2011 17th IEEE Pacific Rim International Symposium on Dependable Computing

A Dependability Solution for Homogeneous MPSoCs

Xiao Zhang, Hans G. Kerkhoff
Testable Design and Test of Integrated Systems Group,
Centre of Telecommunication and Information Technology (CTIT), University of Twente,
Enschede, the Netherlands
x.zhang@utwente.nl and h.g.kerkhoff@utwente.nl

***Abstract*—Nowadays highly dependable electronic devices are demanded by many safety-critical applications. Dependability attributes such as reliability and availability/maintainability of a many-processor system-on-chip (MPSoC) should already be examined at the design phase. Design for dependability approaches such as using available fault-free processor-cores and introducing a dependability manager infrastructural IP for self-test and evaluation can greatly enhance the dependability of an MPSoC. This is further supported by subsequent software-based repair. Design choices such as test fault coverage, test and repair time are examined to optimize the dependability attributes. Utilizing existing infrastructures like a network-on-chip (NoC) and tile-wrappers are needed to ensure a test can be performed at application run-time. An example design following the proposed design for dependability approach is shown. The MPSoC has been processed and measurement results have validated the proposed dependability approach.**

Keywords- MPSoC; dependability; fault-tolerance; self-repair; reliability; availability; embedded instruments; NoC (TAM); self-test

I. INTRODUCTION

The increasing complexity and power issues of digital systems have led to the need of integrating many identical (homogeneous) processor cores onto a single system-on-chip (Multi-Processor SoC, MPSoC). Recent research has forecasted that MPSoCs with more than a thousand processor cores will be commercially available in the near future [1]. As a result of the rapidly growing transistor density on the chip, ensuring high system dependability has become a real challenge [2].

A system is considered as a "dependable system" when it can correctly deliver the expected services under given conditions [3]. Manufacturing test is the conventional method to check the correctness of chips. Faulty cores in the system can be detected by a thorough manufacturing test and be isolated from the system to provide a "functionally correct" chip. The capability to tolerate a few faulty cores in an MPSoC has been adopted in industry [4]. Recent research shows that a significant effective yield increase and overall cost reduction can be achieved by adding a few dynamically reconfigurable spare cores into an MPSoC [5], [6]. The term 'spare' should be interpreted here as cores which may also carry out few or non-crucial tasks. Using these resources for repair, could have slight consequences for the system Quality of Service (QoS).

During the life time of an IC, permanent faults such as time dependent dielectric breakdown (TDDB) and electro-migration can occur as a result of chip degradation and material wear-out [7]. Permanent faults in processor cores can cause a system failure in the field. While using spare cores can help to tolerate manufacturing defects, the dependability of an MPSoC becomes a real challenge without a proper mechanism to detect a faulty processor core in its life cycle.

As it is usually infeasible, if not impossible, to perform a manufacturing test with automatic test equipment (ATE) in the field, chips are often equipped with some self-test features. For example, software based self-test method (SBST), which uses the on-chip processor to execute test programs for core "mutual diagnosis" [8], [9]. The SBST method reuses the on-chip resources and carries out the test via software and hence it saves silicon area. A disadvantage of this method is that functional tests instead of structural tests are performed, which implies a lower fault coverage. This is not desirable as it has been concluded that high fault coverage of in-field testing and advanced fault-recovery mechanisms are essential for a dependable MPSoC [5]. In our paper, we propose to use structural test as our dependability test approach.

Memory built-in self-test (BIST) has been commonly adopted to check the correctness of on-chip memories. As for the logic part of an SoC, methods such as Logic BIST (LBIST) or Deterministic Logic BIST (DBIST) can generate pseudo-random or deterministic test vectors on the chip to test the target circuits then compact the test responses using Multiple-Input Signature Registers (MISR) and compare them with reference signatures [10], [11]. The disadvantage of using a MISR is that test response information is lost in the process of compaction thus the effective fault coverage is decreased. In this paper we exploit the fact that in a homogeneous MPSoC identical processor cores will generate identical test responses if supplied with the same test stimuli. By comparing the test results from several processor cores we can identify the faulty core by majority-voting. This method also eliminate the silicon real estate cost for MISR circuits and signature storage.

In previous research, the real-time characteristic of an on-chip self-test has often been neglected. Usually safety-critical applications allow only a very short system down time; this gives a real-time requirement for fault detection and system recovery. In this paper, we propose a generalized and scalable dependability solution for MPSoC with identical processor cores (homogeneous MPSoC). First, spare cores in the MPSoC should be available as potential resource to meet the reliability requirements. Second, an optimized infrastructural IP is included into the MPSoC to function as a dependability manager (DM) for core-level self-test. The DM performs

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

978-0-7695-4590-5/11 $26.00 © 2011 IEEE
DOI 10.1109/PRDC.2011.16

IEEE computer society

periodic *structural* tests on available cores. These cores can be made available, by shifting their tasks to other cores under the control of software. If the cores being tested are fault-free, processing load can be shifted to them to free other cores for test. If one core is tested to be faulty, system resource management software can exclude it from usable resource list of the system and application will be reconfigured to avoid the faulty core. This effectively makes the system *fault-tolerant.*

An example MPSoC platform using the Xentium tile processors from Recore Systems [12] has been developed with the dependability approach proposed in this paper included. We have attempted to balance parameters such as fault coverage, dependability self-test time and the cost of extra silicon area to achieve a highly dependable MPSoC. The remainder of this paper is organized as follows. In Section II, dependability concepts of an MPSoC are introduced. The impact of the dependability self-test on dependability attributes is analysed in Section III. Section IV presents an example MPSoC design, which has been developed for streaming data applications, such as beamforming, following the proposed dependability approach. The example MPSoC has been processed in 90nm UMC CMOS technology. Measurement results and discussions of several dependability test scenarios are given in Section V. Section VI concludes the paper.

