On-Line Dependability Enhancement of Multiprocessor SoCs by Resource Management

T.D. Ter Braak [1], S.T. Burgess [2], H. Hurskainen [2], H.G. Kerkhoff [1], B. Vermeulen [3], X. Zhang [1]

[1] University of Twente, CTIT, Enschede, the Netherlands

[2] Tampere University of Technology Tampere, Finland

[3] NXP Semiconductors, Eindhoven, the Netherlands

***Abstract*—**

This paper describes a new approach towards dependable design of homogeneous multi-processor SoCs in an example satellite-navigation application. First, the NoC dependability is functionally verified via embedded software. Then the Xentium processor tiles are periodically verified via on-line self-testing techniques, by using a new IIP Dependability Manager. Based on the Dependability Manager results, faulty tiles are electronically excluded and replaced by fault-free spare tiles via on-line resource management. This integrated approach enables fast electronic fault detection/diagnosis and repair, and hence a high system availability. The dependability application runs in parallel with the actual application, resulting in a very dependable system. All parts have been verified by simulation.

I. INTRODUCTION

The advances in digital processors are often related to many-cores processors[1], using more (dual, quad etc.) than one processor intellectual property (IP) blocks in a processor system-on-chip (SoC). In order to cope with the huge data communication requirements between these cores, the cores are often interconnected by a Network-on-Chip (NoC). If the cores are identical, they are often referred to as 'tiles'. In the case of many embedded processors, very aggressive nanometer CMOS technology is required, which in turn has proven to be less reliable than more matured technologies [1].

On the other hand, these highly complex SoCs are increasingly used in safety-critical applications, like in the automotive and medical arena. This demands ultra dependable processor SoCs [2].

A. A dependable processing platform

Embedded systems tend to become more complex, and have to interact with an unpredictable environment in a dynamic fashion [3]. Applications, like e.g. Global Navigation Satellite Systems (GNNS), thus require a high degree of flexibility from the underlying processing platform. Such a platform is developed within the CRISP project [4]. Figure 1 shows a Multi-Processor SoC (MPSoC), internally in our project referred to as Reconfigurable Fabric Device (RFD), consisting of nine reconfigurable processing tiles, each being a Xentium processor core [5] and its associated memories, interconnected by a high performance NoC. The Xentium is a highly reconfigurable, power efficient, pipelined streaming-data processor core running at 200MHz [6]. Multiple of these MPSoCs can be connected to a General Purpose Device (GPD). In our case an ARM926-based SoC has been used. The GPD controls and configures the individual tiles.

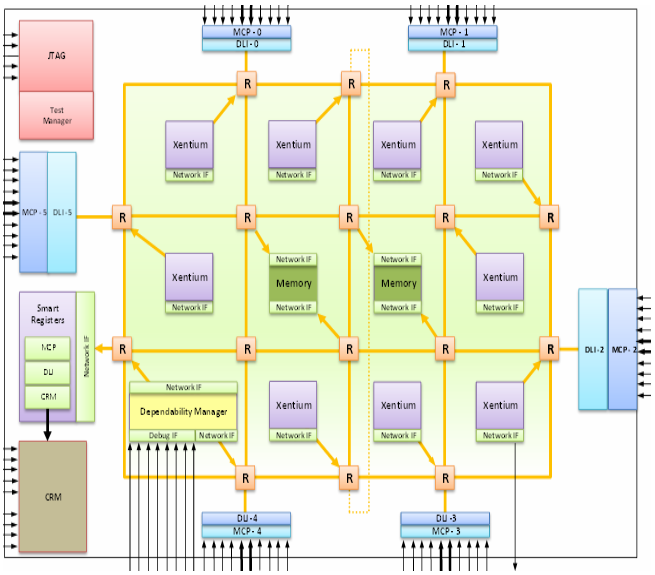

Figure 1 The MPSoC hardware platform

Efficient resource management is required to allow applications to benefit from the processing capabilities provided by the platform. The available resources should be allocated to the active applications based on their

[1] This research is conducted within the FP7 Cutting edge Reconfigurable ICs for Stream Processing (CRISP) project (ICT-215881) supported by the European commission.

computational needs. The resources should be partitioned in a way that should shield applications from interference caused by shared resources [7], but should also protect the applications from hardware failures in the platform. One obtains a more dependable processing platform if some resources are allowed to fail.

One approach to accomplish this goal is the incorporation of a dedicated Dependability Manager (DM) because of the many tasks to be performed. As the DM is not related to any functional task of the SoC, it is referred to as an Infrastructural IP (IIP). This type of integration allows for a higher degree of dependability compared to a software-only or manual approach.

B. *The attributes of dependability*

Dependability is the extent to which a system can be relied upon to perform its intended functions under defined operational and environmental conditions at a given instant of time or given interval [8]. Dependability is more than just reliability. Without treating dependability in depth, important attributes in our case are:

1) *Reliability*

Reliability of a system can be specified as a number after a certain life time, e.g. 0.9875 after 20 years; this indicates that 1.25 percent of the systems are expected to fail after 20 years.

