TOWARDS A DEPENDABLE HOMOGENEOUS MANY-PROCESSOR SYSTEM-ON-CHIP

Xiao Zhang

Members of the dissertation committee:

Prof.dr.ir.	G.J.M. Smit	University of Twente (promotor)
Dr.ir.	H.G. Kerkhoff	University of Twente (co-promotor)
Prof.dr.	P.M.G. Apers	University of Twente (chairman and secretary)
Prof.dr.ir.	J.P. Katoen	University of Twente
Prof.dr.	J.L. Hurink	University of Twente
Prof.dr.ir.	J.Pineda de Gyves	Eindhoven University of Technology
Prof.dr.	Z. Peng	Linköping University
Dr.ir.	H.G.H Vermeulen	NXP Semiconductors

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

Centre for Telematics and Information Technology
P.O.Box 217
7500AE Enschede
The Netherlands

The cover page was designed by KTidea (www.ktidea.com), China. The thesis was printed by Gildeprint, the Netherlands.

ISBN 978-90-365-3772-8
DOI 10.3990/1.9789036537728

TOWARDS A DEPENDABLE HOMOGENEOUS MANY-PROCESSOR SYSTEM-ON-CHIP

DISSERTATION

to obtain
the degree of doctor at the University of Twente
on the authority of the rector magnificus,
prof. dr. H. Brinksma,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday 30th of October 2014 at 16:45

by

Xiao Zhang

born on 17th June 1981
in Yantai, China

This dissertation is approved by

Prof.dr.ir.	G.J.M. Smit	University of Twente (promotor)
Dr.ir.	H.G. Kerkhoff	University of Twente (co-promotor)

Abstract

Nowadays, dependable computing systems are widely required in mission-critical and human-life critical applications. While the advance in CMOS technology enables smaller and faster circuits, the dependability of modern ICs has worsened as a result of the shrinking dimensions of MOS transistors and the increasing complexity of semiconductor devices. For those very complex SoC with many processor cores, dependability enhancement approaches are especially important.

In this thesis we first examine the important attributes of a dependable MPSoC. We then explore the possible approaches to enhance these attributes. The cost of the chosen dependability approach in terms of performance and resource (silicon area/energy) overhead are evaluated. The proposed dependability approach is implemented in silicon and its effectiveness is assessed using experiments and actual measurement results.

In the scope of this thesis, the dependability of an MPSoC is defined as its ability to deliver expected services under given conditions. Three important dependability attributes being reliability, availability and maintainability are identified. Reliability denotes the probability that the MPSoC will fail after a certain period of time. For an MPSoC, maintainability refers to the isolation/bypass of faulty components and reconfiguration of the fault-free spare parts to maintain its functionality. Availability denotes the readiness of the MPSoC to provide correct service.

The reliability of an MPSoC can be improved by using processor cores as spare. Theoretically, system reliability greatly increases as more cores are used as spares. At the same time, the area overhead for reliability enhancement also increases. Maintainability can be realized by incorporating fault detection and self-repair features into an MPSoC. By dynamically detecting faults and reconfiguring the system to circumvent them, the system can be regarded as functionally correct

with a possible drop in performance. The time spent for fault detection and system repair is combined as system down time. Faster fault detection and repair operations will decrease system down time and enable a highly available MPSoC.

The dependability approach proposed in this thesis involves test aiming stuck-at faults performed at the processor core level at application run-time. Once detected, faulty resources can be isolated by the so-called resource management software and core-level system repair can be performed by means of resource reconfiguration.

In order to validate the feasibility of our dependability approach, a homogeneous MPSoC platform with multiple Xentium processing cores was adopted as the vehicle of our experiment. A stand-alone infrastructural IP block, namely the Dependability Manager (DM), has been designed and integrated into the MPSoC platform. The DM can generate the test vectors for the Xentium cores, broadcast them via a Network-on-Chip and then collect the test responses from the cores under test. Since the cores under test have identical architecture, a faulty core can be detected by majority-voting the test responses. Dedicated test wrappers and NoC (reused as a TAM) were included into the platform MPSoC as well. A modified scan-based test scheme was used for a back-pressure style test data flow control by pausing and resuming the test data in the NoC.

The MPSoC platform was fabricated as a Reconfigurable Fabric Device using UMC 90nm CMOS technology. The dependability overhead in terms of silicon area is about 1%. Experimental results show that the dependability test can be carried out at application run-time without interrupting the function of other applications. The inclusion of the DM into the RFD makes it a maintainable MPSoC with very short stuck-at and memory fault detection time (21ms) and reasonable MDT (hundreds of milliseconds).

In conclusion, our proposed dependability approach and dependability test methods have proven to be feasible and efficient. The successful integration of the DM into the RFD and its correct operation indicate that our dependability approach can be applied to other homogeneous MPSoC platforms for dependability improvement.

To my parents

Acknowledgements

This thesis would not have been accomplished without the help of many people. I would like to express my sincere gratitude for their kind support in a memorable period of my life.

First and foremost, my gratitude goes to my supervisor Dr. Hans Kerkhoff. I first met Hans when I was looking for an individual project as a Master student of the University of Twente. Then I did my Master thesis project (BioDrop) and my Ph.D research work with him all along. Not only because his research projects attracted me, but his patience, motivation, enthusiasm, and immense knowledge gave me a great deal of courage to explore in the scientific world. His guidance helped me in all the time of research and accomplishing this thesis. I can not imagine having a better advisor and mentor for my oversea study.

I would like to thank my promoter, Professor Gerard Smit, for his guidance in the CRISP project and his help with my thesis. I also highly appreciate the direct and indirect help of my colleagues such as Xiaoqin Sheng, Jinbo Wan, Yong Zhao, Mark Westmijze, Timon ter Braak, Hermen Toersche, Alireza Rohani, Muhammad Aamir Khan, etc. with my research work in the CAES group.

Since I arrived in the Netherlands in year 2004, I have spent eight years studying in the University of Twente and living in the city of Enschede. My life was made enjoyable by my Chinese and Dutch friends here. I would like to thank them all for showing up in my life and enjoying it with me.

Last but not least, I would like to thank my parents who have loved and supported me in every possible way for so many years.

Contents

List of Acronyms

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
BIST	Built-In Self-Test
CFR	Constant Failure Rate
CG	Clock Gate
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
CRISP	Cutting edge Reconfigurable ICs for Stream Processing
CUT	Circuit Under Test
DfDEP	Design for Dependability
DfT	Design for Test
DM	Dependability Manager
DMR	Dual Modular Redundancy
EM	Electromigration

LIST OF ACRONYMS

FC	Fault Coverage
FIT	Failures In Time
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GNSS	Global Navigation Satellite System
GPD	General purpose Processor Device
GPP	General Purpose Processor
GSP	General Stream Processor
GUI	Graphical User Interface
HCI	Hot Carrier Injection
HW	Hardware
I/O	Input / Output
IC	Integrated Circuit
IIP	Infrastructural IP
IM	Infant Mortality
IP	Intellectual Property
LFSR	Linear Feedback Shift Register
MCP	Multi-Channel Ports
MDT	Mean Down Time
MISR	Multiple-Input Signature Registers
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MPSoC	Many-Processor System-on-Chip

MTBF Mean Time Between Failure

MTTD Mean Time To Detect

MTTF Mean Time To Failure

MTTR Mean Time To Repair

NBTI Negative Bias Temperature Instability

NI Network Interface

NoC Network-on-Chip

PBTI Positive Bias Temperature Instability

PCB Printed Circuit Board

PI Primary Inputs

PLL Phase-Locked Loop

PO Primary Onputs

PRPG Pseudo-Random Pattern Generator

QoS Quality-of-Service

RFD Reconfigurable Fabric Device

SBST Software Based Self-Test

SIM Scan Input Multiplexer

SIU Surround Input Unit

SoC System-on-Chip

SOU Surround Output Unit

SRAM Static Random Access Memory

SW Software

LIST OF ACRONYMS

TAM Test Access Mechanism

TDDB Time Dependent Dielectric Breakdown

TMR Triple Modular Redundancy

TPG Test Pattern Generator

TRE Test Response Evaluator

TSG Test Stimuli Generator

VC Virtual Channel

VCH Virtual Channel Handlers

VHDL VHSIC Hardware Description Language

VLSI Very Large Scale Integration

XTW Xentium Tile Wrapper

Chapter 1

Introduction

For decades, computing systems have been widely used in almost all fields of human activities, both for production and for everyday life. Dependable computing architectures are of crucial importance for mission-critical or human-life critical applications such as aerospace and automotive industry, railway transport, defence systems or banking and stock-trading systems. Undependable computing systems can not only cause financial and environmental disasters but also loss of human life. For example, the Tokyo Stock Exchange has experienced a malfunction in its computer servers due to a hardware failure in February 2012. The failure has knocked out trading on the Japanese stock market four hours [Bloo 12]. In July 2011, two high-speed trains collided with each other in Wenzhou, China, which resulted in 40 people killed and at least 192 injured. The cause of the accident were failures in the signalling and control systems due to lightning strike [Chin 11].

Apparently, the characteristics of a dependable system are the continuous operation and the capability to deal with possible faults. However, the shrinking dimensions of MOS transistors (less than 22nm) and the increasing complexity of semiconductor devices (e.g. Intel Itanium processor with 3.1 billion transistors [Toms 12]) have worsened the dependability of modern ICs. For instance, previous studies [Whit 08b] indicated that the wear-out failures appear much earlier in products using newer CMOS technology nodes as compared to those using older nodes (see Figure 1.1 (a)). Meanwhile, the constant failure rate (CFR) and chance of infant mortality (IM) also increase as the technology scales down

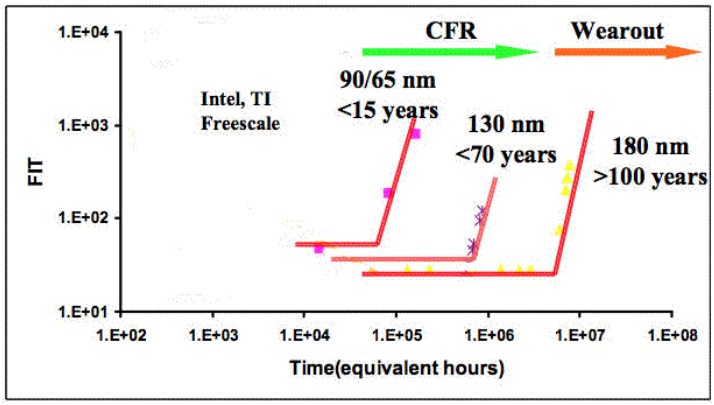
(Figure 1.1 (b)).

An apparent trend in the semiconductor industry is to integrate many components and functional blocks into a single chip to meet the area and power-consumption requirements of target applications. An increasing number of applications also need more than one processing core for complex computations. Many-processor system-on-chip (MPSoC) is becoming a popular solution for these applications. Experts have predicted that MPSoCs with more than a thousand processing cores may come to market in the near future [Bork 07]. How to guarantee the dependability of an MPSoC with billions of transistors is attracting increased research interests.

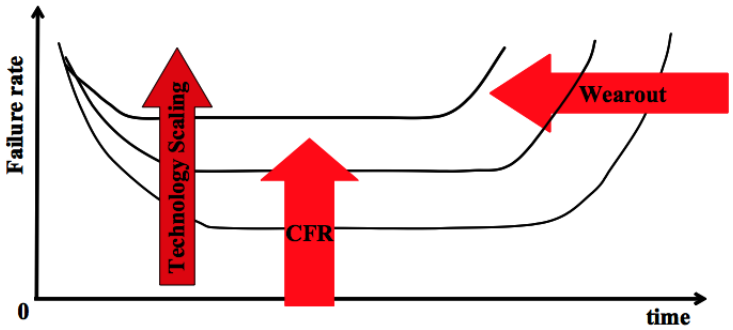
The source of the reliability issues in an MPSoC includes transient faults caused by alpha particles or cosmic rays and permanent faults caused by material aging or system wearout effects. Below the 45nm technology node, negative effects such as negative bias temperature instability (NBTI), hot carrier injection (HCI) or time dependent dielectric breakdown (TDDB) are becoming increasingly noticeable [Whit 08a]. These effects can cause the degradation of internal components and accelerate the occurrence of permanent faults [McPh 06]. A single transistor defect can result in the malfunction or complete failure of an MPSoC. For example in 2011, a major CPU provider has called back a processor product due to a transistor hard failure in the chipset of the processor and suffered a financial loss of about 1 billion US dollars as a result of this [Cnet 11]. More catastrophic consequences could be expected if these processors would have been shipped to customers and used for critical applications. Therefore, we explore in this thesis methods to enhance the dependability of MPSoCs and how to deal with permanent failures during its life time.

1.1 Research problem statement

As MPSoCs are playing an important role in modern safety-critical applications, the dependability of these applications is of crucial importance. In this thesis, the dependability of MPSoCs and approaches for dependability enhancement have been studied. The main research problems addressed in this thesis are:



(a) Normalized manufacturers data on product level failure rate as a function of technology nodes. Below the nodes, the product life time (in equivalent hours) is depicted.



(b) Product failure rate trend as technology scales down: chances of infant mortality increases, constant failure rate goes up and possible wear-out failures will occur earlier.

Figure 1.1: The worsening product dependability as technology scales down [Whit 08b]

- What are the important attributes of a dependable MPSoC and how to measure its dependability in a quantative way?

- What are the possible approaches to enhance the dependability of an MPSoC? What are the costs of the chosen dependability approach in terms of performance and resource (silicon area/energy) overhead?

- How does the implementation of the chosen dependability approach in silicon improve dependability and how can its effectiveness be evaluated using experiments and actual measurement results.

1.2 The CRISP project

Due to the rapid development of all kinds of standards, protocols and algorithms, application providers tend to welcome reconfigurable platforms to implement their applications. A reconfigurable platform offers a convenient solution to update or change the hardware platform in case the application requires flexibility. Field Programmable Gate Arrays (FPGAs) have been very well known for their flexibility and the ease of fast prototyping. However, FPGAs are usually inferior in terms of speed, area and power consumption when compared to their counterparts, the application-specific integrated circuits (ASICs) [ASIC 07]. Hence, there is an increasing need for reconfigurable ICs which combine the flexibility of FPGAs and the performance advantage of ASICs [Heys 04].

The Cutting edge Reconfigurable ICs for Stream Processing (CRISP) project (FP7, ICT-215881) aims to develop a scalable, dependable and reconfigurable many-processor system-on-chip (MPSoC) for a wide range of data streaming applications [CRISP 07]. Stream processing is a digital signal processing technique which is widely used for wireless communication, multimedia and intelligent antennas, etc. Two examples of streaming applications which have been used in the CRISP project are digital radar beamforming and Global Navigation Satellite System (GNSS) reception.

It is the goal of the CRISP project to implement a reconfigurable massive multi-core platform, being a General Stream Processor (GSP), for tomorrow's

streaming applications [Burg 11]. One of the important features which distinguishes the GSP from other digital signal processors is that measures have been taken such that during design-time and run-time its dependability can be enhanced [Ter 11]. Dependability approaches such as static and dynamic detection and localization of faults and dynamically circumventing identified faulty hardware have been proposed, implemented and validated in the CRISP project [Zhan 11]. An example application (radar beamforming) has been chosen for defining the boundary conditions (dependability specifications) of the dependability approach and for dependability attributes evaluation [Zhan 09b]. This thesis partly describes the dependability approach of the CRISP project.

1.3 Contribution of this thesis

The first contribution of this work is a study on MPSoCs from a dependability perspective. Important dependability attributes such as reliability, availability and maintainability of an MPSoC are carefully studied in this thesis. A dependability matrix is proposed to evaluate the dependability parameters and determine the cost for dependability enhancement.

The second contribution of this work is the proposed dependability approach. First a review of existing MPSoC dependability improvement methods has been carried out. Theoretically, the dependability of an MPSoC can be enhanced by determining faulty parts via a self-test and eliminating any faults in the MPSoC and remapping the tasks in the application to fault-free resources. Hence the dependability approach proposed in this thesis comprises of two major parts: self-test and self-repair. Periodic structural scan-based tests can be performed on the processing cores using the NoC as a test access mechanism. Thanks to the homogeneous structure of the target MPSoC, test responses can be compared with each other to determine a faulty core by carrying out majority-voting. The major innovation of our work is to carry out dependability tests via the NoC at run-time. This ensures a high level of availability of the target platform. The dependability of the network-on-chip (NoC) in the MPSoC is a prerequisite of the proposed dependability approach and the NoC is tested by using functional tests.

The third contribution is the design of an infrastructural IP dependability manager. The dependability manager consists of three major building blocks being a test-pattern generator, a test-response evaluator and an FSM for control of the test process. The dependability manager has been designed as generic as possible and a tool-chain has been developed to automate the design process. A new design can be automatically generated given the test-pattern set of the target processor tile under test and detailed requirements such as fault coverage or maximum silicon-area.

To validate the feasibility of the dependability manager architecture, it has been synthesized using UMC 90nm CMOS technology and the resulting netlist has been thoroughly simulated with an MPSoC framework as a testbench. Lastly, a nine-core MPSoC with our dependability infrastructures has been fabricated using the UMC 90nm CMOS technology. The developed dependability software successfully runs on the platform and the complete dependability test flow has been validated through measurement results on a prototype chip.

1.4 Outline of this thesis

As predicted by Moore's law, the size of MOSFET transistors continues to shrink and the density of integrated circuits increases already for half a century. In Chapter 2, this down-scaling trend in CMOS technology and system design will be further elaborated. In addition, its impact on system dependability will be discussed. Background information such as the production and life cycle of semiconductor devices, basics of integrated circuit testing and faults which can occur during lifetime of a chip will also be briefly discussed.

In Chapter 3, the basic principle of system dependability will be introduced. Important dependability attributes such as reliability, availability and maintainability will be examined in detail. Moreover, we will discuss the possible options to enhance system dependability in order to satisfy the dependability requirements of a specific application.

Chapter 4 presents our proposed dependability approach to enhance the dependability of an example MPSoC. The basic idea is to perform a dependability test on chip at runtime for fault detection and to use proper resource management

software for system reconfiguration. The design of essential infrastructures and software needed by the dependability test are discussed in detail.

The most critical part of our dependability approach is an infrastructural IP called the dependability manager. Its responsibilities include test-vector generation, test-response evaluation and the control of the complete dependability test process. The architecture of the dependability manager and the design of its key building blocks will be described in detail in Chapter 5.

In Chapter 6, the design flow, verification and implementation details of an MPSoC equipped with our dependability manager are presented in detail. The design has finally been implemented as a UMC 90nm nine-core MPSoC with dependability enhancement features. Measurement results on a prototype chip showed the correctness of the dependability infrastructure design and we have validated the effectiveness of our dependability approach.

Chapter 7 concludes the thesis with a summary of the presented work and gives some suggestions for future work.

Chapter 2

Trends in CMOS Technology, System Design and Dependability Challenges

ABSTRACT - *In this chapter, the impact of CMOS scaling on system dependability will be introduced. First, a brief overview of the terminology, principles and mechanisms related to semiconductor faults and defects will be given. Typical aging effects which occur during lifetime of a chip and their influence are discussed. Furthermore, some existing test methods related to our dependability test in later chapters will be briefly revisited.*

2.1 Dependability challenges

2.1.1 CMOS scaling and complexity

In the 1960s, it was predicted by Moore's law that the transistor density of integrated circuits will roughly double for every two years. As shown in Figure 2.1, the prediction has been accurate for half a century and is the de facto guideline for the research and development of the semiconductor industry since the late twentieth century. The continuous and aggressive shrinking of MOSFET dimensions resulted in a constant performance increase per unit silicon area and a decrease of the supply voltage. Nowadays, 20nm CMOS technology has been implemented

in industry and 11nm technology is likely to follow in the year 2016. It was predicted in the 2011 International Technology Roadmap for Semiconductors (ITRS) report that the transistor gate length will scale below 10nm and new structures such as multi-gate MOSFETs will be implemented within 10 years [ITRS 11].

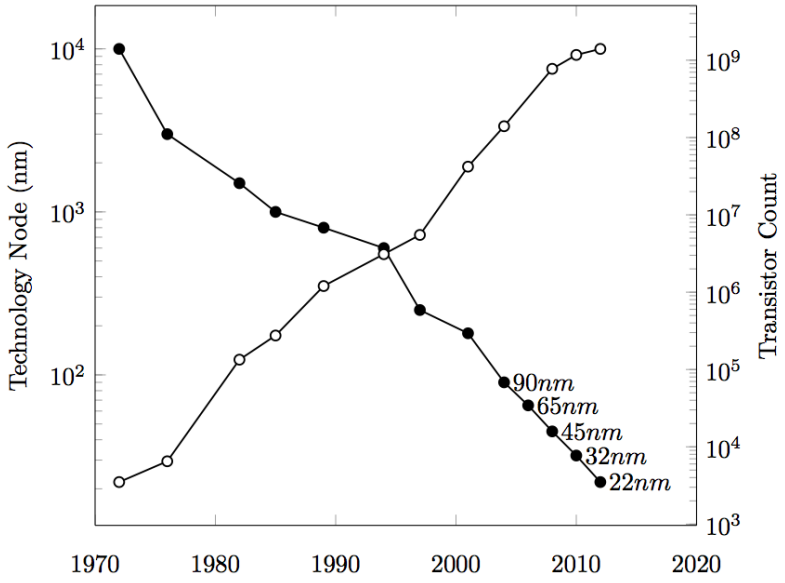


Figure 2.1: Moore's law: CPU transistor count and technology nodes

The endeavour to make smaller transistors has resulted in allowing more system complexity and faster devices. However, the down-scaling of transistor parameters is negatively impacting the reliability of the building blocks of an integrated circuit. For example, the continuous reduction of the thickness of the MOS transistor gate oxide layer leads to a dielectric as thin as 1nm, which is equivalent to 3-4 monolayers of atoms. The ultra-thin gate oxide increases the leakage current as well as the chance of a gate dielectric breakdown. As silicon-based oxide has reached its limit, new materials have been adopted to maintain the scaling demand of industry. Starting from the 45nm technology node, high-K transition metal oxide has been used as dielectric material and metal gates adopted instead of polysilicon gates. New structures such as FinFETs [Hu 12] and extremely thin silicon-on-insulator (ETSOI) transistors are required to maintain the down-scaling trend continuing beyond the 15nm node [Stat 10].

The use of new materials and structures changes the behaviour of well-known failure mechanisms such as time-dependent dielectric breakdown (TDDB) [Whit 08a] and also bring in new reliability concerns e.g. random telegraph noise (RTN). In the ITRS 2011 report [ITRS 11], it is already regarded as a major challenge to sustain current IC reliability level with future devices.

Another direct consequence of the scaling of the MOSFET transistor is the increased integration density. The ever increasing need for higher computation power leads to more and more transistors and interconnects implemented into the same silicon area, which naturally results in more potential fault sites.

On the other hand, the technology advances enabled the integration of a number of processing cores into a single silicon die, which is known as the multi-processor or many-processor system-on-chip (MPSoC). Today, the MPSoC has been widely adopted for desktop as well as embedded and mobile products. Industry expects a single chip with more than a thousand processing cores to be on the market in the near future [Bork 07]. Increasing dependability challenges as a result of the growing structural complexity and strict timing schedule of multi/many-core interaction have already been shown [Axer 11]. Meanwhile, the flexible structure of the MPSoC makes it possible to use fault detection and resource reconfiguration mechanisms to tolerate a certain number of faults and enhance system dependability as will be shown later.

In conclusion, the continuous advances of semiconductor technology have brought enormous dependability challenges. At the same time, new possibilities are also emerging from the MPSoC domain for the enhancement of dependability.

2.1.2 Application reliability and availability requirements

Nowadays, more and more high-end IC products are being used in a harsh environment for life or mission critical applications, such as automotive, military, aerospace and medical [Hinc 10, Rabb 10]. Different from desktop or normal mobile applications, much more severe external stress factors such as temperature, shock and radiation are often expected in these applications. For example, a growing trend in the aviation industry is to replace traditional centralized engine controllers with distributed control systems. This replacement can result in much

simpler interconnections and save hundreds of pounds of aircraft weight. Consequently the control electronics are placed closer to the engine and must function correctly with a temperature range from −55 °C to 200 °C. The automotive industry is another example requiring an increasing number of high-temperature electronics as mechanical and hydraulic parts are being replaced by electromechanical systems. For instance, the transmission controller and wheel sensors need to work with an ambient temperature of around 200 °C. And the exhaust sensors must function properly at a peak temperature of 850 °C [Wats 12].

There are also applications which require very low system down-time (high availability requirement) such as ICs used for telecommunication base stations, banking, trading and stock-market servers. For example, many multinational continuous process manufacturing companies need to manage worldwide operations and business transactions from clients and suppliers all day everyday. Stock trading and investments companies which manage billions of U.S. dollar assets can tolerate almost zero downtime of their IT infrastructure. Another example is the Chinese online train-ticket sale system. In recent years before the Chinese Spring Festival, the train-ticket online sale website must be able to handle over five million concurrent transactions for weeks. A few minutes unavailability of the website server will cause catastrophic social and political consequences. As such, reliability and availability are becoming major requirements in electronic systems when used in these particular applications.

2.2 Production and life cycle of semiconductor devices

2.2.1 A brief introduction to semiconductor manufacturing

The semiconductor manufacturing flow comprises of a number of closely related stages. The flow begins with circuit design and wafer production. The target circuit is implemented during wafer fabrication. Then it is assembled, packaged and tested before the final IC product is delivered to the customer. Each man-

ufacturing stage involves a number of processes. Hundreds of process steps are required in the complete flow. For example, wafer fabrication includes oxidation, deposition, lithography and etching; wafer testing consists of wafer sorting and laser repair, etc.

Due to the complexity of technology, the need for highly precise process steps and the vulnerability to contamination, defects can be expected in every stage of the semiconductor manufacturing chain. Defects can cause faults and failures in IC products. If detected during the manufacturing flow, chip defects will result in yield loss. If faults and failures occur after the chips have been shipped to customers, they can cause system malfunction and damage the reputation of the semiconductor manufacturer [Bush 05]. In the next section, some basic taxonomy and terms related to defects are introduced.

2.2.2 Taxonomy and terminology

In the semiconductor world, defect, fault and failure are often used to denote incorrect parts or behaviours in a device or a system. A defect is a physical flaw or imperfection which violates the design specification. The cause of defects can be incorrect manufacturing operation, material contamination during fabrication or structure wearout during the product life cycle. A defect is the root of a fault, but not all defects result in faults. For instance in Figure 2.2, conducting particles A and B are defects introduced during the wafer fabrication stage. Particle A can cause an unexpected short connection between the data line and the ground line. However, particle B will not cause such a problem. Thus a fault is the result of a critical defect which can cause unacceptable fluctuation of performance or incorrect functional behaviour.

Faults in semiconductor devices commonly fall into three main categories: permanent, intermittent and transient faults. Permanent faults are continuous and irreversible faults which persist regardless of time. Examples of permanent faults include missing material, bridging wires and broken oxide layers, etc. Intermittent faults are usually caused by internal parameter degradation or material instability. Intermittent faults often precede the occurrence of permanent faults as the degradation progresses. A gate dielectric soft breakdown is an example of

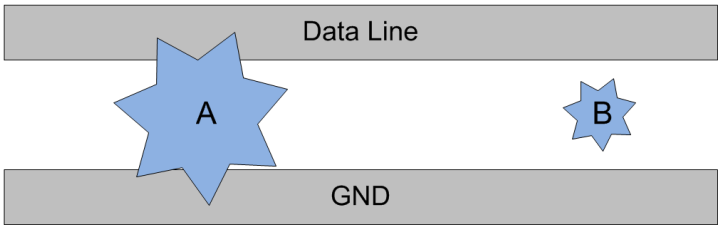


Figure 2.2: Example of critical and non-critical defects

an intermittent fault. Transient faults are also known as random faults. They usually occur as a result of temporary environmental conditions, such as temperature variations, effect of high energy particles or electromagnetic interference. Most tests in digital circuits are based on permanent faults.

A fault in a system can make it fail to deliver the expected service. If a failure occurs in an IC, it can no longer comply with specified functions. The incorrect state and output generated during a system failure is sometimes also referred to as error. The terms mentioned above are similar but not interchangeable. Defects are physical and structural; errors are logical and functional. A fault is local whereas a failure is global.

2.2.3 IC life cycle and failure rate

Three distinctive stages can be identified during the lifetime of an IC, being the infant mortality, working life and wearout stages. The probability an IC will fail in each stage is indicated by the *failure rate*. The failure rate $\lambda(t)$ of an IC varies with time and exhibits the well-known bathtub curve as shown in Figure 2.3.

2.2.3.1 Infant mortality

As Figure 2.3 shows, the curve begins with a high failure rate after manufacturing which gradually decreases. The infant mortality stage can last from weeks to as long as half a year. Infant defects are often caused by extremely marginal structures during the assembly process. These defects somehow passed the basic production test but can result in faults and failures shortly after. ICs at the infant mortality stage should not be shipped to customers in order to avoid field

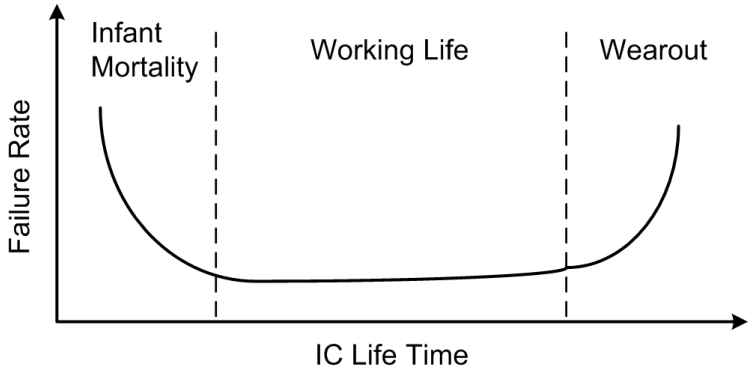


Figure 2.3: Bathtub-shaped failure rate distribution during IC life time

failure and product return. Burn-in is an engineering method for screening out early failures in the factory before the products reach the customers [Kuo 84, Kim 09, Tsai 11]. During the burn-in process, a device is stressed under constant thermal and electrical conditions in order to accelerate the appearance of early failures. Other methods such as vibration, power and temperature cycling are also used to carry the chip through the infant mortality stage. Notice that in many non-critical ICs, burn-in is not done. Instead, IDDQ tests which measure the quiescent supply current are used to weed out potential reliability weaknesses.

2.2.3.2 Working life

The main characteristic of the normal working life stage is the low and nearly constant failure rate (slight variation in practice). Faults will still occur at random spots in this timeframe but the chance is much smaller than in the infant mortality stage. The failure rate of this stage is commonly referred to by the term of Failures in Time (FIT) in the semiconductor industry. FIT is defined as the number of failures that can be expected in one billion (10^9) device-hours of operation. One billion device-hours can be combinations like 1,000 devices for 1 million hours each or 1 million devices for 1,000 hours each, etc. For example, the failure rate $\lambda(t)$ of a chip with 10 FIT is:

$$\lambda = \frac{10}{1 \times 10^9} = 1 \times 10^{-8} \text{ failures per hour} \tag{2.1}$$

The expected average time before a component or system fails after initialization is defined as the mean time to failure (MTTF). For the majority of electrical systems with a constant failure rate, the failure distribution is exponential during its working life. This causes the MTTF to be the reciprocal of the failure distribution rate $\lambda(t)$. To continue with the above example, the MTTF of the system is:

$$MTTF = \frac{1}{\lambda} = 1 \times 10^8 \text{ hours} \tag{2.2}$$

Some systems can repair their failure and return to their normal function. Such a system is defined as a repairable system [Asch 84]. The time frame after the occurrence of a system failure until it is fully restored is defined as the system down-time. Mean time between failures (MTBF) is used to denote the expected average time between two successive failures of a system. MTBF is an important concept in reliability engineering and should not be confused with MTTF. MTBF is only applicable for repairable systems whereas MTTF is commonly used for non-repairable systems which can fail only once. In some literature MTBF is referred to as the mean time *before* failures. Only in this case, the MTBF is equivalent to MTTF. Note that in this thesis, MTBF is always defined as the mean time interval *between* two system failures.

2.2.3.3 Wearout

Ultimately, the failure rate of an IC will start to increase as its internal parts begin to fatigue and wearout. The occurrence of the wearout stage of an IC depends on many factors such as technology, heat, input signals (work load in case of a processor) and power-supply stress conditions. Wearout defects such as electromigration or time-dependent dielectric breakdown often first appear as intermittent faults and then gradually become permanent faults and cause system failure. Several typical wearout mechanisms will be briefly introduced in following sections.