II. DEPENDABLE MPSOC CONCEPT

The definitions of dependability are many and subject to the field of research. For modern computing systems, dependability can be generally accepted as the ability of a system to deliver expected services under given conditions [13]. Some important dependability attributes such as reliability, availability and maintainability will be discussed. In this section, the assumption is made that processor cores are the only parts that will fail in an MPSoC to simplify the discussion. Other parts that could fail such as on-chip memory or NoC are covered in Section IV.

A. MPSoC Reliability

An important attribute in dependability is reliability. A common technique to make a reliable system is to add redundant resources in order to tolerate certain amount of faulty components. Typical architectures of MPSoCs with some redundancy (spare cores) are a massive redundant system, a gracefully degrading system or a standby redundant system [14].

A massive redundant system uses technique such as triple-modular redundancy (TMR) to improve system reliability. The same computing task is separately executed on several identical processor cores and the output of each core is compared to vote for the faulty core. Major disadvantage of this method is the amount of required redundant resources.

A gracefully degrading system uses all the fault-free cores in the chip to execute tasks. If a faulty core is detected, the system software tries to reconfigure the task to run it only on the fault-free cores ensuring system reliability at the cost of degraded performance. The system is considered as dependable until its performance drops to an unacceptable level.

In a standby redundant system, the fault-free cores are either in operational or in standby modes. The application task is executed on all the operational cores. If an operational core is detected as faulty, it is replaced by a fault-free standby core to maintain the same computing performance. The system performance will not degrade until all the standby cores have been used.

With regard to the reliability attribute, an MPSoC organized as gracefully degrading system or standby redundant system can be generally modelled as a load sharing K-out-of-N: G system [15]. A K-out-of-N: G system has in total N processor cores and the system can correctly perform its required function (a Good system) if at least K cores are working properly ($K \le N$). The processing load of the system is distributed among the working cores. The system will not be considered as failing until *N-K+1* cores have become faulty.

The reliability of a system is often described as a probability *R(t)* that the system can provide correct service over a certain period of time. For example, *R(t) = 0.9* over ten year means that the probability that the system will function correctly after ten years is 90%.

Before the MPSoC reliability can be calculated, the reliability of an individual core has to be studied first. For a single core in the MPSoC, if for simplicity a constant failure rate λ is assumed, its reliability can be computed with exponential distribution as $R_t(t) = e^{-\lambda t}$.

In the case of a load sharing K-out-of-N: G system, we assume all the cores in the MPSoC are identical and have the same independent reliability distribution $R_t(t)$, which means $R_t(t)$ is i.i.d. (independent and identically distributed). The system reliability $R_{system}(t)$ is equal to the probability that the number of working cores is greater than or equal to K [16]:

$$R_{system}(t) = \sum_{i=K}^{N} \binom{N}{i} R_t^i(t)\left[1 - R_t(t)\right]^{N-i}. \qquad (1)$$

A special case is when *K=N*, this means the system requires all the cores in the chip to perform a certain task. This results in no spare processor resources in the MPSoC and hence the system fails when any of the working processors fail. In this case, the system reliability $R_{system}(t)$ degrades to:

$$R_{system,K=N}(t) = \prod_{i=1}^{N} R_i(t) = R_t^N(t). \qquad (2)$$

For example, suppose six Xentium processor cores (K=6) are strictly required to perform a computing task for a beamforming application. Assuming $R_t(t) = 0.9$ over 10 years for each core, the system hardware reliability of a 6-core MPSoC after 10 years is $R_{system}(t) = 0.9^6 \approx 0.53$, assuming other parts are fault-free. If the MPSoC is built with 3 cores as spare (9 cores in total, N=9), the system reliability can be calculated using equation (1) as

$$R_{system}(t) = \sum_{i=6}^{9} \binom{9}{i} \times 0.9^{i} \times (1-0.9)^{9-i} \approx 0.99$$

It is obvious that adding a few more spare cores almost doubled the system reliability over a 10 years period.

It should be noted that, the simplified reliability model (1) and (2) all assume an identical constant failure rate for all cores in the MPSoC. In reality, this will not be the case. In addition, due to the aging of the chip, the failure rate is expected to increase instead of being constant. Different cores may also have different failure rates as a result of varying temperature and voltage over the chip (environmental as well as operational conditions). A more precise model for computing MPSoC reliability has been studied in another paper [17].

The above global reliability calculations all assume that, once a faulty core appears in the system, it will be detected. In practice, the real system reliability is also affected by the fault coverage of the selected in-field test method. This will be discussed in Section III.

B. MPSoC Availability & Maintainability

Another two attributes in dependability are availability and maintainability. Availability refers to the readiness of a system to correctly perform required functions. If a fault appears in the system, it is first required to be detected and subsequently the system has to be alerted. In the conventional case, the system will be taken offline, repaired and brought online again after fault elimination. The time spent for fault detection and repair is defined as system down time because the system is not performing the required function during this period of time. The system Mean Down Time (MDT) is an important measure of the availability attribute. Modern systems usually have an availability requirement of more than 99.9% and the MDT always needs to be minimized. Figure 1. summarizes the important terms regarding the system availability time.

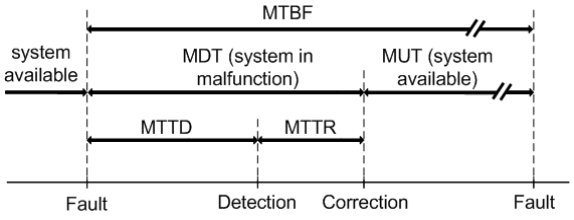

Figure 1. System availability chart. MTBF represents the Mean Time Between Failures; MDT is the Mean system Downtime; MUT is the Mean system Uptime; MTTD is the Mean Time To Detect a fault and MTTR is the Mean Time To Repair the system.

For an MPSoC, it is usually very difficult to physically repair a faulty core in the chip package in field. In that sense, there is no maintainability at core level. At system level, an MPSoC can be considered as a repairable system if the faulty cores can be detected and isolated and the computing tasks can be remapped to fault-free processor cores.