2) *Maintainability*

In the case of an embedded IP block in an SoC, maintainability (and potential) repair in the classical sense is still not feasible. However it is possible in the sense that faults in an IP block can be detected on-chip, potentially be internally compensated, or if not feasible, the IP block could be isolated / bypassed and its function taken over by similar internal resources via electronic rerouting.

3) *(Un)availability*

Unavailability can be indicated as an amount of time (s) in a year or after a fault event; it is the time over lifetime during which the system is being repaired (and hence unavailable). Because of the electronic nature of this process in our approach, the unavailability can be very low, i.e. within milliseconds.

In the ideal case, the reliability of a fully dependable system is 1 (100%) during its specified lifetime (e.g. 20 years), and is never unavailable (0 s).

In Section II, state-of-the-art material on the concepts used to build the platform is provided. We illustrate the use of the resource manager described in Section III with the GNSS application from Section IV. The dependability test applications from Section V and VI are used to detect faulty resources in the platform. These concepts are illustrated with some experiments in Section VII.

Finally, conclusions are provided.

II. STATE-OF-THE-ART

A. *Network-on-Chip*

NoC is proving to be an attractive approach to MPSoC implementation on account of its inherent high bandwidth performance, versatility, and scalability, to name just a few of its merits [3]. This communication-centric paradigm has consequently created an urgent demand for new methodologies that focus on the testing of the network itself. Accordingly, there has been an increasing amount of attention being given to developing methods for the detection of faults in the NoC infrastructure which includes both routers [9] and their interconnecting links [10].

However, practical fault mitigating techniques that seek to exploit the use of spare NoC resources through run-time application mapping require a form of testing that not only can discover the presence but also the location of faults. The diagnosis of links has been investigated in the form of on-line testing [11], [12]. In reference [13], a non-scan based method for locating faulty links inside NoC switches is presented. However, the disadvantage of each of these, as well as most other proposals that we are aware of, is the fact that they either require design-for-test (DfT) structures or are only effective for limited fault cases, or assume a regular topology (e.g. 2D mesh). This makes them not ideally suited to handle the realistic dependability scenarios and/or for implementation within an arbitrary NoC system.

B. *Dependability of IPs*

Dependability of large scale MPSoCs is becoming a critical issue especially when they are used in mission-critical applications. Previous research studied methods to enhance the dependability of such an MPSoC. Ideas such as the Know-Good-Tile concept [14] and majority-voting among processing tiles [15] have been proposed to investigate the correctness of an MPSoC given that only identical processing tiles are used.

These methods rely on an infrastructural IP (IIP) [5] that is designed and integrated into the MPSoC to perform a periodic structural scan-based test on the processing tiles. Since all the tiles in the MPSoC are identical, the same test responses are expected from all tiles under test if they are fault-free. If at least three tiles are tested in parallel, one can identify a faulty tile by differences in its test responses from the other ones (assuming that only one tile becomes faulty at a time). Each tile is wrapped in a dependability wrapper that allows using a tile either in functional mode, or in test mode.

The multiple wrapped, processing tiles in the MPSoC are interconnected by a packaged-switched NoC. This NoC is also reused as a Test Access Mechanism (TAM) [16] to transport test stimuli and test responses when the processing tiles are being tested at application run-time. Since the total bandwidth of the NoC is shared between the application data and the dependability test data, dynamically pausing and resuming of the scan-based test is used to ensure sufficient NoC bandwidth is available for the running applications.

C. On-line resource management

Resource management for real-time embedded applications is conventionally only performed at design-time, as in [17], [18]. Hansson et al. [7] build composable platforms using virtual platforms to reason about applications independently. The flexibility of the application at run-time then depends on what configurations have been prepared beforehand. In this approach, it is difficult to deal with scenarios where many applications co-exist, or when faulty resources must be considered. Although much effort may be spent to explore the many possible configurations and trade-offs, some inherent limitations can only be overcome by doing online resource management.

When applications can be started and stopped by externally triggered events, the resource management problem is a non-clairvoyant scheduling problem, for which no optimal solutions can be guaranteed [19]. Therefore, most online resource management approaches use heuristics to solve this complex problem [20], [21]. In reference [19], the heuristics are explained used to build the resource manager described in this paper.

Synchronous Data Flow (SDF) is used to model and reason about applications; this, in order to provide quality of service guarantees. Many performance analysis techniques exist for SDF [22], [23].

III. On-line Resource Management

If a system is capable of monitoring the dependability of its components, proactive measures can be taken to enhance the dependability of the entire system. Conventional embedded systems may have to be decommissioned when critical components become faulty. The main reason for this failure is not only the lack of redundancy in resources, but also the inflexibility in controlling the system; the mapping of applications to hardware is often still done at design-time and can not be changed while the system is operational. Therefore, online resource management is required to benefit from dependability features built into high demanding systems. In this section, it is described how dependability services may interact with an online resource manager to provide user applications with fault-free resources. The predictable architecture of the CRISP project allows the resource manager to provide QoS guarantees to the applications.