2.3 Faults occurring during IC lifetime

The aggressive technology scaling has resulted in an extremely high level of device density and computational performance boost as well as accelerated circuit degradation and wearout during their operational life time. Consequently, chips which have passed the final production test can fail during their life cycle. The major degradation mechanisms of semiconductor microelectronic devices are negative/positive bias temperature instability (NBTI/PBTI), gate oxide breakdown, also known as time-dependent dielectric breakdown (TDDB), hot carrier injection (HCI), and electromigration (EM). These mechanisms are briefly reviewed below.

2.3.1 Negative/positive bias temperature instability

Negative bias temperature instability is a wearout mechanism mainly observed in p-channel MOS transistors since they usually operate with negative gate-to-source voltage. NBTI is accelerated by elevated temperature and voltage levels and manifests itself as an increase in threshold voltage and a decrease of the drain current and transconductance [Schr 03]. As a result of its adverse impact on the critical parameters of the PMOS transistor, NBTI has become a serious CMOS reliability concern.

Recent research suggested NBTI is physically caused by two tightly coupled mechanisms: interface state generation and holes trapping in the oxide traps [Gras 09]. Fast advances in CMOS processing technology did not alleviate the NBTI effect, instead, they made it worse. For example, nitrogen was incorporated into MOSFET gate oxide to reduce the gate leakage current. However, the introduction of nitrogen turns out to accelerate the NBTI degradation. In addition, transistor size down-scaling resulted in stronger internal electric fields and higher temperature, which also aggrevated the NBTI effect.

The positive bias temperature instability (PBTI) on the other hand affects n-channel MOS transistors in the case they are positively biased. It has a similar mechanism as NBTI and can also negatively impact transistor reliability. In practice, NBTI is the dominant degradation problem. Delays faults will develop as a result of the BTI effect.

2.3.2 Hot carrier injection (HCI)

Carriers (electrons or holes) are able to gain substantial kinetic energy as they travel through a region of a high electric field. For instance, if current flows through the source-drain channel in a MOSFET, carriers can become sufficiently energetic to be *hot*, which is a term used to measure their energy level instead of temperature. Hot carriers can gain so much energy that they are injected into the gate oxide bulk of the MOSFET via scattering or impact ionization [Terr 85, Maha 00, Here 88].

The injection of hot carriers into the oxide can cause various physical damage and change the characteristic parameters of a MOSFET, such as a shift of its threshold voltage as a result of interface-state generation. A device suffering from HCI will eventually fail as defects accumulate. HCI has been regarded as a critical reliability concern which will adversely impact the reliability of semiconductor devices.

The HCI effect is strongly affected by the internal electric field distribution of a transistor. As the down-scaling of the supply voltage is far slower than the shrinking of the channel length and oxide thickness, the internal electric field intensity continuously increases and the reliability issues caused by HCI become worse. It will result at some stage in delay faults.

2.3.3 Time-dependent dielectric breakdown (TDDB)

Dielectric breakdown is the irreversible change of the dielectric property of the gate oxide layer of a MOSFET [Bern 06]. As the dielectric layer of a MOSFET is subject to electric field stress, structural degradation slowly develops in the oxide. As a consequence, the electrical property of the dielectric will gradually change until a hard breakdown takes place. This form of dielectric breakdown is referred to as time-dependent dielectric breakdown.

The general mechanism of TDDB is that under the stress of an electric field, charges are trapped in various parts inside the oxide layer and at its interface. As a result, stress induced leakage current (SILC) is produced, which flows through the dielectric material and creates a heating effect. The heat accumulation can gradually cause thermal damage to the oxide and increase the density of charge

traps (soft breakdown). This positive feedback loop will eventually cause a permanent conductive path within the dielectric, which shorts the gate with the substrate material or the source/drain and results in a faulty transistor [Stat 02]. In such a case, a hard breakdown, i.e. a permanent fault occurs.

The rate of the TDDB defect generation is proportional to the current density flowing through the oxide layer, which is accelerated by the increase of supply voltage and temperature. As a result of the down-scaling of device dimensions, the thickness of the gate oxide continuously decreases, which also leads to an early dielectric breakdown.

2.3.4 Electromigration (EM)

Unlike the previous three wearout effects, electromigration does not take place within the MOS transistors. Instead, the degradation is found in the chip's metallization. Electromigration refers to the transport of metal atoms in the metal thin-film as a result of the flow of electric current. The movement of the metal atoms can cause depletion of the metal material in some places (e.g. cathode side) and accumulation in other places (e.g. anode side). Consequently, high resistances or broken wires can result from the EM effect, which will lead to time-related faults or permanent open wire faults. On the other hand, accumulation of metal atoms can cause permanent shorts between interconnections.

In [Blac 69], the determining factors of the EM-related MTTF has been concluded: $\mathrm{MTTF_{EM}}$ is proportional to the area of the cross-section of the wire; $\mathrm{MTTF_{EM}}$ also has an inverse relationship with the current density through the wire. As device features continuously shrink and wire current density stays high, wearout of wires due to EM will inevitably result in various permanent faults in a chip and continue to be a serious chip reliability threat.

2.3.5 Fault models

Faults caused by the former degradation effects are generally known as *aging* faults. Typical consequences of aging faults in digital circuits can either be logic faults or delays faults. Logic faults are caused by transistors or wires which are open or shorted to (stuck at) logic 1 or 0. A delay fault means that the delay of

one or more paths exceeds the clock period. It can result in incorrect logic values under a specified clock frequency. Notice that by lowering the clock rate, one can eliminate some of the delay faults. All the aging effects introduced earlier can first lead to intermittent delay or logic faults, and then eventually progress into permanent faults [Rade 13].

In this thesis, the intermittent faults are treated as permanent faults as well because the approaches we adopt for testing permanent faults remain effective when the intermittent faults develop into permanent ones. The stuck-at fault model has been widely used for permanent faults detection and diagnosis in a logic circuit. Thus it will be used as the fault model in our dependability test which will be introduced in the following chapters.

2.4 Basics of testing

The continuous down-scaling trend and the increasingly complex manufacturing process of integrated circuits both result in higher chances of defects in semiconductor products. A single defect can easily lead to a fault in a component and eventually in a complete failure of a billion-transistor MPSoC. Thus testing is required to guarantee that only fault-free devices are delivered to the end users.

Different from design verification, manufacturing testing is not intended to verify the correctness of the design itself but to check the soundness of the manufacturing process. Note that the result of testing can be used as more than just a device pass/fail indication. It can also be used as the basis of fault diagnosis, which tries to locate the defects and origin that caused the fault and helps to eliminate this in order to improve the production yield. In fact, testing can also be performed *during* the life cycle of an integrated circuit by means of e.g. built-in self-test (BIST) [Bush 05]. As such, the correctness of an IC can be periodically checked during its working life and repair operations could be enabled in case of any faults. This is particular useful for applications with the requirement of a high level of dependability.

Tests performed after the wafer fabrication generally fall into two categories: parametric tests and functional/structural tests. Examples of parametric tests are DC and AC tests which are carried out subsequently to detect potential electrical

and timing problems before the test of the logic functions of the IC [Bush 05]. The basic idea of functional/structural test is to apply specific test stimuli (test vectors or test patterns) to the circuit under test (CUT) and to separate the good devices from the faulty ones by examining their test responses. If the test responses of a CUT do not agree with those of a known-good circuit, the target CUT is considered to be faulty. For an n-input combinational CUT, there are 2^n possible combinations of test vectors in total, which is usually too large to test for a modern circuit with many inputs. In practice, only a subset of all the test vector combinations are used to test a circuit. One possibility (denoted functional test) is to use the input combinations if a CUT operates in a real system as test vectors. One can get an indication of the CUT's correct functional behaviors using this method. However, the thoroughness (fault coverage) of functional test is often quite limited. In addition, there is no real quantitative measure of the defects which can be detected out of the total set of defects in the case of a functional test.

As a result the manufacturing test uses the structural information of the CUT instead of its designed functionality. In structural testing, faults can be regarded as the deviation of logic values caused by critical structural defects. Various fault models such as single stuck-at (SSA) fault, bridging fault and delay fault have been established to represent the consequence of defects in a CUT. As such, a physical defect can be modeled at the logic level. Take a 4-input AND gate as an example, if any input pin is shorted with the ground line, a stuck-at-0 fault model can be used to model the fault. The logical consequence of the defect is that the output level of the AND gate is fixed to a logic zero. As different defects can yield the same logical fault, one structural test vector is capable of detecting thousands of faults in a large circuit, which makes it a very efficient and effective approach. The effectiveness of a structural test can be measured by its fault coverage, which is defined as:

$$\text{Fault coverage} = \frac{\text{Number of detected faults}}{\text{Number of all possible faults}} \tag{2.3}$$

The goal of test pattern generation is to find a minimum set of test vectors which can detect all the possible faults of a chosen fault model(s), namely to

achieve a 100% fault coverage. The test pattern fault coverage is calculated by performing fault simulations. A fault simulator can emulate target faults in the CUT, apply test vectors to its input and compare the simulated test responses with the reference responses. In this way, it can determine which faults are detected by a given set of test vectors. Mature commercial tools (e.g. TetraMAX from Synopsys) are available for both automatic test pattern generation (ATPG) and fault simulation. Using the netlist of the CUT as an input, one can obtain its test patterns which meet a specific fault coverage requirement.

However, the complexity of ATPG algorithms and the huge number of fault sites require excessive computational power. As a result, design for test (DfT) approaches are often adopted at the design phase, which can greatly reduce the computational requirement. DfT efforts aim to increase the controllability (setting a certain node in the CUT to a desired value) and observability (observing the logic value of an internal node) of the target circuit. Typical DfT examples are scan path and core-wrapper design and insertion. In addition to the increase of ATPG efficiency, DfT can also greatly enhance ATPG fault coverage. In this thesis, several DfT techniques have been adopted to help enhancing the dependability of a target MPSoC. Details of DfT architectures, test application and scheduling will be discussed in Chapter 4. Test vector generation will be covered in Chapter 5.

2.5 Conclusion

The continuous down-scaling trend of CMOS technology nodes leads to even smaller transistor dimensions and as a result much more complex systems. Increasingly aging faults as a result of various degradation effects will occur during the life time of an IC. As a consequence, permanent logic and delay faults will appear much earlier than before. The increased complexity has enabled the introduction of multi-processor SoCs. This offers new possibilities for dependability enhancement via on-chip replacement of faulty processors. As an increasing number of advanced ICs are being used for life/safety-critical applications, their dependability requirements must be treated carefully. Conventional structural test methods for testing stuck-at faults can be adapted for permanent fault detection

as part of our dependability approach. More details will be given in subsequent chapters.

2. TRENDS IN CMOS TECHNOLOGY, SYSTEM DESIGN AND DEPENDABILITY CHALLENGES

Chapter 3

The Dependable MPSoC Concept

ABSTRACT - *As indicated in Chapter 2, the continuous downscaling trend of MOS transistors has led to unreliable building blocks for an integrated circuit. The question how to make a dependable MPSoC boils down to how to build a dependable system using undependable components. The term dependable IC has been used in various occasions without a clear definition. In this chapter, a first step is made to analyze and measure the dependability of MPSoCs in a quantitative way.*

This chapter is organized as follows. In Section 3.1, the definition of a dependable computing system is introduced. In Section 3.2 and 3.3, three main attributes of a dependable MPSoC are examined in detail. A theoretical analysis on the enhancement of each attribute is also given. Section 3.4 presents our design for the dependability concept as well as the design-space exploration to be used for the design of a dependable MPSoC.

Parts of this chapter have been presented at the 2010 International Symposium on System on Chips, Tampere, Finland [Ter 10].

3.1 Introduction into dependable computing systems

Five fundamental characteristics of a computing system were concluded in [Aviz 04], namely functionality, usability, performance, cost and dependability. The definition of computer system dependability has been evolving for several decades. Various definitions of dependability exist, with emphasis on different aspects of interest. Historical definitions of dependability have undergone a change of focus from the capability of system operation to successful task accomplishment [Parh 88]. The International Federation for Information Processing (IFIP) work group 10.4 on dependable computing and fault tolerance defined dependability as:

- The trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers [IFIP 69].

Whereas the International Electrotechnical Commission (IEC) provided its definition of dependability as:

- a collective term used to describe the availability performance and its influencing factors: reliability performance, maintainability performance and maintenance support performance [IEC 12].

Based on the definitions given in previous studies, we adopt the following definition of dependability in this thesis:

- The justifiable confidence that a computing system can perform its specified functions and deliver its specified services in time under given operational and environmental conditions.

In contrast to a dependable system, an undependable computing system is one that operates too slow, performs incorrect functions or delivers incomplete services. System functions and promised services can be accurately specified by the system provider in a similar way as an Internet service provider can promise a service level agreement (SLA) to their users. The notion of dependability presented

above is a general description of system characteristics in a non-quantitative way. Dependability attributes such as reliability, availability, maintainability, safety and confidentiality can be used to quantitatively measure the dependability of a computing system. In the scope of MPSoC dependability, three main dependability attributes reliability, availability and maintainability are studied in detail in the following sections. As safety and confidentiality are strongly application-scenario dependent, they will not be treated in depth in this thesis.

In addition to the dependability attributes, threats to dependability and means to improve system dependability have been summarized in previous studies [Aviz 04]. As indicated in Chapter 2, main threats to system dependability include defects which can result in faults and failures. Four important means to deal with these threats are fault prevention, fault tolerance, fault removal and fault forecasting.

- Fault prevention aims to prevent defects and faults from occurrence by e.g. use of more mature and reliable semiconductor processing technologies for IC fabrication.

- Fault tolerance is the endeavor to maintain system functions and avoid system failures in the presence of faults.

- Fault removal means to eliminate as many faulty parts in the system as possible.

- Fault forecasting means to estimate and determine whether faults are likely to take place in future by monitoring critical parameters such as temperature, current or voltage of the parts of interest.

Fault tolerance and fault removal are the main measures used in this thesis in order to improve system dependability. Note that fault removal in this research refers to the isolation of an entire processor core which contains faulty parts.

3.2 MPSoC reliability

3.2.1 The reliability function

Reliability is, by definition, the ability of a system to correctly perform specified functions under designated conditions for a specific period of time [IEC 07]. It can be determined by the metric of the probability that a system can correctly operate within a given time frame. The time frame is the mission time of a system, ranging from several years to several tens of years. If the mission time of a system is not clearly specified, the time frame can be the specified life time of the system. For example, a sedan can have a lifetime of 15 years and a truck 22 years.

The reliability of a system can be expressed as a *reliability function* $R(t)$ for the mission time t. Similarly, the probability that the system will fail over the same time period can also be described as a function of time, i.e. the *failure distribution function* $F(t)$. According to general probability theory [Bazo 04], it is obvious that:

$$R(t) = 1 - F(t) \tag{3.1}$$

As introduced in Chapter 2, the failure rate $\lambda(t)$ is a parameter used as a metric for the occurrence of system failures. It is a variable which reflects the probability of a system failure at a certain moment t. A typical method to determine the failure rate of a semiconductor device is e.g. usage of accelerated stress tests on a set of samples of devices randomly selected from its production population. In a simplified approach, the calculation of the device failure rate can be performed as following: assume that we observe 10 out of 1,000 devices fail after the stress test finishes, it can be concluded that an individual device will fail with a probability of 1% by the end of the same test under the same conditions. The failure rate obtained from the sampled devices under accelerated stress test is then extrapolated to actual operating conditions to estimate the failure rate of the device when used in field operation. Different failure mechanisms and statistical acceleration models have been discussed in detail in previous studies [Bern 06] so they will not be further discussed in this thesis.

Previous studies have established the relation between $R(t)$ and $\lambda(t)$ [Bazo 04]

as:

$$R(t) = e^{-\int_0^t \lambda(t)\mathrm{d}t} \tag{3.2}$$

The main characteristic of the normal operational life of a semiconductor device is the low and nearly constant failure rate (see Figure 2.3). We confine our reliability computations within this operational period and hence a constant failure rate value λ can be assumed. Equation 3.2 can be easily transformed as:

$$R(t) = e^{-\lambda t} \tag{3.3}$$

Hence, assuming a constant failure rate, common semiconductor devices have a reliability function in the form of a decreasing exponential function. The larger the failure rate λ, the faster the exponential function decreases.

As discussed in Chapter 2, the FIT (failures in time) number is commonly used in industry to specify the reliability of a semiconductor device. FIT represents the number of failures that can be expected in one billion device-hours' operation. It holds that:

$$R(t) = e^{-\lambda t} = e^{-FIT \times t \times 10^{-9}} \tag{3.4}$$

An example on how to calculate the reliability of chips with FIT information is given below. Suppose an IC manufacturing company produces a batch of chips with a FIT number of 100. To calculate the reliability of these chips in 10 years, i.e. 87,600 hours, Equation 3.4 is used:

$$R(87600) = e^{-100 \times 87600 \times 10^{-9}} \approx 99.1\%$$

Therefore the probability that these chips will correctly function after 10 years is about 99.1%.

It should be noted that the specified FIT number of mature modern IC is normally quite low, less than 100 or even less than 10 under commercial environmental and operational conditions. However, their actual failure rate will rise given severe environmental stress found in e.g. marine, aviation or space applications. Therefore in the following sections, exaggerated FIT values (e.g. 1,000 or even 10,000) are assumed in the reliability calculations to highlight the system reliability difference of various system configurations under harsh conditions.

3.2.2 Realistic MPSoC reliability and our assumption

The reliability function of a single component has been discussed in the previous section. However, for a very complex system with multiple sub-blocks, it is usually difficult to calculate the system reliability by treating it as a flattened entity. A logical approach is to decompose the system into smaller components or functional blocks, compute the reliability function of each sub-block, then derive the system reliability function based on the system structure in which these constituting parts are interconnected.

In a typical MPSoC, processor cores occupy most of the silicon area, hence they can be considered as the main sub-components of the system. As a result of the way they are processed, processor cores on the same die are inherently correlated with each other. In addition, due to the process variations and different load history, each core will have a distinctive reliability distribution. Therefore in a real MPSoC, processor cores are dependent and non-identical components. For mathematical accuracy, the reliability of an MPSoC with non-i.i.d. (independent and identically distributed) components with arbitrary reliability distributions have been studied in [Liu 98] and [Huan 08], etc.

In this thesis, we explore the MPSoC reliability improvement at the system level instead of at the component level. As such, our interest lies in the change of system reliability as a result of various sub-blocks' combination approach such as series, parallel or hybrid. From a system perspective, we are more concerned with the average reliability distribution of the entire MPSoC instead of the exact distribution of each individual core. Moreover, the analysis of system reliability can be mathematically simplified with the assumption that the reliability of each core is independent and identical.

Based on these two reasons, we make the assumption that all the processing cores in a homogeneous MPSoC are the same and have identical and independent reliability distribution (i.i.d.) in our following calculations.

3.2.3 Series and parallel system reliability

A system with series structure describes a system whose correct functional operation depends on the sound operation of every and each constituting block. It

should be noted that the sub-blocks are not necessarily electrically interconnected in a serial fashion, but rather the accomplishment of the system function requires the correct operation of each part. For simplicity, such a system is referred to as a series system in the following context.

Take a system with three sub-blocks as an example. Since all three blocks must be correctly operational in order for the system to be functional, the probability that the system works is the probability that all three blocks work at the same time. Hence, the system reliability in this case is the product of the reliability of each block.

$$R_{sys} = R_1 \times R_2 \times R_3 \tag{3.5}$$

Equation 3.5 can be written in a more generic form. For a series system with N sub-blocks, with the reliability of each block being $R_1(t)$, $R_2(t)$ till $R_N(t)$, the system reliability is:

$$R_{sys}(t) = \prod_{i=1}^{N} R_i(t) \tag{3.6}$$

In case all the sub-blocks are identical, i.e. all blocks have the same reliability function $R(t)$, the reliability of a series system with N components can be simply calculated as:

$$R_{sys}(t) = R^N(t) \tag{3.7}$$

From the equations above, it can be concluded that the reliability of a series system is lower than each of its components. The more blocks are involved in a series system, the lower its overall reliability becomes.

In contrast to the series structure, a system with a parallel structure can have its function accomplished via multiple individual sub-blocks. A system with parallel structure can be described as a system which can correctly perform its specified functions if at least one of its constituting blocks works properly. According to its definition, the parallel system fails only if all of its sub-blocks fail at the same time. As discussed in Section 3.2.1, the failure distribution function can be expressed as: $F(t) = 1 - R(t)$. Hence the reliability of a three-component

parallel system can be expressed as:

$$\begin{aligned} R_{sys}(t) &= 1 - F_{sys}(t) = 1 - F_1(t) \times F_2(t) \times F_3(t) \\ &= 1 - [1 - R_1(t)] \times [1 - R_2(t)] \times [1 - R_3(t)] \end{aligned} \tag{3.8}$$

Equation 3.8 can be written in a more generic form. For a parallel system with N sub-blocks, with the reliability of each block being $R_1(t)$, $R_2(t)$ till $R_N(t)$, the system reliability is:

$$R_{sys}(t) = 1 - \prod_{i=1}^{N} [1 - R_i(t)] \tag{3.9}$$

To better illustrate how the series and parallel structure impact the system reliability, the following example is given. Suppose one processor provided by a certain semiconductor company has a FIT value of 1,000. By using Equation 3.3, the reliability (R) of this processor within 10 years is calculated and depicted in Figure 3.1. Assume five such processors are separately used in two systems, one with series structure and the other with parallel structure. The equivalent system reliability of each system in 10 years can be computed using Equation 3.6 and 3.9; the result is also shown in Figure 3.1. By observing the figure, it is quite obvious that the series structure harms system reliability whereas the parallel structure improves the system reliability tremendously. This figure also explains why the introduction of redundant resources (parallel structure) can improve the system reliability.

Figure 3.2 shows a more exaggerated situation if the FIT of a single processor is assumed to be 10,000. The reliability of one processor drops to nearly 40% after 10 years. However, the system reliability with a parallel structure is still quite high even in the presence of very unreliable components.

3.2.4 K-out-of-N:G system reliability

Modern MPSoCs usually consist of a number of processing cores interconnected by advanced communication fabrics e.g. a network-on-chip (NoC). Applications can be mapped to the MPSoC platform at run-time and the positions of functional

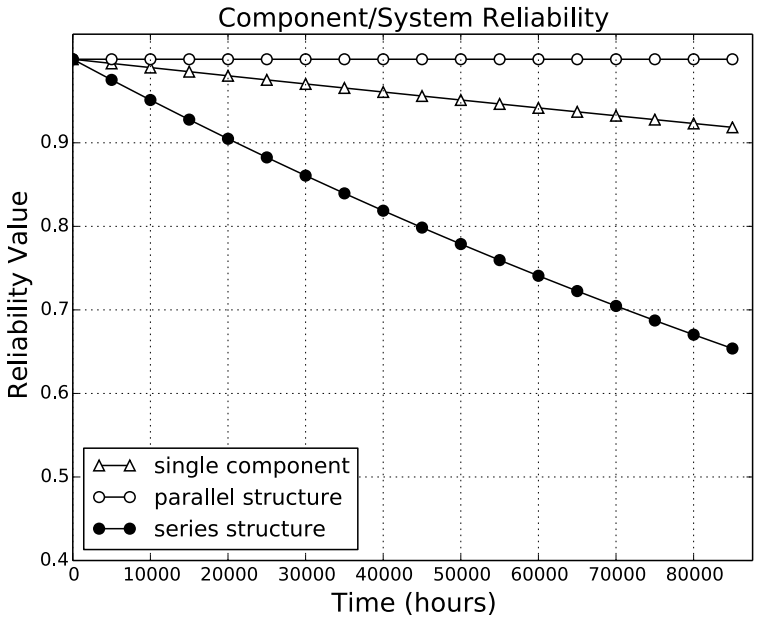


Figure 3.1: Component and system reliability with 5 cores during 10 years; processor FIT = 1,000.

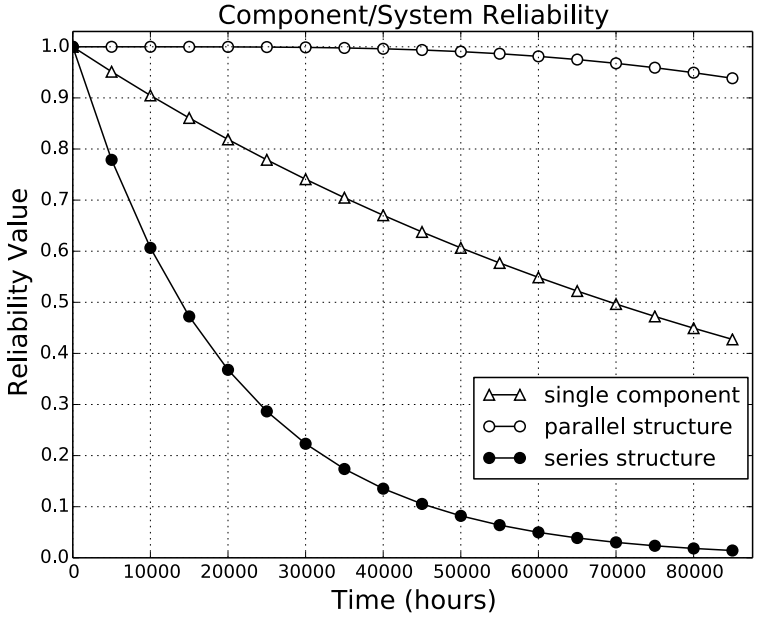


Figure 3.2: Component and system reliability with 5 cores during 10 years; processor FIT = 10,000.

blocks can change depending on the actual mapping. In this case, it is difficult to assert whether these cores are organized in a series or parallel fashion. With regard to the system reliability, a more intuitive expression could be: to perform specified functions or to provide required services, a system needs at least K out a total number of N processor cores to be fault-free. As the load of the application is shared among the working processor cores, such an MPSoC can be modeled as a load-sharing K-out-of-N: G system [Shao 91]. A K-out-of-N: G system has in total N components (in our case processor cores) and the system can correctly perform its required function (being a Good system) if at least K components are working properly ($K \leq N$).

The reliability of a K-out-of-N:G system with i.i.d. reliability components can be calculated as following. The probability that exactly K components work out of the total N components follow the *binomial distribution*:

$$\begin{aligned} P_K &= \binom{N}{K} R^K (1-R)^{N-K} \\ &= \frac{N!}{K!(N-K)!} R^K (1-R)^{N-K}, (K = 0, 1, ..., N) \end{aligned} \tag{3.10}$$

In the equation, R is the reliability of each component. The K-out-of-N system will be a good system if K or (K+1) until N components work correctly. Therefore the reliability of the system is equal to the probability that the number of working components is greater than or equal to K [Lu 06]:

$$R_{sys} = \sum_{i=K}^{N} \binom{N}{i} R^i (1-R)^{N-i} \tag{3.11}$$

It is obvious that the series and parallel structures are two special cases of a K-out-of-N:G system. Given all components have i.i.d. reliability, a series system is a K-out-of-N:G system with $K = N$. And a parallel system is a K-out-of-N:G system with $K = 1$. By substituting K with N and 1 in Equation 3.11, one can get Equation 3.7 and Equation 3.9 respectively.

In order to continue with the example given in the previous section, assume a system comprising of five processors (a K-out-of-5:G system) with the FIT value of each processor being 10,000 (as stated previously, this is an exaggerated value).

The system reliability can then be calculated using Equation 3.11. The results are depicted in Figure 3.3. It is obvious that the system reliability decreases if more processors are required to finish its specified function. If all five processors are needed, the system reliability distribution is the same as the reliability distribution of the series system which degrades greatly over time. If only one processor is required for proper operation, then the system reliability distribution equals that of the parallel system which stays high over ten years.

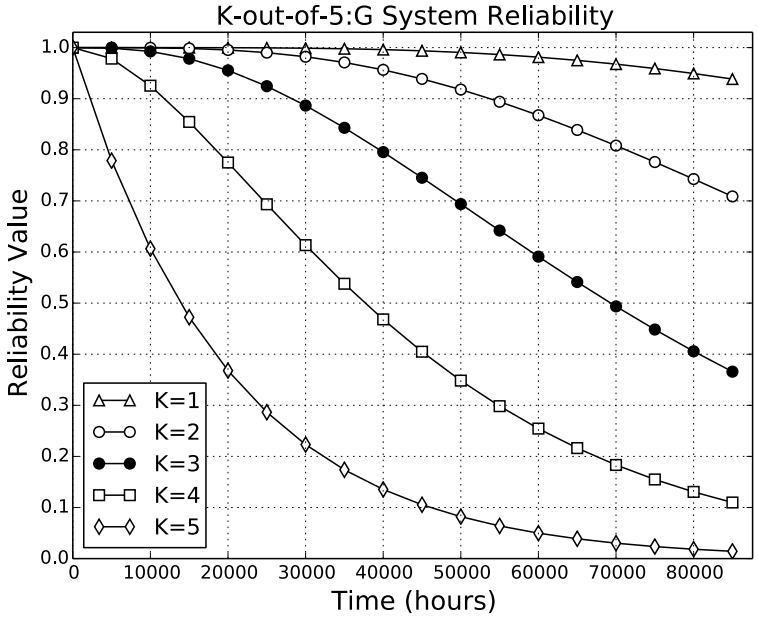


Figure 3.3: Reliability distribution of a K-out-of-5:G system during 10 years, processor FIT = 10,000.

3.2.5 Improving MPSoC reliability

In section 3.2.3, some calculations have been carried out and the results suggest the parallel structure can greatly improve the reliability of a system. The essence of the parallel structure is the introduction of alternative paths for the accomplishment of system functions. This idea is the basis of the fault-tolerant theory: the introduction of spatial redundancy in a system can enable its continuous operation in the presence of faulty components and hence increases its

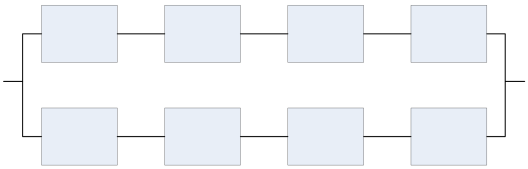
reliability. Information redundancy such as the use of error detection and correction codes (EDCC), or time redundancy such as repeating key operations have their pros and cons but they will not be treated in detail in this thesis. Instead, usage of hardware redundancy in an MPSoC to improve reliability and hence dependability is the main target of our research.

Dual modular redundancy (DMR) or triple modular redundancy (TMR) are classic fault-tolerant approaches which add hardware redundancy in a target system. The idea is to carry out the same operation by parallel hardware blocks at the same time and compare their results against each other to determine the correctness of each parallel path. In case a faulty path exists, a voter component will eventually determine the correct paths by using majority voting. In the following discussions, the term *smart switch* is used to refer to such a component which can determine the correctness of each parallel path and choose the fault-free path to perform the required functions. More discussions on the "smart switch" mechanism will be covered in the next section.

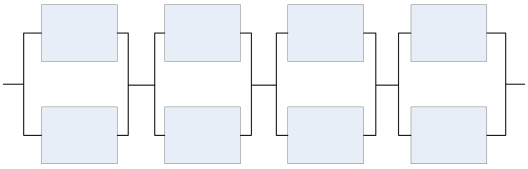
Although expensive approaches such DMR and TMR have been known for a long time, various methods to implement these approaches can still be explored in modern MPSoCs. For example, redundancy can be added to a MPSoC on different hierarchical levels. Parallelism can be achieved at the system level (e.g. using multiple sets of identical systems) or at the component level (e.g. using multiple identical components for one specific system function). Figure 3.4 (a) and (b) demonstrate the dual modular redundancy at the system level and at the component level respectively.

Alternatively, spare units can be added to the system as backups (see Figure 3.4 (c)). If one unit becomes faulty, it can be replaced by the spare unit. The introduction of spares makes the original system a K-out-of-N:G system in which K is the number of processors required by the application and $(N - K)$ is the number of spares. In this case, the system reliability distribution can be computed using Equation 3.11.

