases and/or treat healthy units as defective. Thus, the probability that a processor core is properly self-detected as faulty is not high enough for many practical applications. Another straightforward approach to fault detection is that a processor occasionally sends probe signals to its neighbors and marks neighbor cores as defective if no

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

acknowledgment is received within an established period of time. Such a neighbor-checking approach is also relatively simple to implement. However, it cannot provide better testability as compared to self-checking, because processor nodes test their peers independently and healthy/faulty decisions are made locally with no inter-processor cooperation.

A more complex fault detection mechanism, the mutual inter-unit test, has been specifically designed to improve multiprocessor testability by employing checking schemes with neighbor cooperation, by which each processor node is occasionally checked by a number of its neighbors and the final faulty/healthy decision is made according to the majority operator rule [41]. With this approach and depending on the topology of the multiprocessor, the probability of successful fault detection can be increased by at least 10% as compared to self-checking mechanisms. To improve the utilization of testing the hardware across a multiprocessor mesh and to make an additional increase in the probability of successful fault detection, a multiplexed mutual inter-unit test method based on a similar checking scheme to that of [41] and combined with distributed self-checking has been proposed [42]. Each test unit of each processor is now allowed to check a pair of its neighbors, A and B (not necessarily direct neighbors). The checking time period is split 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, idle time is minimized and the testing hardware is used more efficiently in a time division manner. With this approach, the number of testing neighbors at each processing node is double of that in [41] with small hardware overhead, allowing a higher probability of successful fault detection. The main drawback of the multiplexed mutual inter-unit test is the assumption that each processor node (including those with non-direct processor cores) has many extra connections, which drastically increases the complexity of the communication network and may pose a serious problem if more dimensions are assumed. The inter-unit test methods presuppose that testing neighboring processor subsets check corresponding tested processors asynchronously, which may lead to collisions (two or more testing cores trying to check the same tested node at the same time); thus, special care must be taken to eliminate them.

Here, we propose another extension to the mutual inter-unit test methodology for *d*-dimensional VLSI multiprocessors by using a similar cooperating neighbor-checking scheme as that of [41]. Our main contribution is the use of a novel collision resolution mechanism (the round-robin collision resolution scheme), which guarantees that any processor core is tested by only one neighbor node at a time, so that no special care need be taken to choose the moments when test actions should start. In the following sections, we formally state the proposed method for a *d*-dimensional VLSI multiprocessor to concurrently detect faulty/defective nodes across a mesh. A parallel inter-unit test algorithm based on the proposed formal approach is presented and the dedicated test hardware implementing the above algorithm is diagrammed and briefly discussed. We also take a closer look at the round-robin collision resolution scheme, which is the cornerstone of our method. At the end of the paper, we compare our approach to the distributed self-checking technique and existing inter-unit test methods.

2. MUTUAL INTER-UNIT TEST AND COLLISION RESOLUTION FUNDAMENTALS

The idea of the mutual inter-unit test is straightforward. Each processor core is occasionally checked by its neighbors (referred to as "testing neighbors"). It is also assumed that a processor core periodically performs self-testing. The faulty/healthy decision for each processor is made according to the majority operator rule applied to the individual faulty/healthy decisions arriving from the testing neighbors and self-test hardware. The set of testing neighbors for each processor is formed according to the number of dimensions (*d*) of the multiprocessor topology, whose cardinality should be odd to make the majority operator applicable. The mutual inter-unit test procedure is carried out concurrently across the mesh, so that the faulty processors are detected and the corresponding signals are immediately transferred to the physical neighbors in order to isolate the faulty/defective cores in a timely manner. Unlike the distributed self-checking and neighbor-checking methods, the mutual inter-unit test mechanism provides for the operability of the test hardware itself to be tested implicitly. For example, if one of the testing processors issues a wrong faulty/healthy decision for its neighbor, then the tested neighbor (which is, in fact, healthy) will not be assumed as faulty by mistake, because the resulting faulty signal is formed by the majority operator. Therefore, the probability of successful fault detection increases.

The problem with the inter-unit test mechanism is that collisions may occur when several testing

neighbors start checking the same tested processor at the same time. Known inter-unit test schemes do not introduce any dedicated procedures and hardware to resolve the collisions; so, extra software-level solutions are necessary to calculate the time windows when a processor is allowed to test its neighbors with no collisions. However, this may drastically slow down the test process across the mesh, but may work against the multiprocessor's reliability. Hence, we propose an extended version of the inter-unit test with no possible inter-processor collisions. We assume that a hardware-level collision resolution mechanism, which we refer to as *the round-robin arbitration scheme*, is added to each processor core.

The idea of the collision resolution mechanism is for each testing neighbor (including the self-test units) of a given processor to be assigned an arbitration flag whose high value grants permission to start the test procedure. If this flag is zero, then the corresponding testing neighbor is not permitted to perform the test. The set of arbitration flags of each tested neighbor are organized as a ring shift register containing only the high value, which moves along the ring in a given direction and activates only one testing neighbor at a time. When a testing neighbor is about to initiate the test, it first polls the arbitration flag. If the flag is high, then the testing neighbor commences the test procedure and the flag stops moving until the test is finalized. If the flag is low, then the test is not initiated. The testing neighbor may be put into a queue to spin until the flag clears or simply leaves and tries to initiate the test the next time. With such an arbitration scheme, each processor is guaranteed to be tested by only one neighbor (or self-test unit) at a time. Therefore, no extra software-level support is required to pre-determine the time for initiating the test.

The remainder of the paper is organized as follows. In Section 3, we formally define the construction rule of the testing neighbor sets for a *d*-dimensional multiprocessor. Section 4 provides details on the proposed inter-unit test procedure. In Section 5, a hardware-level implementation of our approach is discussed. Section 6 provides the necessary details on the round-robin arbitration scheme. Section 7 is dedicated to the evaluation and comparison of our proposed approach to the distributed self-test and existing inter-unit test solutions. Section 8 contains the concluding remarks.

3. THE FORMATION OF TESTING NEIGHBOR SETS

We propose a more straightforward rule to form testing neighbor sets for each processor as compared to those defined in [41] and [42]. We assume that only the direct neighbors of a given core can be its testing neighbors, thereby eliminating extra diagonal connections among the processors and reducing the communication hardware complexity of the multiprocessor. With 4 direct neighbors in a 2-dimensional mesh, each processor has 5 testing neighbors if self-checking capabilities are assumed. With 6 direct neighbors in a 3-dimensional mesh, there will be 7 testing neighbors at each processor node. Analogously, in a *d*-dimensional mesh, each processor will be checked by $2d + 1$ testing neighbors.

Taking into account the processor nodes at the edges of the mesh, we can formally state the above rule as follows. Let us first consider a 2-dimensional multiprocessor. Let $U = \{u_{xy}\}$ be the set of its processors with *x* and *y* standing for the coordinates of a processor relative to the leftmost and lowermost node of the mesh, $x = \overline{0, n-1}$ and $y = \overline{0, m-1}$, respectively. *m* and *n* denote the numbers of rows and columns, respectively, of the mesh structure. Then, the testing neighbor set K'_{xy} of processor u_{xy}, $x \in \{0,1, \dots, n-1\}$, $y = \{0,1, \dots, m-1\}$, will be formalized as:

$$K'_{xy} = K_{xy} \cup \{u_{xy}\}, \tag{1}$$

$$K_{xy} = \left\{ u_{x,(y+1)\bmod m}, u_{(x+1)\bmod n, y}, u_{x+(1-\operatorname{sign}(x))n-1, y}, u_{x, y+(1-\operatorname{sign}(y))m-1} \right\}. \tag{2}$$

In Figure 1, different allocations of testing neighbors for the 2-dimensional case are illustrated. The testing neighbors are shown in grey and the dotted squares denote the testing nodes (which are mapped onto the corresponding cores at the opposite sides of the mesh) missing at the edges of the mesh.