Upon the failure of a core, the system enters a malfunction mode. The mean time spent on faulty core detection (MTTD), isolation and application remapping (MTTR) should not exceed the mean system downtime allowed by the user. Therefore, it is required that:

$$MTTD + MTTR < MDT_{allowed} \quad (3)$$

The time required to isolate the faulty core and remap the application to fault-free cores depends on aspects such as core architecture [12] and the used remapping software algorithm [18]. This is not discussed in detail within the scope of this paper. The MTTD is explained in Section III and related to the Dependability Manager design choices.

III. DESIGN CONSIDERATIONS OF A DEPENDABLE MPSOC

As analysed in Section II, one needs to include some spare cores in an MPSoC to increase system reliability, to adopt a core-level self-test and evaluation mechanism to achieve maintainability and to minimize the time spent on dependability test and system reconfiguration for repair to maximize system availability. Hence, the dependability benefits come with a cost. For example, really redundant (fully non-operational) cores and the dependability self-test and evaluation features in the chip require additional silicon area; a shorter dependability test time to improve availability leads to a lower test fault-coverage. In this section, some of the important design parameters are examined to help optimizing the design for dependability choices.

A. Dependability Self-Test and Evaluation and System Reliability

1) Effective Spare Cores

The common reliability characteristic of a standby redundant system and a gracefully degrading system is that the system is considered as functionally correct until the number of working cores drops below a threshold value *K* as described in a K-out-of-N: G system. While *K* is a fixed number determined by both application requirements and individual core performance (e.g. measured by Million Instructions Per Second, MIPS), one can approximately calculate the number of spare cores needed for a target system reliability value according to Equation (1).

In practice, it is usually difficult to verify the correctness of a core while it is working. Therefore two self-test methods can be chosen: 1) bring the system offline, test all the cores, then bring the system online again; 2) pick a few target working cores, shift their work load to (earlier tested fault-free) spare cores, test them, isolate the possible faulty core then shift back the work load.

The first method should be avoided if possible as it interrupts the normal function of the system and seriously degrades the system availability. The advantage of the second method is that the system is tested at application run-time, which makes the system still available during the dependability self-test. To summarize, using the first method makes the system unavailable whenever a dependability self-test is carried out. But when using the second method, the system only becomes unavailable when a faulty core has been detected and an application remapping is required.

A side effect of the second method is that the number of remaining spare cores in the system needs to be greater than or equal to the number of the cores being tested so that the computational load of the cores being tested can be temporarily held. If the number of cores being tested each time is *M*, then the number of effective spare cores in the MPSoC is actually reduced to *N-(K+M)* if one wants to perform the dependability self-test at application run-time. If the number of fault-free spare cores is lower than *M*, the dependability test is still possible using the first method until all spare cores have been used to replace the faulty ones. Figure 2 shows an example of the dependability test and the application remapping scenarios for both a standby redundant system and a gracefully degrading system.

2) Influence of Dependability Test Fault Coverage on Reliability

The reliability calculation performed in Section II.A assumes that once a faulty cores appears in the system, it can be detected. If the system is operating in the field, this implies a 100% dependability self-test fault coverage. In practice, 100% fault coverage is difficult to achieve for complex processors considering the limited capability of an on-chip built-in self-tester due to either silicon area or test time.

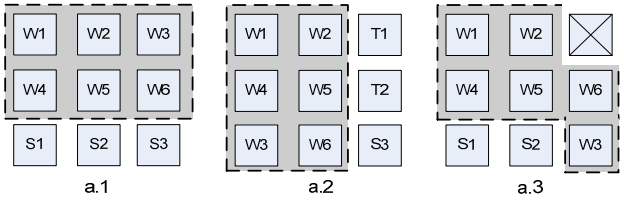

Figure 2a. A standby redundant system in mode: a.1 normally working, a.2 dependability test, a.3 application remapped. Grayed area denotes the application load. W represents working core, S represents standby cores and T represents cores being tested. The core with a cross is the one tested as faulty.

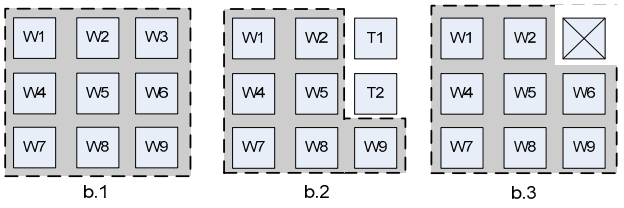

Figure 2b. A gracefully degrading system in mode: b.1 normally working, b.2 dependability test, b.3 application remapped.

If one assumes that every fault has the same probability to occur in the chip, it can be concluded that the dependability self-test fault coverage is proportional to the reliability increase of the system when spare cores are added. Given a *K*-core system with no spares (a K-out-of-K: G system), its reliability is calculated to be $R_t^K(t)$ assuming every core has a reliability distribution of $R_t(t)$ according to Equation (2). By adding *N-K* spare cores in the system, the system reliability can be calculated by Equation (1). Let the improved reliability be $R_I(t)$. Now consider a dependability self-test with a fault coverage of *F %*, then the actual system reliability can be expressed as:

$$R_{system}(t) = R_I(t) \times F/100 + R_t^K(t) \times (100 - F)/100 \quad (4)$$

An interesting example is, if the fault coverage of the dependability self-test is zero (*F=0*); then the system reliability is still $R_t^K(t)$, meaning the reliability increase is also zero despite the fact that spare cores have been added to the system. This indicates that in addition to introducing redundant resources, an effective dependability self-test method is also crucial to improve system reliability.

Software based self-test (SBST) usually cannot provide a very high fault coverage thus it will negatively affect the system reliability. Hence it is preferable that the on-chip dependability test can achieve a fault coverage close to that of the manufacturing test. Therefore, it is proposed in this paper to include an infrastructural IP into the MPSoC to serve as a dependability manager (DM). The DM should be able to perform a high quality (structural) dependability test when needed, to analyse the core test results and to alert the system if a faulty core has been detected.