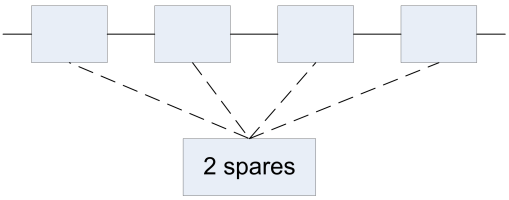
To evaluate the effectiveness and efficiency of different redundancy configurations, the following example is taken to observe the reliability distribution of each approach. Assume an application requires four identical processors to work at the same time in order to perform a certain function. Each processor has an



(a) Dual modular redundancy at the system level



(b) Dual modular redundancy at the component level



(c) 4-out-of-6:G redundant system

Figure 3.4: Various redundancy configurations (all blocks are identical processor cores)

identical and independent reliability distribution and the FIT number of a single processor is set to 1,000. To enhance the reliability of the system, redundancy is introduced by means of dual modular redundancy at system and component level. In addition, spare cores can be added to make the system a 4-out-of-N:G redundant system. Figure 3.4 shows the block diagram of each configuration. For the sake of simplicity, the switch blocks are not shown. Simulation results of the reliability distribution of each case in 10 years is given in Figure 3.5.

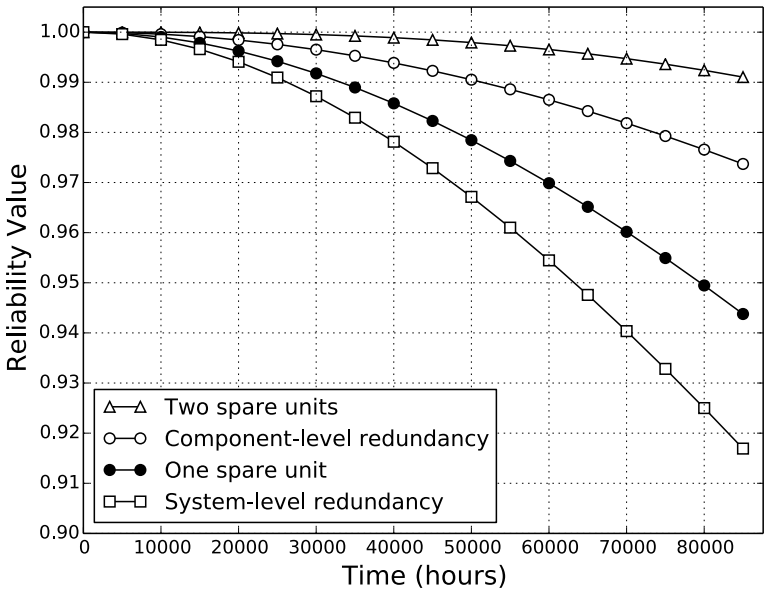


Figure 3.5: Reliability distribution of dual modular redundancy at system and component level and a 4-out-of-N:G system with one or two spares; processor FIT = 1,000.

By examining the calculation results, it can be concluded that redundancy at the system level brings the least improvement to system reliability (the lowest line $R_{sys} \approx 91.8\%$ after 10 years). Whereas component-level redundancy achieves a much better result ($R_{sys} \approx 97.5\%$ after 10 years). By adding one spare unit in the system, the reliability improvement is already better than the system redundant configuration. And by adding two spares, the system reliability becomes even better ($R_{sys} \approx 99\%$ after 10 years) than that of the component-redundant configuration. However, the resource overhead of the K-out-of-N configuration is 25%

(one spare) or 50% (two spares). In comparison, the two parallel configurations both need a 100% resource overhead. If the number of processors required by the target application increases, the benefit in terms of resource overhead efficiency of the configuration with spares (standby redundancy) becomes even more obvious. In conclusion, the K-out-of-N redundant configuration can achieve huge system reliability improvements while adding less resource overhead to the system than the parallel system and component configurations. Therefore, the usage of spare processor cores in an MPSoC has been chosen to improve its reliability.

3.2.6 Fault coverage and reliability

In previous discussions, it is assumed that an ideal switch component exists in a parallel redundant system, which serves two functions:

- to determine the correctness of each parallel path;
- to choose the fault-free path to perform the required functions.

In an MPSoC modeled as a K-out-of-N:G system, a similar component should be able to carry out similar functions:

- to determine the correctness of each processor core;
- to replace any faulty core with a fault-free one.

In the previous section, it is also assumed that the fault detection (i.e. self-test of the core) and core replacement (repair) actions should always take place immediately after any fault occurs. This means the selected fault-detection technique should have a fault coverage of 100% on all possible type of faults; in addition the fault detection and core replacement actions should take almost no time. In practice, it is very difficult to reach either of these two goals. The fact that extra time needs to be spent for fault detection and core replacement, negatively impacts the system availability. More discussion on the implication of system availability will be covered in the next section. In this section the relation between fault coverage and reliability improvement is explored.

Fault detection without 100% fault coverage in an MPSoC implies the possibility that when a processor core becomes faulty, it will not be detected, and

a system failure will still occur even though there are fault-free cores as spares. Consider an extreme example: if the fault coverage is zero, which means a faulty processor cannot be detected at all, the system reliability will not be improved no matter how many spare cores are added into the system.

Given a random circuit with a test fault coverage of 90%. If the total number of possible faults is 100, then 90 of them are detectable and the rest 10 faults are undetectable. To simplify the discussion, it is assumed that every potential fault has the same probability to occur. As such, if one random fault occurs, it will either fall into the detectable category (90/100 = 90% chance) or fall into the undetectable category (10/100 = 10% chance). As such, the test fault coverage (90%) is *numerically* equal to the probability that a specific fault will be detected (also 90%). As shown in the following calculation, the value of fault detection probability is substituted by the fault coverage. Let the probability of a random fault detection be P and the probability that this fault cannot be detected be P', it holds that $P + P' = 1$.

Consider an MPSoC modeled as a K-out-of-N:G system. In an ideal situation, all possible faults can be detected (full fault coverage) and the system reliability R_{sys_full} can be calculated following Equation 3.11; R_{sys_full} is equal to the reliability of a K-out-of-N:G system R_{K-N}. In the worst case, none of the faults can be detected (zero fault coverage) and the system reliability R_{sys_zero} is equal to the reliability of a K component series system ($R_{K-series}$). Since a 100% fault coverage (FC) is usually difficult or too expensive to achieve, the actual system reliability of this MPSoC lies between R_{sys_full} and R_{sys_zero}:

$$\begin{aligned} R_{sys} &= P \times R_{sys\_full} + P' \times R_{sys\_zero} \\ &= FC \times R_{sys\_full} + (1 - FC) \times R_{sys\_zero} \\ &= FC \times R_{K-N} + (1 - FC) \times R_{K-series} \end{aligned} \tag{3.12}$$

The 4-out-of-5:G system in the previous example can be reused to demonstrate the influence of test fault coverage on system reliability. Figure 3.6 shows the calculated system reliability with different test fault coverages as parameter. The relation between test fault coverage and system reliability indicates that in addition to introducing redundant resources, an effective dependability self-test

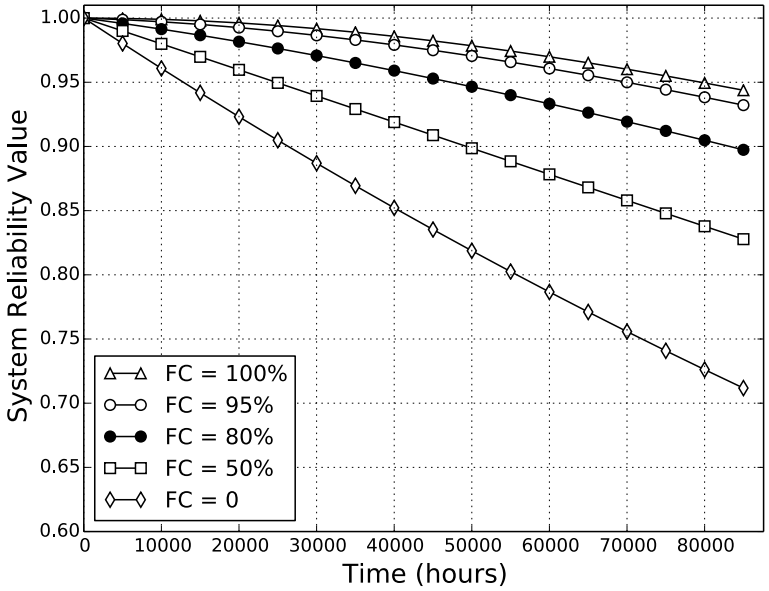


Figure 3.6: Reliability of a 4-out-of-5:G system with test fault-coverage (FC) as parameter during 10 years; processor FIT = 1,000.

method is also crucial to improve system reliability. For example, software based self-test (SBST) [Zhu 09, Chen 03] is flexible but usually cannot provide a very high fault coverage. However, IC manufacturing test is known for its high fault coverage on specific type of faults. Hence, an exploration of suitable MPSoC self-test methods is thoroughly discussed in the next chapter.

3.3 MPSoC availability and maintainability

This section introduces the traditional availability and maintainability concepts first. Then it explores how to achieve system-level maintainability taking the architectural advantage of a homogeneous MPSoC. It then reveals the impact on system availability using the proposed test strategies to detect a fault in an MPSoC.

3.3.1 Introduction to availability and maintainability

A maintainable system is one that can continue to deliver its specified services after undergoing repair operations. Such a system can tolerate multiple occurrences of faults and failures before it is completely discarded as useless. A traditional maintenance operation involves a repairman who can examine the cause of a system failure, repair or replace the faulty component, test the repaired system and bring it back online again. However in the case of a modern MPSoC, field repair of its internal sub-blocks in its package is normally impractical, if not impossible, due to its high level of integration and complexity. In that strict sense, there currently exists no maintainability at the MPSoC core level. Whereas at the system level, spare processor cores can be reserved to substitute for the faulty ones by means of resource reconfiguration and the remapping of application tasks. This substitution can be regarded as a special form of system maintenance, which can restore the MPSoC functions and requires no external repair or test equipment.

Both the fault detection and system maintenance operations cost time, during which the normal system functions cannot be performed, i.e. the system is down. The loss of operation time implies the decrease of system availability. Availability refers to the readiness of a system to correctly perform its specified functions. System availability can be measured by the percentage of time when the system is correctly operational.

$$\text{Availability} = \frac{\text{Uptime}}{\text{Total time}} = \frac{\text{Uptime}}{\text{Uptime + Downtime}} \tag{3.13}$$

Modern reliable systems normally have a high degree of availability (more than 99.99%). However, for real-time industrial systems, the system downtime (unavailability) is usually of particular interest. This is because the correctness of system service not only depends on the logical results but also the actual time it costs to produce the results [Axer 11]. In order to guarantee the real-time property of such systems, the system downtime should be carefully examined.

In Figure 3.7, important terms related to system availability are recapitulated [IEC 07]. The time interval between two successive system failures is defined as the mean time between failures (MTBF). Note that the mean time to failure

(MTTF) is often used in a non-reparable system which can fail only once, while MTBF is particularly used for repairable systems. The MTBF consists of two parts: mean system uptime (MUT) and mean system downtime (MDT). MDT consists of the time spent for fault detection, i.e. mean time to detection (MTTD) and the time spent for system repair, i.e. mean time to repair (MTTR).

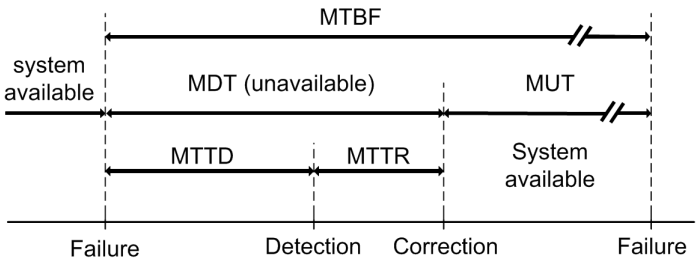


Figure 3.7: Important terms related to system availability [IEC 07]

If an MPSoC is used for industrial systems with real-time requirements, its maximal MDT cannot exceed the limit specified by the user application. Hence it is required that:

$$\text{MTTD} + \text{MTTR} < \text{MDT}_{max} \tag{3.14}$$

Once a faulty core is detected in an MPSoC, the time required to replace it with a fault-free spare core and to remap the application (MTTR) largely depends on the core architecture and the algorithm used by the reconfiguration and remapping software. On the other hand, the MTTD is mainly determined by the selected fault models and the test approach. Within the scope of this thesis, methods to decrease the MTTD are further explored in the following sections. MTTR will be discussed in Chapter 6 where the implementation and evaluation of the entire system will be presented.

3.3.2 A maintainable MPSoC

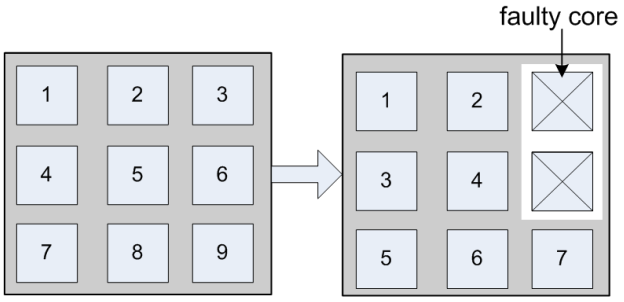
As briefly introduced in the previous section, the traditional system maintainability method, i.e. physical repair within the chip package, is currently not feasible while the MPSoC operates in the field. Instead, maintainability can also be achieved by locating the faulty sub-block, isolating it and using spare blocks as

substitution for it. Note that the level of hierarchy at which the fault detection and isolation take place have a direct impact on the maintenance efficiency and possible overhead. For example in [Schu 05], redundancy is introduced at the *microarchitectural* level of a target processor at the cost of about 10% silicon area overhead. For a single or dual core processor, this is an effective approach as there are not sufficient resources for core-level redundancy. From a silicon area overhead point of view, this approach is not economic for an MPSoC since the total area overhead linearly increases with the number of cores instantiated in the MPSoC. For a homogenous MPSoC with a large number of processor cores, a centralized approach to manage the MPSoC dependability at the core level is apparently a better approach.

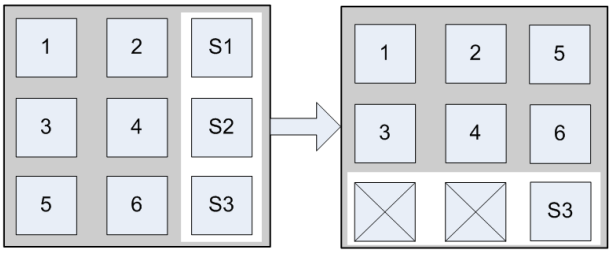
3.3.2.1 Redundancy organization

Graceful degradation and standby redundancy are two typical redundancy configuration schemes for an MPSoC. The graceful degradation system employs all the *available* processor cores in the MPSoC for target applications. If a faulty core is detected, the system will remap the application to fault-free cores and bypass the faulty ones to ensure overall correct service. However, the reduction of the total number of *operational* cores will inevitably cause a loss of system performance. As such, the system actually trades performance for reliability. The performance of an MPSoC can be characterized by its quality of service (QoS). Various QoS models for MPSoC performance evaluation have been proposed in previous studies [Guo 07, Iyer 07]. One typical QoS standard is to examine the number of instructions which can be handled by an MPSoC per second (million instructions per second, MIPS). Modern processors can normally achieve several thousands MIPS. For instance, the ARM Cortex A7 embedded processor can process 2,850 MIPS at 1.5 GHz. Although faults can occur within its internal processor cores, an MPSoC can still be regarded as fault-free at the system level before its MIPS drops below a certain threshold value.

In a standby redundant system, a fixed number of processor cores are employed (assuming full workload for all cores) depending on the actual requirement of the target applications. Other cores are reserved as spares for substitution once a



(a) Graceful degradation scheme (right side: the mission can still be accomplished by the cores in the grey area with increased time)



(b) Standby redundancy scheme

Figure 3.8: MPSoC redundancy configuration schemes. The grey area is where the computation tasks take place.

faulty core emerges. The system QoS will stay unchanged as long as any emerging faulty core can be detected and there are still spares available. If all the reserved spare cores are exhausted, the required system QoS cannot be satisfied anymore and then the system will be considered to fail. A block diagram of a 9 core MPSoC configured as a graceful degradation system or a standby redundancy system is shown in Figure 3.8 (a) and (b) respectively. The grey area represents the cores where the computational operations takes place. Both the gracefully degrading and standby redundant configuration can be used in the same MPSoC. In the case that not all the working cores are fully loaded and a fault is detected, one can take the following steps in sequence: 1) shift the workload of the faulty core to other working cores; 2) use spare cores if the workload is too heavy for the remaining working cores; 3) degrade system workload (gracefully degrading) depending on the actual QoS requirement.

3.3.2.2 Topology and resource remapping

Modern MPSoCs usually adopt a network-on-chip (NoC) as the communication fabric between processor cores due to its high scalability, flexibility and performance. If a fault-free processor core is used to substitute a faulty one, the corresponding control and data signals should be rerouted to a new location of the NoC. The rerouting of signals implies a change of the existing NoC routing table, which can cause problems for the application engineer if he tries to map the existing applications whose signal flow was optimized (e.g. in timing) for the old routing table.

Various solutions have been proposed to deal with the mismatch between the NoC routing table and the application remapping requirement. For example, the concept of a virtual topology was introduced in [Zhan 09a, Zhan 10a]. Virtual topology is an abstraction of the chip's physical topology into functional blocks connected by communication fabrics for the operating system and application programmers. It helps to provide a uniform and consistent topology view for application mapping regardless of the actual organization or *change of the physical topology*. An alternative approach is to design an application in such a way that it is *resource-location* independent. All computing resources are dynamically managed by resource-management software in the MPSoC and sufficient performance for each application is guaranteed by allocating resources to them at run-time [Ter 10, Ter 11].

The latter solution has been adopted in this thesis as a result of its tight and convenient coupling with our dependability approach. By treating the dependability management infrastructures as resources, the system dependability can be better managed together with existing applications. Via a standardized network interface (NI), core-to-core communication can be routed through dynamically configured routes in the NoC. A NoC-based MPSoC can be viewed as a library with a certain amount of processors cores in various conditions. For example, the 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. In this way, system-level maintenance can be carried out to achieve a high level of dependability.

More details are provided in the next chapter on our dependability approach.

3.3.3 Improvement of MPSoC availability

In order to guarantee the system MDT is within a specified range, the first step is to derive a quantitative measure of the mean time to detect a fault (MTTD). As a self-test can be used to detect the type of faults of the users' interest, the MTTD can be estimated from a test perspective. The fault detection procedure can normally be performed by the following sequence:

- Generation of the test-vectors;
- Test stimuli are transported from the tester to the target core under test (CUT);
- Application of the test stimuli;
- Test responses become available at the output of the CUT;
- Test responses are transported back to the tester;
- Analysis of the test results.

These operations need to be repeated until all the processor cores in an MPSoC are tested. The time (T_{test}) required for data transport (both test stimuli and test responses) is determined by the volume of the test data and the bandwidth of the test access mechanism (TAM):

$$T_{test} = \frac{V_{stim}}{B_{stim}} + \frac{V_{resp}}{B_{resp}} \tag{3.15}$$

V_{stim} and V_{resp} is the volume of the test stimuli and test responses respectively. Note that they can be different in case the number of primary inputs and outputs of the CUT is different. The NoC bandwidth (B_{stim} and B_{resp}) available for the test stimuli and the test responses can also be different depending on the actual routing.

The time spent on test stimuli application and test response analysis depends on the actual hardware architecture and the implemented algorithm of related

blocks. If properly arranged, the steps outlined above can be pipelined to make the complete self-test operation much more efficient. For example a word-wise comparison of the test response can be carried out while each response word is collected by the tester. Ultimately, the MTTD is determined by the number of cores under test, the total volume of the test data and the available TAM bandwidth. Assume that in total N processor cores are tested one by one; if the last chosen processor core is faulty and it is only detected by the last test vector, the worst-case detection time T_{max} is N times T_{test}.

The mean time to detect a fault can be approximately calculated as half the worst-case detection time T_{max}. Consider the following example: a self-test has to be carried out on a nine processor-cores in an MPSoC and each time three cores are tested simultaneously. Assume the volume of the test stimuli and test response data is 10MB and the bandwidth of the TAM is 200MB/s. In this case,

$$\text{MTTD} = \frac{1}{2} \times T_{max} = \frac{1}{2} \times \frac{9}{3} \times T_{test} = \frac{1}{2} \times \frac{9}{3} \times \frac{10\text{MB}}{200\text{MB/s}} = 75\text{ms} \qquad (3.16)$$

It should be noted here that in some cases, the test stimuli generation and test response collection process can also almost 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 summary, the MTTD and system availability can be improved with the following methods:

- Test in parallel: test as many cores as possible each time to reduce the number of test iterations.
- Pipeline the test operations: design a pipelined system in which important steps such as test-data transfer, test-application and test-result analysis can be pipelined to reduce the time required per test.
- Reduce the total test data: a self-test with high fault coverage usually requires a large amount of test vectors. The reduction of test data will normally lead to a decrease of the test fault coverage.
- Increase the TAM bandwidth: a TAM based on a traditional bus usually

has a fixed bandwidth. In case the NoC is reused as a TAM, the increase of bandwidth for test data implies less bandwidth for user applications and the probability of a drop of system performance due to a bandwidth bottleneck.

3.4 Design for Dependability

Design for test (DfT) has been known for more than half a century. It is a combination of techniques which add to the testability feature of an integrated-circuit product. Similarly, the concept of Design for Dependability (DfDEP) has been proposed in a previous study [Edmo 90]. We use the DfDEP concept in this thesis with the aim to improve the dependability of today's highly complex IC such as an MPSoC. To summarize the discussions in the previous sections, three main features of our approach of the DfDEP concept can be concluded:

- Processor core redundancy at the proper hierarchical level to improve system reliability;
- Self-test and self-repair infrastructures to make the target system maintainable;
- Real-time measurement and optimization of related operations to increase system availability.

An example of design-space exploration of a dependable MPSoC is illustrated in Figure 3.9. The three dependability attributes are listed in the center of the figure. The parameters which have an influence on a certain attribute are depicted in the surrounding blocks. For instance, parameter (2) the no. of test iterations refers to the number of the dependability tests determined by the total number of cores and the number of cores tested simultaneously each time. This number indicates how many times tests will take place; thus it has a direct influence (represented by the arrowed line) on the availability attribute. The influence of TAM bandwidth (1) and the volume of the test data (3) on availability has been shown in Equation 3.15. Introducing spare cores and using a proper redundancy organization (6) such as graceful degradation and standby redundancy can make the MPSoC maintainable at core level. A good run-time mapping algorithm (7)

can not only make the application remapping easier (maintainability) but also quicker (availability).

The parameters listed in the surrounding blocks can be tuned and decisions can be made at design-time by the system designer. But one should also be aware how tightly these parameters are linked with one another. For example, the number of spare cores (5) and self-test fault coverage (4) will directly impact system reliability. In addition, the parameters can have an influence on each other. For example, a high self-test fault coverage can result in high reliability. However, a high fault coverage self-test usually requires a large amount of test data (3), which will increase the mean time to detect a potential fault and harms system availability. Therefore, it is the designer's objective to carefully balance the dependability parameters according to the actual user requirements. A sound DfDEP approach should cost as little as possible resource overhead (e.g. silicon area) while satisfying the dependability specifications.

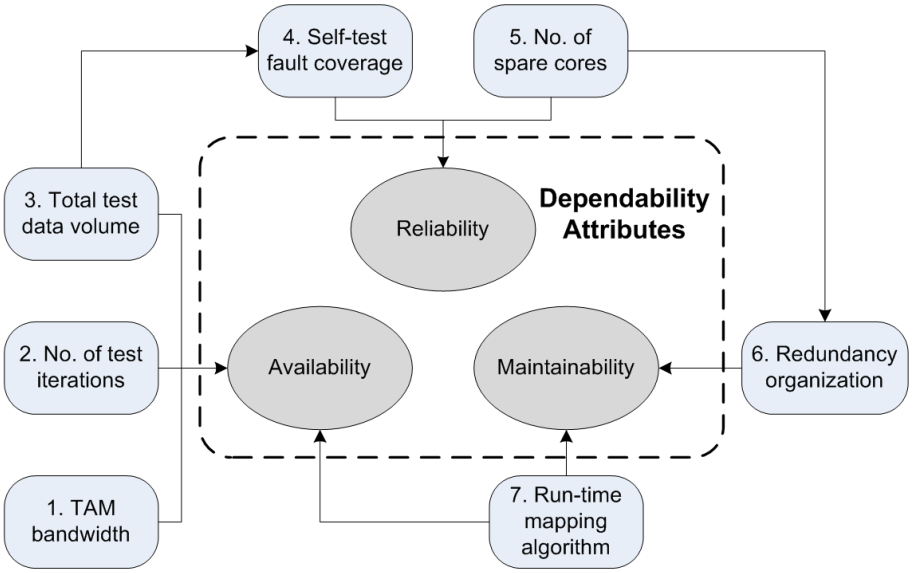


Figure 3.9: Dependable MPSoC design space exploration

As such, the DfDEP endeavors can be conceived as an attempt to solve a series of optimization problems in which one needs to achieve the highest level of dependability within the limit of a given budget (silicon area, power, cost and design time). Using a top-down approach, one can break a system dependability

specification into sub-items and deal with them separately. Consider the following example. One of the dependability specifications for a radar system is 95% reliability after 10 years. Assume this radar system comprises of five basic sub-blocks connected in a serial fashion and each sub-block has the same reliability R. Using equation 3.7, a very rough estimation of the reliability requirement on each sub-block can be calculated as $R = \sqrt[5]{0.95} \approx 99\%$, after 10 years.

In this case one needs to examine whether each sub-block can be built within the budget limit and achieve the required 99% reliability in 10 years. If less reliable components (usually cheaper) are used to build the sub-blocks then redundancy should be added to improve the system reliability.

Ultimately, an optimal DfDEP approach is one that reaches the highest level of system dependability within the budget constraints. In the next chapter, a generic dependability approach is proposed, which aims to enhance the dependability of an MPSoC.

3.5 Conclusion

In this chapter, the definition of system dependability and its three main attributes being reliability, availability and maintainability have been introduced and linked with the design parameters of an MPSoC. It is shown that adding spare cores in a homogeneous MPSoC is a favorable method to improve system reliability. It is revealed that increasing the test fault-coverage will contribute to the increase of system reliability as well. On the other hand, the test-data volume and TAM (Test Access Mechanism) bandwidth will determine the mean time to detect a fault, which is an essential parameter for system availability. By isolating faulty cores from the library of usable resources using resource-management software and remapping the application to fault-free processor cores using the run-time mapping software, the maintainability for an MPSoC can be effectively achieved.

Chapter 4

The Dependability Approach

ABSTRACT - *In Chapter 3, it has been revealed that the key starting point for MPSoC dependability enhancement is the detection of any emerging faults in the system. In this chapter, an example MPSoC platform is adopted to demonstrate our dependability approach. Its key innovation lies in performing a dependability test in the MPSoC at application runtime. This is made possible by carrying out the test using the NoC as a test access mechanism. Because the NoC is shared by the dependability test and user applications, strategies which aim to minimize the impact of the dependability test are also explored in detail.*

This chapter is organized as follows. In Section 4.1, two basic elements of our dependability approach, namely MPSoC self-test and self-repair are introduced. Section 4.2 gives a thorough review of existing integrated circuit self-test methods and presents our dependability test architecture for a NoC-based homogeneous MPSoC. Trade-offs related to the self-test method are also discussed. In Section 4.3, essential dependability infrastructures and software required by the dependability test are introduced. Section 4.4 discusses the scheduling and planning of the self-test and their impact on system performance. An introduction of

Parts of this chapter have been presented at the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN2008) [Kerk 08], the 12th and 13th Euromicro Conference on Digital System Design, Architectures, Methods and Tools [Zhan 09b, Zhan 10b].

the software-based NoC testing scheme is given in Section 4.5. Conclusions are presented in Section 4.6.

4.1 The dependability approach

The dependability approach for an MPSoC is a collection of techniques and infrastructures which are aimed at the improvement of its dependability. In Chapter 3, the main dependability attributes of an MPSoC have been analyzed and the methods for dependability enhancements have been discussed. Theoretically, the dependability of an MPSoC can be enhanced by eliminating any faulty parts in the MPSoC and remapping the application to fault-free resources. Hence the dependability approach proposed in this thesis comprises two major parts: self-test and self-repair. In the context of a homogeneous MPSoC, cores or processing tiles will be treated as the basic unit for fault detection, system reconfiguration and the remapping of applications.

4.1.1 MPSoC self-test

The self-test of an MPSoC refers to the detection of faults in the system at the core level. For safety-critical and mission-critical applications, a high level of availability is often required which means the system needs to maintain its functionality while a test is carried out. As such, the system needs to be operational (online) while the test is carried out. Online testing is favored over offline testing for fault detection as it can be performed without shutting the system down. Note that sufficient time is still required to perform the necessary test.

Depending on the status of the system while a test is carried out, online testing can be further divided into two sub-categories, being concurrent and non-concurrent online testing. In concurrent online testing, the test is carried out simultaneously while the system is performing its normal functions. Classic concurrent online testing methods include hardware redundancy such as TMR, and time redundancy such as performing key operations more than once. Other options are information redundancy such as code-based error detection, or the use of various sensors which monitor the important parameters of the system (current,

voltage, temperature, etc.) to judge or predict its correctness [Nico 98]. Full availability of the system during the test is the obvious advantage of the concurrent online testing method. However, the usually high DfT overhead (silicon area, power dissipation and time) makes it not always suitable for fault detection of an MPSoC.

In non-concurrent online testing, the tests are performed when the system is in idle state. When the system is occupied by a new application, the test can be paused or interrupted. Previous research proposed software-based *functional* test methods for embedded processor cores [Kran 03, Xeno 03]. The benefit of the software test approach is that the test can be carried out using existing components; hence no extra hardware parts have to be added to the system and thus no silicon area overhead results. The disadvantage is that its fault coverage is usually lower than a comprehensive *structural* test. On the other hand, structural tests can usually guarantee a very high fault coverage (100% or close to 100%) with regard to the targeted type of faults (e.g. stuck-at faults), but it requires the circuit under test be brought into a non-functional mode.

In a homogeneous MPSoC, it is often the case that not all the computing resources (cores) are required by applications during a certain time frame. Therefore tests can be performed on the idle cores while the applications run on the other ones. If the test is finished, the computational load can be shifted to the cores which have been tested and other cores can be freed and tested. To realize this transition, additional hardware might be required for e.g. storing the status (internal register values) of the involved operational processor cores. This process can be repeated until all cores have been tested. The test can be invoked by the user or triggered on a periodic basis. This test approach is especially suitable for applications which do not need the immediate detection of faults after their occurrence. In this thesis, the traditional scan-based structural test is modified and used as a non-concurrent online testing approach for the self-test of homogenous MPSoCs. Our new method features the high fault coverage of structural tests and can be performed at application run time, which can keep the system availability at a high level. This test approach will be referred to as *dependability test* in the remainder of this thesis.

4.1.2 MPSoC self-repair

In case an MPSoC operates in the field, it is usually difficult to physically repair it once a fault occurs. In this thesis, self-repair of an MPSoC is defined as the isolation of the faulty core, the reconfiguration of the remaining ones and the remapping of the application to fault-free resources. By treating the basic elements in an MPSoC such as cores and NoC segments as resources, resource-management software can maintain a table of all the fault-free resources in the MPSoC and isolate faulty ones from it based on the dependability test result. Run-time mapping software then remaps the application to fault-free resources. The run-time mapping software and the resource-management software have been developed within the CRISP project and details about the software design and implementation can be found in [Ter 10, Ter 11]. This thesis will not further explore the run-time mapping software design scheme.

With this self-repair approach, it can be ensured that the application is always running using the fault-free resources. However, this approach has its limitations. For a graceful degradation system, faulty cores can lead to the degradation of system performance. For a standby redundancy system, first the workload of the faulty core will be shifted to other working cores if they are not fully occupied. Then the spare cores will be used if the workload is too heavy for existing working cores. Finally, non-primary tasks can be removed from processors for primary tasks from failing processors, thereby reducing the QoS. If the QoS of a graceful degradation system drops under an acceptable level, or the standby redundancy system has depleted all its resources, the MPSoC becomes unrepairable and will be regarded as a faulty system at that moment.

The dependability test, reconfiguration and remapping operations are performed in the background, while the MPSoC is operational in the field. Therefore all self-diagnosis and self-repair hardware and software resources are intended to be on-chip to enable a stand-alone operation in combination with a general purpose processor.

In summary, the dependability approach proposed in this thesis is to perform dependability test at application runtime for core-level fault detection and to make use of resource-management software and run-time mapping software for

core-level system repair. Similarly, the NoC can be tested in a software way and any faulty segments can also be isolated from the system. The focus of the dependability approach is self-test and self-repair. We assume that a core can be stopped in between two task iterations and that the state of a task is empty between these two iteration; system rollback mechanisms will not be explored in this thesis.