Rules (1) and (2) can be directly expanded into a *d*-dimensional multiprocessor case:

$$K'_{x_1 x_2 \dots x_d} = K_{x_1 x_2 \dots x_d} \cup \{u_{x_1 x_2 \dots x_d}\} \tag{3}$$

$$K_{x_1 x_2 \ldots x_d} = \left\{ \begin{matrix} u_{(x_1+1) \bmod n_1, x_2, \ldots, x_d}, u_{x_1, (x_2+1) \bmod n_2, x_3, \ldots, x_d}, \ldots, u_{x_1, x_2, \ldots, (x_d+1) \bmod n_d}, \\ u_{x_1+(1-\mathrm{sign}(x))n_1-1, x_2, \ldots, x_d}, u_{x_1, x_2+(1-\mathrm{sign}(y))n_2-1, x_3, \ldots, x_d}, \ldots, u_{x_1, x_2, \ldots, x_d+(1-\mathrm{sign}(y))n_d-1} \end{matrix} \right\}. \quad (4)$$

It is evident that

$$\left| K_{x_1 x_2 \ldots x_d} \right| = 2d, \quad (5)$$

$$\left| K'_{x_1 x_2 \ldots x_d} \right| = 2d + 1. \quad (6)$$

Formula (6) guarantees an odd number of testing neighbors at each processor and renders the majority operator applicable to faulty/healthy decisions. The collision resolution mechanism, in turn, guarantees that any processor u_{xy} is never checked by more than one peer at a time.

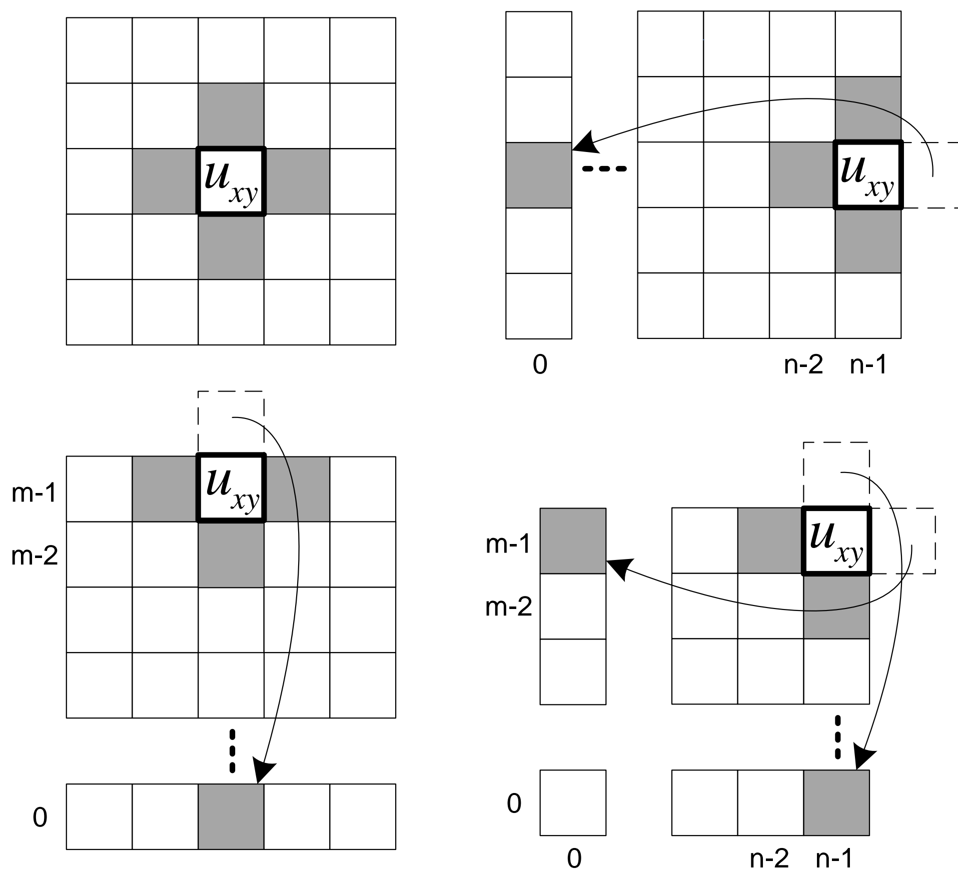

Figure 1. Possible allocations of testing neighbors K_{xy} in a 2-dimensional multiprocessor.

4. MUTUAL INTER-UNIT TEST PROCEDURE

The above conceptual description and formal rules allow the creation of an algorithm representing the process of the inter-unit test occasionally performed by each processor as we explain in detail below.

We consider a *d*-dimensional multiprocessor and its arbitrary processor $u_{x_1 x_2 \ldots x_d}$. For clarity, we indicate $u_{x_1 x_2 \ldots x_d}$. with the superscript 0 and enumerate its consecutive neighbors by the superscripts$1, 2, \ldots, 2d$, respectively. For the 2-dimensional case, we would have 4 neighbors renumbered as $u_{xy}^1, u_{xy}^2, u_{xy}^3, u_{xy}^4$.

The proposed algorithm is presented in Figure 2. All the symbols used in the flow-chart are explained in Table 1.