A. Centralized resource management

In demanding systems, single points of failure should be avoided as much as possible. In most systems however, a single access point is used to start and stop applications, either initiated by the system itself or by a physical user. The platform contains (at least) one GPD, which is running a Linux kernel controlling the entire platform. The resource manager maintains a consistent platform state in shared memory. A larger platform may introduce several of such control points, and a distributed coherence protocol [24] may remove this single point of failure.

B. Everything is an application

The resource manager developed in the CRISP project works on the granularity of applications, while being unaware of the functionality of the application. Applications may interact with the resource manager through a limited set of function calls. Besides the obvious capabilities of requesting and releasing sets of resources, applications may provide information about the status of a resource. Using the same approach, applications may thus perform calculations on their acquired resources, or test those resources for correctness.

C. Application structure

An application is described as a task graph. The amount and type of resources required for operation is specified per task. One of these tasks is the entry point for the application; this task is started on the GPD at start-up of the application. It interacts with the resource manager to request the required resources for the other tasks in the application.

Applications thus explicitly have to request a set of resources, possibly containing memory sizes, processor time, I/O ports and interconnect facilities. Analogue to the explicit memory management required in programming languages such as ‘C’, this request for resources may fail under certain conditions. If a resource request is rejected, then the application may revert to a lower QoS level or try again later in time. The logic to handle a rejection is implemented in the control task as well. Upon success, the application continues under normal conditions.

Once the other tasks are configured and started, the control task may sleep, monitor the application, or execute part of the functionality of the application. Afterwards this task is responsible for stopping the application and releasing its resources.

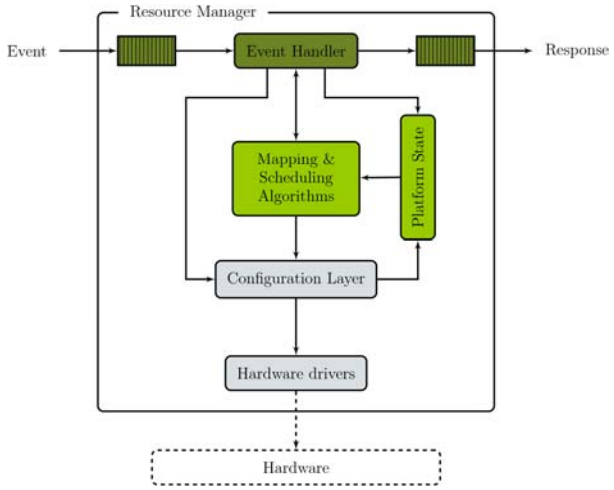

Figure 2 The interaction interfaces and control flow in the resource manager

D. *Event-response based interaction*

An event-response based interaction between applications and the resource manager allows for arbitration over possible multiple, near-simultaneously occurring events. Events are handled first on their priority, and second in order of arrival. The type of an event determines its priority; for example, releasing resources has priority over allocating new resources. Special applications, such as the dependability test software, may have a special, configurable priority that distinguishes them from normal user applications. This distinction is not required, but it allows for prioritizing between the intended functionality of the system and provided services such as the dependability enhancement.

The interaction flow in the software stack is illustrated in Figure 2 by three applications; a GNSS application, a NoC dependability application, and a core test application. For each application, the internal functionality and concepts are described first, after which the interaction with the resource manager is explained.

IV. THE GNSS APPLICATION

Satellite navigation applications of today range from cheap receivers embedded in mobile phones to expensive and highly accurate scientific ones. Indeed, there is an analogy between the scalability of our platform and GNSS receivers depending on the targeted application.

In the CRISP project, the GNSS application is specified and designed to support existing GPS (U.S. based NAVSTAR Global Positioning System) and future Galileo (European system) signals transmitted in the L1 frequency band centred at 1.575 GHz. The three main blocks in any GNSS receiver are identified to be i) a radio front end for analogue signal processing, ii) a digital baseband processing part and iii) navigation calculus to determine PVT (position, velocity and time) from measured pseudo ranges [25]. The implementation steps towards the CRISP GNSS application are explained in more details in [26], [27]. Figure 3 shows the tasks and communication streams of the GNSS application.

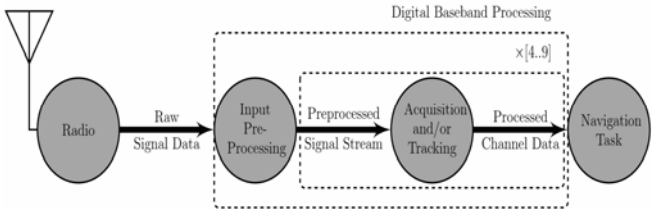

Figure 3 The structure of the GNSS application

The GNSS application is able to solve the PVT of the receiver if four or more satellites are tracked successfully [25]. Thus, the receiver application should always (after an initial acquisition stage) have four or more cores running a tracking process to enable navigation. If a satellite is lost, no satellite data is available until the re-acquisition procedure is again performed and concluded.