B. Dependability Self-Test and System Availability

A DM is designed to test the type of faults of the users' interest in the MPSoC. As the mean time to detect a fault (MTTD) is strictly constrained by the acceptable system down time (MDT), one needs to carefully plan the self-test for minimum MTTD.

The mean time to detect a fault can be approximately calculated as:

$$MTTD = \frac{T_{MAX}}{2} = \frac{1}{2} \times \left[\frac{N_{total}}{N_{pertest}} \right] \times \frac{(V_{vector} + V_{response})}{B_{TAM}} \quad (5)$$

T_{MAX} is the maximum time to perform a full dependability test on all the cores in the MPSoC. T_{MAX} is determined by the number of times the dependability test is performed and the amount of time spent per test. The number of tests is determined by the total number of processor cores in the MPSoC (N_{total}) and the number of cores being tested each time ($N_{pertest}$). The time spent per test can be calculated by the sum volume of the test vectors (V_{vector}) and test response data ($V_{response}$) divided by the bandwidth (B_{TAM} , MByte/s) of the test access mechanism (TAM). For example, dependability testes need to be performed on nine processor cores in an MPSoC. Each time three cores are tested. The volume of test vector and test response data is 10MB. The bandwidth of the TAM is 200MB/s. In this case,

$$MTTD = \frac{1}{2} \times \frac{9}{3} \times \frac{10MB}{200MB/s} = 75ms.$$

It should be noted that, in some cases, the test vector generation and test response collection process can take place in parallel, e.g. in a conventional scan-based test. This should be taken into account while calculating the time spent per test. In addition, the time spent to shift the load from working cores to spare cores and remapping the application tasks is not considered within the scope of this paper.

The fault coverage of the dependability self-test is often closely related to the data volume of test vectors. A high fault coverage self-test usually requires a large amount of test vectors, which slows down the dependability test and will also cost more silicon area to store or generate these test vectors.

C. MPSoC Architecture and System Maintainability

In the case of MPSoC, the traditional system maintainability is interpreted as the presence of a built-in self-repair mechanism. Core-level redundancy of an MPSoC enables the possibility to tolerate potential faulty cores and allows "repair" at system level. For a conventional bus-based many-core system, replacing a working core with a spare core at another location will likely change the system topology and this can cause (e.g. speed) problems for application developers. This problem can be solved by adopting a Network-on-Chip (NoC) infrastructure as an on-chip communication fabric (e.g. configured as guaranteed throughput connection). The NoC has become popular in MPSoCs as a result of its high bandwidth, scalability and flexibility compared to traditional bus interconnections. A recent survey on the research works of NoC can be found in [19].

Via a standardized network interface (NI), core-to-core communication can be routed through dynamically configured routes in the NoC. The intrinsic reconfigurability of the NoC enables a higher level abstraction of the system for application developers. A NoC-based MPSoC can be viewed as a library with a certain amount of processing power (namely, the cores). Core status can be categorized as operational, in test, standby or faulty. System-level resource management software can map new tasks to standby cores, put cores into dependability test mode or isolate a faulty core. This way, a system level maintenance can be achieved by arranging dependability self-test and resource reconfiguration.

Core-level dependability self-test can use the IEEE 1500 standard for embedded core testing. Previous studies have suggested that the NoC can be reused as a Test Access Mechanism (TAM) to avoid dedicated test buses [20]. Recent research has shown simulation results of performing a scan-based structural test using the NoC as a TAM at application run-time in an MPSoC [21].

D. Design for Dependability Considerations

The design for dependability parameters as previously discussed have been summarized in Figure 3. Increasing dependability parameters such as the number of spare cores or the volume of test vectors will benefit system dependability. But it also increases silicon area overhead and test time. A balance between dependability requirements and resource overhead should be evaluated at the system design phase.

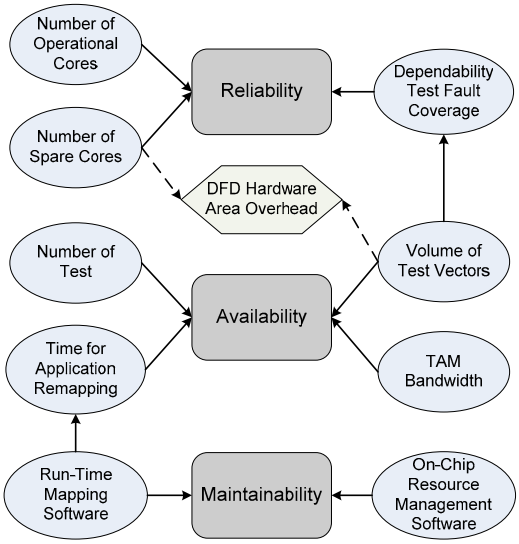

Figure 3. Design for dependability parameters and their relations with resources.

IV. Case Study: Design and Implementation of a Dependable MPSoC

A. The CRISP Platform

The CRISP (Cutting edge Reconfigurable ICs for Stream Processing) project researches optimal utilization, efficient programming and dependability of reconfigurable many-core processors for streaming applications [22].

1) General Stream Processor (GSP)

The envisioned platform for CRISP is a General Stream Processor (GSP) dedicated for virtually any streaming application. The current implementation of the GSP includes a General Purpose Device (GPD) and five Reconfigurable Fabric Devices (RFD) individually assembled in 400BGA package and interconnected on a PCB as shown in Figure 4. The GPD consists of an ARM-9 based processor and runs an embedded Linux OS to support the run-time mapping software and the dependability application programming interface (API). The reconfigurable many-core architecture in the GSP has been separated into several smaller packages (RFD) for the sake of reduced risk and cost. A Xilinx Virtex-4 FPGA device is incorporated on the PCB for user-defined applications.