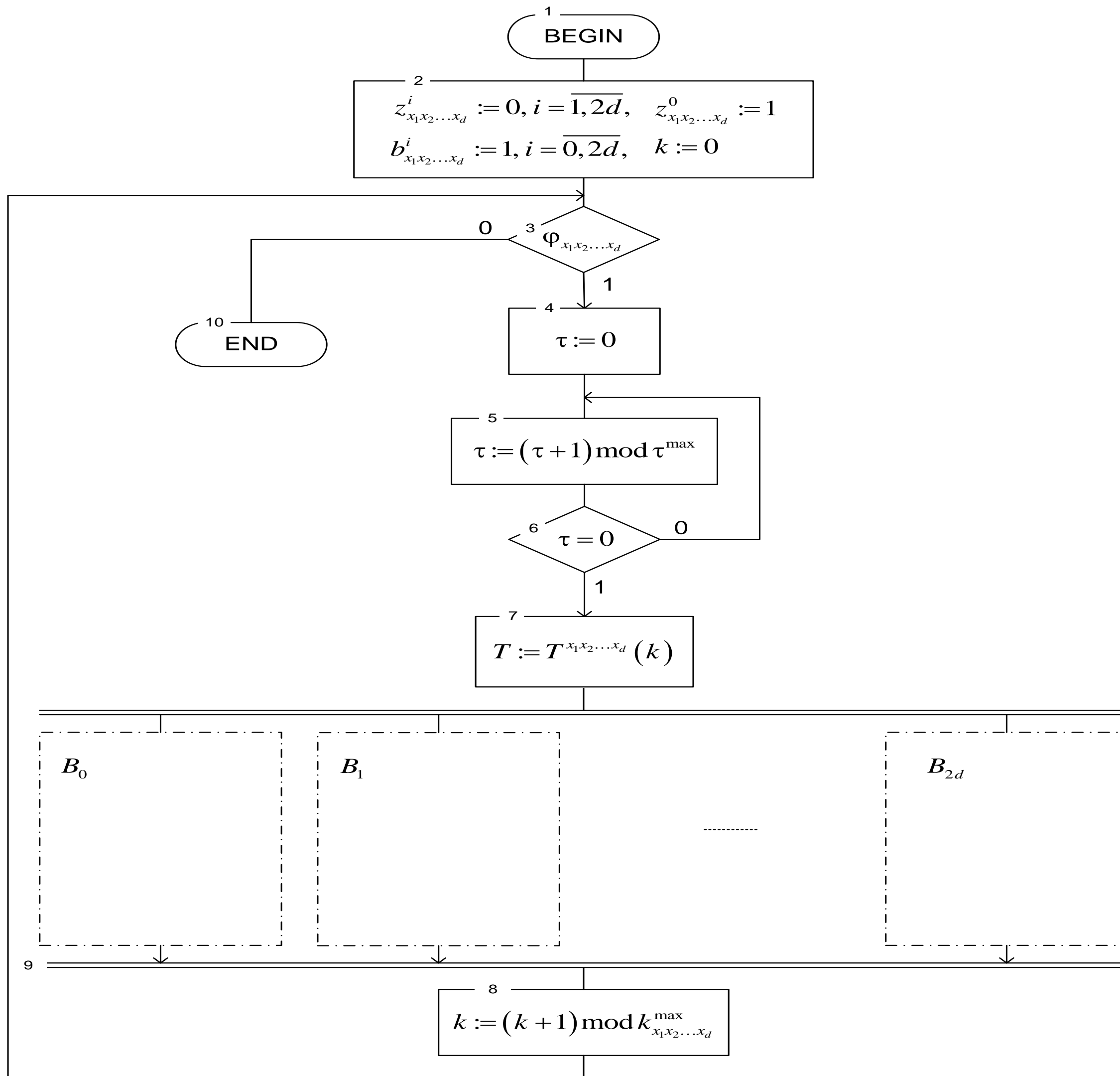

Figure 2. The proposed mutual inter-unit test algorithm.

Table 1. Symbols used in the flowchart shown in Figure 2.

No.	Symbol	Meaning
1	$T^{x_1x_2\dots x_d}(k)$	k^{th} test signature issued by processor $u_{x_1x_2\dots x_d}$
2	$k,\ k = \overline{0, k^{\max}_{x_1x_2\dots x_d} - 1}$	Test signature counter
3	T	Current test signature to be transferred to the tested neighbors
4	$k^{\max}_{x_1x_2\dots x_d}$	Number of test signatures supported by node $u_{x_1x_2\dots x_d}$
5	τ	Test loop start timer
6	$\tau^{\max}$	Time between two adjacent test loops (in clock ticks)
7	$z^{i}_{x_1x_2\dots x_d}$	Test enable flag for testing node $u_{x_1x_2\dots x_d}$ and tested node $u^{i}_{x_1x_2\dots x_d}$
8	$b^{i}_{x_1x_2\dots x_d}$	Test enable flag for tested node $u^{i}_{x_1x_2\dots x_d}$ and other testing neighbors (not including $u_{x_1x_2\dots x_d}$)
9	$\varphi_{x_1x_2\dots x_d}$	Healthy/faulty flag of node $u_{x_1x_2\dots x_d}$
10	$B_0, B_1, \dots, B_{2d}$	Separate parallel test threads corresponding to the tested neighbors and the self-test hardware of node $u^{i}_{x_1x_2\dots x_d}$
11	:=	Assignment/transfer operator

The algorithm contains an outermost loop, including a parallel section and commences execution with the initialization (see Statement 2) to set up the collision resolution flags properly. Flags $z^{i}_{x_1x_2\ldots x_d}$ are reset to zero, which means that no neighbor of the current processor is allowed to start the test routine. In contrast, flag $z^{0}_{x_1x_2\ldots x_d}$ is set to logical "1," which means that it is allowed to perform the self-test.

As soon as the initialization ends, the algorithm enters the loop and executes until the current node is assumed to be healthy (see Condition 3). When the next iteration begins, the test loop timer starts counting down first (see Vertices 4–6). As soon as the timer has finished (the predefined time interval τ^{max} has elapsed), the next test signature $T^{x_1x_2\ldots x_d}(k)$ is fetched to initiate the test actions in the tested neighbors of the current node. A test signature may be interpreted as a pointer (address) to the test routine to be executed. All test routines are assumed to have been predefined and distributed among the processor cores in advance.

As soon as test signature $T^{x_1x_2\ldots x_d}(k)$ is read out, the algorithm enters the parallel section and threads $B_0, B_1, \ldots, B_{2d}$ to start the execution (see dashed–dotted squares in Figure 2). All these threads are identical. Thread B_i corresponds to the ith tested neighbor and thread B_0 is mapped onto the current node (self-test). When all the threads terminate, Statement 8 executes to increment the test signature counter k and the next iteration begins. As soon as all $k^{\max}_{x_1x_2\ldots x_d}$ test signatures are fetched and processed, the algorithms roll back to test signature $T^{x_1x_2\ldots x_d}(0)$, assuming $k = 0$.

Each thread B_i $\left(i = \overline{0,2d}\right)$ can be represented as a separate algorithm, as shown in Figure 3. All the symbols used in the flow-chart of Figure 3 are explained in Table 2.

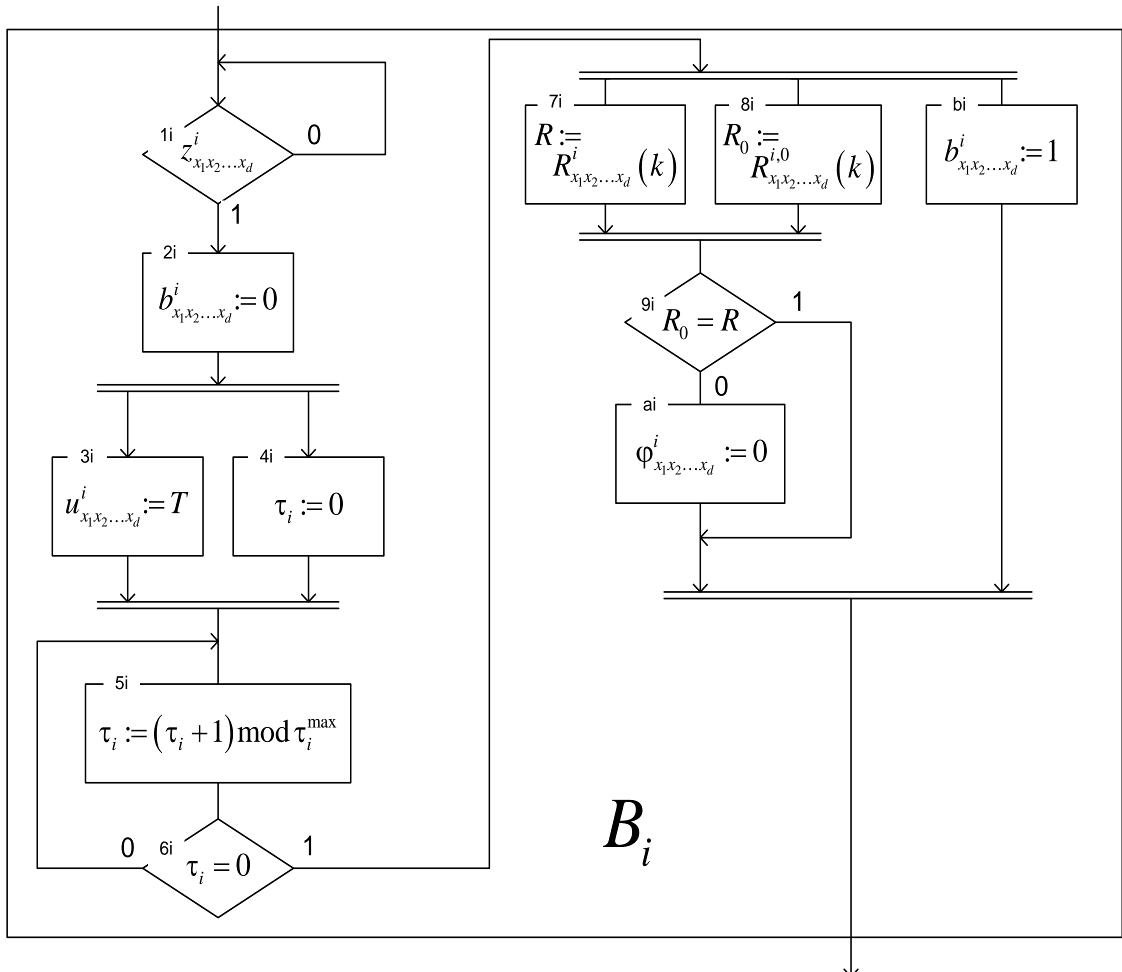

Figure 3. Flow-chart representing thread B_i.