A. *Dynamic resource allocation*

After starting the GNSS application, the navigation task on the GPD raises an event of type ALLOCATE_RESOURCES together with a task graph that describes the communication structure and the required resources for operation. Based on the event type, the event handler forwards the request to the mapping and scheduling algorithms after an unpredictable, but finite delay. While consulting the platform state, a set of fault-free resources is allocated to the application, such that the application meets its QoS specification. Through the platform specific drivers, the configuration layer configures the newly allocated resources on the platform and starts the other tasks in the application on their assigned processors.

In the next two paragraphs, the testing of the NoC and Xentium cores in the MPSoC as used in the GNSS application will be discussed in detail.

V. THE NOC TESTING APPLICATION

The NoC dependability application is a GPD-hosted utility for diagnosing faults within the communication fabric. By using a SW-only approach, the need for specialized DfT structures within the network has been eliminated. This saves Silicon area and increases the ease with which the system may be deployed onto arbitrary platforms.

A. *NoC Fault Representation*

The domain for NoC testing is understood to be all inter-router connections (link components) as well as all of the internal intra-router routing paths (switch components); there are up to 20 such paths per 5 bi-directional port router used in our prototype MPSoC platform. We refer to either link or switch components as path components with the combined number modelled for our platform being 333.

An arbitrary path through the network can modelled as a series of interconnected path components, C_1 to C_n. The overall fault status of a path, P(C), may then be expressed as the logical AND of each constituent path component as shown by Equation 1, where each C may take on the value of '1' for good or '0' for faulty.

$$P(C) = C_1 \bullet C_2 \bullet C_3 \bullet C_4 \bullet .... \bullet C_n \qquad (1)$$

A faulty path component is assumed to permanently malfunction by either corrupting the packet payload or causing the packet itself to be miss-routed and/or dropped.

B. *Test Concept*

Although there are $2^{(n-1)}$ possible fault configurations (where n is the number of components along some arbitrary route) satisfying the P(C) = 0 condition, the fact is exploited that there is only a single solution that satisfies P(C) = 1 which occurs if all C = 1. The practical application of this is that if a packet can be successfully routed over a defined path it will be known that the specific set of path components comprising that route are good. The primary task then becomes one of attempting to successfully route test packets

through each good path component resulting in a diagnosis rendered on of a faulty until proven good basis.

In the case that the GPD and MPSoC are on separate dies, the first phase of testing begins by checking the connectivity between the two devices. This is accomplished by the GPD injecting one or more packets (as required) into the MPSoC along a looping path that terminates back at the GPD as illustrated by route wr0 in Figure 4. The exact route that these packets follow is based on the principle of a random self-avoiding walk, which has been constrained by:

- Explicitly defined start and end points
- The physical boundary of the network topology
- A maximum specified number of router hops.

A route so devised provides an effective means for the discovery of good paths through a network containing arbitrarily distributed faults. Successful reception by the GPD of a previously injected packet verifies the link and enables the testing of the MPSoC communication infrastructure to proceed.

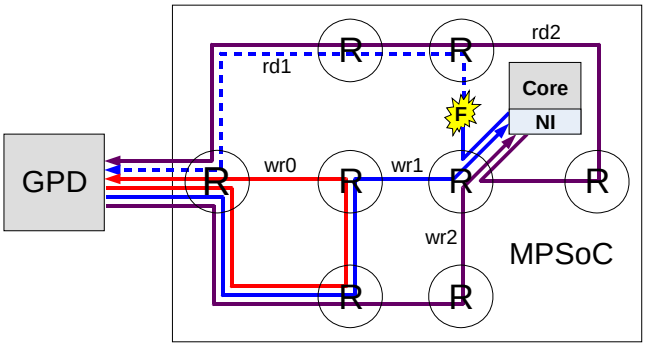

Figure 4 Routing of test packets through the MPSoC

This next testing phase follows a similar approach as previously employed except that here the test packets are sent through the network along paths to and from designated cores in the form of write/read operations. Ports on the router connected to the core are systematically selected so as to exercise each core related switch path. Switch paths for all other input-output combinations (i.e. non-core related) are exercised indirectly as part of the write/read operations performed on neighbouring cores. Two example iterations from the many that are possible for this procedure are also illustrated in Figure 4. In the first case, the GPD writes data to the core by way of route wr1 followed by an attempted read which fails because of a fault along route rd1. During a subsequent operation data is written to the core by way of route wr2 which is then successfully read back over route rd2. This verifies all path components along wr2/rd2 which is also seen to include certain sections of the previously unverified rd1.

Upon complete execution of this algorithm, the expected result is that all good network path components will have been identified leaving what is left to be deemed the faulty elements.