High speed chip-to-chip connections (C2C) and Multi-Channel Ports (MCP) form the interconnection ports between GPD and RFDs. A user is able to debug the system via a serial debug interface on the board. Part of the data flowing in the NoC of one RFD can be rerouted to the FPGA device and measured using a Logic Analyzer through a MICTOR connector (Figure 4).

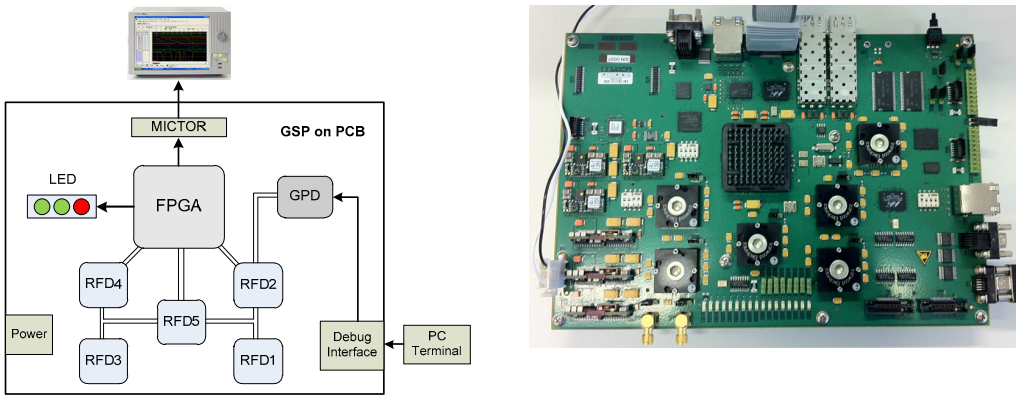

Figure 4. GSP Platform: simplified diagram of major functional blocks (left) and photo of the platform on a PCB (right).

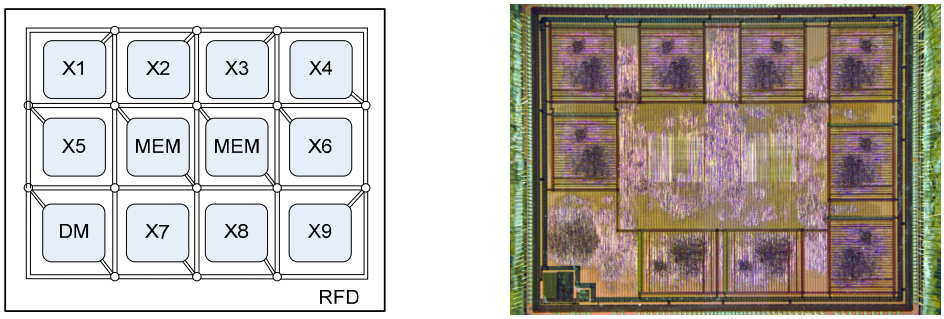

Figure 5. Reconfigurable Fabric Device (RFD) : floorplan (left), X denotes a Xentium tile processor, MEM represents the embedded SRAM and DM stands for the Dependability Manager. Photo of the RFD silicon is shown at right side.

2) Reconfigurable Fabric Device (RFD)

The major part of the GSP consists of five RFDs. Each RFD holds nine identical high-performance Xentium tile processors [12], two SRAM tiles and a Dependability Manager (DM) infrastructural IP as shown in Figure 5. The RFD chip has been produced in 90nm UMC CMOS technology and measures about 43.8mm^2. In the RFD, the area of a single Xentium tile is roughly 1.88mm^2 while the DM occupies around 0.4mm^2. The specified clock speed is 200MHz.

The functional blocks in the RFD are interconnected with GUARVC, a virtual-channel NoC architecture for streaming applications with a router capable of concurrently providing both guaranteed and best-effort services over a shared network infrastructure [23]. The NoC has a data width of 32-bit operating at 200MHz.

B. Design for a Dependable RFD

One of the themes of the CRISP project is dependability, which enables the GSP platform to be used for safety-critical applications. An RFD is a good example of a homogeneous MPSoC with nine identical processor cores (Xentium tile processor). As an example, a beamforming application has been mapped to each RFD, which requires six working cores in one RFD. This leaves three Xentium cores as spare parts for non-crucial tasks or self-repair.

The dependability parameters listed in Figure 3 have been examined to determine the dependability design choices. User specified reliability requirements demand that the dependability self-test fault coverage must be higher than 90%. In the scope of this paper, the stuck-at fault model was targeted as the main fault model. The RFD availability requirement is at least 99.0% with a 500 milliseconds allowed MDT in the case of a faulty tile processor.

Each Xentium tile processor has 32 parallel scan-chains for conventional manufacturing tests and dependability structural tests. An IEEE 1500 compliant dependability wrapper has been developed to switch the Xentium tiles to normal, manufacturing test and dependability test modes [21]. Other important dependability infrastructures include BIST for the SRAM block and a software-based self-test for the interconnection nodes such as the NoC and MCPs. Run-time mapping software, resource management software and the DM API have been developed and run in the ARM-based GPD [18].

C. Dependability Tests and Automatic DM Design

The goal of the dependability test is to reproduce the deterministic test vectors (for stuck-at faults) on-chip to achieve the required fault coverage and ensure a small silicon area overhead. Since the Xentium tile processors in the RFD are identical, and if we assume that only one faulty tile can appear after the previous test, one can test more than one

Xentium tile at a time and compare the test results to determine a possible faulty tile by majority voting. In this case, more than one Xentium tile is required per test. A major advantage of this method is the removal of the reference structural test results storage on the chip.