Table 2. Symbols used in the flowchart shown in Figure 3.

No.	Symbol	Meaning
1	$\tau_i^{\max}, i=\overline{0,2d}$	Node $u^i_{x_1x_2\ldots x_d}$ response time limit
2	$\tau_i, i=\overline{0,2d}$	Node $u^i_{x_1x_2\ldots x_d}$ response time counter
3	$R^i_{x_1x_2\ldots x_d}(k)$	Node $u^i_{x_1x_2\ldots x_d}$ response token corresponding to test signature $T^{x_1x_2\ldots x_d}(k)$
4	$R^{i,0}_{x_1x_2\ldots x_d}(k)$	Node $u^i_{x_1x_2\ldots x_d}$ expected response corresponding to test signature $T^{x_1x_2\ldots x_d}(k)$
5	$\varphi^i_{x_1x_2\ldots x_d}$	Node $u^i_{x_1x_2\ldots x_d}$ faulty/healthy decision made by processor $u_{x_1x_2\ldots x_d}$
6	$u^i_{x_1x_2\ldots x_d}$	i^{th} tested node of the current processor
7	R, R_0	Extra buffers

In the first step, thread B_i starts to spin while waiting for condition $z^i_{x_1x_2\ldots x_d}=1$ to become true (see Vertex 1*i*). Immediately, the test (self-test) routine begins for tested neighbor (current node) $u^i_{x_1x_2\ldots x_d}$. Then, flag $b^i_{x_1x_2\ldots x_d}$ is reset (Vertex 2*i*), leading to no other nodes being allowed to test $u^i_{x_1x_2\ldots x_d}$. As a result, no collisions occur, because for any testing unit trying to start checking, the ith tested neighbor $z^i_{x_1x_2\ldots x_d}$ is clear. During the next step, test signature $T^{x_1x_2\ldots x_d}(k)$ is transferred to $u^i_{x_1x_2\ldots x_d}$ (Statement 3*i*) and the timer counts down until $\tau_i^{\max}$ elapses (see Vertices 4*i*–6*i*). As soon as $\tau_i=0$, test response $R^i_{x_1x_2\ldots x_d}(k)$ arrives (or does not in some cases) from tested node $u^i_{x_1x_2\ldots x_d}$ (Statement 7*i*). Concurrently, expected test response $R^{x_1x_2\ldots x_d}(k)$ is fetched (Statement 8*i*) to be compared to $R^i_{x_1x_2\ldots x_d}(k)$ (see Condition 9*i*). If the test response (or whatever has arrived) differs from what is expected to arrive, then $u^i_{x_1x_2\ldots x_d}$ is assumed to be faulty and flag $\varphi^i_{x_1x_2\ldots x_d}$ is reset to zero (Statement a*i*). Otherwise, nothing happens and $\varphi^i_{x_1x_2\ldots x_d}$ remains high. In parallel, flag $b^i_{x_1x_2\ldots x_d}$ is again set high (Statement b*i*), making it possible to self-test or for the other neighbors to test node $u^i_{x_1x_2\ldots x_d}$.

One must mention that all the statements and conditions in the above algorithm are based on simple atomic operations, such as increment, assignment, set/reset, compare and test for zero/one. Therefore, it can be directly implemented in hardware.

5. Hardware-level Implementation

In this section, we discuss the possible hardware-level implementation of the above mutual inter-unit test mechanism. We consider both test units and collision resolution hardware. Taking into account that all processor cores in the multiprocessor are identical, we pick up an arbitrary node for consideration. Assuming a 2-dimensional multiprocessor, we can represent the structure of the test hardware, as shown in Figure 4.

The unit presented in Figure 4 contains 4 identical neighbor check units (NCU1–NCU4) and a self-test unit (STU). The operation of these units is based on the thread algorithm shown in Figure 3. A test organization unit (TOU) is needed to store and fetch test signatures mapped onto the current node in order to control the delay between adjacent test cycles and to coordinate the operation of the NCUs and

STU. TOU implements all the sections of the proposed test algorithm (Figure 2) except for parallel threads $B_0, B_1, \dots, B_{2d}$. Arbitration flip-flops AF0–AF4 organized in a ring shift register are required to perform the proposed round-robin collision resolution scheme to guarantee that two or more neighbors never start checking the same processor within the overlapping time frames. Only one AF can be high at a given moment. This high value moves from one flip-flop to its neighbor and lets each neighboring processor check the current node in a time division manner. A clock pulse generator (CPG) is used to synchronize the operation of the test hardware components. A separate CPG may be employed or the processor's main pulse generator may be considered as a CPG.

The neighbor check units form and issue healthy/faulty flags φ_{xy}^i for the corresponding neighbor nodes according to the following rule: $\varphi_{xy}^i = 1$ if processor u_{xy} makes a decision that neighbor u_{xy}^i is healthy and $\varphi_{xy}^i = 0$ otherwise (we assume that $u_{xy}^0 \equiv u_{xy}$). The same is carried out by the testing neighbors of the current processor. As an addition, node u_{xy} occasionally performs a self-checking routine that results in a healthy/faulty flag φ_{xy}^0. Finally, a generalized faulty/healthy flag φ_{xy} is calculated by the majority rule:

$$\varphi_{xy} = \#\left(\varphi_{xy}^0, \varphi_{xy}^1, \varphi_{xy}^2, \varphi_{xy}^3, \varphi_{xy}^4\right), \tag{7}$$