A final post processing procedure is performed whereby the resulting link and switch path based diagnosis result is transformed into a link-only representation for compatibility with the real-time mapping resource manager.

VI. THE XENTIUM CORE TESTING APPLICATION

The core testing application is an essential part of the dependability enhancement approach for the MPSoC. The application consists of dependability software running on the GPD and extra hardware, such as the infrastructural IP (DM) and the dependability wrappers around the Xentium tiles.

The IIP shown in Figure 5 consists of a test pattern generator (TPG), a test responses evaluator (TRE) and a finite state machine (FSM). The FSM controls the IIP and communicates with the dependability software running on the GPD. Deterministic test patterns for the Xentium tile (with 32 parallel scan-chains) have been generated at design-time using a commercially-available ATPG tool [5]. The TPG in the IIP reproduces these deterministic test patterns by using a linear-feedback shift register (LFSR) combined with a reseeding technique. Stuck-at fault coverage of 90% for the Xentium tile processor can be reached using the scan-based test of the TPG. It is stressed that this fault coverage is sufficient for the current dependability test in our *prototype* MPSoC. Increased fault coverage in the final version can be obtained at the cost of an increase in Silicon area for the TPG and Xentium. A phase-shifter (PS) packs the test stimuli into 32-bit words to suit the 32-bit wide NoC. Each bit of such a 32-bit word fills one scan flip-flop in each of the 32 scan chains. At this point, the silicon area of the IIP is below 1% of the total area of the MPSoC.

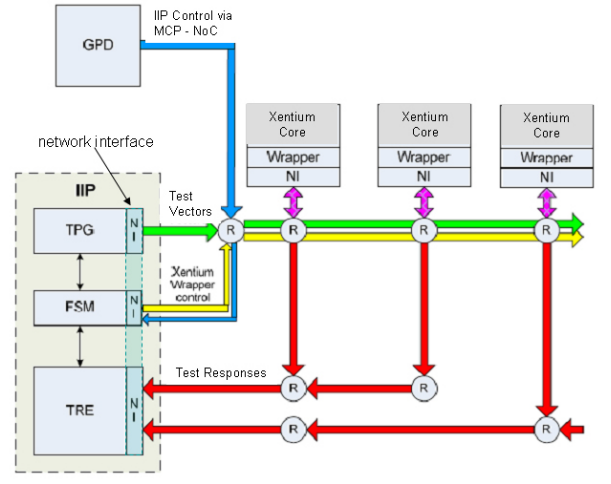

Figure 5 The core test application environment

The Xentium Tile Wrapper (XTW) can switch between functional mode and dependability test mode via commands issued over the NoC. In functional mode, the XTW transparently delivers the data from the NoC to the functional inputs of the tile/core, and passes the data from the functional outputs to the NoC. In dependability test mode, it delivers the NoC data to the test inputs (scan-chain inputs and primary inputs) of the tile. A similar operation is performed at the output of the tile to capture the data from the scan-chain outputs and primary outputs of the tile, and deliver this data to the NoC.

The core dependability application on the GPD starts the test activities. It requests sufficient resources to use the IIP together with two or three Xentium cores. The resource manager also allocates and configures the required communication routes between them. The IIP has no knowledge of which cores are being tested; this is completely determined by the communication routes. A possible test scenario is shown in Figure 6. Writing the control register in the FSM of the DM starts the test sequence.

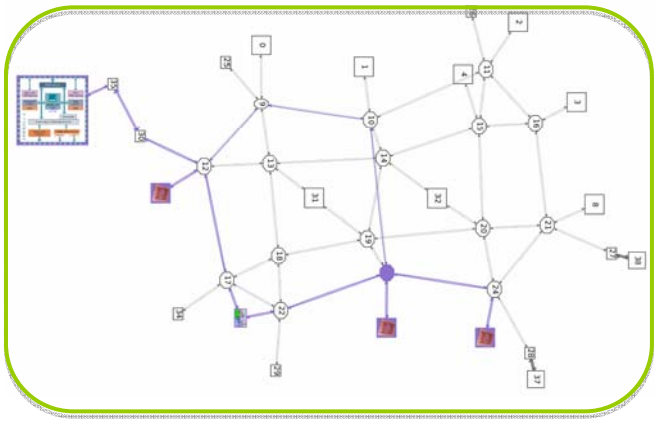

Figure 6 A dependability test scenario with 3 Xentium cores

The IIP will subsequently switch the wrappers of the chosen Xentium tiles to dependability test mode, preparing the Xentium tiles for receiving the test stimuli generated by the TPG of the IIP. The test stimuli are broadcasted via the NoC to the target Xentium tiles and the test responses are collected and compared in the TRE of the IIP-DM.

The test sequence is halted as soon as a difference is detected in the test responses from the Xentium tiles under test. The Xentium tile that generated a different test response from the others is identified as the faulty tile. This information is encoded into a fault status report. This status report is read by the dependability software in the GPD. The faulty Xentium tile can then be isolated or removed from the usable resource table of the resource management software.

