DeSyRe

on-Demand System Reliability

Design of the DeSyRe online testing, graceful degradation and Virtualization support

CONTRACT NR	287611
TYPE OF DOCUMENT	Deliverable D3.3
AUTHOR	I. Sourdis, S. Tzilis, A. Malek, G. Gaydadjiev, R. Shafik, G. Rawerda, D. Pnevmatikatos, D. Theodoropoulos
ABSTRACT	This document describes the final design for online testing, graceful-degradation, and virtualization. In addition, this document describes the DeSyRe programming and execution model.
WORKPACKAGE	WP3
DISSEMINATION LEVEL	Restricted/Public

DOCUMENT HISTORY

Release	Date	Comments	Status	Distribution
v0.1	23-05-2013	Document creation	Draft	All Partners
v0.2	20-06-2013	Adding on-line testing and gracefull degradation text	Draft	All Partners
v0.3	26-06-2013	Complete draft	Draft	All Partners
v0.4	01-07-2013	Incorporating comments from the quality check control	Draft	All Partners
v0.5	02-07-2013	Incorporating comments from the quality check control	Draft	All Partners
v0.6	10-07-2013	Incorporating comments from the quality check control	Draft	All Partners
v1.0	11-07-2013	Finalizing the document	FINAL	EC, All partners

Contents

1. Executive Summary

This document describes the design of the DeSyRe online testing, graceful degradation, and Virtualization support. First, the Online Testing approach as defined by the consortium is presented in detail. We start with describing all of the techniques used for online testing of the different DeSyRe system components. More precisely, we start with the Xentium processor, which – similarly to other DeSyRe components - is isolated from the rest of the system and tested through its Network interface. The Xentium datapath and tightly coupled memory are tested using a software-implemented test mechanism (a gold test sequence of instructions). The SiMS core uses a similar software-oriented approach exploiting the NOP slots of a program to insert online test instructions. For testing DeSyRe distributed memory blocks, a centralized Built-In Self Test (BIST) facility is used that will "walk through" the memory blocks and test them in isolation using the IEEE1500 test access mechanism over the DeSyRe NoC. This provides both significant saving in area and additional flexibility for the BIST engine implementation. The memory BIST engine in a particular DeSyRe system implementation will be programmed to generate the set of merely small yet high-coverage tests for the specific memory blocks organization, size and realization technology. The NoC is tested by selected techniques that use error detection codes to trigger a fault characterization step. The Fault characterization will detect permanent faults and will thereafter diagnose the faulty link or router in order to update the fault-map with the detailed fault information needed for the graceful degradation decisions. To reduce testing complexity, the fine-gran fabric will be tested only in the context of the particular functionality instantiated at the time of testing. This requires that each different function targeting this substrate has to be accompanied with the set of test vectors that the centralized BIST engine will use when needed.

All of the test results of the different components will be translated into corresponding updates to the DeSyRe system fault-map (described in D4.2) that plays a central role in the Graceful Degradation process. The DeSyRe system uses heuristics to provide graceful degradation and close-to-optimal system configuration that strikes the balance between the HW resources, the SW workload and the task mapping. The solution space is walked through in a smart way by considering three relevant metrics: *Functionality*, *Performance*, and *Energy* using a DeSyRe specific cost model.

Furthermore, the task based execution model of DeSyRe is described along with its impact on check-pointing as well as on virtualization and context switching. The execution model divides the application in tasks, through code annotations, mapped to different system components. Communication is allowed only at the task boundaries. This has a direct impact to checkpointing since checkpoints need to include only the inputs of tasks and a pointer to the task graph, rather than components' architectural state. In addition, virtualization is achieved by maintaining multiple versions of task descriptions at the runtime system. In doing so, a task can be mapped to different types of components by choosing the proper binary, targeting a certain component type. Context switching and task migration are also supported by just maintaining different versions of task descriptions. In a DeSyRe system, a task cannot be resumed from any arbitrary execution step, instead it requires to be fully restarted using its initial input set; this consideration avoids the need of saving and restoring architectural state.