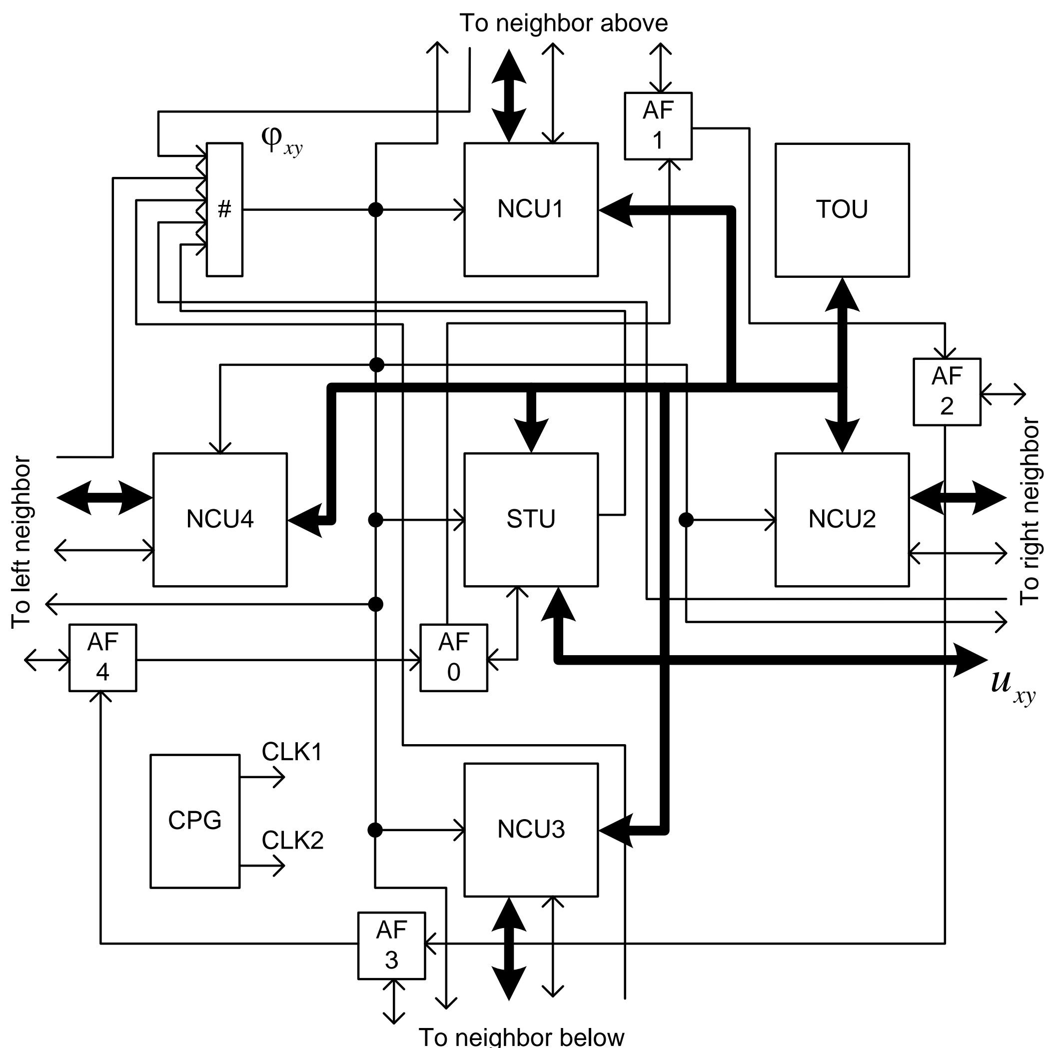

Figure 4. The organization of processor node test hardware in a 2-dimensional multiprocessor.

where # denotes the majority operator. If $\varphi_{xy} = 1$, then u_{xy} is assumed to be healthy. If $\varphi_{xy} = 0$, then it is further treated as faulty.

The structure shown in Figure 4 can be extended to a multiprocessor of any given dimension $d \geq 2$. In a general case, a processor's test hardware includes $2d$ NCUs operating in parallel, $2d$ bidirectional links needed to transfer test signatures to the tested neighbors and to receive response tokens, as well as to receive test signatures from the testing neighbors and transfer response tokens, $2d$ input terminals required to receive healthy/faulty flags from the testing neighbors and $2d$ output terminals to issue the generalized healthy/faulty flag to the direct neighboring processors. The count of the majority gate's input terminals is calculated by Formula (7).

6. THE ROUND-ROBIN ARBITRATION MECHANISM

Using the above conceptual representation, we have developed a functional diagram embodying the round-robin arbitration mechanism necessary to avoid inter-processor test collisions. The scheme of a 2-dimensional multiprocessor is shown in Figure 5, which includes five JK flip-flops with input invertors needed to store the test enable flags. Consequent flip-flop enumeration is adopted and corresponds to the enumeration of the neighbor nodes (the ith flip-flop together with its inverter corresponds to the AFi unit in the block diagram of Figure 4). The flip-flops are connected to each other to form a ring shift register whose operation is clocked by pulse chain CLK1 issued by CPG (not shown in Figure 5 for simplicity). The AND gates are introduced to block pulse chain CLK1 from clocking the flip-flops when the current node is being tested by a neighbor or being self-tested.

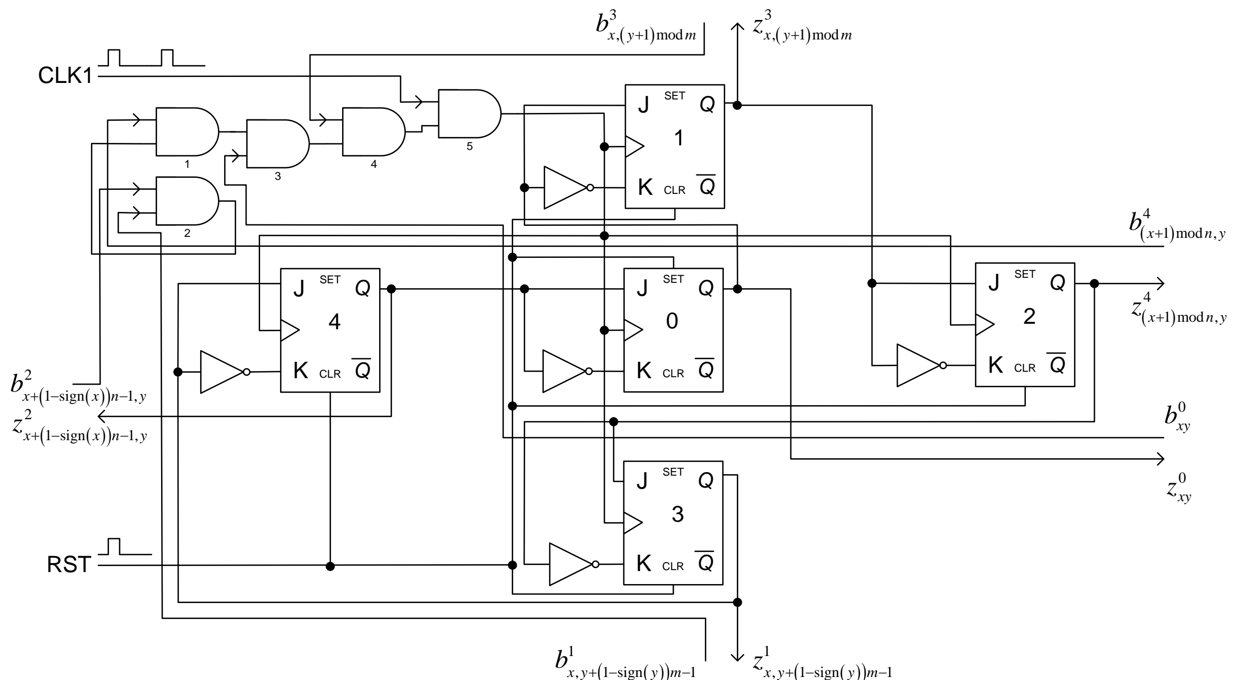

Figure 5. The round-robin arbitration hardware functional diagram for a 2-dimensional multiprocessor.

Initially, a system reset pulse arrives to initialize the flip-flops. As a result, flip-flop 0 is set to "1" while the remaining flip-flops are reset to "0." Thus, the initial state of the test hardware will be in accordance with Statement 2 of the above algorithm (see Figure 2); i.e., all processors are initialized to start self-test routines.

If the current processor is about to start a self-test, then STU (see Figure 4) issues flag $b^0_{xy} = 0$ (which implies that Vertex 2i of thread B_0 will execute). As a result, gate 5 is blocked and no pulse CLK1 is able to clock the flip-flops. Flip-flop 0 remains set while the rest of the flip-flops are reset until the self-test routine terminates. As soon as self-test is done, high flag $b^0_{xy} = 1$ arrives (see Vertex bi of the above algorithm) and the AND gates are unblocked, making it possible to clock the flip-flops. Note that flag b^0_{xy} may remain set and the AND gates may be open if the test routine start timer has not finished counting down yet.

Another pulse CLK1 passes by AND gate 5, feeds the clock inputs of all the flip-flops and transfers the value of logical "1" to flip-flop 1 from flip-flop 0. In turn, flip-flop 0 is reset to zero because of the low

state of flip-flop 4. As a result, flip-flop 1 is set while the remaining peers are reset. After that, flag $Z^3_{x,(y+1) mod\ m} = 1$ arises and allows neighboring processor $u_{x,(y+1) mod\ m}$ to initiate the test process for the current node (superscript "3" implies that current node u_{xy} is the third neighbor of processor $u_{x,(y+1) mod\ m}$).

When processor $u_{x,(y+1) mod\ m}$ starts checking the current node, its NCU3 issues flag $b^3_{x,(y+1) mod\ m} = 0$ (which corresponds to Vertex $2i$ in thread B_3 of the proposed algorithm). AND gate 5 becomes blocked and pulses CLK1 can no longer feed the clock inputs of the flip-flops. Flip-flop 1 stays high while the rest remain clear until the test terminates. As soon as the test is finished, flag $b^3_{x,(y+1) mod\ m} = 1$ arrives (corresponding to Vertex bi of our algorithm), the AND gates are unblocked and clock pulses CLK1 start feeding the flip-flops. If flag $b^3_{x,(y+1) mod\ m}$ is not reset to zero (the test routine start timer has not counted down yet), then the AND gates remain open.