If no test-response differences are detected during the dependability test, the tested tiles are considered fault-free. They will stay in the usable resource table of the resource management software. This dependability test process is repeated until all Xentium tiles are tested. This way, the faulty tiles, if any, are identified and isolated from the system ensuring that the remaining processing tiles hardware are fault-free. Depending on the dependability requirements (i.e. acceptable mean system down time / unavailability) from the end user, the dependability test activities can be performed at a desired frequency.

VII. Experimental Results

A. *The NoC testing results*

Performance of the NoC test SW was investigated in conjunction with a C language network model for the MPSoC platform introduced in Figure 1. A varying number of randomly inserted path faults (link and switch types) were configured into the model after which a diagnostic simulation was performed. Figure 7 shows a representative set of results in relation to two key performance metrics - the number of equivalent link faults reported and number of test packets injected. Each corresponding value for a given fault level is to be interpreted as an independent iteration.

The utility of the method is manifest in the success with which the system can diagnose arbitrary multi-fault configurations. Note the fact that one-to-one correspondence between inserted and reported faults is generally not observed and is not of itself indicative of an incorrect diagnosis but rather a consequence of fault equivalency. As the number of faults increase from 0 to 10, the required number of injected test packets grows from a baseline value of about 1.6k, up to about 3.4k. This increase derives from the greater effort required to successfully route packets through an increasingly compromised network.

This “faulty until proven good” approach ensures 100% fault detection assuming that there exists a network path to or from every path component and the GPD. However, if multiple cores are connected to a single router, as with the Smart Registers and Dependability Manager of our prototype platform, the switch path that directly connects the two is found not to satisfy that criterion and is thus not testable. This means that of the 333 uniquely modelled path components for our platform, only 331 can in fact be exercised reducing test coverage to 99.4%.

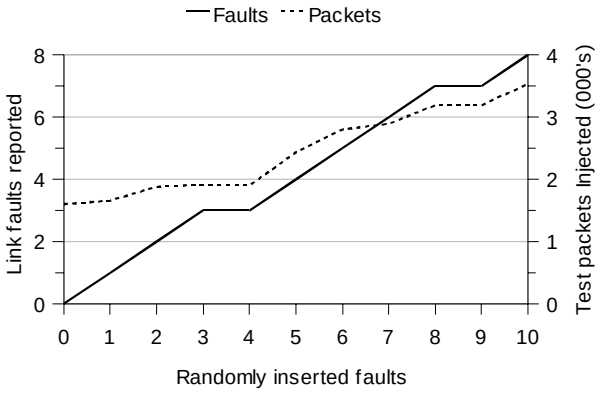

Figure 7 The results of NoC testing procedure

B. *The core testing results*

In order to verify the core testing application, the complete MPSoC platform has been designed and implemented in synthesizable VHDL. A complete simulation of the test stimuli generation and test response collection is shown in Figure 8.

After the dependability software in the GPD activates the dependability test application, the test stimuli data are generated, packed into multiple 32-bit data flits and multicast to the three target Xentium tiles via the NoC. The NoC data flits are disassembled at the network interface of the Xentium tile and the test stimuli data are loaded into the 32 parallel scan-chains of each tile. When the test responses are produced, they are collected again via the NoC and compared by the TRE of the IIP as shown in Figure 8.

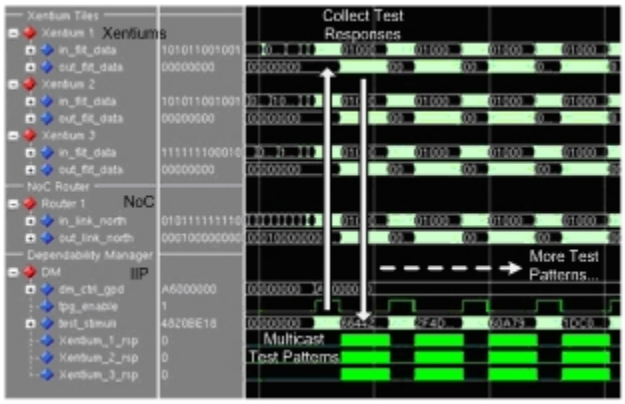

Figure 8 The results of core testing procedure

When the comparison is completed and no response difference is identified, more test stimuli will be generated from the IIP-TPG until a first difference is detected or the dependability test finishes.

C. *Resource management under hardware failures*

Figure 9 again shows the MPSoC, together with an active GNSS application. The radio used to receive GPS signals is indicated with the rightmost gray block. The antenna data is first transported to the GPD on the left, which forwards the pre-processed satellite data to the Xentium cores. Figure 9 shows an example scenario where a part of the chip is malfunctioning, indicated in red. The faults are isolated to a single Xentium core and the NoC router nearby. The GNSS application is still able to run with eight tracking processes (shown in purple).