4.2 Dependability test concept and architecture

4.2.1 Previous research on MPSoC testing

For decades, scan-based structural test has been the most popular DfT scheme used in the semiconductor industry for the test of digital logic circuits. To implement a scan design, one replaces all the basic storage elements (registers) in a sequential digital circuit with scan registers and connects them into one or more scan-register chains. The scan chains can be used to shift the test stimuli into the circuit and shift the test responses out. The introduction of scan chains can greatly improve the controllability and observability of the original circuit [Bush 05].

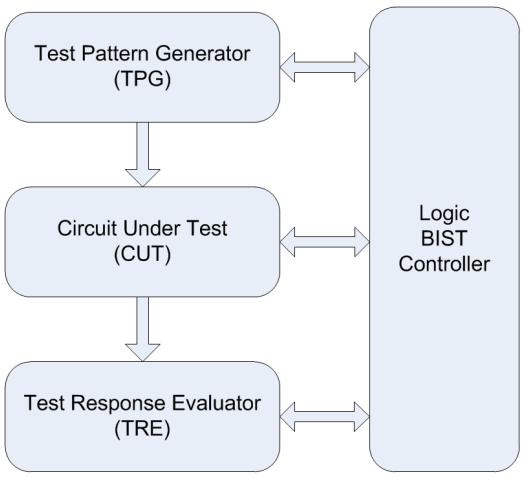


Figure 4.1: A common logic BIST architecture

As the complexity of modern VLSI increased, scan-based logic Built-In Self-

Test (BIST) was developed to reduce the cost of using expensive external Automatic Test Equipment (ATE) and to provide on-chip and on-site testing possibilities. A common logic BIST architecture is illustrated in Figure 4.1. It usually comprises of a test pattern generator (TPG), a test response evaluator (TRE) and a logic BIST controller. The TPG can be implemented as a pseudo-random pattern generator (PRPG) which generates and applies pseudo-random test-vectors to the Circuit Under Test (CUT). Multiple-Input Signature Registers (MISR) can be used to construct the TRE which compacts the test responses of the CUT into a signature. The resulting signature can then be compared to the simulated reference signature to determine the correctness of the CUT. The BIST controller manages the activities of the TPG, TRE and the CUT and generates a pass/fail signal once the test is completed.

The communication channel between the CUT and the logic BIST circuit is defined as the test access mechanism (TAM). Test stimuli from the TPG, test response to the TRE and control signals to the CUT are all transported via the TAM. Dedicated wiring is a natural choice to implement the TAM. In previous research, a structured Test Bus framework has been proposed in [Varm 98]. In [Mari 98], the authors introduced the TestRail concept to provide access from the chip pins to the CUT. Despite their reliable access to the target CUTs, extra silicon area overhead limits the use of a dedicated TAM in very complex systems. Hence the reuse of the existing resources in a chip as the TAM is an attractive alternative. For example, it has been suggested that the existing AMBA bus can be reused as a TAM [Harr 99].

A modern MPSoC often consists of a number of embedded cores for data processing. The interconnection between the cores has evolved from a traditional bus to an on-chip network, referred to as the Network-on-Chip (NoC). The NoC gained popularity as a result of its stable performance, flexible configuration and its scalable structure [Beni 02]. Research on how to use the NoC as a TAM has also been carried out [Cota 03]. Test stimuli and test-response data can be packed into NoC packets, which are then transported via the NoC to and from the cores under test. Special wrapper logic is needed to separate the cores under test from other parts of the system during the test. This so called *test-wrapper* can switch the cores from the normal functional mode to test mode and vice versa.

A standard test-wrapper design approach has been included into the IEEE 1500 standard for embedded core test [IEEE 05].

This modular way to test the embedded cores in an MPSoC is often referred to as core-based testing. Apart from the TAM and wrapper design, the order in which the cores in an MPSoC are tested and the planning of routing of the test data also have an impact on the total test time and test-power consumption. Different test-scheduling schemes have been proposed with the focus on e.g. minimizing the test time [Kora 02] or meeting specific power constraints [Zhao 03].

The identical core architecture of a homogenous MPSoC enables different test strategies from the previous research. Special requirements of the online dependability test also brings in different test constraints to satisfy. These issues will be discussed in detail in the following sections.

4.2.2 Dependability test architecture for a NoC-based homogeneous MPSoC

A homogeneous MPSoC contains a large number of structurally identical cores. Test parallelism can be achieved by broadcasting the same test stimuli to multiple cores simultaneously instead of applying them to each core individually. For example, the two-core UltraSPARC processor has been tested in such a way that the test-vectors are applied to the two cores concurrently in a scan lockstep mode. Then the test responses scanned out of the two cores are compared within the processor and a mismatch between any response bits is indicated by a fail pin [Paru 02]. In [Maka 07], the authors introduced the test of the Vega2 processor using the AZSCAN architecture. Multiple cores are also tested in parallel and the test response is compared on-chip with reference test responses. In our earlier research, we have proposed to use the NoC as a vehicle to transport the test responses of identical embedded processing tiles and compare them with a golden reference tile (a known-good tile) [Kerk 08].

The on-chip comparison of core test responses is possible in a homogeneous MPSoC owing to its unified structure. In fact, the comparison of the test responses can be carried out on any number of identical cores given that they are

brought to the same state (e.g. by performing a soft reset) before the test starts and the same test-vectors are applied to them in the same sequence. This mutual comparison scheme has the obvious advantage over the comparison with the reference test responses in that it saves the ATE or on-chip storage for the reference test responses. It also has the big advantage that a signature comparison requires a whole test to be performed, instead the mutual comparison scheme can have the test data flow paused while the comparison takes place. Mutual comparison is also superior to the response compaction and signature analysis approach as a result of the reduced design effort and hardware overhead. It should also be noted that the test-response compaction is a lossy process which means information is always lost during the compaction [McCl 85]; this can result in a reduced overall test fault coverage. Whereas no information will be lost in a mutual comparison of raw test responses.

The block diagram of an example MPSoC with 64 processing tiles is illustrated in Figure 4.2, showing the main functional blocks of the NoC based MPSoC [Zhan 09b]. The 8×8 array of tiles (squares) are responsible for most of the data processing tasks. The tiles are interconnected by the NoC (lines). The intersection points of the NoC lines are the NoC routers (R in Figure 4.3). A general purpose processor (GPP) is included in the system to handle basic control tasks. To facilitate the dependability test, an Infrastructural IP (IIP) is incorporated into the MPSoC to function as the on-chip test-pattern generator (TPG) and the test-response evaluator (TRE). In the following part of this thesis, this IIP will be referred to as the Dependability Manager (DM) because it is added into the system for its role in dependability improvement. The DM interfaces the NoC via a standard network interface identical to the ones used by the processing tiles and the GPP. It can broadcast test stimuli to the tiles under test and collect their test responses via the NoC.

The absolute minimum number of cores needed for the test-response comparison scheme is two and in case of any response mismatch, a third core is needed to determine which core is faulty. Majority-voting can be used to determine the faulty core if three or more cores are tested together, assuming that only one core fails each time a test takes place. Note that the voting circuit has to be fault-free to realize the majority-voting scheme; a number of self-checking voter designs

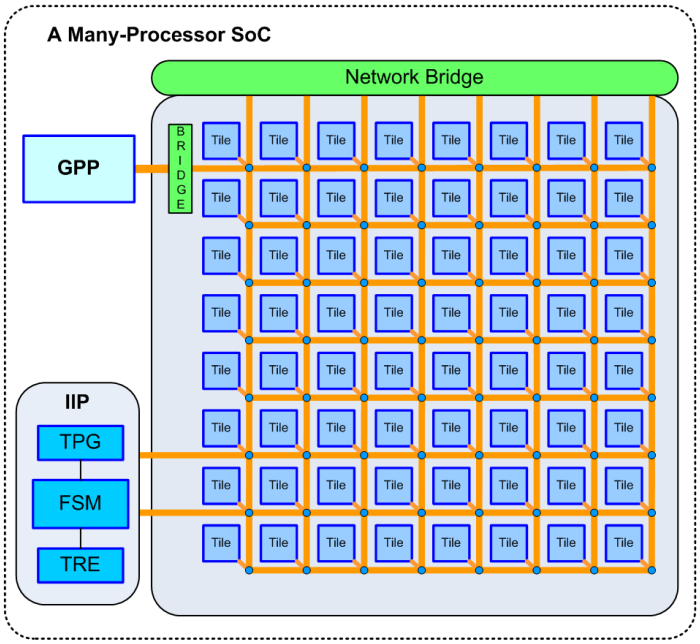


Figure 4.2: A NoC-based MPSoC with 64 processing tiles, GPP and dependability features.

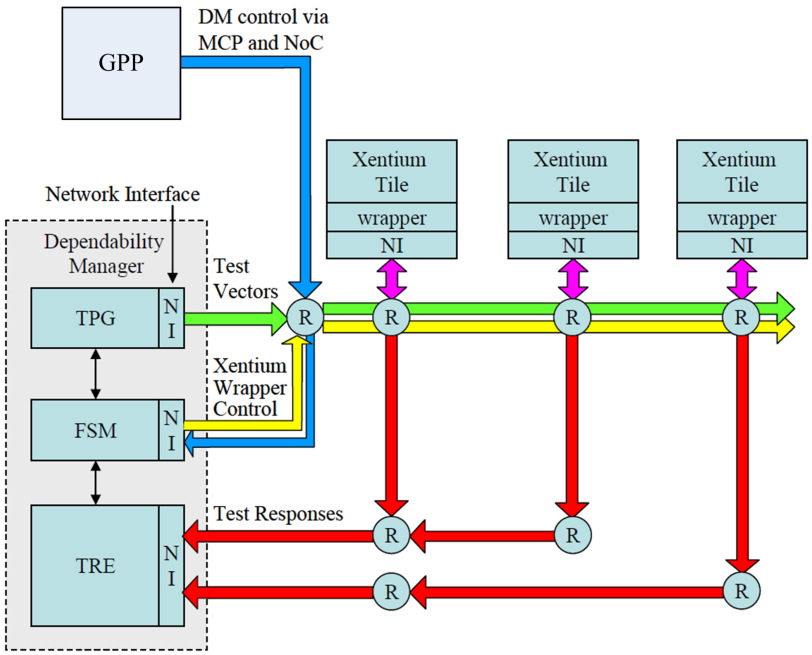


Figure 4.3: Signal flow of the dependability test; three processing tiles are simultaneously tested. R are routers.

have been proposed in literature [Caze 04]. Figure 4.3 highlights the signal and data flow while using the DM to perform the dependability test on three identical processing tiles. Deterministic test patterns can be produced by the TPG in the DM and broadcasted to the three tiles under test via the NoC. The TPG does not store the raw test patterns internally due to storage limits. Instead, it only stores a small amount of seeds which can be expanded into deterministic test patterns using a test-pattern compression technique such as reseeding (more details are discussed in Chapter 5). After one complete test-vector has been shifted into the scan chains of the target tiles, test responses will be generated and returned to the TRE of the DM via different paths of the NoC. Bitwise test-response comparison then takes place in the TRE. If one tile generates different test-responses from the other two, it will be determined as faulty. The result of the dependability test and the ID of the faulty core will be reported to the finite-state machine (FSM) within the DM. This FSM communicates also with the GPP via the NoC. Dependability software in the GPP will take care to collect the dependability test result and use it to guide the resource-management software and the run-time mapping software in the GPP to take subsequent actions such as faulty-core isolation and application remapping.

The previous paragraph describes the basic architecture of the dependability test. Note that only three cores are needed during each test. Other cores in the MPSoC can run user applications simultaneously as the test takes place. However, conflicts can occur, if the dependability test competes with user applications for the use of the same NoC bandwidth. In Section 4.4, care is taken to avoid this conflict by carefully scheduling the test and e.g. incorporating a pause/resume feature to test related infrastructures. As such, the dependability test can be regarded as non-concurrent online testing.

In addition to its architecture, requirements and trade-offs of the dependability test are discussed in the next section. Infrastructures related to the dependability test can be divided into two groups: the DM itself and other parts such as the NoC reused as a TAM, the core test-wrapper and related software. Details about the DM itself, such as design of its internal blocks, its network interface and its overall control mechanism will be presented in Chapter 5. While the other parts such as the usage of the NoC as a TAM and the tile-wrapper design will be

explained in Section 4.3 of this chapter.

4.2.3 Dependability test requirements and trade-offs

In this section we review the main requirements for our dependability approach. The following requirements and trade-offs have to be considered while devising the dependability test scheme.

4.2.3.1 Fault model and fault coverage

The type of faults to be dealt with in this thesis is confined to permanent faults resulting from chip-aging effects. Notice that other type of faults can also be detected by making some modifications to our approach. For instance, transient faults can be detected by repeating the test. As the embedded cores and the NoC occupy most of the silicon area in an MPSoC, the dependability test will focus on these parts. Scan-based structural test will be adopted to test the logic part of the cores and the core internal memory will be tested using its own memory BIST engine. Both tests need to be coordinated by the DM. A software-based test approach will be used to test the NoC while the system boots up.

As analyzed in chapter 3, the fault coverage obtained from the dependability test has a direct impact on the reliability improvement of the system as well as the test time and silicon overhead. Thus the DM-TPG needs to generate deterministic test-vectors instead of pseudorandom patterns and should achieve a high fault coverage under the premise that the design does not violate silicon area limits.

4.2.3.2 Timing considerations

One of the important measures of the dependability test efficiency is the fault detection latency, which is defined as the time between the occurrence of a fault and the detection of the fault. Assuming the dependability test is performed on a periodic basis, then the worst-case fault detection scenario is that a fault occurs right after one test is completed and this fault is only detected by the last set of test-vectors of the next dependability test. The worst-case fault detection latency is the sum of the dependability test occurrence period (T_p) and the full

dependability test time (T_t). The fault detection latency will decrease when T_p and T_t become shorter. Since T_p is usually much longer than T_t (order(s) of magnitude), the fault detection latency is primarily determined by the dependability test occurrence period. However, it should be noted that a higher test occurrence frequency will also cause an increase of power dissipation and decrease of system performance.

4.2.3.3 Silicon area overhead

The silicon area overhead of the dependability approach is mainly contributed by the incorporation of the DM infrastructural IP and the wrappers for each processor tile. After discussing with our industrial partners of the CRISP project, an acceptable area overhead was determined that both the DM and wrapper should be smaller than that of one processing tile (a Xentium hard macro occupies around 1.88 mm^2) in the MPSoC.

4.2.3.4 Performance loss

The dependability test can cause a decrease of the system performance in two aspects being the usage of cores for test and the occupation of the NoC bandwidth for test-data transport. If the dependability test is performed on the idle cores in the system using unoccupied NoC segments, the overall system performance will not be affected. Otherwise, a performance decrease is expected. Since the dependability test is intended to be performed at application run-time, the minimization of the system performance overhead is of crucial importance. Proper test planning and scheduling can help to retain the performance. This will be explained in detail in section 4.4.

4.2.3.5 Scalability and Adaptability

The dependability test scheme is a scalable method by nature. As the test data flow can be mixed with the functional data flow in the NoC, the test can be regarded as a special application which requires the cores to be in a special mode (test mode). Therefore for a similar NoC-based MPSoC with the same type of

embedded cores but a different core instance number or a different topology, no change to the proposed dependability test architecture is necessary.

The proposed test scheme can also be easily adapted to an MPSoC with a new type of processing core. In this case, the TPG of the DM needs a redesign to generate proper test patterns for the new core. This requires the DM to be designed in a modular way for the ease of the replacement of the TPG. In chapter 5, the design of a tool chain for the automatic generation of the DM-TPG for new embedded cores will be explained. In addition, some minor changes with regard to the core wrapper will be necessary to accommodate the new I/Os.

4.3 Dependability test infrastructure

The dependability manager IP, the core test-wrapper and the NoC reused as a TAM are the three main structural elements of the dependability test scheme. The usage of a packet-switched NoC to perform a dependability test and the design of a generic core test-wrapper will be explained in more detail in this section.

4.3.1 Background of the NoC

4.3.1.1 Terminology and general architecture

Network-on-Chip, in analog to terms of the macro-world computer networks, is a recent evolution of on-chip communication fabrics. The NoC is gaining more and more popularity in the MPSoC paradigm due to its high throughput, scalable architecture and high level of parallelism. Figure 4.2 already showed an example MPSoC with its embedded processing tiles interconnected by a NoC. A closer examination of that example reveals the major building blocks of a NoC, being routers and links.

Routers (R in Figure 4.3) are switches which direct the data to various directions according to the routing information. Links are the physical wires which connect the routers into a certain topology. The way how a message travels through the topology is defined as its routing algorithm. If two messages compete for the same NoC resources (router, link), an arbitration mechanism is necessary.

A detailed view of the NoC and its core connection part is illustrated in Figure 4.4. The processing tile communicates with the NoC via a dedicated Network Interface (NI), which packs the data payloads with the destination information and sends them to the router connected to the core. Other functional IPs such as a general purpose processor (GPP) or the dependability manager can also access the NoC via *dedicated* network interfaces.

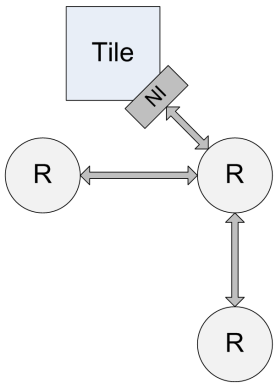


Figure 4.4: NoC router and processing tile connection. R: router; NI: network interface.

4.3.1.2 Basics of NoC flow control

Data generated by the processing cores are encapsulated with routing and control information as NoC packets by the network interface (NI) before injected into the NoC. The NoC flow control mechanism determines how the traffic is managed for the packet data flow. Mainstream flow control approaches fall into two categories, packet-switching and circuit-switching, depending on whether the flow control is buffered or bufferless [Wolk 09].

A circuit-switched NoC creates a dedicated channel between the sender and the receiver of the information before the communication starts. The NoC resources along this channel will be reserved and are not usable by any other party until the end of the communication. Thus there is no need to buffer the communication as long as the sender does not exceed the bandwidth limit. A circuit-switched NoC benefits from its simple architecture and guaranteed data transport

latency. Its disadvantage is the apparent lack of routing flexibility if multiple communications take place in parallel.

On the other hand, a packet-switched NoC does not need a reservation of a communication path in advance. Data packets can be buffered in the NoC routers if there is congestion in the subsequent route to the destination. Although more flexible than the circuit-switched approach, packet-switched NoC needs a large number of buffers in its router design. One of the widely accepted packet-switched flow control schemes is virtual channel (VC) flow control [Dall 87], which allows several NoC data flows to share the same physical channels using Time Division Multiplexing (TDM). The data stream is labeled with unique so-called VC identifiers, so that different data flows going into the same router port are buffered in separate parallel buffers. Our discussions will be based on the reuse of the NoC as a TAM in a packet-switched NoC using VC flow control.

As a 2-D meshed topology is the common NoC topology for a homogenous MPSoC [Rhee 04], hence the following discussion will be based on a NoC with 2-D meshed topology.

4.3.2 Reuse a GuarVC NoC as a TAM

Guaranteed Service (GS) and Best Effort (BE) are two important NoC service types [Wolk 09]. Streaming applications often require the NoC to provide a predictable performance, such as guaranteed throughput and bounded latency. GS can be used to serve these applications. For less important communications, such as the dependability test, BE service can be used.

The GuarVC NoC is a packet-switched NoC architecture which uses the VC flow control mechanism. Its router can provide both GS and BE services [Wolk 09]. The architecture of a typical GuarVC router is shown in Figure 4.5. As depicted in the figure, the router has five data ports in five directions (North, East, South, West and Local). Each data port includes an input port and an output port; each input/output supports four time-multiplexed virtual channels. Each channel contains several internal buffers for temporary data storage. A virtual channel allocator matches the VC of the input port with the VC of the output port based on the destination of a data packet and the routing algorithm

being used. The crossbar switch in the middle of the router determines the duration of the time slot that is allocated to an established channel. The channel with a larger time slot has a larger bandwidth.

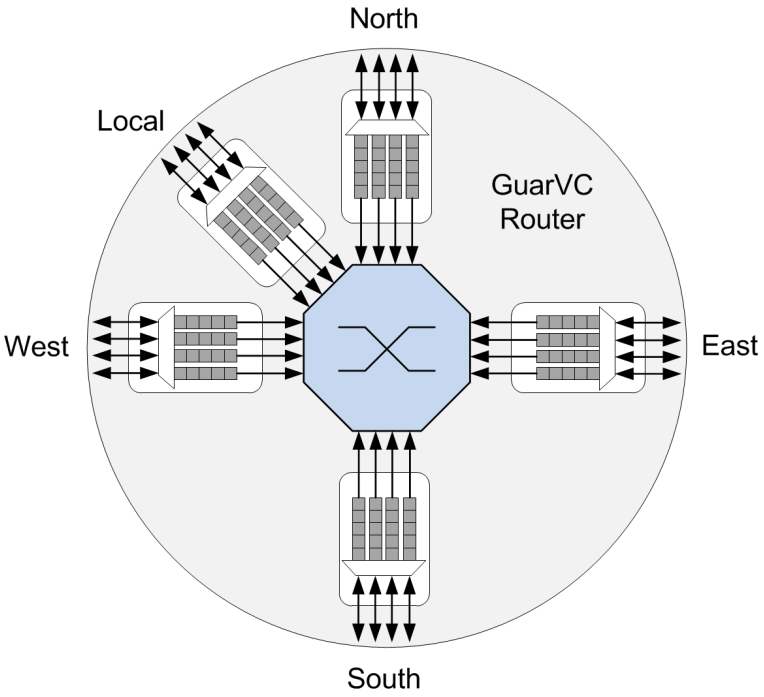


Figure 4.5: A typical router architecture of a GuarVC NoC [Wolk 09]

Each NoC port comprises some data lines and control lines. For the sake of simplicity, only the data lines (arrowed lines) are shown in Figure 4.5. The data lines consist of 32-bit data and a 3-bit flow control digit (flit) ID. The control lines comprise of 2-bit VC selection (4 VCs in total) and a 4-bit VC buffer status signal (one bit for each VC). The data signals form the so-called flow control digit, which is the basic unit transported by the NoC in each clock cycle. The structure of a flit is shown Figure 4.6. A control field Flit ID is attached to the beginning part of each flit to identify its property, followed by a 32-bit data payload. Four important flit types can be identified: *Header*, *Tail*, *Idle* and *Payload*.

A complete NoC data packet usually consists of a large number of flits. A common NoC data packet structure is shown in Figure 4.7. A data packet starts with a number of header flits which contain the routing information to its destination. The payload flits carry the actual data that need to be transported. The

Figure 4.6: General NoC flit structure

data flow is terminated by a tail flit which releases the virtual channel after the communication. If a packet transport starts, each router consumes a header flit and passes the remaining flits of the packet into the next router designated by the header flit it consumes. When the data flow arrives at the destination router, all routing information has been consumed. The last router sends the data payload to the IP core connected to it via the network interface.

Figure 4.7: A general NoC packet structure. The header, payload and tail parts each consist of a number of NoC flits of corresponding type.

For a GuarVC NoC with 32-bits wide data lines, the size of the flit data payload is 32-bits. An optimal test-data loading and shifting scenario can be achieved if the associated cores have 32-bit parallel scan chains. Hence in every clock cycle, one NoC flit is delivered to the embedded processor core and unpacked, and one bit of the test-vector is shifted into each of its 32 scan chains. The same applies to the test responses when they are shifted out of the scan chains.

As identical cores are simultaneously tested in the dependability test, the same test stimuli have to be *broadcasted* to multiple embedded cores at the same time. Therefore, we have developed a dedicated multicast protocol for the NoC in order to transfer the same test-vectors to multiple targets. A multicast packet is a special packet which can split its route and copy its payload to different paths. In this way, the test-vectors generated by the DM can be sent to multiple cores under test. As for test-response collection, the DM needs to read back the test responses from the cores under test via three *separate* paths.

In order to cope with the situation in which the NoC has to be shared between the application data and the test data, we have innovated a *back-pressure* alike flow control mechanism which is of key importance for the scheduling of the

dependability test [Zhan 10b]. It works as following. In case a route used to transport the test data is occupied by an application with a higher priority, the test data will start to accumulate in the local buffers of the router. If the local buffers become full, a "buffer full" signal will be issued to preceding routers along the planned path. This process will continue until the source of the test data (i.e. the TPG of the dependability manager) is notified; it will then pause its test data generation. More details on the dependability test scheduling at application run time will be covered in Section 4.4.

4.3.3 Core test-wrapper

4.3.3.1 Introduction

The basic function of the core test-wrapper circuit is to provide an interface layer between each embedded core and the SoC. It is located between the corresponding I/O ports of the core and its network interface. The following two important test-wrapper operational modes can be identified: the functional mode and the test mode. If the core test-wrapper is set to functional mode, the embedded core can perform its normal function and communicate with its network interface as the wrappers are transparent. If the test-wrapper is set to test mode, it isolates the core from the rest of the system and puts the core under test by delivering the incoming test data to the corresponding test ports (namely the primary input and scan input ports) of the core.

The IEEE standard 1500 for embedded core test has been used to guide the design of core test-wrappers [IEEE 05]. In Figure 4.8, a typical IEEE Std. 1500 wrapper component is shown. It consists of a mandatory wrapper serial port and three groups of registers being the wrapper instruction register, wrapper bypass register and wrapper boundary register. Test data and wrapper control instructions can be shifted via the serial port into the wrapper boundary register and instruction register respectively. The test-wrapper for several embedded cores can be daisy-chained together and the cores which do not need to be accessed can be bypassed by directing the test data via the bypass register. It is also briefly mentioned in the standard that an optional wrapper parallel port can be added to the wrapper design for faster data transport.

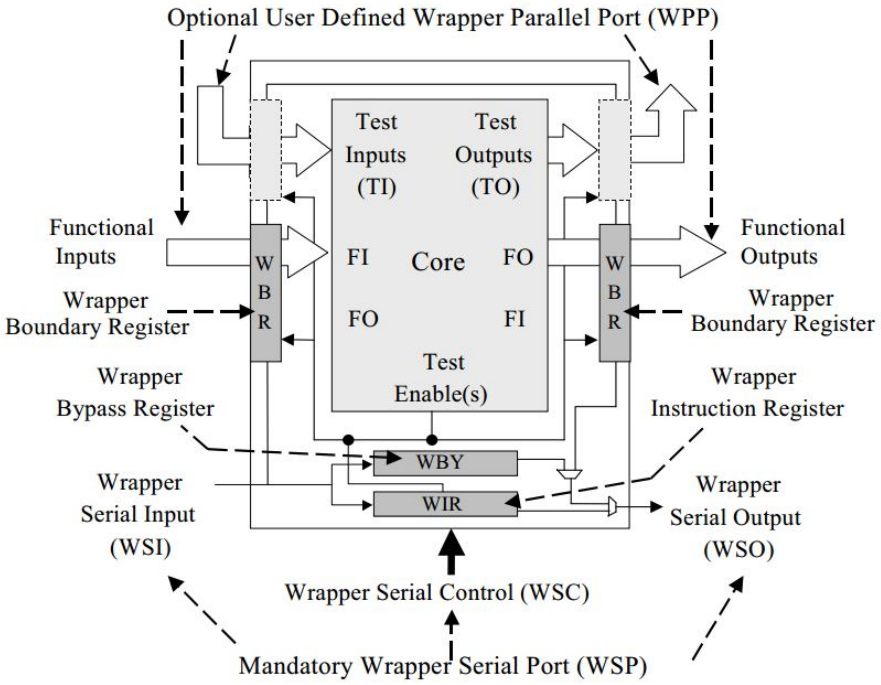


Figure 4.8: Standard IEEE 1500 wrapper components [IEEE 05]

Since the NoC-based dependability test uses a different TAM approach as compared to the IEEE Std. 1500, a new wrapper design has been proposed by us for accessing the embedded cores connected by the NoC [Zhan 10b]. A number of notable changes can be distinguished from the IEEE Std. 1500 test-wrapper. First, the wrapper serial port is completely removed due to the fact that the minimal data unit sent over the NoC is the 32-bit NoC flit. The test data are applied to the embedded core and the test responses are collected from it both in a parallel fashion. Second, control of the wrapper activity is made possible by writing to the configuration register of the wrapper and reading from its status register. The access to the wrapper registers can be performed by the DM via the NoC instead of using dedicated wires. To design the core test-wrapper in a generic way, important test parameters such as the length of the scan chains or the number of primary input/output can be communicated to the wrapper configuration register by the DM at run-time. To reduce the control-signal traffic during the dependability test, a built-in wrapper controller will control the operation of the wrapper and the embedded core once the test starts. For cores with embedded BIST engines for internal memory test, the wrapper controller can initiate the memory BIST and load the test result into its status register. In addition, the bypass register is removed due to the flexible feature that the test data can be routed into the NoC. The external test mode (EXTEST) is also discarded because the TAM (NoC) is tested using a centralized software approach to be discussed later.

4.3.3.2 Core test-wrapper architecture

A block diagram of the proposed core test-wrapper design is shown in Figure 4.9. The main functional blocks of the wrapper and its interaction with the wrapped core are shown. Our core test-wrapper consists of five main components.

- The Scan Input Multiplexer (SIM) is responsible for multiplexing the dependability test scan vector input to the input of the scan chains of the embedded core.
- The Surround Input Unit (SIU) takes care of providing the test-vector input signals to the primary inputs of the core.

- The Surround Output Unit (SOU) captures test responses from the core functional output and shifts them out.

- The Clock Gate (CG) module controls the clock signal of the embedded core which can be halted in case the test stimuli are not arriving in time or if the test responses are not fetched in time by the network interface. Care is taken during post-synthesis layout and clock-tree insertion to ensure that the additional delay introduced by the CG does not cause problems in the communication between the core and other top-level blocks, e.g. the NoC.

- The Wrapper Controller orchestrates the test operations of the core and associated memories, interfaces with the scan chains and BIST engines with the on-chip network, with support from the SIM, SIU, and SOU components.

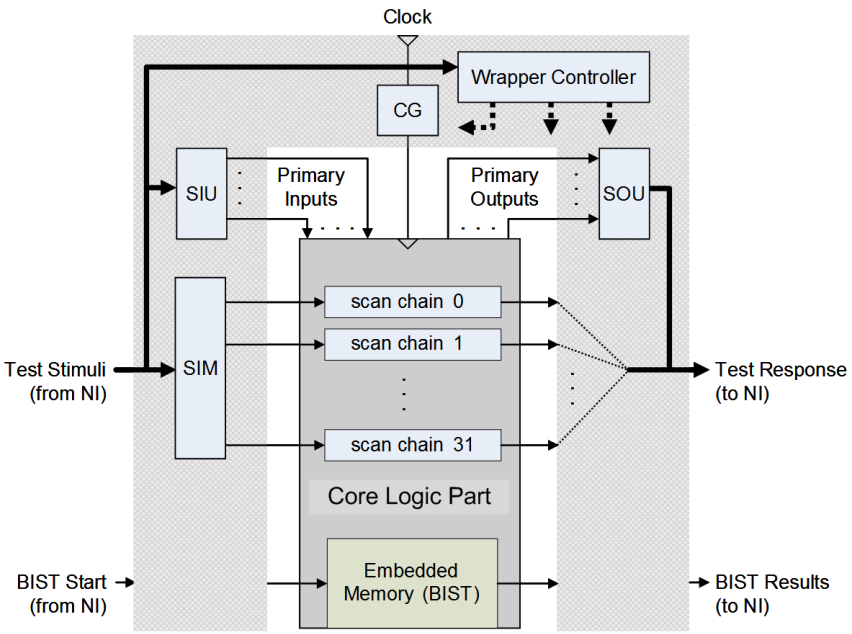


Figure 4.9: Proposed core test-wrapper architecture

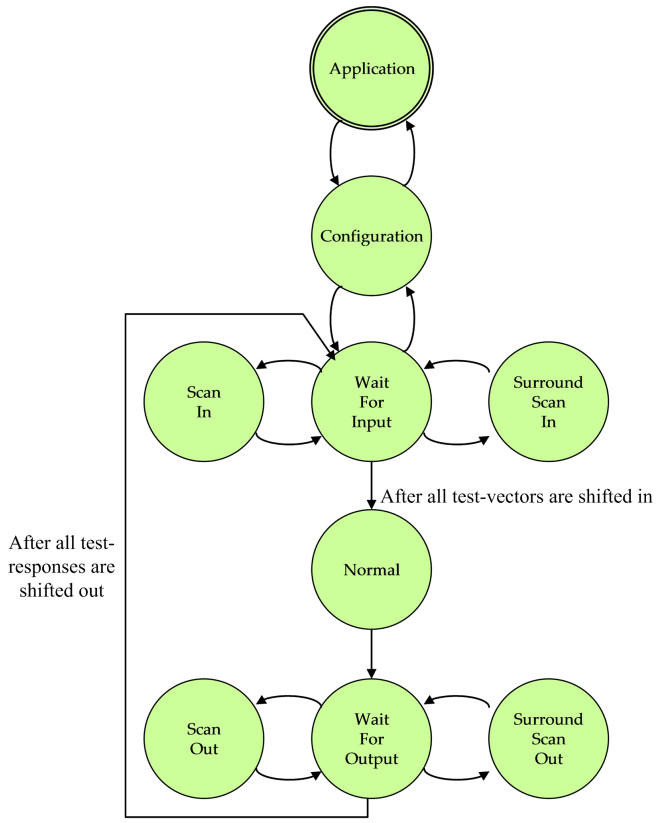


Figure 4.10: State transition diagram of the wrapper controller (some transition conditions have been left out for simplicity).