Analogously, the high-level value is transferred from flip-flop 1 to flip-flop 2, then travels from flip-flop 2 to flip-flop 3 and finally returns to flip-flop 0 from flip-flop 4, meaning that another arbitration loop is complete. The operation of the test hardware stays the same as discussed above. If flip-flop 2 becomes high, then node $u_{(x+1) mod\ n,y}$ gains the right to start testing the current processor. In turn, when flip-flop 3 is high, neighbor $u_{x,y+(1-sign(y))m-1}$ will check the current processor.

Note that the round-robin arbitration hardware can be easily extended to a d-dimensional case, but would include more flip-flops, inverters and terminals. For a 3-dimensional multiprocessor, 7 flip-flops are necessary, whereas 9 flip-flops are required in a 4-dimensional case.

7. Comparison of The Proposed Approach

7.1 Probability of Successful Fault Detection Evaluation

To compare the proposed inter-unit test method to the existing alternatives, the probability of successful fault detection is theoretically evaluated first and its dependencies on the multiprocessor dimension and reliability of separate test units are explored. The results are compared to the distributed self-checking and known mutual inter-unit test methods.

We take into account that a faulty node may be erroneously reported as healthy ("hidden fault") by a test unit and that a healthy processor may be mistakenly treated as faulty ("false fault"). All faults that are neither hidden nor false are known as "explicit faults;" i.e., detected faults that really exist. Thus, we define the probability of successful fault detection as a measure of the probability of properly detecting faulty nodes that really exist in the multiprocessor.

Let $\pi(t)$ be the probability that a separate test unit of a processor properly detects a faulty neighbor node (or the current node in the case of self-checking). Let $\pi^-(t)$ and $\pi^0(t)$ be the probabilities that a separate test unit is unable to detect a faulty node and claims a healthy node to be faulty, respectively. Then, the following fundamental relation will take place:

$$\pi(t) = 1 - \left(\pi^-(t) + \pi^0(t)\right). \tag{8}$$

Assuming $\pi(t)$, $\pi^-(t)$ and $\pi^0(t)$ are the same for all the multiprocessor's nodes across the mesh, we deduce the probability of successful fault detection formula for the proposed approach.
For the simplest 2-dimensional case, $|K'_{xy}| = 5$ and we obtain:

$$P(t)\big|_{d=2} = \sum_{i=3}^{5} P_5^i(t) = \sum_{i=3}^{5} C_5^i \pi(t)^i \left[1-\pi(t)\right]^{5-i}, \tag{9}$$

where $P_5^i(t)$ denotes the probability that i out of $|K'_{xy}| = 5$ testing nodes properly detect a faulty neighbor and C_5^i is the number of i item selections out of 5 items. For a 3-dimensional multiprocessor, $|K'_{xy}| = 7$ and we have:

$$P(t)\big|_{d=3} = \sum_{i=4}^{7} P_7^i(t) = \sum_{i=4}^{7} C_7^i \pi(t)^i \left[1-\pi(t)\right]^{7-i}, \tag{10}$$

where $P_7^i(t)$ denotes the probability that i out of $|K'_{xy}| = 7$ testing nodes properly detect a faulty neighbor. Using formulae (9) and (10), we deduce

$$P(t) = \sum_{i=\lceil (2d+1)/2 \rceil}^{2d+1} P_{2d+1}^i(t) = \sum_{i=\lceil (2d+1)/2 \rceil}^{2d+1} C_{2d+1}^i \pi(t)^i \left[1-\pi(t)\right]^{2d-i+1}. \quad (11)$$

Using Formula (11), we can investigate the dependencies of the probability of successful fault detection on the multiprocessor dimension and reliability of the separate test units. Having deduced the same formulae for existing approaches, it is possible to compare these approaches to our method on various factors.

Let us first compare our approach to the distributed self-test method. The self-test is able to detect faulty nodes with probability $\pi(t)$, because there is only one test unit in a given processor across the mesh. Therefore, this unit is sufficient to evaluate and explore the relation $\varphi(t) = P(t)/\pi(t)$ in order to compare the proposed method to the distributed self-test. Figure 6 shows the $\varphi(t)$ *versus* probability $\pi(t)$ graphs obtained using Formula (11).
Figure 6 shows that the proposed approach has the greatest advantage when $\pi(t) \approx 0.7$. For a 2-dimensional multiprocessor, the maximum $\varphi(t)$ value attained is 1.1956, which takes place at $\pi(t) = 0.7$ and means that the probability of successful fault detection increases by almost 20% as compared to the distributed self-test approach. The higher the multiprocessor dimension *d,* the greater the advantage of our method. For example, in a 5-dimensional mesh-connected multiprocessor, $\varphi(t)$ becomes higher than 1.3 at $\pi(t) = 0.7$, which signifies a 30% advantage. For higher values $\pi(t) \geq 0.9$, our method becomes less advantageous than a simple self-test. However, higher reliability test units are hard to build up in practice. For lower values $\pi(t) \leq 0.6$, our method also works worse and for $\pi(t) \leq 0.5$, it does not work at all. However, the case $\pi(t) \leq 0.5$ corresponds to "extremely unreliable" test units, which (according to Formula (8)) would claim faulty units to be healthy and/or treat healthy units as faulty in most cases such that a self-test would not be feasible with such units.

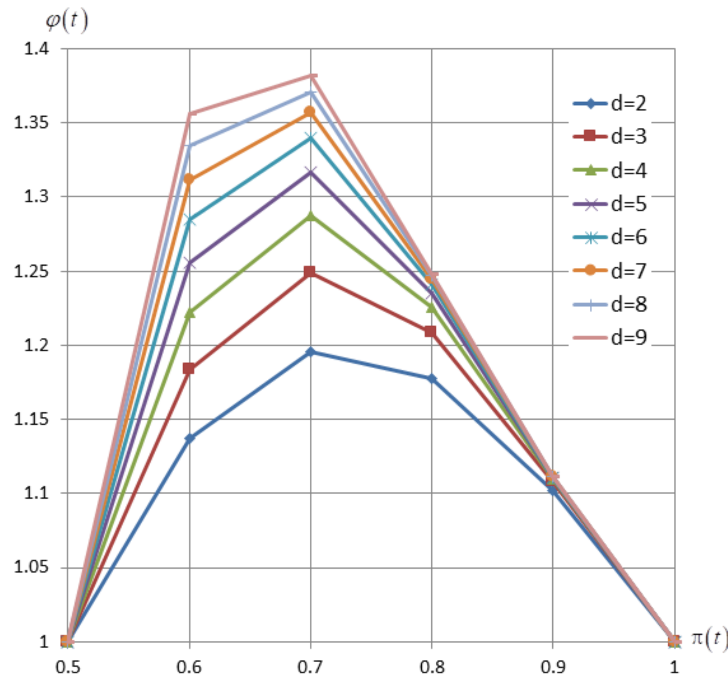

Figure 6. $\varphi(t)$ *versus* $\pi(t)$ graphs for fixed $2 \leq d \leq 9$.

Let us now compare our approach to the mutual inter-unit test method presented in [41] under the same assumptions as those formulated above. Let $P_0(t)$ denote the probability of fault detection attained when the mutual inter-unit test [41] is employed. Then, it is sufficient to evaluate and explore the relation $\psi(t) = P(t)/P_0(t)$ to compare the proposed method to the mutual inter-unit test. Figure 7 presents the $\psi(t)$ *versus* probability $\pi(t)$ graphs obtained using Formula (11) and a similar formula found in [41].