All scenarios of a single fault in either a Xentium processor core or a network router were generated. For seven out of the thirty-nine generated faults, the GNSS application failed to run on the platform. These faults resulted in a failure of the application, because the faults occurred on the input or output chain of the chip. Both the GPD and the radio are single points of failure, together with their connection to the chip. The NoC within the chip has a high degree of connectivity and therefore has sufficient redundant communication resources if part of it fails. The GNSS application can work with a variable number of Xentium cores (albeit at least four); failure of one of them still allows the application to run.

VIII. Discussion and Conclusions

In this paper a new approach towards dependable design of homogeneous MPSoCs has been presented in an example GNNS application. First, the NoC dependability is functionally verified via embedded software. Then the Xentium processor tiles and associated embedded SRAMs are periodically verified via on-line self-testing techniques, by using a new IIP Dependability Manager. Faulty tiles are electronically bypassed and replaced by other Xentium resources via embedded resource manager software based on the DM results. This integrated approach enables fast electronic fault detection/diagnosis and repair, and hence a high MPSoC availability (unavailability: 10us). The dependability application runs in parallel with the actual GNNS application, resulting in a very dependable MPSoC (reliability: 0.9783, after 15 years). All parts have been verified by simulation. The MPSoC has been designed in UMC 90nm technology and will be soon available for dependability and functional evaluation.

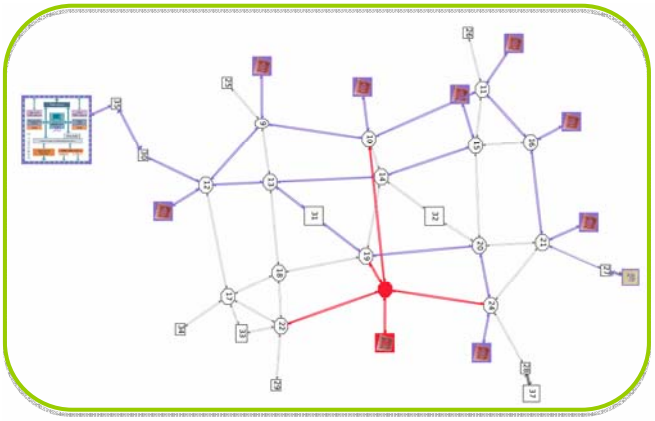

Figure 9 The GNSS application running (8 trackers) with resource failures in one of the Xentium cores and the attached NoC router

Acknowledgment

The research in this paper has been conducted within the FP7 Cutting edge Reconfigurable ICs for Stream Processing (CRISP) project (ICT-215881) supported by the European commission. The authors want to acknowledge the many contributions of all the other partners in the CRISP consortium, being Thales Netherlands, Atmel Automotive GmbH and Recore Systems.

References

[1] Y. Cao, P. Bose, and J. Tschanz, “Reliability challenges in Nano-CMOS Design”, IEEE Design & Test of Computers, pp. 6-7, 2009.

[2] S. Sakai, M. Goshima, and H. Irie, “Ultra Dependable Processor”, *IEICE Trans. Electronics*, vol. E91-c, no. 9, pp. 1386-1393, 2008.

[3] G. Buttazzo, “Research trends in real-time computing for embedded systems.” *SIGBED Rev.* 3, pp. 1-10, 2006. doi: http://doi.acm.org/10.1145/1164050.1164052

[4] http://www.crisp-project.eu

[5] H.G. Kerkhoff and X. Zhang, “Design of an Infrastructural IP Dependability Manager for a Dependable Reconfigurable Many-Core Processor,” *in Proc. DELTA 2010*, HCM City Vietnam, pp. 270-275, Jan. 2010.

[6] http://www.recoresystems.com

[7] A. Hansson, K. Goossens, M. Bekooij, J. Huisken, “CoMPSoC: A template for composable and predictable multi-processor system on chips.” *ACM Trans. Design. Autom. Electron. Syst.* vol. 14, pp. 1-24, Jan. 2009.

[8] A. Avizienis, J-C. Laprie, B. Randell and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing”, IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pp. 11-32, 2004.

[9] F. Gilabert, D. Ludovici, S. Medardoni, D. Bertozzi, L. Benini, G.N. Gaydadjiev, "Designing Regular Network-on-Chip Topologies under Technology, Architecture and Software Constraints," *Complex, Intelligent and Software Intensive Systems, 2009. (CISIS '09). International Conference on* , pp.681-687, March 2009.

[10] A.M. Amory, E. Briao, E. Cota, M. Lubaszewski, F.G. Moraes, "A scalable test strategy for network-on-chip routers," *Test Conference, 2005. Proceedings. (ITC 2005). IEEE International* , sess. 9, pp.-599, Nov. 2005.

[11] C. Grecu, P. Pande, A. Ivanov, R. Saleh, "BIST for network-on-chip interconnect infrastructures," *VLSI Test Symposium, (VTS06). Proceedings. 24th IEEE* , sess.6, pp. -35, May 2006.