4.3.3.3 Wrapper control mechanism

The core test-wrapper control interface consists of one *configuration* register and one *status* register. The dependability scan test and BIST operations can be controlled and observed via this interface over the NoC.

The state transition diagram of the wrapper controller is presented in Figure 4.10. The core test-wrapper stays in the *application* state until configured by the DM to perform a test. After entering the test state, it first waits for the test stimuli (the *wait for input* state) transported from the NoC. Then the test-wrapper writes a single, 32-bit test-vector word to SIM in Figure 4.9 and the bits in this word are shifted into the 32 scan chains of the embedded core, one bit for each chain per written word. This process is repeated until the scan-chain is fully filled (the *scan in* state). The same holds for the SIU in Figure 4.9 which applies the primary input stimuli during the *surrounded scan in* state. A counter determines the number of times the writing operation takes place and generates a *write complete* signal (not shown in the figure due to limited space) based on the scan and primary input length information provided at the beginning of the test. After all the input test data have been written to the core under test, the wrapper controller transitions the core to dependability normal mode (the *normal* state), captures the core primary outputs in the SOU (the *surrounded scan out* state) and shifts the scan test-responses out (the *scan out* state).

BIST operations with respect to the memories associated with the embedded core can also be performed, although due to I/O isolation limitations, this cannot be done in parallel with the dependability scan operation. During the BIST operation, the wrapper controller configures all the BIST engines and monitors the execution of the BIST algorithms. Upon completion, the wrapper controller captures the test results from the BIST engines, and makes them available in the wrapper status register. The DM is then able to retrieve these results by reading this register via the network interface.

Figure 4.11 shows the layout of the configuration register of the core test-wrapper. The configuration register resets to the value 0, which disables the scan operation. The register contains eight fields in total. The scan operation is enabled by setting bit 31. This will cause the controller in Figure 4.9 to gate

the clock to the wrapped core and wait for the arrival of test patterns via the network interface. Fault injection (discussed in the next section) is enabled by setting bit 30, and programming the fields Faulty Word Index, and Faulty Bit Index. The Initiator ID bit indicates whether the configuration data comes from the DM or from a user command in the GPP. If the configuration data are sent by the DM, the fault injection fields are masked to ensure that the DM cannot initiate a fault injection operation. The Scan Length, Surround Scan Out, and Surround Scan In fields are used by the wrapper controller to determine when to switch from scan into normal mode, and from scan-out mode to scan-in mode. These parameters are assigned by the DM at runtime before each dependability test starts.

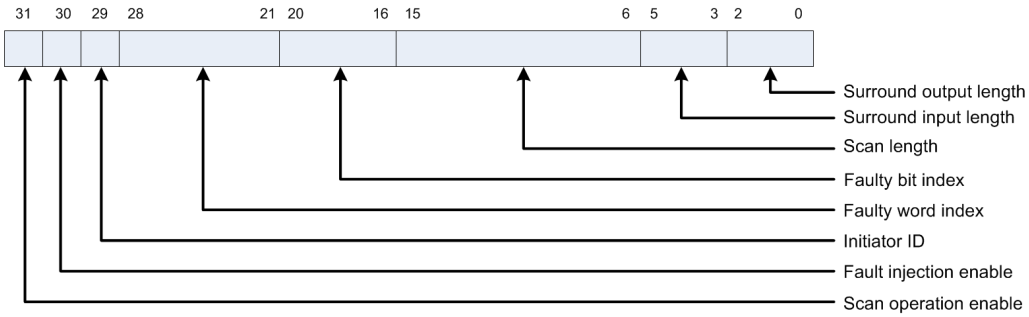


Figure 4.11: Layout of the core test-wrapper configuration register

The status register of the core test-wrapper contains a few fields to report the test result of the internal memory BIST and the current state of the wrapper controller. The DM can check the wrapper status via polling before issuing more commands to it.

4.3.3.4 Built-in Fault Injection Support

In this section we first review several methods for the verification of fault injection. The verification of the dependability concept involves a controlled experiment to check the operation of the embedded cores in the presence of a fault. In practice, it may be very problematic to control a fault in a core or its associated memory. The following options have been analyzed concerning the controlled fault-injection into the core logic part:

- **Hardware insertion**: insert additional hardware at a certain node location inside the core just for fault injection to electronically force a stuck-at fault at a particular node under programmable control. This involves adding non-functional hardware inside the core.
- **Mechanical modification**: make use of focused ion beam (FIB) techniques later on, to remove and/or add connections inside a core, and emulate a stuck-at fault via mechanical actions. This requires a careful selection of the node during layout to enable FIB on wires, as these should be accessible at the top level. Although this is technically feasible, it will place additional constraints on the layout of the core. In addition, this operation suffers from potential risks of destroying the chip and introducing significant costs.
- **Fault emulation via the wrapper**: modify the test responses of a core inside its wrapper under programmable control before it is collected by the DM. This is feasible due to the fact that the test responses of a dependability test are always read by the DM via the core test-wrapper. Most stuck-at faults in the core will result in an inversion of one or more test response bits in the scan chains. Hence, if a bit could be flipped in the output of the wrapper connected to a scan chain, a particular stuck-at fault can be emulated. This requires an instruction for the wrapper to allow the flipping of a bit while reading out scan data. This instruction should come from a third party (e.g. the GPP) and keep the DM unaware of the decision to inject a fault. Hence it guarantees an independent role for the Dependability Manager in the dependability test scenario.

The third option has been chosen and was experimented during the test phase of the MPSoC chip. It leaves the embedded core untouched and thus completely compatible with IP-based SoC design from our industrial partner. It also places the least amount of constraints on the synthesis and layout stages of the MPSoC.

The position of the bit flip determines which fault in the core under test is being emulated and has been provided by the GPP in dedicated software. In this manner, many realistic faults can be emulated in a software-based manner which is excellent to demonstrate the dependability concept in practice.

4.4 Dependability test at application run-time

As discussed in previous sections, one of the important features of our dependability approach is trying to cause as little as possible interference to normal system operations. This is achieved by testing processor cores while they are in idle state using the NoC segments unoccupied by user applications. However, it is not always possible to find unused NoC routes from the DM to the cores under test. Therefore, measures have to be taken to avoid communication conflicts between the dependability test and user applications. In this section, test scheduling and planning in a NoC-based MPSoC are introduced. Modifications to the existing structural scan-based test approach and additional pause/resume functions are proposed to deal with potential communication conflicts. The impact of test planning decisions on system performance has been quantitatively evaluated.

4.4.1 Dependability test scheduling

Test scheduling in a NoC-based environment involves the planning of the path for the test data, the allocation of resources (cores, NoC bandwidth) and the coordination of the test process. The goal of an optimized test scheduling is to carry out the test while meeting specific constraints such as power dissipation, test time or resource constraints.

A dependability test in a NoC-based homogeneous MPSoC can be performed in either a pre-emptive or a none-pre-emptive fashion. In the latter test [Cota 06, Liu 05], dedicated NoC resources such as routers and channel bandwidth are reserved for the test. The test-data pipeline will not be interrupted while the test proceeds. The resources are released only after the test is finished. This test scheduling can be easily realized in a circuit-switched NoC. In a GuarVC NoC, guaranteed throughput (GT) service can be used to achieve a non-pre-emptive scheduling too. An obvious disadvantage of this test scheduling is the fixed usage of the NoC resources. As the dependability test needs to be performed at application run time, the NoC bandwidth is shared between the normal applications and the dependability test. Of course, it is required that the dependability test intrudes as little as possible with the communication in other functional appli-

cations. Fixed reservation of NoC paths can cause routing difficulties for other applications and sometimes it can block the communication channel of untested cores and lead to an application failure.

Hence, pre-emptive scheduling is the preferred scheduling manner. In a pre-emptive test scheduling [Cota 03], the test data can be sent to the destination using best effort service. The test data flow can be interrupted by the communication data of another application with a higher priority and arbitration can be established within the NoC router. The test can be resumed when sufficient NoC bandwidth (e.g. at least one free virtual channel) becomes available again. The pre-emptive test scheduling enables the dependability test to adapt to the varying NoC bandwidth. However, a closer examination of the test data pipeline reveals the difficulty to implement the traditional scan-based test approach using pre-emptive test scheduling.

A traditional scan-based test [Bush 05] begins with shifting test-vectors into the scan chains of the core under test. When the scan chains are fully filled, test stimuli are applied to the primary inputs (PI) and the core runs autonomously for one clock cycle. Then the test result of the primary outputs (PO) are unloaded and the scan chain test responses are shifted out. At the same time, the second test-vector can be shifted in. As such, each time the test response of the Nth test pattern is shifted out for one bit, the test-vector of the $(N+1)th$ test pattern needs be shifted in for one bit. The test time is thus minimized by the parallel scan-in and scan-out operations. This is not difficult to achieve in the case dedicated test lines instead of the NoC are used.

If the same test is to be performed in a NoC environment, careful timing is crucial to ensure the matching between test-pattern application and test-response collection actions. To avoid any data loss, one test-vector flit must arrive at the core under test while one response flit is shifted out. If the input flit arrives early, the previous response flit (waiting to be collected) will be overwritten by the shifting of the scan chain. If the input flit arrives late, a wrong input vector will be shifted into the scan chain. Both can result in an incorrect test result. The fact that several cores are tested together using the same test patterns broadcasted from the DM makes things even more complicated. The dependability test can only succeed if all cores under test and their NoC communications run at the same

pace. Previous research claimed that it is possible to make a successful scheduling by labeling each channel in the NoC with time information to determine when to generate the next test-vector [Cota 04]. In addition to the very complex top-level control, this method hardly works if more than one core is involved in the test because the test data synchronization among the cores under test with unpredictable NoC bandwidth is very difficult.

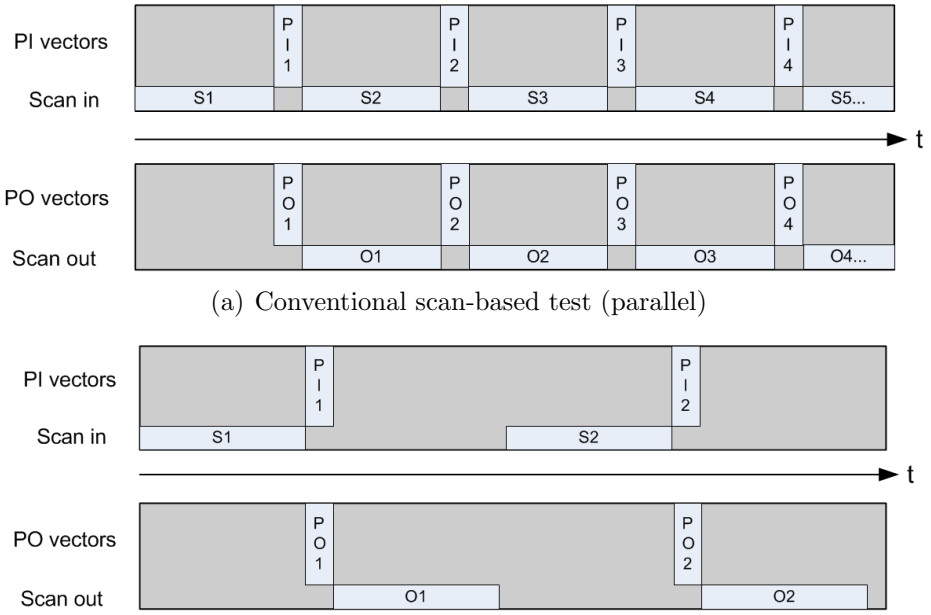
To solve this test scheduling dilemma, we have proposed to decouple the test-pattern application and test-response collection operations. Combined with the back-pressure flow control mechanism of the NoC, the dependability test scheduling can become much simpler. Note that the new scheduling scheme focuses on the co-existence of the dependability test and normal applications hosted by the system. This makes it different from traditional schemes which are often performed off-line and stress minimal ATE test time.

4.4.2 The modified scan-based test

4.4.2.1 Decoupling the scan-based test

As mentioned in the previous section, in a conventional scan-based structural test, the scan-in and scan-out operations are performed in a parallel way to minimize the test time. This approach is feasible because both operations take exactly the same number of clock cycles to complete. In Figure 4.12(a), the flow of a normal scan-based test is shown. On the top are the test-patterns including the scan vectors (S1, S2...) and the primary input stimuli (PI1, PI2...). Below are the test-responses including the scan out response (O1, O2...) and the primary outputs (PO1, PO2...). Scan vector S(N) is shifted into the scan chain while the scan response O(N-1) is shifted out. Top-level test scheduling software needs to synchronize the test-pattern flow and the test-response flow. The blockage of either flow will result in data loss of the other flow and a failed test.

A scan-based test with the test-pattern application and test-response collection operations decoupled is shown in Figure 4.12(b). In this scenario, the DM will not generate a new test pattern until the test responses of the previous pattern have been fully received. The test takes place in a serial fashion. A direct comparison with the regular parallel approach shows that the test time is doubled



(a) Conventional scan-based test (parallel)

(b) Decoupled scan-based test(serial)

Figure 4.12: Parallel and serial scan-based test. S: scan input vectors; O: scan responses; PI: primary input stimuli; PO: primary output responses.

in the new approach. However, the bandwidth requirement of the serial approach drops to half of the parallel approach. More importantly, the top-level scheduling only needs to manage either the test input pattern flow or the test-response flow at a certain moment. The removal of the correlation between the two operations greatly simplifies the test scheduling and brings more flexibility to the routing algorithm.

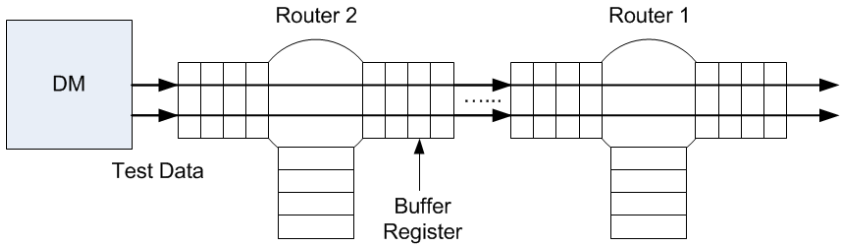
4.4.2.2 Pause/resume of the dependability test

As the dependability test and normal functional applications share the NoC for communication purposes, it is essential that the competition for NoC resources will not cause a congestion of functional/test data or a test failure. Traditional scan-based test is a continuous process. But in a NoC environment, it is possible that the interruption and resume of tests can take place from time to time. The view of the dependability test scheme proposed in this thesis is that both the source and the sink of the test data can cope with this interruption. As such, the

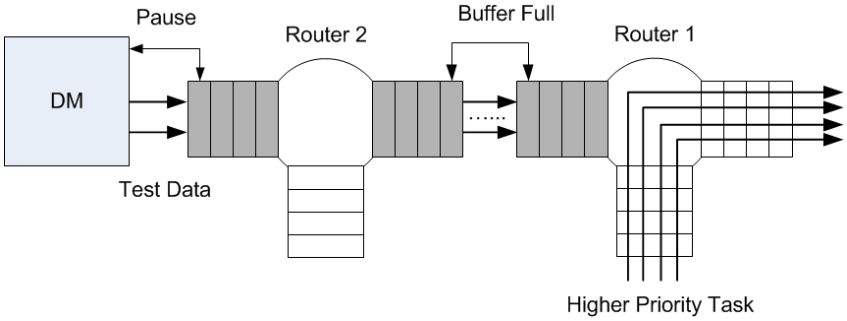
system can treat the dependability test as a special application with lower priority and allocate the NoC bandwidth to more important functional tasks if necessary. Test data can accumulate within the router buffer registers if the downstream path is temporarily unavailable. When the buffer in a router becomes almost full, a "buffer full" signal will be generated and passed to the routers along the path to the source of the test data. Ultimately, this "buffer full" signal will propagate to the source and cause it to pause the data generation.

This back-pressure style flow control mechanism works for both the test input pattern generation as well as the test-response collection processes. These two scenarios will be examined separately by an example. Figure 4.13 shows the back-pressure flow control in the test-pattern generation process. In Figure 4.13(a), the dependability test input patterns are the only data stream that occupy the bandwidth of the two routers. In Figure 4.13(b), the data stream of a high priority task claims all the bandwidth of the east port of Router 1, causing the test input patterns to accumulate in the buffers of Router 1. Router 1 and 2 successively generate the "buffer full" signal and eventually cause the DM to pause the test pattern generation. As long as the next test input vector does not arrive at the core under test, the core test-wrapper will hold the core clock and prevent any shifting operations of the core scan chain. This is achieved by the Clock Gating (CG) module introduced in Figure 4.9. Later, when the router becomes available again, the "buffer full" signal will be revoked and the test-pattern generation operation will be resumed.

Similarly, test-response data-flow control is also necessary in case of insufficient bandwidth. Such an example scenario is depicted in Figure 4.14. In this example, back-pressure flow control is used to adjust the pace of response collection for each core. In the figure, two routers (R1 and R2) are available for the DM to access the NoC. Test responses of core C can be delivered to the DM twice as fast as compared to core A and B because A and B have to share the same router port (east port of R2). But the comparator in the DM-TRE will perform one comparison only after the test responses of the same sequence (1, 2, 3) from all three cores have arrived. Therefore, the test-response collection of core C is paused repeatedly after its buffer channel in the DM-TRE becomes full. As such core C can match the response collection pace of the other two cores. The "buffer



(a) Sufficient bandwidth for test data



(b) Higher priority task uses all bandwidth; dependability test is paused

Figure 4.13: Back-pressure driven test data flow (test-pattern generation part)

full" signal will arrive at the core test-wrapper which controls the shifting of the scan responses by clock gating of the core under test (core C).

4.4.3 Impact of the dependability test

If one assumes the dependability test will not be interrupted by any other application, the test time of one scan-based test is determined by the NoC bandwidth available for the test-data transport. The overall dependability test time also depends on the number of times the test is performed. Figure 4.15 shows the relationship between the overall test time and the number of NoC virtual channels allocated to the DM and the number of cores tested each time in the 64-cores MP-SoC platform. The test time increases if less NoC virtual channels are available or if less cores are tested each time simultaneously.

On the other hand, the system performance will drop if the use of NoC bandwidth for the dependability test and the number of cores tested each time increase. This trend is shown in Figure 4.16. The system performance is calculated by the

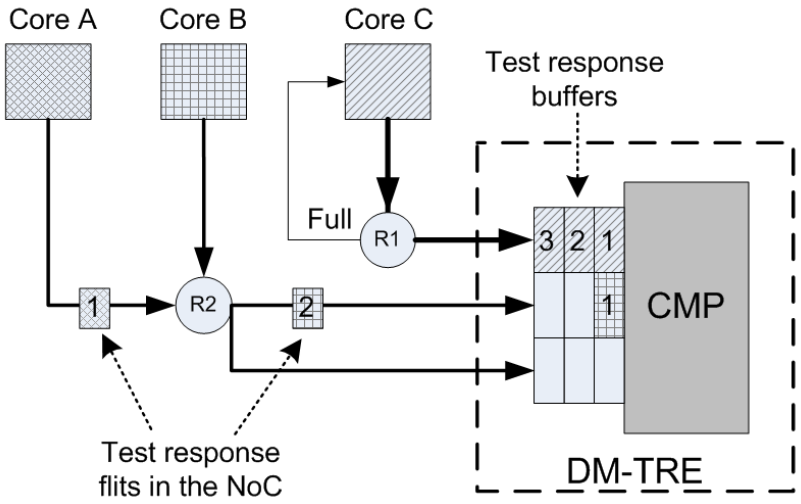


Figure 4.14: Test response collection from three cores A, B and C under test. CMP denotes a comparator. R1 and R2 are routers. Cores A, B and C are simplified without the NI and wrapper.

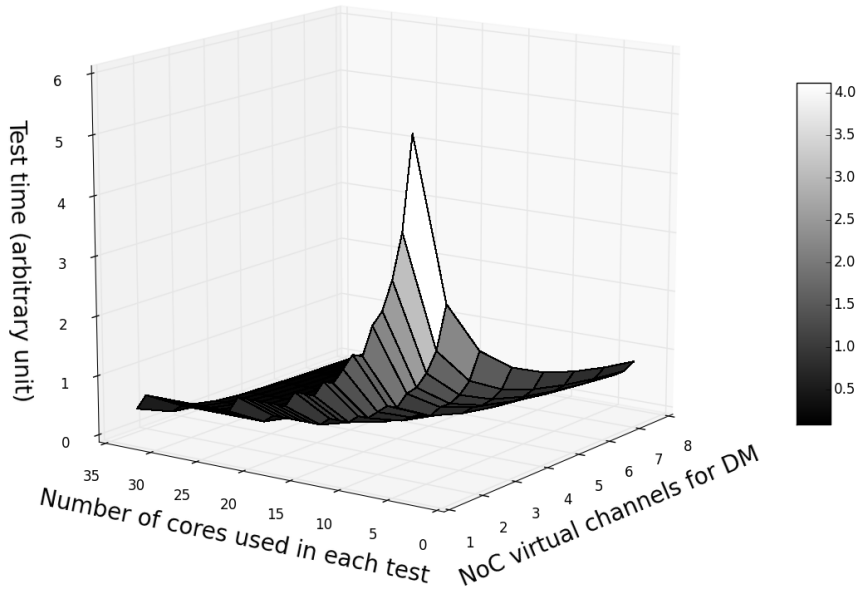


Figure 4.15: Dependability test time (arbitrary unites) v.s. NoC virtual channels for DM and the number of cores tested each time. The vertical bar on the right side indicates the grey scale corresponding to the test time.

percentage of processor cores and NoC virtual channels available for user applications. An optimal resource-allocation plan is one that balances the dependability test time and the system performance and meets the specific requirements of the user. The user can adjust these parameters in a convenient way by changing the parameters of the dependability test software which was used to generate the test time and performance plotting. By specifying parameters which have to be satisfied such as the level of system performance and the number of cores in each test, the third parameter (e.g. NoC bandwidth for test) can be calculated. For example, if three cores are tested each time and a minimum of 0.7 system performance is required, then as much as 0.25 of total NoC bandwidth can be used for the dependability test.

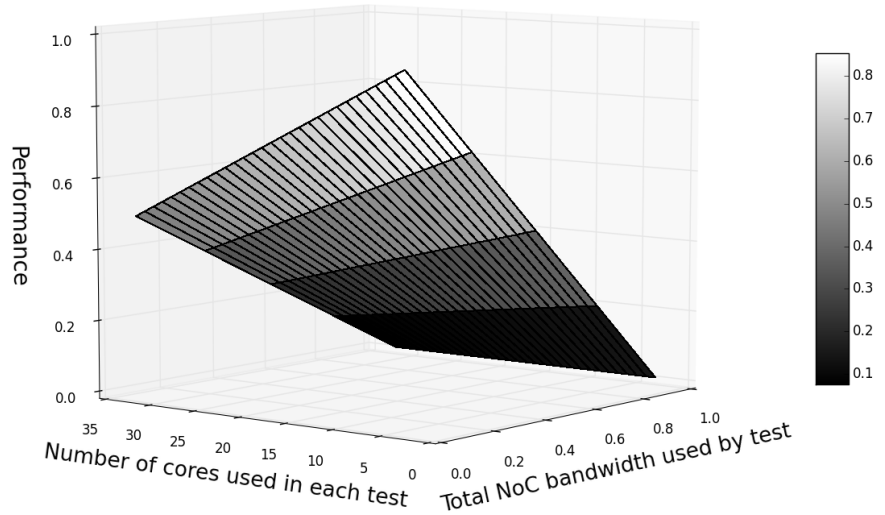


Figure 4.16: System performance (percentage of total computing power) v.s. NoC bandwidth (arbitrary units) and the number of cores tested each time simultaneously. The vertical bar on the right side indicates the grey scale corresponding to system performance.

4.5 Testing the NoC

As the dependability test on the embedded cores uses the NoC as a TAM, it is essential to guarantee its correctness. Hence, the NoC has to be thoroughly tested before it can be used to transport the test data for the dependability test of

embedded cores. Previous research have generated test strategies for NoC routers [Grec 05, Amor 05] and NoC interconnecting links [Grec 05, Cota 08]. Functional tests on the NoC components have also been proposed [Stew 06].

A software-based NoC test approach has been proposed in the CRISP project [Burg 11]. Any faulty link, router switch component or even the network interface of a processor core can be detected. In addition, both the test and diagnosis of the NoC can be accomplished, which means the detection and localization of a potential fault. The main idea of this software-based test approach is briefly introduced in the following part as it is crucial for our dependability approach.

4.5.1 NoC fault modeling

As already introduced, the major building blocks of a NoC are routers and links. Links connect a number of routers into a certain topology while each router handles five links to five directions. The internal switch components of a router can set up communication paths between any two links. Both routers and links can be modeled as path components in which data packets will travel during a communication round. As such, the entire NoC infrastructure can be modeled as a collection of path components. Accordingly, any arbitrary path in a NoC can be represented as a series of interconnected path components.

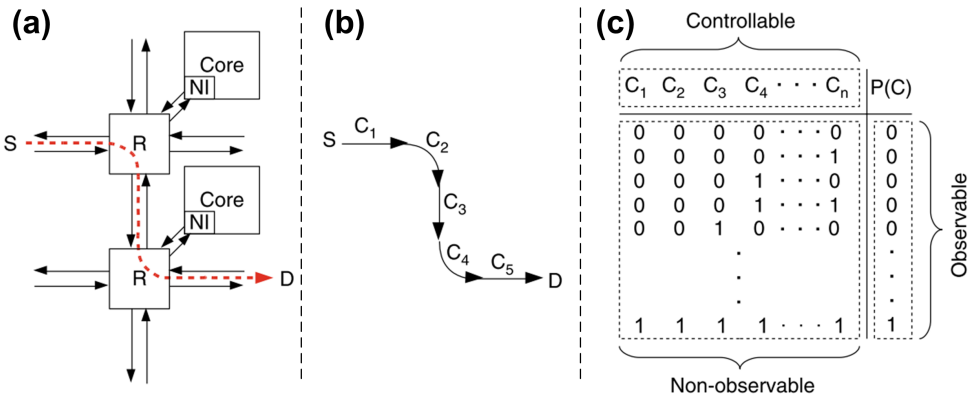


Figure 4.17: Modeling of the NoC (a) An arbitrary NoC path; (b)Path represented as a component graph with each link or router modeled as a component represented by Cn; (c) Truth table of the logic and path function $P(C)$ [Burg 11]

An example of this NoC modeling method is given in Figure 4.17. An arbitrary NoC path has been shown in Figure 4.17(a), which involves two router switches (the R block) and three links along the red dotted arrowed line. In total five path components can be used to model this path (component C as shown in Figure 4.17(b)). The case $C = 1$ is used to represent a good component and $C = 0$ for a faulty component, the status of this arbitrary path can be expressed as the logic AND of all its path components:

$$P(C) = C_1 \cdot C_2 \cdot C_3 \cdot ... \cdot C_n \tag{4.1}$$

P(C) is defined as the path function which is an observable unit. For a random path, $P(C) = 1$ denotes its fault-free status and $P(C) = 0$ means the path is faulty. A link is considered faulty if it fails to deliver a message from one end to the other end. And a router switch component is regarded faulty if it fails to route the incoming packet to the designated direction. In this thesis, the possibility is excluded that one path component can be repaired and recovered after a fault occurs, which means a faulty unit will be permanently isolated from the NoC topology by the resource-management software.

4.5.2 NoC test and diagnosis concept

Due to the complexity of the NoC in a large MPSoC, direct test of each individual path component is infeasible. But the truth table in Figure 4.17(c) suggests that if one packet can be successfully routed via a specific path without being corrupted, all path components along that path can be regarded as functionally fault-free. Such a fault-free path is defined as a *good path*. Hence the test of the complete NoC can begin with the assumption that all path components are faulty. By routing multiple data packets through the NoC such that every good component is part of at least one good path, all the fault-free components can ultimately be recognized and those unrecognized components can be regarded as faulty.

This idea appears to be simple and straightforward at first glance, but it is quite difficult to realize this as the result of the unpredictable adverse effect of any faulty component. For example, if a faulty link component generates a reversed result compared to its input (similar behavior as an inverter), and if the same

data packet traverses this link component twice, the two reversing effects will cancel each other out. The test result may suggest it is a good path, but actually it is not. For this reason, the self-avoiding walk (SAW) technique [Haye 98] has been adopted as the guideline for NoC path generation. The basic idea is that the generation of any random path must follow the rule that the test data packet will not go through the same path component more than once. Note that multiple traversals of the same router is allowed as long as they do not go in the same switch direction. Some additional constraints for path generation are as follows:

- Explicitly defined start and end points;
- The physical boundary (the outermost routers) of the network topology;
- A maximum specified number of router hops (the number of routers in a sequential order).

By sticking to these constraints, one can compute a random path which will include one or several target path components as long as sufficient number of hops is given. The test and diagnosis of the entire NoC can be accomplished with a number of such test paths.

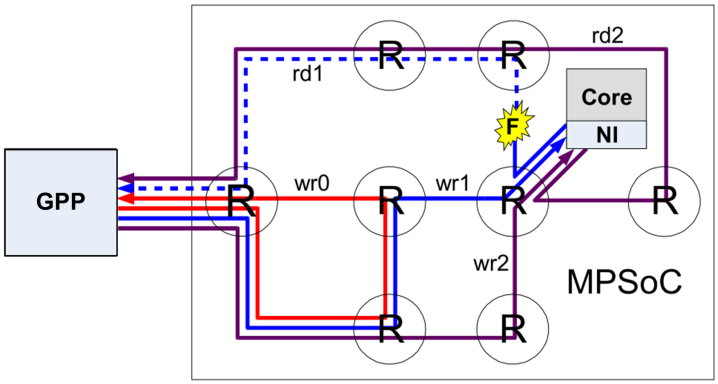


Figure 4.18: Routing of the NoC test data packets in an MPSoC; wr = write, rd = read [Ter 10].

Figure 4.18 shows an example which uses the NoC test approach introduced above in the MPSoC environment shown in Figure 4.2. Here the GPP serves as the test input vector generator to inject test-data packets into the NoC and also as

the test response analyzer to check whether the same data packets have returned. Two test strategies can be used in different circumstances. In the first case, the GPP specifies a proper path to test and sends the test data packet to travel through that path and then return to the GPP itself. An alternative method is to *write* the test data packet into a specific processor core, and then *read* the content back. Both methods can used in parallel with user applications as long as sufficient NoC bandwidth is available for the test. The first method does not require the utilization of any processor core but it can only test the links and routers. The second method can also test the network interface of the processor core but it has to utilize the core for writing and reading the data packets. In Chapter 7, it will be stated that the test of the NoC of a nine-core MPSoC took about several hundred milliseconds with a 200MHz main clock frequency. More details about the algorithm and implementation of the used NoC test approach can be found in [Burg 10, Burg 11, Ter 10] and will not be further discussed in this thesis.

4.6 Conclusions

In this chapter, the two basic parts of our proposed dependability approach being the self-test and self-repair of a target MPSoC, have been examined. The traditional scan-based structural test has been modified and used as a dependability testing approach for the self-test of homogenous MPSoCs. It has the advantage of a very high fault coverage and can be performed at application run-time. The self-repair of an MPSoC is defined as the isolation of faulty cores, the reconfiguration of the good remaining cores and the remapping of the application software. Resource-management software has been designed within CRISP to record the status of each core in an MPSoC. Run-time mapping software has been designed to reallocate applications to fault-free resources.