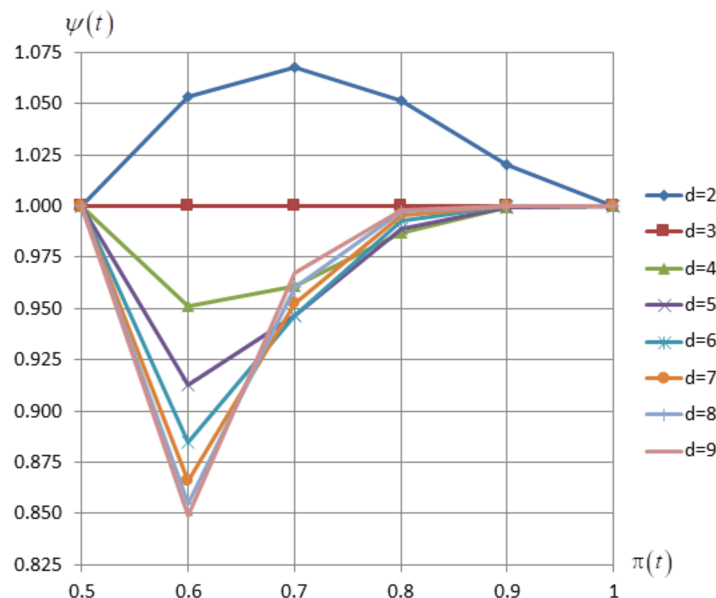

Figure 7. $\psi(t) versus\ \pi(t)$ graphs for fixed $2 \leq d \leq 9$.

Analyzing the graphs of Figure 7, we can see that our approach works better for 2-dimensional multiprocessors. It is evident that these results are from the higher testing neighbor set cardinality. With $d = 2$ and $\pi(t) = 0.7$, the advantage of our approach is about 7%. In the 3-dimensional case, our method demonstrates the same probability $P(t)$ values as does the inter-unit test. For $d \geq 4$, our method becomes slightly worse than the inter-unit test at $\pi(t) \geq 0.7$ and loses at $\pi(t) \approx 0.6$, for which the probability is actually a bit too low for practical cases.

7.2 Connection Complexity Evaluation

Excessive connectivity is the main drawback of known mutual inter-unit test approaches. To deploy an inter-unit test environment, each processor node needs many external connections (input and output terminals) to communicate to its peers while performing test routines. This connectivity depends highly on the multiprocessor dimension and test unit parameters and so, may become a serious concern when complex systems are being manufactured.

In what follows below, we compare the connectivity factor in our approach to those in the known mutual inter-unit test methods and demonstrate that our proposed method can drastically decrease multiprocessor connectivity. The connection complexity is formally defined as the required number of extra direct connections between a given processor and all its peers to perform test routines. Only extra test connections are considered. It is assumed that the "regular" connections required for inter-processor data exchange and control are the same whichever test method is used, but only external connections are under consideration. For example, the links between the processor core and STU are not taken into account, because they are internal.

According to Figs. 4 and 5, i^{th} NCU needs $\Omega_i = W_R + W_T + 4$ input/output terminals, where W_R and W_T are the widths of response packets and test signatures, respectively. Thus, taking into account the round-robin arbitration unit connections, the majority gate terminals and the backward test/response buses, the total number of extra input/output terminals of a processor may be calculated as:

$$\Omega = 2d\left(2\left(W_R + W_T + 4\right) + 1\right). \tag{12}$$

In the same fashion, the connection complexity of known mutual inter-unit test schemes can be evaluated. For the inter-unit test method presented in [41], the following formula will take place:

$$\Omega_0 = 2\left[\left(d\left(d-1\right)+1\right)\left(W_R + W_T + 1\right) + d\right]. \tag{13}$$

Formula (13) takes into account all extra test connections to the peers of a given processor.

To compare our approach to the inter-unit test method of [41], we calculate the relation $\xi = \Omega_0/\Omega$ for different values of the multiprocessor dimension d and fixed W_R and W_T (We assume that $W_R = W_T$ for simplicity reasons). In Figure 8, ξ *versus* d graphs are shown for $W_R + W_T \in \{32,64,128,256\}$. According to the graphs of Figure 8, our approach requires a few more extra connections in the 2-dimensional case. For $d > 2$, our method works better than that of [41]. For example, given a 4-dimensional multiprocessor, the inter-unit test method would require 55% more extra connections than does our approach. Thus, with our proposed test method, higher dimension fault-tolerant multiprocessors would be significantly easier to implement because of the lower extra connectivity.

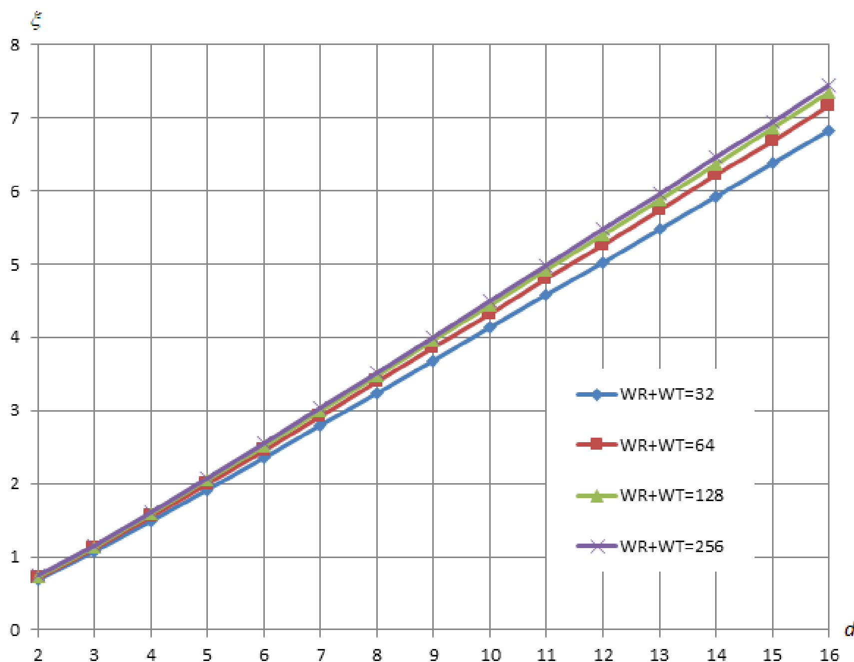

Figure 8. ξ *versus* d graphs for fixed $W_R + W_T \in \{32,64,128,256\}$.

8. CONCLUSION

In this paper, we proposed a new approach of a mutual inter-unit test with round-robin collision resolution to improve the testability of mesh-connected multiprocessors by increasing the probability of successful fault detection as compared with simple distributed self-checking. Compared with other mutual inter-unit test methods, such as [41] and [42], our approach automatically resolves the collision problem when two or more neighboring processors are about to start checking the same peer during overlapping time windows. Our method can be applicable to multiprocessors of arbitrary dimensions, with 2-dimensional ones having the maximum effectiveness, which matches the technological limitations of modern VLSI multiprocessors. For future scaling, our approach must allow drastic reduction of the multiprocessor connectivity with respect to known mutual inter-unit test methods. For example, in a 4-dimensional system, we need 55% less extra connections with our approach, while in a 5-dimensional case, we reduce extra connectivity by over 90%. The new mutual inter-unit test technique allows for the online hardware-level testing of all processor nodes across the mesh in parallel, thereby significantly contributing to the performance of the test environment.

REFERENCES

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, "Digital Systems Testing and Testable Design," IEEE Press, Piscataway, NJ, 1994.

[2] M.K. Aguilera, W. Chen and S. Toueg, "Failure Detection and Consensus in the Crash-Recovery Model," Distributed Computing, vol. 13, no. 2, pp. 99–125, 2000.

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

[4] L. Benini and G. De Micheli, "Networks on Chips: A Paradigm," IEEE Transactions on Computers,

vol. 35, no. 1, pp. 70–78, 2002.