The DM consists of three major parts being a Test Pattern Generator (TPG), a Test Response Evaluator (TRE) and a Finite State Machine (FSM). Deterministic test vectors are generated by the TPG using techniques such as Deterministic Logic BIST [25]. In the TPG, a linear feedback shift register (LFSR) is combined with a reseeding [26] or bit-flipping [27] functional block to reproduce the deterministic test vectors using limited seed or bit-flipping information to achieve vector compression. The generated test vectors are organized in a phase-shifter into 32-bit data flits and multicasted to the Xentium tiles under test via the NoC. Test responses from multiple tiles are sent back to the TRE and compared to determine the potential faulty tile. The FSM communicates with the dependability test API to control the test process and will inform the on-chip resource management software upon the detection of a faulty core.

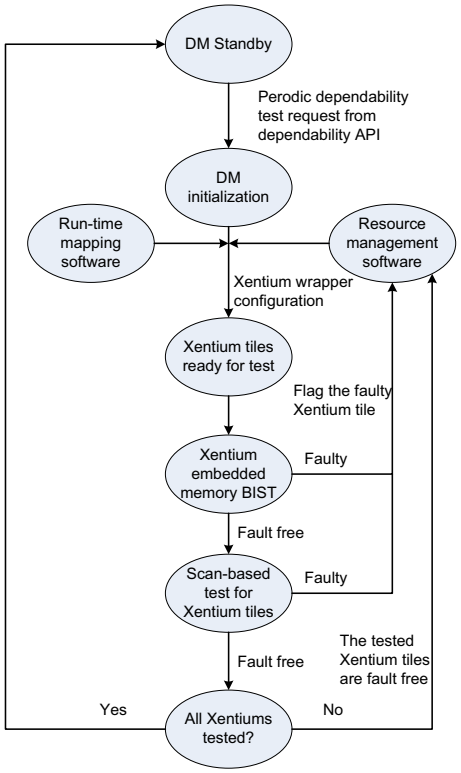

Figure 6. Dependability test flow

An example design of the DM sub blocks can be found in [24]. A recent progress is the development of a software tool which automates the design process of the DM. The input options to the tool include fault coverage requirement of the dependability test, LFSR length, compression method (reseeding or bit-flipping) and a test pattern file of the core under test generated by a commercial ATPG tool. Given the required input, the tool can generate synthesizable VHDL codes for the DM. The test vectors generated by DM can be stored into formatted test pattern file by the tool, which can then be imported into the ATPG software together with the design under test to check the DM-TPG fault coverage. As a result, the DM is able to reproduce a major part of the original ATPG structural test vectors and reach 99% of its fault coverage.

A DM design has been generated by the tool for the Xentium tile processor test. The total number of test vectors is 413; each test vector contains 398 32-bit data blocks for the 32 scan chains in the Xentium. Given the full bandwidth of the 32-bit 200MHz NoC, the subsequent test time can be calculated as $413 \times 398 \div (200 \times 10^6) \approx 0.8$ milliseconds. If nine Xentium tiles can be tested in three groups, the MTTD is 2.4 milliseconds (the specified MDT is 50 milliseconds). The dependability test can still be performed with a longer test time if the DM can only claim part of the NoC bandwidth. The method to tolerate varying NoC traffic with the presence of application data in the NoC has been discussed earlier in [21].

The DM can be designed to cooperate with other BIST methods. For example, it has been designed to start the BIST engine of the *embedded* memories in the Xentium tile and to check the memory BIST results. More fault types such as delay faults can be covered by the DM in the future [28]. The complete dependability test flow on the RFD is shown in Figure 6.

V. Dependability Experimental Results

The RFDs have been processed by UMC and passed a structural manufacturing test by Atmel Automotive GmbH. A detailed microphotograph of our DM block in the RFD is shown in Figure 7. Note that the DM has not been implemented as a hard macro, but as glue logic. On the left upper corner, the clock PLL is shown. The area of the DM is less than the area of a Xentium tile processor and the DM overhead in terms of silicon area in the RFD is acceptable (about 1%).

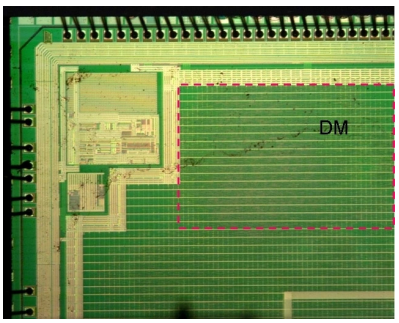

Figure 7. Silicon photo: DM in RFD

The GSP platform (Figure 4) has been assembled and went through basic factory tests and some functional tests. Using the debug interface, a user is able to interact with the dependability API running in the GPD and command the DM to perform various dependability test actions. A special on-chip hardware circuit to emulate Xentium faults has been added at the design phase, just for dependability experiment purpose. The following experiments have been carried out to validate the

function of the DM and the dependability approach proposed in this paper.

A. DM Function Validation

In order to validate the DM can deliver its designed functions, special test procedures have been prepared. Part of the NoC traffic in RFD2 has been redirected to the Multi-Channel Port (MCP) of the FPGA device on the PCB and then looped back. The NoC data in RFD2 was thus made observable via the 34-pin MICTOR connector connected to the FPGA (see Figure 4). An Agilent 16823A logic analyzer was used to capture the data stream. Examples of measurement results using the logic analyzer are shown in Figures 8, 9 and 10. It should be noted that the clock frequency of the MCP in the FPGA was set to be 100MHz due to the limitation of the FPGA device.

The user can run their own dependability software (in C codes) at the PC terminal to issue commands such as "start embedded memory BIST for No.1 Xentium tile" or "perform a scan-based test on No. 2, 3 and 4 Xentium tiles" to the DM API in the GPD. The DM API then sets up a communication channel (a virtual channel) between the GPD and the DM over the NoC and command the DM to perform the requested action(s) accordingly by send 32-bit command word to the DM control register. In the measurement result shown in Figure 8, the command word "9600 0000" was sent to the DM over the NoC. This command is interpreted by the DM as "to start up the embedded memory BIST on 3 selected Xentium tiles".