The dependability test is carried out by our dedicated IIP incorporated into the MPSoC. It generates scan-test input patterns and in an innovative way multicasts the stimuli for the cores under test. It compares the test responses from multiple identical cores and generates the test results accordingly. The NoC has been reused as a TAM which can transport both the functional data and the test

data simultaneously. To cope with potential bandwidth competitions between the functional and the test data, a start/stop flow control mechanism has been created to manage the scan-based test via NoC.

The design of important dependability test infrastructures such as the NoC and our design of the dedicated test wrappers have been thoroughly presented. The trade-offs of the dependability test are also discussed and the dependability test software API is designed to aid the user to balance these parameters. The test and diagnosis of the NoC can be implemented using the self-avoiding walk technique using software running on the GPP. The GPP can function as a test vector generator and a test response analyzer for the test of the NoC. As such the faulty parts of the NoC can also be isolated from the system.

Chapter 5

Dependability Manager Architecture

ABSTRACT - *In Chapter 4, the general idea was introduced how to carry out the dependability test. The structural scan-based test on the Xentium tile logic part is performed first and the Xentium tile memory test follows. We have also introduced how to use the NoC as a TAM, as well as the wrapper design for the Xentium tile so that one can perform the dependability test at application run-time via the NoC. In this chapter the design of the dependability manager (DM) is presented in detail. The DM is designed in a generic way as a stand-alone IP which can be integrated into any NoC-based MPSoC framework to perform a dependability test.*

5.1 Introduction

5.1.1 DM overview

The motivation to introduce a Dependability Manager (DM) in an MPSoC is that there is a need to build an on-chip dependability test environment through

Parts of this chapter have been presented at the 5th IEEE International Symposium on Electronic Design, Test & Applications (DELTA 2010) [Kerk 10].

which the correctness of the internal processor cores/tiles of an MPSoC can be verified. This goal is achievable with software running on a general purpose processor in the system in a similar way as the software dependability test approach designed for the NoC in Chapter 4. However, the software test approach has some disadvantages as will be discussed below.

Limited fault coverage is the first drawback of the software approach. As the NoC mainly consists of wires and switches, a software functional test is usually sufficient to cover most of the faults. But a processor core is more complex than the NoC due to its large logic blocks and embedded memories. Much more sophisticated test-patterns (e.g. structural scan-based test-patterns) and dedicated tester hardware are required to perform a test with a very high fault coverage.

Furthermore, it is possible that the user requires the dependability test to be performed periodically at a high frequency. For a software approach, this implies that the general purpose processor will be frequently occupied by the dependability test mission, which makes it unavailable for other important user tasks. This is usually unacceptable for mission-critical applications. On the other hand, if a separate general purpose processor is added to the system solely for the dependability test purpose, the cost in terms of silicon area is usually much higher than its ASIC counterpart, namely a dedicated dependability manager.

Therefore, in the CRISP project it has been chosen to implement the DM as a hardware Infrastructural IP (IIP) in the target MPSoC. The idea is to construct the DM as a flexible design with customizable parameters such as the type of faults it can test and the fault coverage it can achieve. The interface of the DM should be made sufficiently generic so that it can interact with the other parts of the MPSoC via a standard network interface as introduced in Chapter 4.

In the scope of this thesis, the common stuck-at fault model is taken as an example to demonstrate the dependability test concept and to validate our proposed dependability approach. Note that more fault models can be covered by our approach with some modification. For example, by performing the scan-based test twice in a short time interval, one can detect potential transient faults. The dependability manager comprises a test-vector generation and a test-response evaluation infrastructure targeting the structural stuck-at faults of the Xentium tiles. A Finite State Machine (FSM) manages the internal activities of the DM

and communicates with the dependability software which currently runs on external General purpose Processor Device (GPD, e.g. an ARM926).

The DM is devised in a generic and modular way, using the hardware description language VHDL. Figure 5.1 shows a block diagram of its internal blocks. Three major functional blocks of the DM can be identified being the test-pattern Generator (DM-TPG), the test-response Evaluator (DM-TRE) and the Finite State Machine (DM-FSM). These three components are separately designed and simulated and subsequently integrated by a top level VHDL entity to provide the complete DM. A Matlab program has been developed to automate the design process of the key component DM-TPG. The input for this program is the post-synthesis layout netlist file of the target processor which contains the final order of the scan-chains inserted. As a result, a new DM-TPG block can be generated in an automatic way to provide test-pattern sets for a different processor as the Xentium.

The interaction of the DM with its surrounding environment is also shown in Figure 5.1. The DM transports its input and output data, such as test-vectors or test-responses, via the NoC using a dedicated Network Interface (NI). An external GPD can issue control instructions to the DM and read its test report also via the NoC. Alternatively, the DM can be directly controlled by external pins by setting the `IIP_mux` multiplexing signal in case of debugging. The debugging pinsare dedicated pins only for the sake of verification and validation of the DM concept. After that, those pins will not be used anymore and will be removed in future designs.

Important features and specifications of the DM Infrastructural IP include:

- Deterministic scan-based test-vector generation for the Xentium processing tile
- Xentium tile memory BIST handling
- Up to three channel comparison of scan-based test-responses
- Adapted new pause-and-resume function during test-pattern generation and test-response evaluation

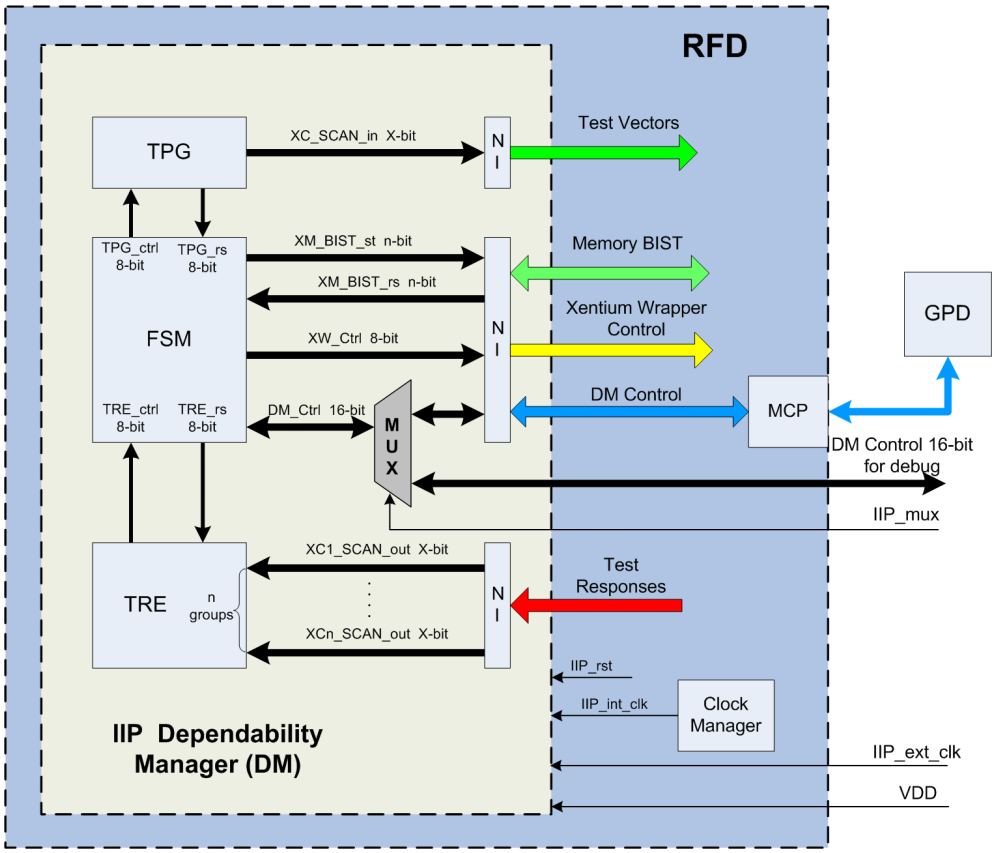


Figure 5.1: Internal blocks of the DM in the RFD

- Register-based programming interface for DM configuration, internal state observation and test result reporting
- Direct top-level debugging interface
- Main clock frequency 200MHz
- Silicon area as small as possible (decided to be smaller than one Xentium tile)
- No hard requirement on power dissipation yet [Zhao 13]

5.1.2 The Xentium tile from a test perspective

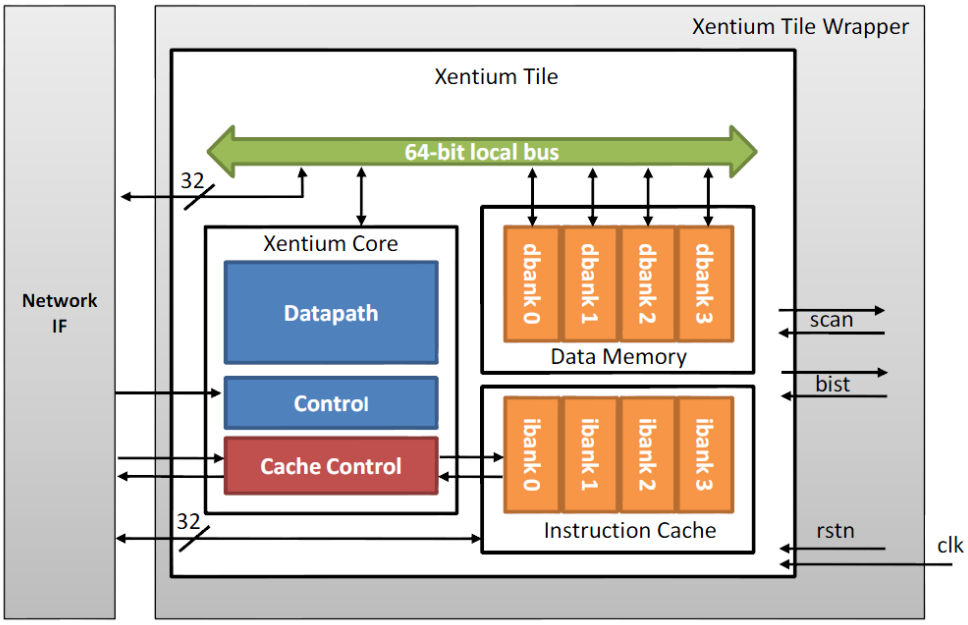


Figure 5.2: Block diagram of the Xentium processing tile from Recore Systems

An MPSoC with many identical Xentium™ tiles has been chosen as a target platform for the evaluation of our dependability approach. The Xentium processing tile is a reconfigurable digital signal processing core [Recore 13]. It features high-performance and energy-efficiency by parallel operation optimization at the

instruction level. A block diagram of the Xentium processing tile is shown in Figure 5.2.

The major building blocks of the Xentium tile include two modules: the Xentium Core (datapath and control) and the Xentium Memory (data memory and instruction cache). In addition, a network interface handles the communication between the Xentium tile and the external NoC. A Xentium tile wrapper (XTW) provides access to the scan-chains of the Xentium core and the BIST engine of the Xentium memory. Details of the Xentium network interface and the Xentium tile wrapper have been introduced in Chapter 4.

Table 5.1: Testability information of the Xentium processing tile

Item	Detail
Number of logic gates	200K
Number of Primary Inputs (PI)	168
Number of Primary Outputs (PO)	154
Number of scan-chains	32
Length of each scan-chain	413
Number of stuck-at faults	729K
Number of ATPG test-vectors	1275
ATPG test-pattern highest fault coverage	99.3%

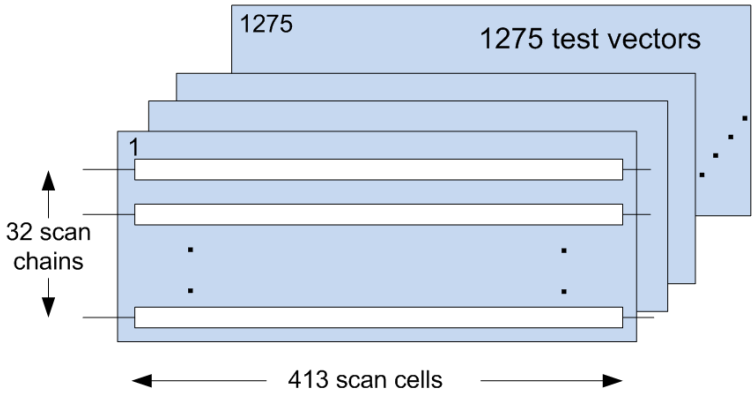


Figure 5.3: Xentium tile scan-chains and test-vectors organization

The dependability test on the Xentium tile consists of two parts being the test of the Xentium Core and the test of the Xentium memory. To facilitate

the dependability test of the Xentium Core, full scan-chain insertion has been carried out on the Xentium tiles and structural test-patterns targeting stuck-at faults have been generated with commercial Automatic Test-Pattern Generation (ATPG) software. The Xentium tile is being provided as a hardmacro IP together with a fixed test-pattern set for our dependability test. Important information concerning the testability of the Xentium tile is summarized in Table 5.1 and visualized in Figure 5.3. Note that the fault coverage of the Xentium tile was less than 100% due to the fact that the design was not optimized for testability.

The given ATPG test-vectors (deterministic) need to be compressed by e.g. a reseeding technique so that they can be reproduced by the DM-TPG without using any external Automatic Test Equipment (ATE). The DM produces one bit of the deterministic test-vector for each scan-chain in the Xentium tile in each clock cycle. These vectors are packed into 32-bit data flits and transported to the Xentium tiles under test via the NoC. The XTW unpacks the NoC data flits, shifts the test-vectors into the scan-chains of the Xentium core and also applies test-vectors to the primary inputs of the Xentium tile. Test-responses from the Xentium tile are then collected by the DM-TRE from the XTW and subsequently evaluated in the DM-TRE.

The control of the Xentium memory BIST engine and the collection of the BIST test results are also performed by the DM via the NoC. The Xentium memory module has been equipped with a Built-In Self-Test (BIST) engine from Atmel Automotive. The memory BIST can be enabled if the Xentium tile wrapper is set in dependability test mode by the DM via the NoC. At the time the memory BIST is completed, the test results are also collected by the DM and a test report is generated.

5.2 Test-pattern compression theory

In Chapter 4, the BIST architecture adopted for our dependability test has been discussed. The Test-Pattern Generator in the DM generates test-vectors and applies them to the Xentium processing tiles via the NoC at application run-time and then the DM evaluates the test-responses. Obviously, the design and implementation of the DM-TPG has direct impact on the dependability test fault

coverage, test time and the DM area.

Many studies have been carried out in the past on test-pattern generation for BIST applications [Bush 05]. For example, a simple counter circuit can be used for exhaustive testing. Given a circuit with 100% testability, an exhaustive test can always achieve 100% fault coverage for stuck-at faults. But it usually takes an enormous amount of time to generate all test-vectors. For instance, given a combinational logic circuit with 168 inputs, 2^{168} test-vector combinations are required to perform an exhaustive test. Suppose the clock frequency for generating the test-vectors is 1GHz, then the test time needed to test this 168-input circuit can be calculated as following:

Test time = $2^{168} \times 1 \times 10^{-9}$ seconds $\approx 1.2 \times 10^{34}$ years. The parameter 2^{168} is the number of test-vectors; 1×10^{-9} is the time (in seconds) of each clock cycle.

Apparently, the test time for exhaustive testing is much too long to be used for our dependability test. A widely used alternative and more realistic scheme is to use a Linear Feedback Shift Register (LFSR) as a Pseudo Random Pattern Generator (PRPG) to perform pseudo-random testing on the target circuit e.g. [McCl 85, Hari 11, Mumt 11]. The LFSR can generate a subset of the exhaustive test-patterns and the fault coverage of the these patterns on a target circuit can be determined by performing fault simulation. In addition to the time-consuming computation of the test length and fault coverage, a major disadvantage of the pseudo-random testing is that some logic circuits contain random-pattern resistant faults (e.g. circuits with large fan-in) which are difficult to detect with random patterns [Eich 83]. This limits the fault coverage of pseudo-random testing. Various techniques are available to increase the fault coverage of pseudo-random testing, such as test point insertion [Sava 60, Toub 96]. These methods require a modification of the original design to eliminate the random-pattern resistant faults, which is usually not feasible if the target circuit is provided as pre-designed hardcores or IPs by core vendors.

Deterministic test-patterns generated by commercial ATPG tools are often provided with pre-designed IPs which can ensure a very high fault coverage (99.9-100%). One will naturally want to reproduce these test-vectors in a BIST application. One of the most straightforward methods is to store these deterministic test sets in an on-chip Non-Volatile Memory (NVM) such as a Read Only Mem-

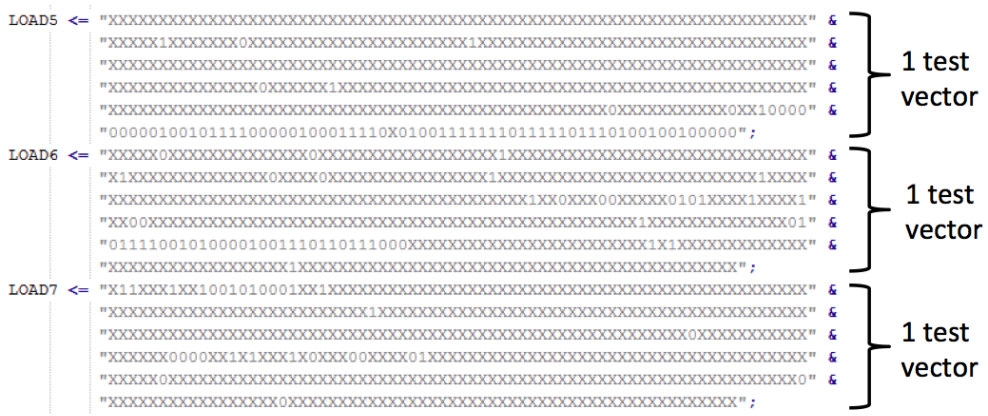


Figure 5.4: Part of the Xentium tile ATPG test-vectors; "X" denotes the "don't care" bits.

ory (ROM) or flash memory and later load them to perform tests. However, the direct storage of the original deterministic test-vectors is a huge waste of silicon area as the majority of the deterministic test-vectors are unspecified bits (X) as can be observed in the test-pattern files of the Xentium tile processor (Figure 5.4). These unspecified bits can be assigned to any value (binary 1 or 0) without influencing the test fault coverage; thus they are called *don't care* bits. In contrast, the bits that have a value of 0 or 1 are called care bits. The values after each `LOADN` signal are the vectors to be loaded into the scan-chains of the Xentium tiles. As shown in the figure, the majority of the test-vectors are don't care bits (with value `X`). The specified bits and don't cares together form a test-vector. A collection of test-vectors is often referred to as a *test cube*.

The large amount of don't care bits in the test cube make it feasible and also necessary to compress the deterministic test-vectors to save the on-chip storage space. Many studies have been carried out in the past on deterministic test-vector compression techniques. In general they fall into two categories, the code-based technique and the linear-decompression based technique both described in [Toub 06].

The *code-based* test-vector compression scheme partitions the original test cube into smaller groups and encodes them into codewords to achieve data compression. During a test, a tester can feed the codewords to a decoder block to

restore the original deterministic test-vectors. Examples of the code-based compression method include the one using run length codes [Chan 03], statistical code based on the frequency of bit occurrence [Jas 99] or Huffman coding which makes use of a Huffman tree [Gonc 03, Jas 03]. Pros and cons of the code-based and linear decompression technique have been evaluated in [Toub 06]. While the code-based technique is applicable to any set of test cubes, linear decompression techniques can better exploit the don't care bits in the deterministic test cube and achieve a relatively higher compression rate.

The *linear decompression* based technique uses a linear test-pattern generator such as an LFSR for deterministic test-pattern compression. A representative example is the reseeding scheme, in which test-vector compression is achieved by storing a group of LFSR initial states (the seeds) instead of the original deterministic test-vectors [Kone 91]. A test begins with loading of the seeds into the LFSR registers and the corresponding test-vectors can then be generated in subsequent clock cycles. One can compute the required LFSR seed for certain test-vectors by solving a series of linear equations based on the feedback polynomial of the LFSR. The LFSR seeds can be stored in a non-volatile memory or embedded as logic.

Instead of using the LFSR seeds to manipulate LFSR outputs, one can also modify the LFSR output contents directly. It has been reported that deterministic test-vectors can be obtained by modifying a number of bits of the pseudo-random test-vectors generated by an LFSR. This method has been referred to as bit-flipping [Wund 96, Kief 98].

Recent research has also proposed a programmable BIST scheme which combines the reseeding and bit-flipping methods [Hakm 07]. Instead of increasing the length of the LFSR, this method ignores some of the unsolvable linear equations while calculating the seeds. The specified bit values of the test-vectors corresponding to the ignored equations are flipped by an external circuit during the reseeding process. While this method generally results in shorter LFSR length and LFSR seeds, it costs extra effort during the design phase and the storage of the flip vectors.

According to the literature outlined above, all three techniques (reseeding, bit-flipping or the mix of the two) are able to achieve full fault coverage for some

of the target circuits mentioned in their research. In this thesis, the reseeding scheme for the design of the DM-TPG will be adopted due to its straightforward design process which leads to easier software automation.

5.3 Reseeding TPG architecture

The reseeding technique for test-vector compression is fault-model independent. For example, with little modification, it can be used for transition faults instead of stuck-at faults. The reseeding technique was first introduced in [Kone 91]. A reseeding TPG consists of an LFSR, reseeding logic and a phase-shifter in the case it is used to feed parallel scan-chains.

5.3.1 LFSR

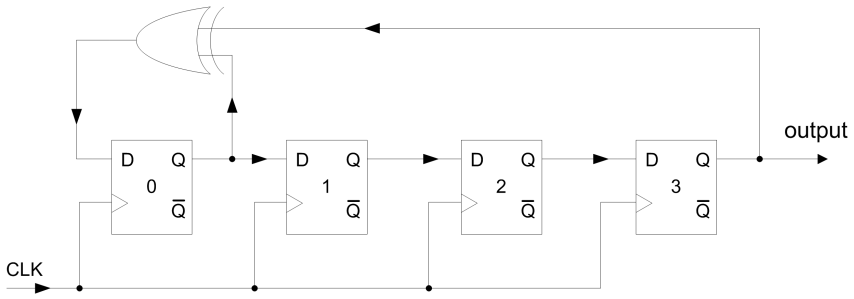


Figure 5.5: A 4-stage Type I LFSR consisting of four D flip-flops

A Linear Feedback Shift Register (LFSR) is a shift register which can shift its contents to the next most significant bit when a clock pulse is provided. An LFSR usually consists of a group of flip-flops connected in a serial way while the output of each flip-flop can be combined with e.g. exclusive-OR gate(s) to form a feedback mechanism (a tap). Figure 5.5 shows an example of a 4-stage LFSR consisting of four D flip-flops and one XOR gate. The output of flip-flop 3 is XOR-ed with the output of flip-flop 0 and fed to the input of the LFSR. The values of all the flip-flops in the LFSR at a certain moment is referred to as the state of the LFSR. When clocked, the initial state of the LFSR can determine its subsequent states; thus the initial state is defined as the seed.

An n-bit LFSR can have as many as $(2^n - 1)$ possible states if properly configured. Given a non-zero seed, an n-bit LFSR, which can cycle through all the $(2^n - 1)$ states, is defined as a maximal length LFSR and all the states as its maximal length sequence (m-sequence). A maximal length LFSR is usually preferred over its non-maximal counterpart of the same length for test-vector generation, as an m-sequence LFSR can produce a maximum number of test-vector combinations.

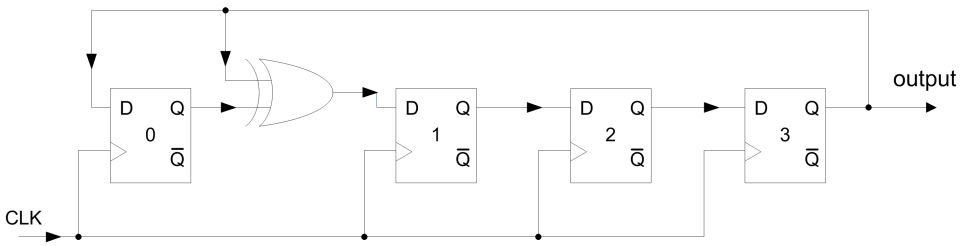


Figure 5.6: A Type II 4-stage LFSR

The total number of states of an LFSR is influenced by its feedback polynomial which is determined by the LFSR taps. For example, the feedback polynomial of the LFSR in Figure 5.5 can be written as: $X_4 + X + 1$, where the plus sign is used to denote a modulo-2 addition introduced by the XOR gate. The coefficients of each stage are either 1 or 0 depending on whether this stage is included into the feedback loop or not. The last 1 in the polynomial represents the input to the LFSR instead of a tap. This LFSR configuration (with e.g. external XOR gates) is usually referred to as the Type I LFSR. Its counterpart, the Type II LFSR, with the feedback XOR gates connected in between the flip-flops stages is shown in Figure 5.6. A Type II LFSR arranged in such manner is referred to as the dual of the Type I LFSR.

In the LFSR reseeding context, target deterministic test-vectors can be regarded as the states which the LFSR needs to generate. The seeds to generate these states need to be computed and stored in an efficient way. Based on the feedback polynomial of the LFSR, a system of linear equations can be derived and used to determine the seed value for each test-vector in a test cube. It is possible that one particular seed can generate a long sequence of states which can cover multiple deterministic test-vectors.

Table 5.2: Propagation of the internal flip-flop states of a 4-stage LFSR (Figure 5.5)

Cycle	FF0	FF1	FF2	FF3 (output)
1	L0	L1	L2	L3
2	L0+L3	L0	L1	L2
3	L0+L2+L3	L0+L3	L0	L1
4	L0+L1+L2+L3	L0+L2+L3	L0+L3	L0
5	L1+L2+L3	L0+L1+L2+L3	L0+L2+L3	L0+L3
6	L0+L1+L2	L1+L2+L3	L0+L1+L2+L3	L0+L2+L3
7	L1+L3	L0+L1+L2	L1+L2+L3	L0+L1+L2+L3
8	L0+L2	L1+L3	L0+L1+L2	L1+L2+L3

5.3.2 Seed Calculation

The design of LFSRs has been extensively examined in previous studies [Bush 05]. The 4-stage LFSR in Figure 5.5 is used as an example to demonstrate the seed calculation for the reseeding scheme. Assume the initial state (seed) of the LFSR is L={L0, L1, L2, L3}, with L0-L3 being the status values of FF0-FF3. The output of the LFSR is connected to the scan input signal of an 8-bit long scan-chain of the circuit under test (CUT). As shown in Figure 5.7, S0-S7 represents the value of the internal scan flip-flops of the scan-chain. In each clock cycle, an output value is generated by the LFSR and serially shifted into the scan-chain. After eight clock cycles, the scan-chain is completely filled by the output bits generated by the LFSR. Table 5.2 shows the values of the four flip-flops of the LFSR during the eight clock cycles. Note that the value of FF3 is the output of the LFSR and is shifted into the scan-chain during each clock cycle.

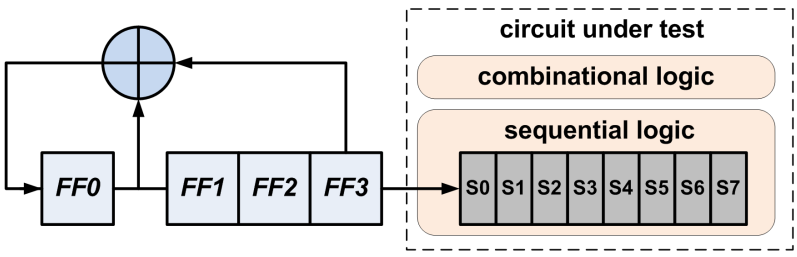


Figure 5.7: Use of a 4-stage LFSR (FF0-FF3) to fill an 8-bit scan-chain (S0-S7)

$$\overset{A}{\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \overset{L}{\begin{pmatrix} L0 \\ L1 \\ L2 \\ L3 \end{pmatrix}} = \overset{S}{\begin{pmatrix} S0 \\ S1 \\ S2 \\ S3 \\ S4 \\ S5 \\ S6 \\ S7 \end{pmatrix}} \tag{5.1}$$

According to Table 5.2, the output bit of the LFSR (value of FF3) in each clock cycle can always be expressed as a combination of the seed value (L0-L3). Based on this table, a system of linear equations showing the relationship between the seed value and the scan cell value can be derived and subsequently expressed in a matrix form as shown in Equation 5.1. The binary matrix at the left side of the equation is determined by the LFSR feedback polynomial. The column matrix {L0-L3} is the seed of the LFSR and the column matrix {S0-S7} represents the eight scan cells in the scan-chain of the CUT. For example, the last output of the LFSR in Table 5.2 is L1+L2+L3 (value of FF3 in cycle 8). It should be assigned to the first flip-flop of the scan-chain (S0 as shown in Figure 5.7). Therefore in Matrix A, {0, 1, 1, 1} is multiplied by {L0, L1, L2, L3} then assigned to {S0} on the right side of Equation 5.1.

Consider the following question. What is the proper seed value L if one needs to generate and shift the test-vector {1, X, 1, X, 0, X, X, 0} into the scan-chain using the 4-bit LFSR discussed above? By observing the test-vector, it is clear that the specified bits are located at S0, S2, S4 and S7 and the other bits are don't care bits (X). Substituting the test-vector value into Equation 5.1 yields in

a reduced matrix as follows:

$$\begin{matrix} A \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix} \times \begin{matrix} L \\ \begin{pmatrix} L0 \\ L1 \\ L2 \\ L3 \end{pmatrix} \end{matrix} = \begin{matrix} S \\ \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \end{matrix} \tag{5.2}$$

This system of linear equations can be solved as follows:

$$\begin{cases} L1 + L2 + L3 = 1 \\ L0 + L2 + L3 = 1 \\ L0 = 0 \\ L3 = 0 \end{cases} \Longrightarrow \begin{pmatrix} L0 \\ L1 \\ L2 \\ L3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{5.3}$$

The final solution for the seed is L= {0, 0, 1, 0}. The solution indicates that if one would load the LFSR with the seed value L= {0, 0, 1, 0} and let it autonomously run for eight clock cycles, the output of the LFSR will eventually fill the scan-chain with the desired deterministic test-vector values {1, X, 1, X, 0, X, X, 0}.

In summary, given a target test-vector S, the LFSR seed L can be determined if the following system of linear equations $A \times L = S$ can be solved. In this equation, matrix A represents equations corresponding to the position of the feedback to the XOR gate of the LFSR and the column matrix S specifies the logic value of the specified bits in the test vector.

5.3.3 Prior studies of the reseeding technique

In the context of BIST, the LFSR will be implemented as a logic circuit and the seeds can be stored in a NVM. An optimized reseeding scheme can increase the encoding efficiency and results in a shorter LFSR, fewer number of seeds and eventually a reseeding design which occupies less silicon area. Given a test-vector with more specified bits (care bits) than the length of the LFSR, the derived linear equation is not always solvable due to the fact that there may be more

equations than variables. In this case, one may not be able to generate the target test-vector with any seed. Previous research suggests that if the length of an LFSR is made 20 bits longer than the maximum number of the care bits in a test cube, the probability that this test cube cannot be coded into a seed drops to 1 out of a million [Kone 91].

Much research has been carried out to explore the methods to reduce the length of the LFSR used for reseeding. In [Hell 92] and [Hell 95], Hellebrand, et al. proposed to use a multiple-polynomial LFSR instead of a single-polynomial LFSR for LFSR reseeding. The feedback part of the LFSR has been designed in such a way that it can be configured into different feedback polynomials by means of programming. By choosing from multiple polynomials, the encoding efficiency can be improved and the length of the LFSR can be shortened. In [Rajs 98], the use of a multiple-polynomial LFSR has been combined with variable length seeds depending on the number of specified bits in the test cube to further increase the encoding efficiency.

The previous reseeding methods have been considered to be static in the sense that a seed is loaded into an LFSR for the generation of a full deterministic test-vector. A dynamic reseeding scheme has been reported in [Kris 01], in which the seed is modified incrementally while the test generation proceeds. In addition to reducing the number of clock cycles needed to generate one test-vector, the length of seeds can also be made shorter than the LFSR. Other research has proposed to use another stage of compression to further compress the LFSR seeds by use of statistical encoding [Kris 02] or by storing only positions of different bits between consecutive seeds [Bala 06].

Another fact about the complete test-vector set is that the number of specified bits in every test-vector usually varies a lot. In the previous example shown in Figure 5.4, it can be seen that the number of specified bits in test-vector Load5 is about twice as many as in Load7. The length of the LFSR is usually determined by the test-vector with the largest amount of care bits, and so is the length of the seeds. As such, the seeds for test-vectors with fewer care bits will eventually contain much useless information. An attempt to solve this issue has been proposed in [Volk 03], in which the seeds are encoded for a fixed number of specified bits instead of for a fixed test-vector. Experimental results suggest

higher seed efficiency of this scheme. A drawback of this method is that in order to deal with the changing number of tester cycles, an extra clock-gating circuit has to be added to implement a so-called *stall clock*, and the original test cube has to be preprocessed into different sizes of bit slices with a similar amount of specified bits.