[5] 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.

[6] R. Bianchini and R. Buskens, "Implementation of On-Line Distributed System-Level Diagnosis Theory," IEEE Transactions on Computers, vol. 41, pp. 616–626, May 1992.

[7] T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," ACM Computing Surveys, vol. 38, no. 1. pp. 1–51, 2006.

[8] D. Blough and H. Brown, "The Broadcast Comparison Model for On-Line Fault Diagnosis in Multicomputer Systems: Theory and Implementation," IEEE Transactions on Computers, vol. 48, pp. 470–493, May 1999.

[9] B. Ciciani, Ed., Manufacturing Yield Evaluation of VLSI/WSI Systems, Los Alamitos, CA: IEEE Computer Society Press, 1998.

[10] S. R. Das, "Self-testing of Cores-based Embedded Systems with Built-in Hardware," IEE Proceedings–Circuits, Devices and Systems, vol. 152, no. 5, pp. 539–546, Oct. 2005.

[11] E. P. Duarte Jr. and T. Nanya, "A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm," IEEE Transactions on Computers, vol. 47, pp. 34–45, Jan. 1998.

[12] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw and D. Sylvester, "Vicis: A Reliable Network for Unreliable Silicon," Proc. of the 46th DAC, pp. 812–817, Jul. 2009.

[13] S. Furber, "Living with Failure: Lessons from Nature?," Proc. of the 11th IEEE ETS, pp. 4–8, May 2006.

[14] T. Horita and I. Takanami, "Fault-tolerant Processor Arrays based on the 1.5-track Switches with Flexible Spare Distributions," IEEE Transactions on Computers, vol. 49, no. 6, pp. 542–552, June 2000.

[15] S. Y. Hsieh and C. Y. Kao, "The Conditional Diagnosability of k-Ary n-Cubes under the Comparison Diagnosis Model," IEEE Transactions on Computers, vol. 62, no. 4, pp. 839 – 843, April 2013.

[16] L. M. Huisman, "Diagnosing Arbitrary Defects in Logic Designs Using Single Location at a Time (SLAT)," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 23, no. 1, pp. 91–101, 2004.

[17] 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.

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

[19] 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.

[20] A. Kohler and M. Radetzki, "Fault-tolerant Architecture and Deflection Routing for Degradable NoC Switches," Proc. of the 3rd ACM/IEEE Int. Symp. NoC, pp. 22–31, May 2009.

[21] 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 IOLTS, pp. 75–80, Jun. 2009.

[22] A. Krstic, W. C. Lai, K. T. Cheng, L. Chen and S. Dey, "Embedded Software-based Self-test for Programmable Core-based Designs," IEEE Design and Test of Computers, vol. 19, no. 4, pp. 18–27, July/Aug. 2002.

[23] J. C. M. Li and E. J. McCluskey, "Diagnosis of Resistive-Open and Stuck-Open Defects in Digital CMOS Ics," Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 11, pp. 1748–1759, 2005.

[24] L. Lin, S. Zhou, L. Xu and D. Wang, "The Extra Connectivity and Conditional Diagnosability of Alternating Group Networks," IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 8, pp. 2352–2362, Aug. 2015.

[25] 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," Proc. Int. Symp. VLSI-DAT, pp. 72–75, Apr. 2009.

[26] P. Maestrini and P. Santi, "Self-diagnosis of Processor Arrays Using a Comparison Model," Proc. Of the 14th Symp. on Reliable Distributed Systems, pp. 218–228, 1995.

[27] J. Mekkoth, M. Krishna, J. Qian, W. Hsu, C.-H. Chen, Y. S. Chen, N. Tamarapalli, W. T. Cheng, J. Tofte and M. Keim, "Yield Learning with Layout-Aware Advanced Scan Diagnosis," Proc. of the International Symposium for Testing and Failure Analysis, pp. 412–418, 2006.

[28] M. Psarakis, D. Gizopoulos, E. Sanchez and M. Sonza Reorda, "Microprocessor Software-based Self-testing," IEEE Design and Test of Computers, vol. 27, no. 3, pp. 4–19, May/June 2010.

[29] J. Raik and V. Govind, "Low-area Boundary BIST Architecture for Meshlike Network-on-Chip," Proc. of the 15th IEEE Int'l Symp. DDECS, pp. 95–100, Apr. 2012.

[30] 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, pp. 776–792, 2004.

[31] S. Rangarajan, A. T. Dahbura and E. Ziegler, "A Distributed System-Level Diagnosis Algorithm for Arbitrary Network Topologies," IEEE Transactions on Computers, vol. 44, pp. 312–334, Feb. 1995.

[32] S. Rangarajan and D. Fussell, "Diagnosing Arbitrarily Connected Parallel Computers with High Probability," IEEE Transactions on Computers, vol. 41, pp. 606–615, May 1992.

[33] A. Sengupta and A. T. Dahbura, "On Self-Diagnosable Multiprocessor Systems: Diagnosis by the Comparison Approach," IEEE Transactions on Computers, vol. 41, pp. 1386–1396, Nov. 1992.

[34] M. Sharma, C. Schuermyer and B. Benware, "Determination of Dominant-Yield-Loss Mechanism with Volume Diagnosis," Proc. of IEEE Design & Test of Computers, vol. 27, no. 3, pp. 54–61, 2010.

[35] C. Stroud, J. Sunwoo, S. Garimella and J. Harris, "Built-in Self-test for System-on-Chip: A Case Study," Proc. of the Int'l Test Conf., pp. 837–846, 2004.

[36] W. C. Tam, O. Poku and R. D. Blanton, "Systematic Defect Identification through Layout Snippet Clustering," Proc. of the IEEE International Test Conference, pp.1, 2010.

[37] H. Tang, S. Manish, J. Rajski, M. Keim and B. Benware, "Analyzing Volume Diagnosis Results with Statistical Learning for Yield Improvement," Proc. of the European Test Symp., pp. 145–150, 2007.

[38] Z. Wang, M. Marek-Sadowska, K. H. Tsai and J. Rajski, "Analysis and Methodology for Multiple-Fault Diagnosis," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 3, pp. 558–575, 2006.

[39] L. Zhang, "Fault-Tolerant Meshes with Small Degree," IEEE Transactions on Computers, vol. 51, no. 5, pp. 553–560, May 2002.

[40] 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.

[41] J. 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.

[42] J. 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:

في هذه الورقة، يتم اقتراح طريقة جديدة للفحص المتبادل بين الوحدات في المعالجات الدقيقة متعددة الأبعاد في نطاق تقنية التكامل واسع النطاق جداً، وذلك لضمان أنّ أي معالج يجري فحصه فقط من قبل العُقدة المجاورة له دون غيرها في الوقت ذاته وأنّه ليست هناك حاجة لبذل عناية خاصة لاختيار اللحظات التي يجب أن تبدأ فيها عمليات الفحص؛ بمعنى أن الطريقة المقترحة تحول دون وقوع تعارض في فحص الوحدات. وقد تم بناء خوارزمية خاصة لهذا الغرض.

وقد أثبتت الطريقة المقترحة أنها تنطوي على تحسين لفحص المعالجات الدقيقة المتصلة في هيئة شبكة؛ من خلال زيادة الاحتمالية للكشف الناجح عن الأخطاء مقارنة بطريقة الفحص الذاتي الموزعة. من جانب آخر، تقلل الطريقة المقترحة بشكل حادّ من الوصلات اللازمة في المعالج الدقيق إذا قورنت بالطرق الأخرى المعروفة للفحص المتبادل بين الوحدات، ومن ثم فهي تسهل من عملية تصنيع المعالجات الدقيقة. فعلى سبيل المثال، في نظام رباعي الأبعاد، تقود الطريقة المقترحة الى تخفيض الوصلات الإضافية اللازمة بنسبة 55%.

This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).