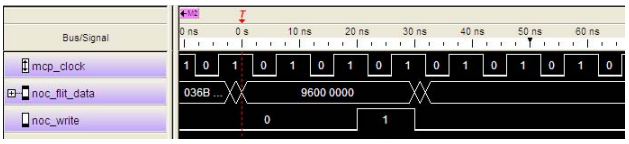

Figure 8. GPD to DM (in RFD2) command: start Xentium embedded memory BIST

Similar procedures have been taken to capture the structural test vectors generated by the DM. The test vectors have been generated in the form of 32-bit data flits and multicasted to several Xentium tiles under test over the NoC. An example test vector data flits passing a NoC router is shown after "noc_flit_data" in Figure 9. All the measured test vectors values have been compared with the reference test vectors in the ATPG test pattern file and it can be concluded that the correct test vectors have been generated by the DM in the RFD chip.

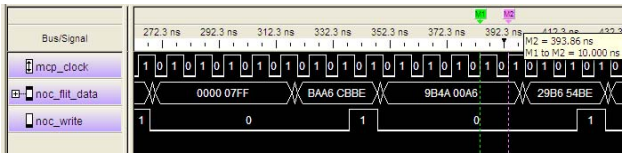

Figure 9. DM Test vectors travel via the NoC

In Figure 10, the data block between M1 and M2 is one complete test vector measuring about 32.7us. A complete structural dependability test consists of 413 such test vectors. Hence, the actual time to generate all the test vectors is $413 \times 32.7 \mu s \approx 13.5 ms$ with a 100MHz clock.

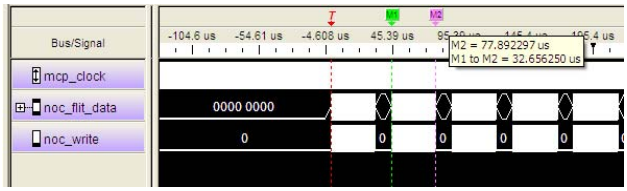

Figure 10. DM Test vectors travel via the NoC zoomed out view

Comprehensive tests have validated that the DM can successfully perform its designed dependability test functions using the NoC as a TAM.

B. Software NoC Test

The NoC in each RFD is tested using a software walk-through test method at the start-up of the GSP platform. In addition, chosen parts of the NoC (routers and interconnections) can also be tested at application run-time if the NoC resource were not occupied by any application. The NoC test results are also reported to the resource management software and faulty segments will be isolated.

More details on the NoC test algorithm and results will be treated in another paper. In this paper, we perform the DM dependability test only after a complete NoC test has been carried out and a fault-free NoC test report is received.

C. Full Dependability Test at Application Run-Time

In this part, the proposed dependability approach is performed at the run-time of a simple application to validate the complete dependable heterogeneous MPSoC concept.

1) The Mailbox Application

A simple mailbox application has been developed and its basic principle is shown in Figure 11. When the application starts up, the GPD sends a mail to one RFD and this mail will loop through three Xentium tiles via the NoC. A LED on the PCB will be turned on when the mail reaches its associated Xentium tile processor. If a Xentium tile becomes faulty or have a hardware fault emulated, the whole application will be terminated.

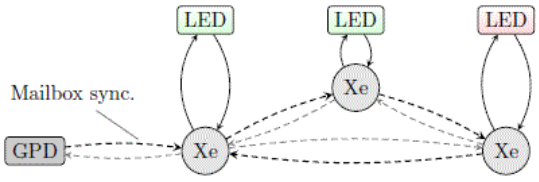

Figure 11. The mailbox application using three Xentium tiles

2) Dependability Test Scenario

The resource management software mapped the mailbox application to Xentium 1, 6 and 7 on RFD5. At a certain moment, a hardware fault was manually emulated on Xentium 7 via a designed fault injection feature in the Xentium wrapper. As the DM has been programmed to perform a periodic dependability test, it immediately identified the faulty Xentium tile. A dependability test report is printed to the PC terminal as shown in Figure 12. The report can be interpreted as: no communication error has occurred during the dependability

test; a faulty Xentium core has been detected and the faulty unit id is Xentium 7.

```
DM FULL test returned: 0x703063e8
---------------------------------
   is error: 0
 error type: No error
 core fault: 1
  mem fault: 0
faulty DUTs: 001
Xentium fault detected: core fault in unit 3.
INFO  [7402] kairos_handle_report_event      : Element 'gsp0.rfd5.xe1' is correct
INFO  [7402] kairos_handle_report_event      : Element 'gsp0.rfd5.xe6' is correct
INFO  [7402] kairos_handle_report_event      : Element 'gsp0.rfd5.xe7' is faulty
INFO  [7402] kairos_raise_event              : Processing event(7) took    4.906 ms.
```

Figure 12. Dependability test report

The resource management software was also notified by DM to isolate Xentium 7 from the system and remap the mailbox application to fault-free Xentium tiles. The whole process has been successfully accomplished with no user action required except for the fault emulation. The user could observe a very short pause of the blinking LEDs when the fault on Xentium 7 was emulated, then the application will recover by itself and the LEDs will be blinking again. This experiment proves the proposed dependability approach can be truly performed at application run-time.

A timer in the GPD measures the time spent for each activity. The dependability test cost around 4.9ms and the remapping of the mailbox application to fault-free resources cost around 90ms. Hence, the system MDT for the mailbox application was around 95ms, shorter than the 500ms MDT requirement.

3) Dependability Attributes Improvement

The mailbox application uses 3 Xentium tiles out of the total 9 Xentiums in one RFD. Therefore it can be considered as a 3-out-of-9: G system. If we make the simplified assumption that each Xentium tile has i.i.d. $R_t(t) = 0.8$ over 15 years and other parts of the MPSoC remain fault-free, the system reliability can be calculated using equation (1) and (2). The system dependability attributes have been are summarized in Table 1. An obvious system dependability improvement is achieved with our dependability approach.