5.3.4 Logic triggered reseeding

The LFSR seeds which are calculated to generate a certain test cube can be easily stored in an on-chip NVM such as a ROM. Alternatively, the seeds can also be encoded into reseeding logic circuits [AlYa 03]. While maintaining the same reseeding efficiency (same seeds), this scheme uses logic circuits to encode the seeds instead of using dedicated ROM blocks which are not always available in some MPSoCs. Using hardware to encode the seeds can be achieved by flipping certain bits of the LFSR. This is accomplished by letting the bits jump to a new state after the previous deterministic test-vector has been generated and shifted out. This scheme is referred to as *logic triggered reseeding* in the remainder of this thesis.

During the reseeding operation, three important states of the LFSR should be noted:

- The End Of Sequence (EOS) state is the moment at which the last bit of a deterministic test-vector is shifted out;
- The EOS+1 state is the moment at which the LFSR runs from the EOS state autonomously for one clock cycle;
- The New Seed (NS) state is the moment at which the contents of the LFSR becomes the target new seed (more specifically, the new seed is loaded into the LFSR).

The basic idea of logic-triggered reseeding is as follows. When the LFSR reaches the EOS state, the generation of a certain test-vector is completed and a new seed needs to be loaded into the LFSR to generate the next test-vector. By comparing the value of the LFSR at the EOS+1 state and the NS state, the LFSR

stages where an inversion (flipping) operation is required can be determined. The reseeding block is a pure combinational logic circuit and it can only be triggered when the LFSR reaches an EOS state. The reseeding logic can make the LFSR to jump from the EOS state to the NS state without going through all the intermediate states. As such, the new seed is loaded into the LFSR (reseeding). This process can be repeated until all the seeds have been loaded and all the deterministic test-vectors have been generated.

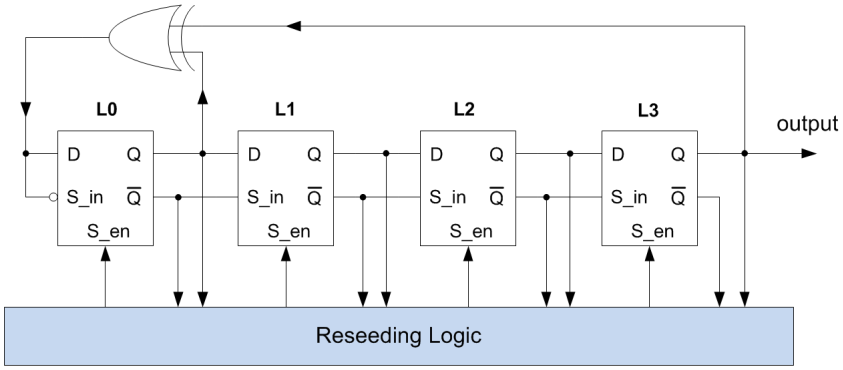


Figure 5.8: A 4-stage LFSR with scan flip-flops and reseeding logic. "S_en" is the scan enable signal, and "S_in" is the scan input signal.

The basic D flip-flop LFSR shown in Figure 5.5 can be easily modified to implement a test-vector generator based on logic triggered reseeding. Figure 5.8 shows such an example in which the clock signals are left out for simplicity. Two major modifications to the basic LFSR can be identified. First, the normal D flip-flops of the LFSR are replaced with standard scan flip-flops such that an opposite value of the previous stage output can be used as the input of the next stage. This is made possible by activating the `S_en` signal which makes the input of a scan flip-flop to be `S_in` which is connected to the $\overline{Q}$ output of the previous stage. Second, a reseeding logic block is introduced which controls the scan-enable signal `S_en` of the scan flip-flops. The reseeding logic takes a combination of the output values at each stage of the LFSR to determine when the scan enable signal of the flip-flops should be enabled, more specifically, to determine at which moment the input value of a certain stage should be $\overline{Q}$ instead of Q. Our scan flip-flop implementation in Figure 5.8 can achieve the same logic function but

takes less area compared to the D flip-flop and multiplexer implementation which were proposed in [AlYa 03].

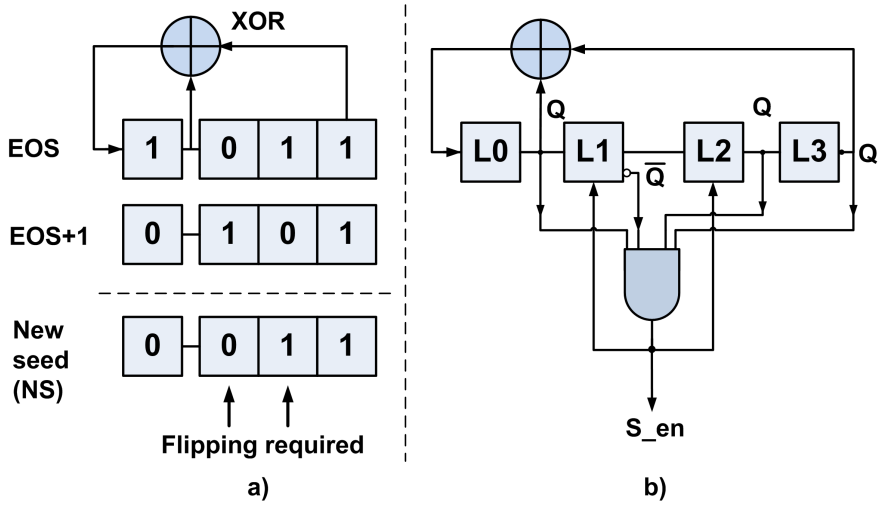


Figure 5.9: a) Calculation of the flipping position and b) the implementation of the the reseeding circuit.

An example design of the reseeding logic is given in Figure 5.9. Let the EOS state of the LFSR be {1, 0, 1, 1}, subsequently the EOS+1 state of the LFSR is {0, 1, 0, 1}. Assume the new seed value is {0, 0, 1, 1}; it can be seen from Figure 5.9 that the L1 and L2 stages of the LFSR have to be flipped in order to jump from the EOS state to the NS state. As such, the `S_en` signal of scan flip-flop FF1 and FF2 needs to be activated. As shown at the right side of Figure 5.9, an AND gate can use the value of the EOS state as a trigger and subsequently enable the `S_en` signal of scan flip-flop FF1 and FF2 to achieve the inversion of the value of FF1 and FF2 (notice that the `S_in` signals in Figure 5.8 are not shown in the figure due to space limits).

The input of the AND gate is determined by the LFSR value at the EOS state. For a random stage, if its value is "1", the direct output of that stage will be used (L0, L2 and L3 in this example). If its value is "0", the negative output will be used (L1 in this example). As the `S_en` selection lines are determined by the difference between the EOS+1 and NS state, the Exclusive Or operation of (EOS+1) and (NS) can be used to calculate the `S_en` selection positions in advance and the logic circuit to generate the selection signals can be designed

according to the calculation results. In this example, {0, 1, 0, 1} XOR {0, 0, 1, 1} = {0, 1, 1, 0}, which indicates a flipping is required at flip-flop FF1 and FF2. Hence, the AND gate output is connected to the `S_en` signal of FF1 and FF2.

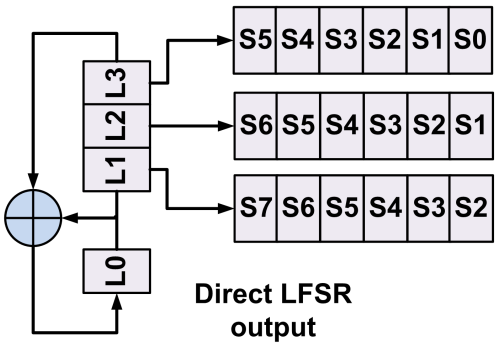
Theoretically, the logic-triggered reseeding scheme can achieve the same fault coverage as the deterministic test-vectors can achieve as long as the selected LFSR is sufficiently long. However, a very long LFSR occupies much silicon area. Therefore, in practice, a short LFSR is always preferred. In the next section, the practical design procedures of a test-pattern generator for the DM using this reseeding scheme will be introduced.

5.3.5 Two-Dimensional Test-Vector Generation and Phase Shifter

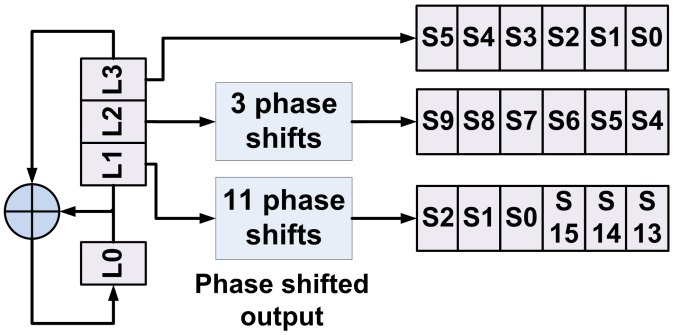
With the complexity of modern SoCs growing, a large and complex design under test may contain a very long scan-chain which can result in a long test-application time in a scan-based test. It is common practice to break up the long scan-chain into shorter parts and to fill them in parallel to save test time. Thus the reseeding TPG is required to perform a two-dimensional test-vector generation for parallel scan-chains.

In general, the contents of each LFSR stage is a few clock cycles' different from each other. Take the 4-stage LFSR in table 5.2 as an example, the contents of FF3 is one clock cycle different from the contents of FF2 and three cycles different from FF0. The term *phase shift* is commonly used to denote the clock cycles' difference between LFSR stages. As such, the output of each LFSR stage is a phase shifted version of each other.

Another example is shown in Figure 5.10 to better illustrate the phase shift concept. Let S0-S15 be the serial output of the 4-stage LFSR during in total 15 clock cycles. The outputs of L2, L1 and L0 are 1-bit, 2-bit and 3-bit phase shifts of L3 respectively (Figure 5.10(a)). Thus if the required phase shift(s) is smaller than the length of the LFSR, the target sequence can be directly generated by the output of certain LFSR stages. If a phase shift no smaller than the length of the LFSR is required, the target sequence cannot be directly generated by the LFSR stage outputs anymore. In this case, a Phase Shifter (PS) circuit can be



(a) Direct outputs from LFSR parallel stages



(b) test-vector generation with a phase-shifter

Figure 5.10: Parallel test-vector generation using an LFSR.

used to generate the target sequence. As illustrated in Figure 5.10(b), a 4-bit phase shifted version of the serial output (L3) can be generated by L2 (which is 1-bit phase shift of L3) with 3 phase shifts. And a 13-bit phase shifted version of the serial output (L3) can be simultaneously generated by L1 (which is 2-bit phase shift of L3) with 11 phase shifts.

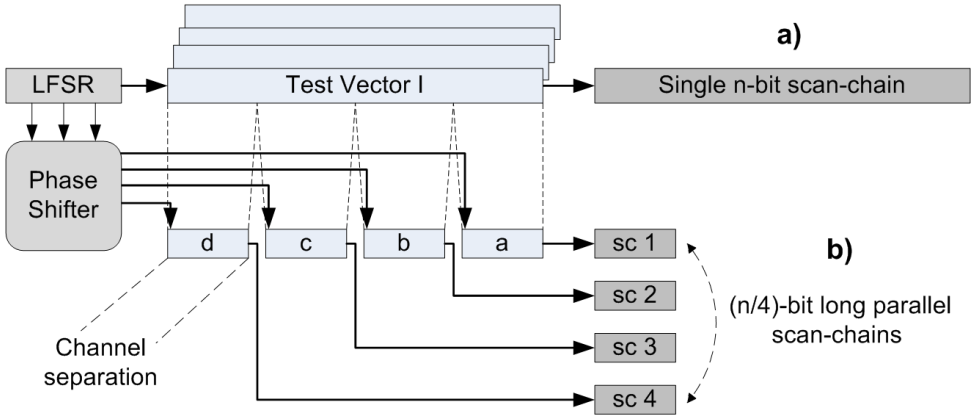


Figure 5.11: Two-dimensional test-vector generation with a phase-shifter. a) Test-vectors generated by the LFSR are shifted into a single scan-chain in serial. b) Four segments of the test vector are generated simultaneously and shifted into four parallel scan-chains.

A common phase-shifter implementation is an XOR-gate network which takes the output of each LFSR stage as input and provides test-vector sequences with a specific number of phase shifts based on the right-most stage of the LFSR (i.e. the serial output). An example scenario of using an LFSR (with reseeding logic) combined with a phase-shifter to fill parallel scan-chains is shown in Figure 5.11. The upper part of the figure shows the original case, in which the serial output of the LFSR generates a long test-vector (Test Vector 1) from its right-most stage in a serial fashion and fills a single n-bit long scan-chain in n clock cycles. The other method is shown in the lower part of the figure, where the n-bit scan-chain is split into four parallel parts and filled up in one-fourth of the time by a phase-shifter circuit.

The phase-shifter circuit works like a hardware pointer, which can point to any position in the original test-vector sequence generated by the LFSR by making the appropriate number of phase shifts. The bits following that position are

then made available for the parallel scan-chains without the necessity to generate the preceding bits first. In the example shown in Figure 5.11, the Test Vector 1 sequence is divided into four segments and generated in four channels at the same time. The number of phase shifts between each channel is defined as the *channel separation*. The larger the channel separation, the less these channels are correlated (sharing the same bits combination). It is obvious that if the channel separation is equal to the length of the parallel scan-chains, the test-vector sequence in each channel has no overlap anymore with each other and are thus least correlated. In this case, the highest efficiency can be achieved for test vector generation since each channel is generating a unique vector sequence.

5.4 Design and Implementation of the DM-TPG

5.4.1 Design of the reseeding TPG

Important Design for Test (DfT) parameters of the Xentium processing tile, such as the number of scan-chains, the length of each scan-chain, test-vector cube and the maximum number of care bits, are subject to change as the design evolves. Thus the TPG of the DM should be designed as generic as possible to accommodate possible changes or even a different type of processor design. The three major building blocks of the reseeding TPG being the LFSR, the reseeding logic and the phase-shifter also need to be designed in a generic way.

In order to automate the design process, a Matlab software tool has been developed to generate synthesizable VHDL codes for the LFSR, reseeding logic and phase-shifter block based on the DfT parameters of the target design under test. The generic DfT parameters are treated as variables and values can be conveniently assigned via a graphical user interface (GUI) by the user. A reseeding TPG can be generated automatically if all the necessary DfT information of the target circuit under test is provided. Figure 5.12 shows the complete design flow of the reseeding TPG hardware. The numbers in the figure correspond to the following steps.

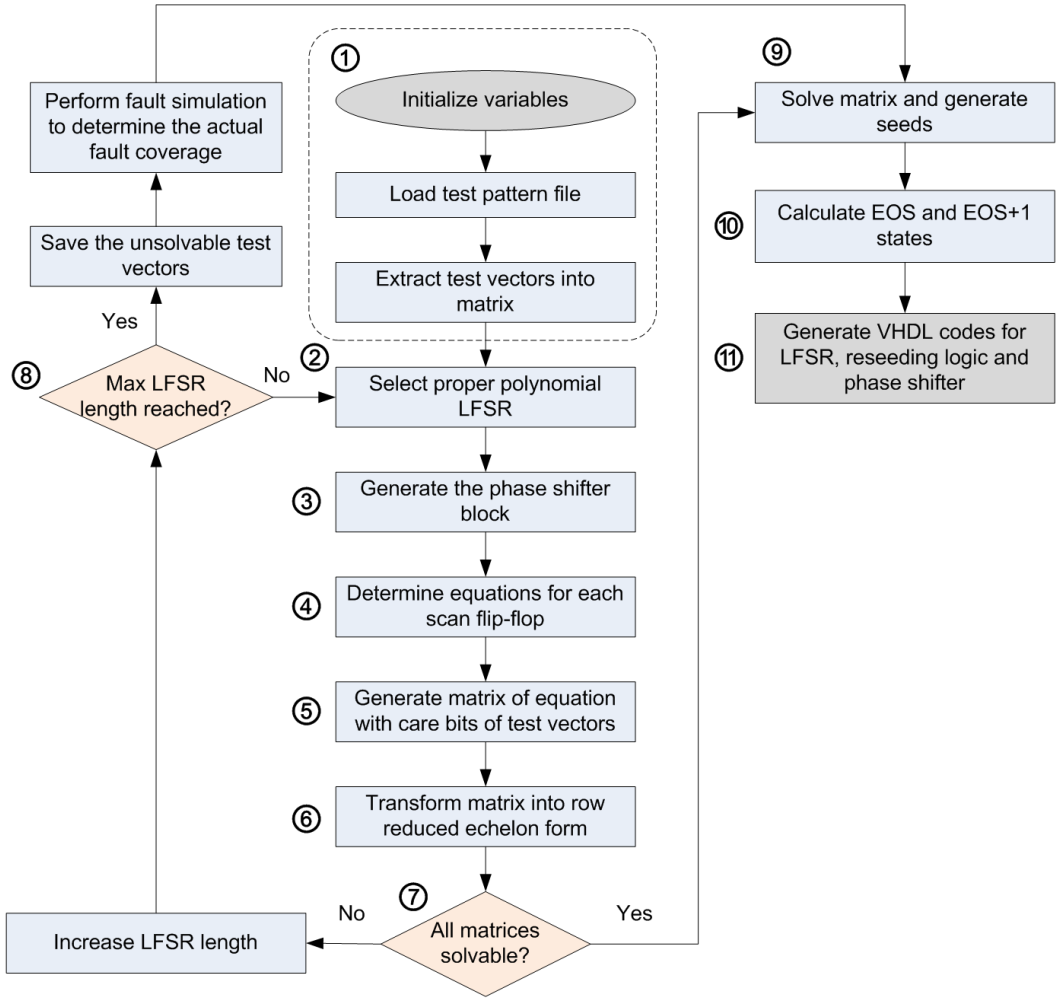


Figure 5.12: The reseeding TPG design flow

Step 1: initialize variables, load the required test-pattern file and extract test-vectors into a matrix.

Important DfT parameters are assigned to the software variables at the beginning of the flow. Two key parameters are the number and length of scan-chains of the target CUT. The software loads the pattern file containing deterministic test-patterns of the target CUT generated by commercial ATPG software such as TetraMax. Test-vector information is extracted from the test-pattern file and stored into two matrices. One matrix contains the information of the position of the care bits in each test-vector; the other matrix stores the value of each care bit (1 or 0).

Step 2: select proper polynomials LFSR

The polynomials of the LFSR determine its feedback taps, which are selected in order to obtain a maximal LFSR length sequence (m-sequence). The Matlab software has been used to calculate the possible feedback taps which can produce an m-sequence of LFSRs ranging from 5 till 1024-bit long [Clar 94]. These LFSR and their m-sequence feedback taps configuration are stored in Matlab matrices tables for the next step.

Step 3: generate the phase-shifter block

As described in Section 5.3.5, the idea of a phase-shifter is to find a linear combination of the parallel LFSR outputs, such that the resulting sequence will be shifted by a specified number of bits with respect to the previous channel. In the scope of this thesis, the length of the parallel scan-chains is the same and is initialized in the beginning of the design process. The length of parallel scan-chains does not have to be identical to use our approach.

An algorithm on how to determine the arrangement of the XOR network of a phase-shifter for a specified channel separation has been introduced in [Rajs 00]. The theorem proving part is rather long so it will not be discussed in this thesis. Only the algorithm for choosing the right LFSR stages to generate a q-bit long channel separation is briefly explained.

Given an n-bit Type I LFSR and a number of q-bit long parallel scan-chains, the phase-shifter generation algorithm can be summarized as follows:

1. Determine the dual version of the given LFSR. For example, the Type II LFSR in Figure 5.6 is the dual of the Type I LFSR in Figure 5.5. The dual of a Type I LFSR can be obtained by inserting an internal XOR gate into the LFSR stage where there is a feedback tap in the Type I LFSR except for the rightmost one.

2. Initialize the dual LFSR with the seed value {1, 0, 0, ..., 0}.

3. Let the dual LFSR autonomously run for q clock cycles.

4. Observe the resulting value of the dual LFSR. The stages which hold the value 1 should be used as input to the phase-shifter structure and the output of this phase-shifter is then a q-bit shift of the reference sequence.

Following this method, the phase-shifter network can be determined for each parallel scan-chain channel.

Step 4: determining the equations for each scan flip-flop

For each flip-flop in the scan-chain(s), there exists a corresponding equation consisting of the bits of the LFSR. An example has been shown in Table 5.2; each scan flip-flop can be represented by an XOR-function of the status of registers of the LFSR. Because the TPG is designed for multiple scan-chains and also include a phase-shifter, the equations for each scan-chain become slightly different as in the case without a phase-shifter. In the presence of a phase-shifter, each scan-chain sequence becomes a linear function of the parallel outputs of the LFSR. To generate the equations for each scan-chain, first the equations (in matrix form) for each LFSR parallel output are generated. Then the appropriate output matrices (depending on the phase-shifter function) are XOR-ed with each other for each scan-chain.

Step 5: generate the matrix of equations with care bits of test-vectors

The equations as determined at the previous step describe each scan flip-flop in the parallel scan-chains with bits of the LFSR. To calculate the seeds for one deterministic test-vector, only those equations that correspond to care bits in the test-vector need to be solved and the equations corresponding to don't care bits can be discarded. Therefore, a new matrix for each test-vector is generated, which contains the values of the care bits in this test-vector and the equations corresponding to each care bit. The matrices containing the care bits information derived in Step 1 are used as references. The new matrices are of the form $A \times L = S$ where the right-hand matrix S contains the care bits value and the left-hand matrix A describes the positions of each care bit in the test-vector.

Step 6: transform the matrix into the row-reduced Echelon form

A common way to solve a system of linear equations of the form $A \times L = S$, is by reducing the augmented matrix $A|S$ (matrix A with column S tagged on) to row-reduced Echelon form and solve the simplified equations [Anth 12]. Ordinary linear algebra uses addition, subtraction and multiplication. In the context of calculating the polynomial for LFSR, the operation on variables is modulo-2 addition (mod-2). The row-reduced matrix is obtained using Gauss-Jordan elimination [Anth 12]. Adding a row to another is performed by XORing the rows with each other (mod-2 addition). The method to obtain the row-reduced Echelon form is basically equivalent to Gauss-Jordan elimination.

Step 7: are all matrices solvable?

Linear equations are not solvable if the equations are linearly dependent. This is the case if a row-reduced matrix in Echelon form contains a pivot element in the last column [Lay 11]. Usually the software determines the maximum number of care bits from the deterministic test cube. A constant value (e.g. 20) is subtracted from the number of care bits, and this new number is used as a starting point of the LFSR-length. If the matrices of certain test-vectors are unsolvable, the length of the LFSR is incremented by one bit and the entire flow is executed

again. This process is repeated until an LFSR-length which is shorter than the maximum length is found under the condition that all matrices can be solved.

Step 8: maximum LFSR length reached?

As the LFSR and its seeds occupy most of the area of a test-pattern generator, one can easily control the size of the TPG by specifying an upper limit to the length of the LFSR. As such, a trade-off can be made between the size of the test-pattern generator and the fault coverage it can achieve. If the software reaches the maximal LFSR length specified by the user, and there are still unsolvable matrices for certain test-vectors, these vectors will be saved in a separate file and no efforts will be made to generate them anymore. A new test cube will be generated without these test-vectors. Normally, the fault coverage of the new test cube will be lower than that of the full test cube. Fault simulations can be carried out to evaluate the actual fault coverage of the new test cube with regard to the target design.

Step 9: solve the matrix and generate seeds

The matrices obtained in the previous step are solved and the seed for each deterministic test-vector is calculated and stored.

Step 10: calculate the EOS and EOS+1 State

The End Of Sequence (EOS) state is required to detect the end of one deterministic test-vector. And the EOS+1 state needs to calculated to determine which LFSR stages have to be flipped in order to load the new seed. In the reseeding scheme, the combinational logic circuit can be used to detect the EOS of a previous test-vector and check whether a specific flag signal should be triggered to load a new seed. The size of this circuit is proportional to the length of the LFSR and the number of seeds to be loaded. For a design with known LFSR length and number of test-vectors, the use of a counter structure is simpler and more area efficient for loading the new seeds. More details of this method are provided in the next section.

The EOS+1 state can be calculated by loading the LFSR with the EOS state and let the LFSR run for one clock cycle. An XOR operation of the LFSR value at the EOS+1 and the NS state will indicate at which LFSR stages a flipping is required. The output of the reseeding logic is connected to the scan-enable signals of those LFSR flip-flops whose value needs to be flipped to load the new seed.

Step 11: generate VHDL codes

The major building blocks of the reseeding TPG are the LFSR, the reseeding circuit and the phase-shifter block. The LFSR consist of a chain of standard scan flip-flops with proper feedback taps for maximal sequence. The reseeding circuit consists of logic-gate networks which can trigger the LFSR flip-flops to enter the desired states. The phase-shifter block is an XOR-gate network which combines the LFSR parallel outputs to realize the proper phase shift. We have developed a Matlab program to generate synthesizable VHDL codes of all the blocks in an automated fashion.

5.4.2 DM-TPG implementation and simulation results

It should be noted that the TPG had to be integrated into the DM as its core part and had to be implemented in a real chip to demonstrate our dependability approach. As a result of limited design time, our goal was not to achieve e.g. highest seed utilization efficiency or complete fault coverage of the deterministic test-patterns. Instead, priority has been given to the following aspects:

- **A generic design approach**: the design approach is compatible with different DfT design parameters such as the number of primary inputs and scan-chains. Mainstream ATPG test-pattern file formats are supported.
- **A highly automated design process**: a straightforward design process has been chosen for automation to minimize the user interference while generating a new design.
- **Important parameters**: in the case implemented with mainstream UMC/TSMC 90nm technology, the speed of the TPG should be no less than

200MHz. The area of the DM (TPG) should be as small as possible; the upper limit was set to the size of a hard macro of a Xentium processing tile, which is $1.88mm^2$.

- **Fault coverage**: the fault coverage of the TPG will be treated as a tunable dependability parameter to trade off with other parameters, e.g. silicon area of the TPG design. The maximum TPG fault coverage can be as high as that of the deterministic test-patterns.

Two additional blocks, being the control logic and the data-selector blocks, have been implemented with the reseeding circuit to build a complete test-pattern generator. The control block incorporates two counters, i.e. a bit counter and a seed counter. The bit counter counts the number of bits generated by the LFSR to determine when to load the next seed. The seed counter determines which seed shall be loaded next time. Furthermore, the control logic handles all the control signals of the TPG such as: start, stop, restart, pause or resume the test-pattern generation. One important unique feature of the TPG is that its activity can be paused or resumed depending on the status of the network traffic.

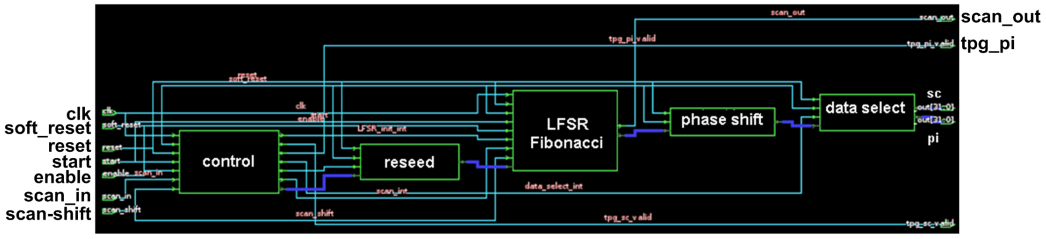


Figure 5.13: Block diagram of the TPG implementation (Cadence software)

The data selector has been incorporated into the TPG to distinguish between the test-vector stream for the scan-chains and the primary inputs of the Xentium tile. In Figure 5.13, an example implementation of the TPG is shown. From left to right, the main blocks are the control block, the reseeding logic, the LFSR, the phase-shifter and the data-selector block. In Figure 5.14, the simulation results of the TPG in a customized testbench is given. In the simulation, the TPG has finished generating the scan-test-vector (`tpg_sc_out`) for the Xentium tile scan-chains and then switches its output to generate the test-vectors for the primary

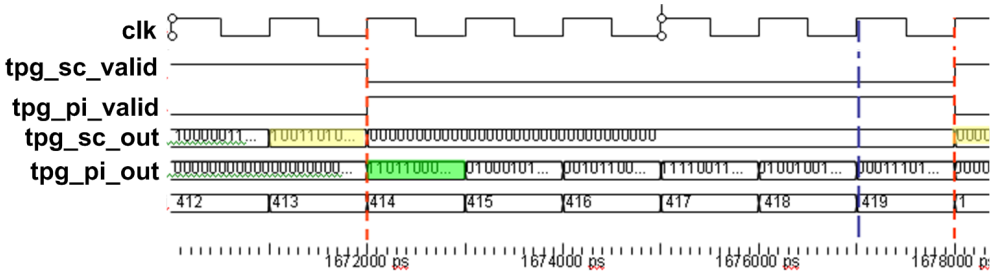


Figure 5.14: Simulation results of the test-vector generation process of the re-seeding TPG

inputs of the Xentium (signal `tpg_pi_out`) for the next six clock cycles. The signals `tpg_sc_valid` and `tpg_sc_valid` indicate the period at which moment the `tpg_sc_out` output or the `tpg_pi_out` output are valid respectively.

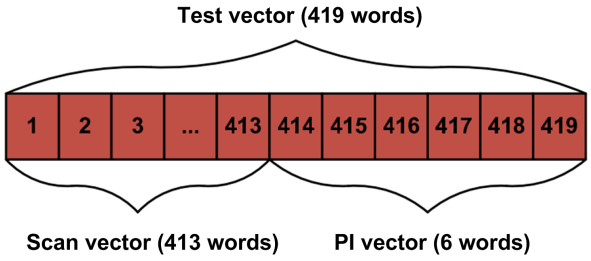


Figure 5.15: TPG output data frames of a complete test-vector

Figure 5.15 shows the output-data frame of the TPG while generating a complete test-vector. Each numbered block represents a 32-bit data word generated from the 32-bit TPG output in each clock cycle. In the first 413 clock cycles of a test, 413 data words are shifted into the 32 parallel scan-chains of the Xentium. In the next 6 clock cycles, test-vectors for the 168 primary inputs are sent to the Xentium in six 32-bit data words (as shown in Figure 4.9 in Chapter 4). More extensive verifications of the TPG functions are covered in the next chapter.

5.4.3 Efficiency of the reseeding method

In order to evaluate the performance of the reseeding TPG, the design has been implemented for different number of test-patterns and synthesized in the TSMC

90nm CMOS technology library able to run at 500MHz. A statistical analysis has been performed on the synthesis results. In the following experiment, the reseeding part of the TPG is synthesized to generate 50, 96, 189, 569 and 1002 test-vectors out of the complete test cube which contains all 1275 ATPG test-vectors for the Xentium processing tile. Figure 5.16 shows the results of the synthesized scenarios.

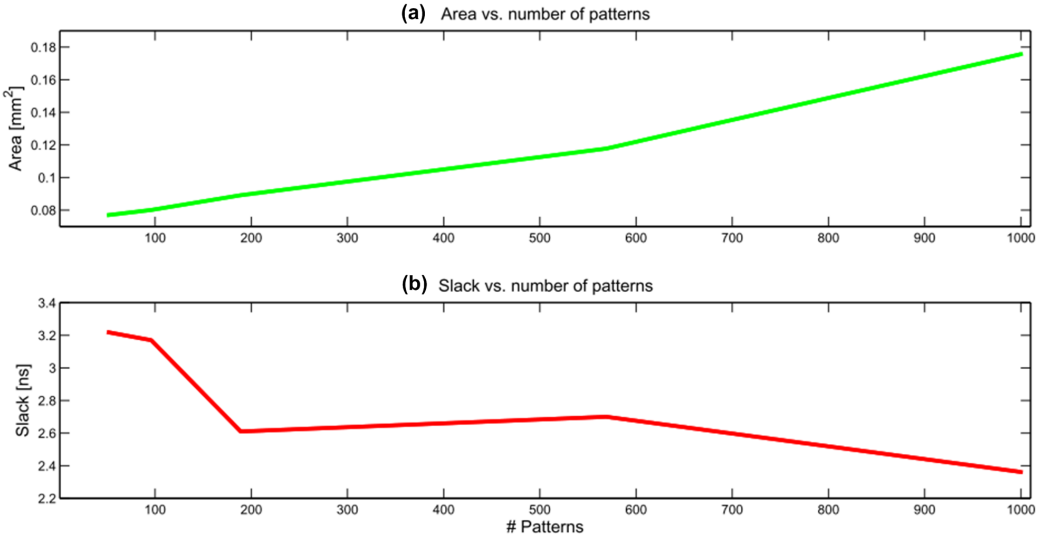


Figure 5.16: Results of the TPG synthesis

In Figure 5.16 (a), it is quite clear that the silicon area of the TPG is proportional to the number of deterministic test-vectors that need to be generated. The slope of the line reflects the length the LFSR; the longer the LFSR, the steeper the slope.