[12] M. Herve, E. Cota, F.L. Kastensmidt, M. Lubaszewski, "Diagnosis of interconnect shorts in mesh NoCs," *Networks-on-Chip, 2009. (NoCS 2009). 3rd ACM/IEEE International Symposium on*, pp. 256-265, May 2009.

[13] J. Raik, V. Govind, R. Ubar, "Design-for-testability-based external test and diagnosis of mesh-like network-on-a-chips," *Computers & Digital Techniques, IET* , vol.3, no.5, pp. 476-486, September 2009.

[14] H.G. Kerkhoff, O. Kuiken, and X. Zhang, “Increasing SoC Dependability via Known Good Tile NoC Testing,” *IEEE Intern. Conf. on Dependable Systems and Networks (DSN08),* Anchorage USA, 2008.

[15] X. Zhang and H.G. Kerkhoff, “Design of a Highly Dependable Beamforming Chip,” *in Proc. Euromicro on Digital System Design (DSD09)*, pp. 729-735, Aug. 2009.

[16] X. Zhang, H.G. Kerkhoff and B. Vermeulen, “On-Chip Scan-Based Test Strategy for a Dependable Many-Core Processor Using a NoC as a Test Access Mechanism,” *in Proc. Euromicro on Digital System Design 2010 (DSD10), Lille, France, 2010 (to appear).*

[17] A. D. Pimentel, C. Erbas, and S. Polstra, “A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels”. *IEEE Trans. Comput.* 55, 2 (Feb. 2006), pp. 99-112, 2006. doi: http://dx.doi.org/10.1109/TC.2006.16

[18] C. Lee, S. Kim, and S. Ha, “A Systematic Design Space Exploration of MPSoC Based on Synchronous Data Flow Specification”. *J. Signal Process. Syst.* 58, 2 (Feb. 2010), pp.193-213, 2010.

doi: http://dx.doi.org/10.1007/s11265-009-0351-6

[19] T.D. ter Braak, P.K.F. Hölzenspies, J. Kuper, J.L. Hurink, and G.J.M. Smit, “Run-time Spatial Resource Management for Real-Time Applications on Heterogeneous MPSoCs,” *in Proc. Of the Conf. on Design, Automation and Test in Europe (DATE 2010)*, Dresden. pp. 357-362, Mar. 2010. EDAA. ISBN 978-3-9810801-6-2

[20] C. Chou, and R. Marculescu, “User-aware dynamic task allocation in networks-on-chip”. In *Proceedings of the Conference on Design, Automation and Test in Europe* (DATE '08), Munich, Germany, ACM, New York, NY, pp.1232-1237, 2008.

DOI= http://doi.acm.org/10.1145/1403375.1403675

[21] E. Carvalho, C. Marcon, N. Calazans, and F. Moraes, “Evaluation of static and dynamic task mapping algorithms in NoC-based MPSoCs”, *Proceedings of the 11th international Conference on System-on-Chip*, Tampere, Finland, J. Nurmi, J. Takala, and O. Vainio, Eds. IEEE Press, Piscataway, NJ, pp. 87-90, October 2009.

[22] S. Stuijk, T. Basten, M.C. Geilen, and H. Corporaal, “Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs”, *Proceedings of the 44th Annual Design Automation Conference (DAC '07)* (San Diego, California, June 2007). ACM, New York, NY, pp. 777-782, 2007.

doi: http://doi.acm.org/10.1145/1278480.1278674

[23] A. H. Ghamarian , M. C. W. Geilen , S. Stuijk , T. Basten , B. D. Theelen , M. R. Mousavi , A. J. M. Moonen , M. J. G. Bekooij, “Throughput Analysis of Synchronous Data Flow Graphs”, *Proc. of the Sixth International Conf. on Application of Concurrency to System Design*, pp. 25-36, June 2006.

[24] O.E. Theel, and B.D. Fleisch, “A Dynamic Coherence Protocol for Distributed Shared Memory Enforcing High Data Availability at Low Costs”, *IEEE Trans. Parallel Distrib. Syst.* 7, 9, pp. 915-930, 1996. doi:http://dx.doi.org/10.1109/71.536936

[25] M. Braasch and A. J. Van Dierendonck, “GPS Receiver Architectures and Measurements”, Proceedings of the IEEE, vol. 87, no. 1, pp. 48-64, Jan. 1999.

[26] H. Hurskainen, J. Raasakka, T. Ahonen, and J. Nurmi, “Multicore Software-Defined Radio Architecture for GNSS Receiver Signal Processing, ” EURASIP Journal on Embedded Systems, vol. 2009, Article ID 543720, 10 pages, 2009. doi:10.1155/2009/543720

[27] J. Raasakka, H. Hurskainen, T. Paakki, and J. Nurmi. "Modeling Multi-Core Software GNSS Receiver with Real Time SW Receiver" in Proceedings of ION GNSS 2009. Savannah, Georgia, 2009.