Table1. Mailbox application dependability attributes:

Mailbox Application Dep. Attributes	No Dependability Approach	With Dependability Approach
Reliability (15 years)	51%	99%
Availability (MDT)	N.A.	95ms
Maintainability	N.A.	Yes

VI. CONCLUSIONS

In this paper, we have examined the important attributes of a dependable MPSoC. How design for dependability choices can affect these attributes have been analysed in detail. Equations for the calculation of dependability attributes in simplified situations have been discussed.

A dependable MPSoC (RFD) has been designed following the proposed dependability approach. The device has been processed using 90nm UMC CMOS technology. Experimental results have validated our dependability approach as well as the DM design and associated software. More fault models will be supported by the DM in the future.

ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of Recore Systems on the Xentium tile processor, back-end simulations and RFD SoC integration, NXP for the Xentium wrapper design, and Atmel Automotive for floor-planning and RFD structural manufacturing tests. Tampere University contributed to NoC software testing. Timon ter Braak and Hermen Toersche from the University of Twente have been involved in the run-time mapping software design and helped with system test and measurements.

REFERENCES

[1] S. Borkar, “Thousand Core Chips - A Technology Perspective”, in Proc. ACM/IEEE Design Automation Conference (DAC), pp. 746–749, 2007.

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

[3] IEC standard 60300-3-4, “Application guide to the specification of dependability requirements”, Sep. 1, 2007.

[4] M. Riley, L. Bushard, N. Chelstrom, N. Kiryu and S. Fergusson, “Testability Features of the First-Generation Cell Processor”, in Proc. IEEE International Test Conference 2005 (ITC 2005), pp 119, Nov. 2005.

[5] S. Shamshiri, P. Lisherness, S Pan and K. Cheng, “A Cost Analysis Framework for Multi-core Systems with Spares”, in Proc. International Test Conference 2008 (ITC 2008), pp. 1-8, Oct. 2008.

[6] L. Huang and Q. Xu, “Test economics for homogeneous manycore systems,” in Proc. International Test Conference 2009 (ITC 2009), pp.1-10, Nov. 2009.

[7] S. Borkar, “Challenges in Reliable System Design in the Presence of Transistor Variability and Degradation”, IEEE Micro, vol. 25, no 6, pp. 10–16, 2005.

[8] P. Parvathala, K. Maneparambil, W. Lindsay, “FRITS - A Microprocessor Functional BIST Method,” in Proc. International Test Conference 2002 (ITC2002), pp. 590-598, 2002.

[9] J. Collet, et al., "Chip Self-Organization and Fault-Tolerance in Massively Defective Multicore Arrays," in IEEE Transactions on Dependable and Secure Computing, No. 99, pp. 1-11, 2010.

[10] G. Hetherington, T. Fryars; N. Tamarapalli, M. Kassab, A. Hassan, J. Rajski, “Logic BIST for large industrial designs: real issues and case studies,” in Proc. International Test Conference 1999, pp.358-367, 1999.

[11] G. Kiefer, H. Vranken, E.J. Marinissen, H.-J. Wunderlich, “Application of deterministic logic BIST on industrial circuits,” in Proc. International Test Conference 2000, pp.105-114, 2000.

[12] Recore Systems Xentium architecture, http://www.recoresystems.com/technology/xentium-technology/xentium-architecture/

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

[14] M.D. Beaudry, “Performance-Related Reliability Measures for Computing Systems”, in IEEE Transactions on Computers, vol. C-27, no.6, pp. 540-547, June 1978.

[15] J. Shao and L. R. Lamberson, “Modeling a shared-load k-out-of-n: G system”, in IEEE Transactions on Reliability, 40(2):205–209, June 1991.

[16] Z. Lu, W. Liu, “Reliability Evaluation of STATCOM Based on the k-out-of-n: G Model,” International Conference on Power System Technology, 2006. PowerCon 2006, pp.1-6, Oct. 2006.

[17] L. Huang and Q. Xu, “On Modeling the Lifetime Reliability of Homogeneous Manycore Systems”, in Proc. 14th IEEE Pacific Rim

International Symposium on Dependable Computing, 2008 (PRDC 08), pp. 87-94, Dec. 2008.

[18] T.D. ter Braak, et al., "On-Line Dependability Enhancement of Multiprocessor SoCs by Resource Management", in Proc. of the 2010 International Symposium on System-on-Chip, Tampere, Finland. pp. 103-110, Sep. 2010.

[19] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip", ACM Comput. Surv., vol. 31, no. 1, 2006.

[20] E. Cota, M. Kreutz, C. A. Zeferino, L. Carro, M. Lubaszewski, and A. Susin, "The impact of NoC reuse on the testing of core-based systems," in Proc. 21st VLSI Test Symposium, pp. 128-133, Apr. 2003.

[21] 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. Digital System Design: Architecture, Method and Tools (DSD2010), pp. 531-537, Sep. 2010.

[22] CRISP project: http://www.crisp-project.eu/

[23] P. T. Wolkotte, "Exploration within the Network-on-Chip Paradigm," PhD. thesis, University of Twente, 2009, ISBN 978-90-365-2757-6.

[24] 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.

[25] P. Wohl, J. A. Waicukauski, S. Patel, and M. B. Amin, "Efficient compression and application of deterministic patterns in a logic BIST architecture," in Proc. Design Automation Conference, pp. 566-569, Jun. 2003.

[26] A. A. Al-Yamani and E. J. Mccluskey, "Built-in reseeding for serial bist," In Proc. IEEE VLSI Test Symposium (VTS), pp. 63-68, 2003.

[27] H. Wunderlich and G. Kiefer, "Bit-flipping bist," In Proc. IEEE/ACM international conference on Computer-aided design, pp. 337-343, 1996.

[28] A.D. Singh, "Scan Based Testing of Dual/Multi Core Processors for Small Delay Defects," International Test Conference 2008 (ITC 2008), pp.1-8, Oct. 2008.