In Figure 5.16 (b), the relation between the slack and the number of test-vectors is shown. *Slack* is defined as the difference between the expected time and the actual arrival time of signals at a certain node within the TPG circuit. A positive slack means the circuit can meet the speed requirement of the synthesis (500MHz in this case) whereas a negative slack means the synthesized circuit fails to meet the speed requirement. In general, TPG gets slower if more test-vectors need to be generated. Note that the main optimization constraint for the synthesis is not the slack and hence any number above zero was acceptable by the synthesis software.

The maximal length of the LFSR has been set to be 1024. In this case, 273 of the 1275 deterministic test-vectors cannot be solved. This means these 273 vectors cannot be generated by the 1024-bit long LFSR. The area of the logic gates for the resulting TPG is about 10% of the Xentium processing tile. The TPG has shown to be able to operate correctly at 500MHz via post synthesis simulations. A new test cube is generated after excluding the 273 unsolvable test-vectors and a fault simulation has been performed on the Xentium tile using a commercial ATPG tool. The new test cube can achieve 88% of the fault coverage of the complete deterministic test-patterns. It should be noted that the length of the LFSR can be increased so that all the deterministic test-vectors can be reproduced by the TPG to achieve a higher fault coverage, of course at the cost of increased silicon area. Using the software we developed, one can fine tune the length of the LFSR, to meet a specific fault coverage, area or time requirements.

5.5 Design of the DM-FSM

5.5.1 Functional overview

All the activities of the DM are controlled by a finite-state machine, referred to as the DM-FSM. The DM-FSM periodically analyzes the instructions given in its *configuration* register and performs tasks e.g. a dependability test accordingly. It also reports the status of the DM or dependability test results in a *status* register to the outside world. Important functions of the DM-FSM are:

- Communication with external blocks;
- Coordination of the activities of the DM-TPG and DM-TRE;
- Function as a BIST controller for the online scan-based test;
- Adjust dependability test activities according to the NoC traffic intensity;
- Control the Xentium tile wrapper and interact with the internal memory BIST engine of the Xentium.

5.5.2 DM-FSM I/O and communication protocol

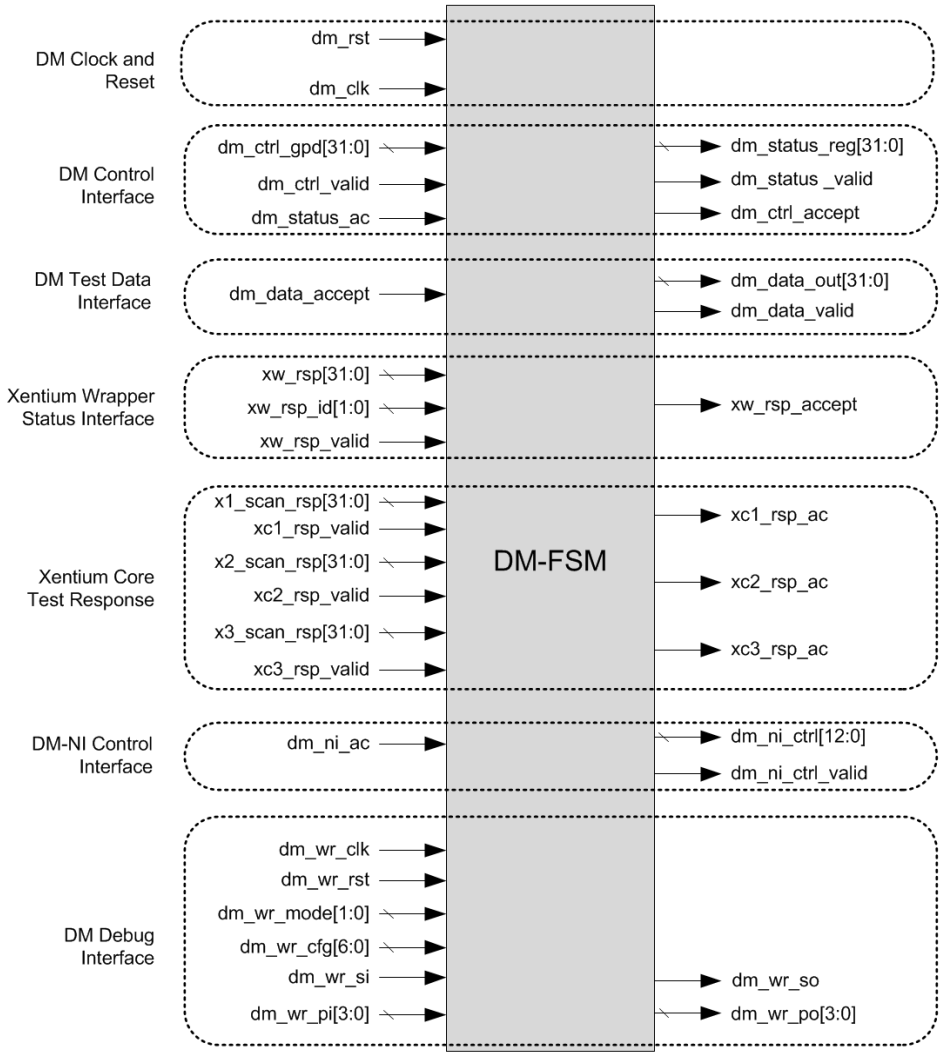


Figure 5.17: I/O interface of the DM-FSM

The complete list of input/output signals of the DM-FSM is given in Figure 5.17. The clock and reset signals of the DM are connected to the global clock and reset pin of the system. A software reset of the complete FSM is possible by writing the reset instruction to the configuration register. To fully control the activities of the DM, instructions are given to the FSM configuration register by an external General Purpose Device via the DM control interface. Similarly, the

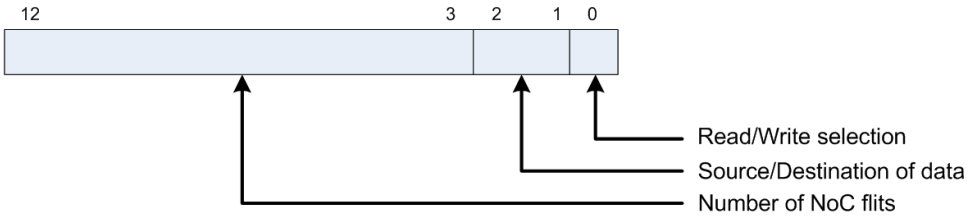
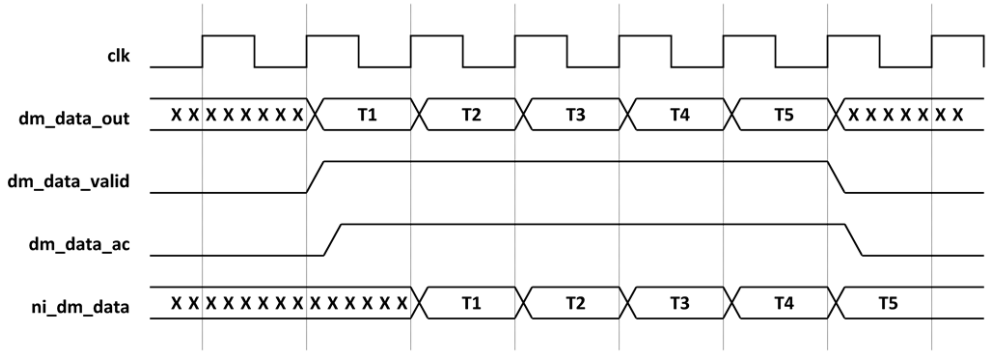


Figure 5.18: Structure of the FSM network control signal

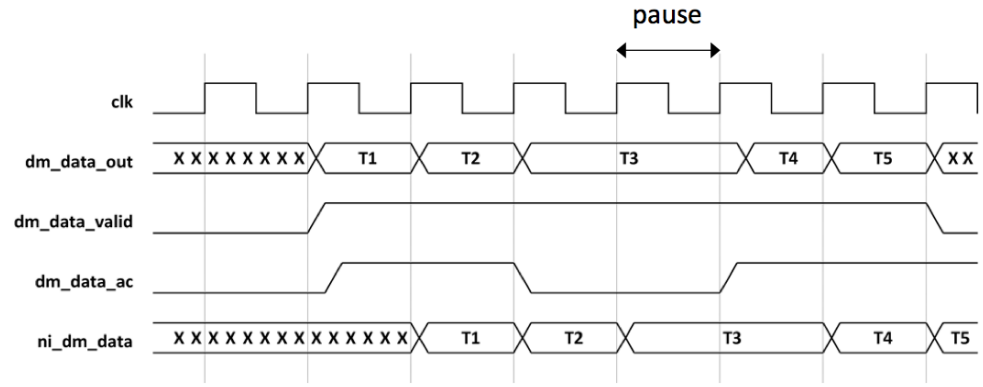
FSM can report its internal status and dependability test results via the FSM status register. Details of the FSM configuration and status registers will be discussed in the next section.

Important DM-FSM outputs include the scan-test-vectors, Xentium tile wrapper (XTW) configuration data and the Xentium memory BIST commands which are all generated via the DM Test Data Interface. It should be noted that, in order to reduce the number of DM outputs, the test-vector output of the DM-TPG has been multiplexed by the DM-FSM to share the same top-level `dm_data_out` port with some other control signals. Meanwhile, the FSM reads the XTW status information and scan-test-responses from the Xentium wrapper Status Interface and the Xentium Core test-response Interface respectively.

The DM-FSM accesses the Network-on-Chip via a dedicated interfacing block, the DM Network Interface (DM-NI). All the communication between the DM-FSM and the DM-NI is handled by a centralized Communication Unit within the FSM. The Communication Unit prepares all the output data which the DM-FSM needs to write and specifies all the input data which the DM needs to read. The property of the data (such as read/write, or data length) between the FSM and the NI is denoted by the `dm_ni_ctrl` signal of the DM-NI Control Interface. Figure 5.18 explains how the 13-bit `dm_ni_ctrl` signal has been organized. The first field from the Least Significant Bit (LSB) specifies whether it is a reading or writing operation between the FSM and NI. The second field indicates the source or destination of the data. It should be noted that the DM-FSM only specifies the name of the target locations. The exact NoC routes are designated by the dependability software running in the GPD and made known to the DM-NI only. The last field is 10 bit long. It states the number of NoC flits (each 32 bit) of the



(a) Normal test-vector generation



(b) Paused/resumed test-vector generation

Figure 5.19: Timing diagram of DM and DM-NI communication. Tx denotes a 32-bit test-vector.

FSM and NI communication; the maximal size of one FSM to NI communication sequence is 1024 NoC flits.

The communication between the DM-FSM and DM-NI follows a simple handshake protocol. A pair of valid and accept signals have been used to synchronize the data flow between the two blocks. Figure 5.19(a) shows an example of the communication between the DM-FSM and the DM-NI. In the example, the signal `dm_data_out` represents the 32-bit test-vectors generated by the DM. The signal `dm_data_valid` accompanies the first output data T1 (test-vector) to notify the DM-NI of the arrival of the test-vectors (i.e., `dm_data_valid` becomes high). Under normal circumstances, the DM-NI should respond by making the accept signal `dm_data_ac` active (i.e., high) and start accepting the test-vector data.

The sampling takes place at the rising edge of the clock and the data contents become available in the DM-NI at the next clock cycle.

If the DM-FSM receives the active `dm_data_ac` signal, it starts to send the next test-vector T2 and continues to send new vectors to the DM-NI as long as the `dm_data_ac` signal is active. The DM-NI can use this feature to control the output flow of the DM. For example, at a certain moment, a communication problem such as NoC congestion occurs and the test-vector generation needs to be paused. The DM-NI thus makes the `dm_data_ac` signal inactive (i.e., low) as shown in Figure 5.19(b). Consequently, the DM-FSM will enable the pause signal of the DM-TPG (which is not shown in this figure) and stall the test-vector (T3) generation. If the communication problem is over, the DM-NI will make the `dm_data_ac` signal active again, and the test-vector generation process will be resumed. The same handshake protocol and pause/resume mechanism also apply to the collection of the test-responses and all other communication between the DM-FSM and the DM-NI [Kerk 10].

5.5.3 FSM architecture

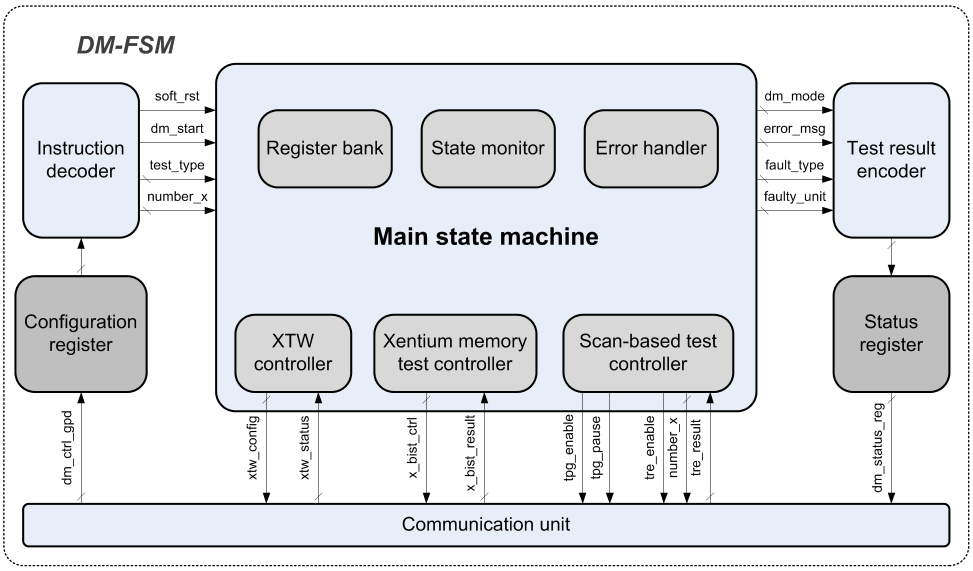


Figure 5.20: The module block diagram of the DM-FSM

The module block diagram of the DM-FSM is depicted in Figure 5.20. It

comprises of a main state machine, a communication unit and configuration/status registers. External instructions are sent to the configuration register, subsequently interpreted and proper control commands are given to the main state machine. The state machine performs the requested dependability test activity and encodes the test results in the status register. In the previous section, the usage of the communication unit to interact with the DM-NI was introduced. The other parts of the DM-FSM will now be discussed in the following section.

5.5.3.1 The configuration register

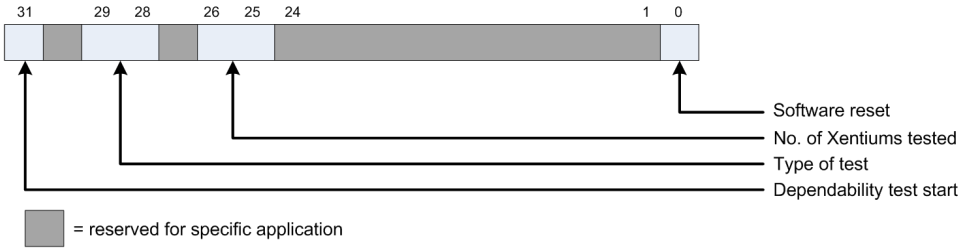


Figure 5.21: Fields of the DM-FSM configuration register

The configuration register of the DM-FSM functions as the interface where external control instructions can be issued to the DM. Upon an FSM reset, the 32-bit register resets to the value 0x0000 0000 which keeps the DM-FSM in idle state and awaits for incoming instructions. The setup of the configuration register is shown in Figure 5.21.

The configuration register contains four important fields:

1. **Dependability test start**: the dependability test can be started by setting the configuration register bit 31 (to logic 1). Clearing this bit will stop the current dependability test.

2. **Type of test**: the DM supports three types of dependability tests being the test of the Xentium core (control logic and datapath), the test of the Xentium internal memory or a combination of the previous two tests. The user can choose which type of test to perform depending on the actual requirement.

3. **Number of Xentiums to be tested**: the maximum number of Xentium tiles which can be tested within one dependability test is three in order to limit the complexity of the DM design as well as the dependability test. The minimum number of Xentium tiles which can be tested is one in a Xentium memory test or two in a Xentium core test. This is a result of the comparison approach of the scan-based test-response which requires at least two Xentiums.

4. **Software reset**: the DM can perform a software-based reset before the next dependability test starts to make sure all the internal registers have been set to default values.

The instructions written into the configuration register will be decoded and proper control information will be extracted and passed to the main state machine as shown in Figure 5.20. To give an example, the instruction with value 0xB600 0000 (hex) will command the DM to perform a full dependability test on three selected Xentiums.

5.5.3.2 The main state machine

After the decoded instructions have been passed to the main state machine, it starts up to perform the requested functions. The main state machine consists of a top-level state loop and several sub-state machines for executing specific tasks. The designated state-transition diagram of the main state machine is shown in Figure 5.22.

One important feature of the DM-FSM is the incorporation of an error-handler module. The error handler uses a hardware timer to time the execution time of each state. In case of unexpected situations, such as a very long NoC communication delay or the case that no responses are provided within the maximum allowed time, the error handler will abort the on-going dependability test, generate an error report and reset the state machine to avoid deadlocks. In Figure 5.22, the DM error state is depicted with a dark color. In case of an error, the FSM can jump to the DM error state from any other state (the transitions are not shown for simplicity).

Other important states of the state machine are:

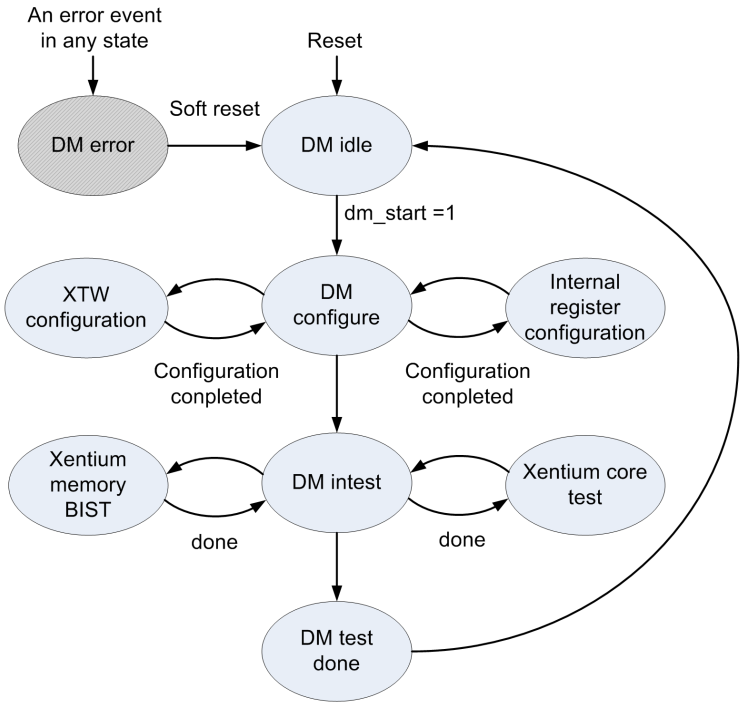


Figure 5.22: Simplified state transition diagram of the DM-FSM

1. **DM idle**: the DM-FSM will enter the idle state after a synchronous reset. It will remain in this state until commanded to start the dependability test (i.e. if `dm_start = 1`). The state machine subsequently switches into the DM configure state.

2. **DM configure**: in this state, the FSM will examine the type of requested dependability test and update the contents of its internal registers to prepare the correct sub-states. The DM-FSM then uses the XTW controller to write to the configuration register of the tile wrapper of target Xentiums and set them into dependability test modes via the NoC. If any failed communication occurs with the target Xentium tiles or if the returned XTW status is not as expected, the error handler will be notified and an error message will be generated. The DM will then enter the DM error state.

3. **DM intest**: the DM intest state is the main state where the dependability tests are carried out. It contains two sub states: the scan-based test for the Xentium core and Xentium memory BIST. Proper tests will be performed according to the type of test specified in the configuration register.

   The Xentium core test state will activate the scan-based test controller in the FSM. The controller coordinates the activity of the DM-TPG and the DM-TRE to perform the test via the NoC. As described in Chapter 4, the test-vector application and test-response collection process of the scan-based test have been decoupled to reduce the high-level control efforts.

   In Figure 5.15, the structure of one test-vector generated by the DM-TPG for the Xentium core was illustrated. The normal test flow for one complete test-vector runs as follows. First, the scan-test vectors are generated by the DM-TPG one by one. These scan vectors are then multicasted to the target Xentiums via the DM-NI as NoC data flit. The XTW receives this data flit, and shifts it into the scan-chains of the Xentium tile. Meanwhile, the XTW counts the number of NoC flits it has received. After all 413 scan vectors have been received, the XTW starts to apply the PI vectors to the primary inputs of the Xentium tile. The DM-TPG is paused after the generation of one complete test-vector. The XTW sets the Xentium tile

into normal mode and let it run autonomously for one clock cycle. Then the DM reads the test-responses from the three Xentiums under test via the NoC, unpacks them at the DM network interface and performs bit to bit test-response comparison to determine whether there are mismatches between them. If the test-responses from the Xentium tiles are identical, the next test-vector will be generated and the test process will repeat until all test-vectors are generated. If the test-response comparison detects a difference, the test process will be immediately terminated and the fault information will be compiled into a fault report and updated to the status register of the DM-FSM. This method can drastically reduce the test time.

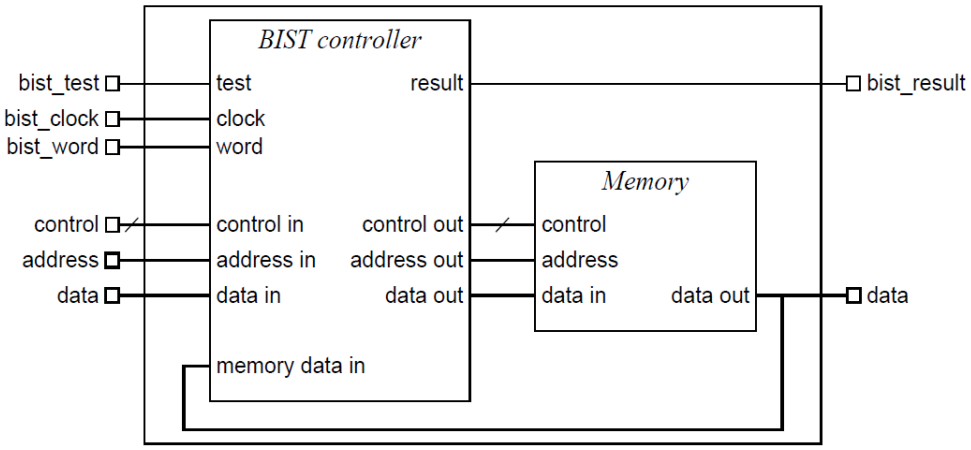


Figure 5.23: Xentium memory BIST controller (ATMEL)

The Xentium memory BIST is carried out separately from the scan-based test. The Xentium memory modules have been equipped with a Built-In Self-Test (BIST) engine from Atmel Automotive. Figure 5.23 shows a block diagram of an example BIST controller for the Xentium memory. A typical Xentium memory test is performed in the following sequence: the DM-FSM writes into the configuration register of the XTW to activate the `bist_test` signal of the memory BIST controller. The BIST controller will start the memory BIST engines and tests all memory blocks in one Xentium. The cumulative test result (pass or fail) becomes accessible from the `bist_result` signal when the test has finished (Figure 5.23). The DM-FSM reads the test result and compiles it into the status register of the FSM

accordingly. The structure of the status register will be treated below.

The FSM will autonomously switch to the next state if all the requested dependability tests have been performed.

4. **DM test done**: if all the requested dependability tests have been carried out, the state machine enters this state and encodes the test results or FSM operation errors into the status register. The dependability software periodically checks the contents of the status register. If it detects that all the tests have been finished, the software will collect the contents of the status register and the soft reset will put the DM-FSM into the DM idle state.

5.5.3.3 The status register

The status register of the DM-FSM serves two main purposes: first, it can indicate the operational status of the DM, including the current FSM state and possible malfunction events. Second, it is used to store the dependability test report compiled from the dependability test results.

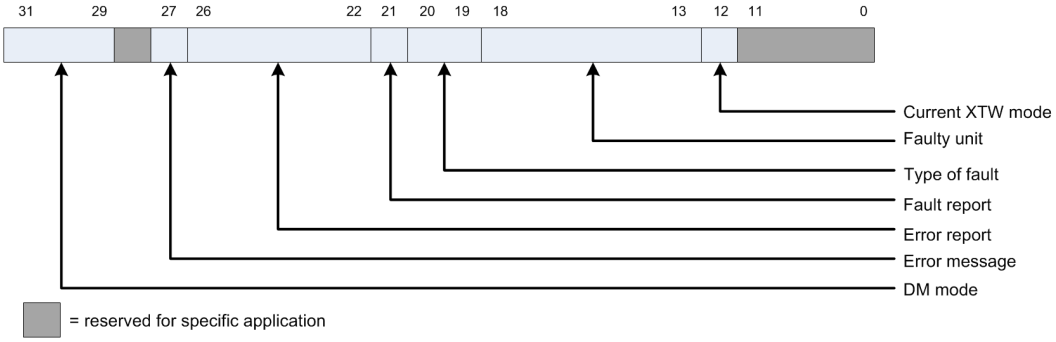


Figure 5.24: Fields of the DM-FSM status register

The structure of the status register is shown in Figure 5.24 and it contains the following fields:

1. **DM mode**: the current state of the state machine is reported in this field of the status register. The four main states of the state machine and its sub states are all included.

2. **Error handling**: the second and third fields of the status register are used to store the reports from the error-handler module. Bit 27 will be set if an error has been detected and the next field will store the detailed information of the error such as the error location and type to assist further debug. Possible type of errors includes communication timeout or wrong command format, etc.

3. **Dependability test report**: bit 21 to 13 of the status register are divided into three fields to hold the dependability test report. Bit 21 indicates whether a fault has been detected in the test or not. If indeed a fault has been detected, bit 20 and 19 describe the type of the fault, either a fault in the Xentium core or in its memory. The last fields indicate the faulty unit number of the Xentium tiles which have been tested.

4. **XTW mode**: the dependability software can verify the status of the XTW after the dependability test to make sure that they have been switched to the normal mode from the dependability test mode. Only after this check is completed, the Xentium tiles can be used by normal applications once more.

As an example, the status register value 0x6036 0000 means: a dependability test has been finished and the DM-FSM is currently in *DM test done* state. There were no errors while running the dependability tests. One fault has been detected while doing the scan-based test on the Xentium core, and the faulty unit ID is No.3.

5.5.3.4 Implementation of the DM-FSM

Due to the complexity of the DM-FSM, it has been designed and implemented using the high-level state-machine design software called StateCAD from Xilinx [Xilinx 07]. All states of the DM-FSM are drawn in the software and transition criteria from one state to another state are specified, as shown in Figure 5.25. The detailed state diagram of the DM-FSM can be found in Appendix A. The StateCAD software contains a simulation environment in which the state transitions of the FSM can be verified. After the behavior of the state machine has been

extensively verified, synthesizable VHDL code of the design can be automatically generated. A top-level VHDL entity design glues the DM-FSM, DM-TPG and DM-TRE together to construct the complete dependability manager.

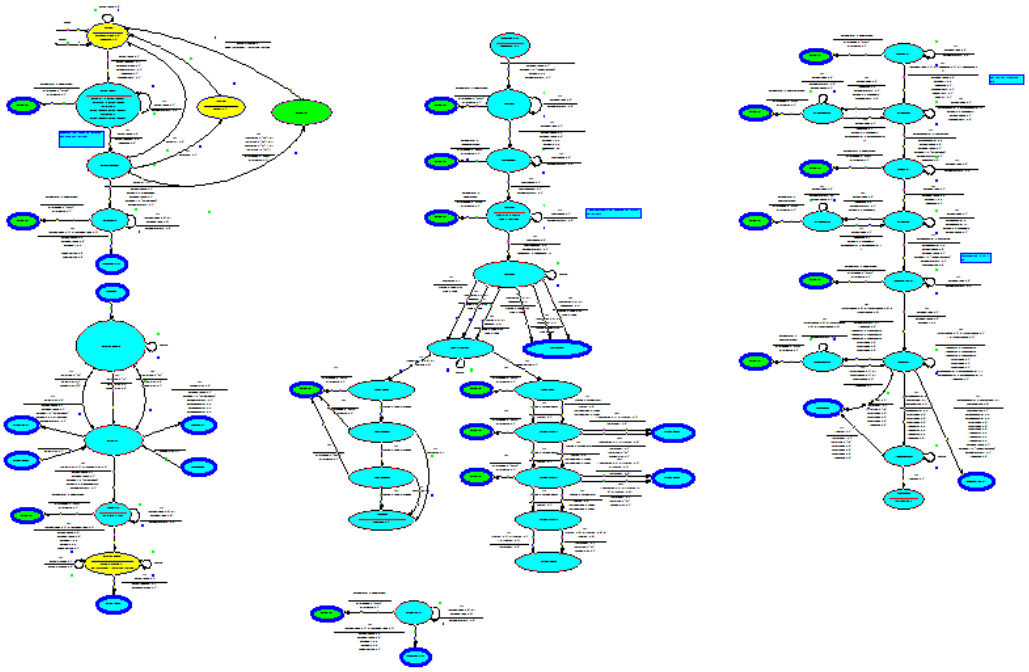


Figure 5.25: Overview of the DM-FSM state diagram in StateCAD of Xilinx (details shown in Appendix A)

5.6 Design of the DM-TRE

In Chapter 4, our motivation was discussed to compare the raw scan-test responses from multiple Xentium tiles instead of checking the compacted signatures. The fact that a large number of identical processor units are present in a homogenous MPSoC enables the use of the comparison method to check the correctness of the dependability test results from multiple processors at the same time. The complete dependability test on the Xentium processing tile begins with the application of the test-vectors to the Xentium tiles, let them run autonomously for one clock cycle and then collect and evaluate the test-responses. As part of the dependability manager, a test-response Evaluator (TRE) has been

designed to carry out the test-response comparison task.

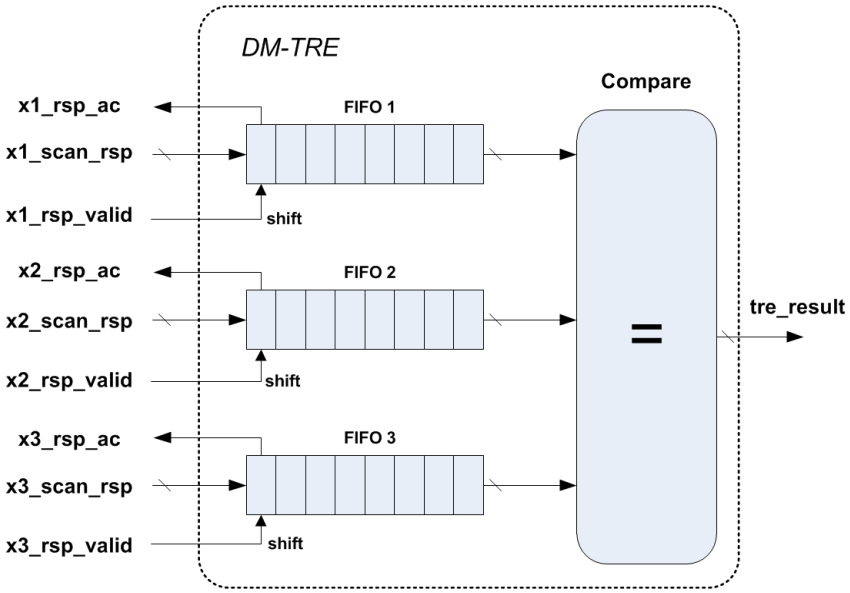


Figure 5.26: Block diagram of the DM-TRE

A simplified block diagram of the TRE has been depicted in Figure 5.26. The design of the TRE essentially comprises a 3-input 32-bits comparator, preceded by three FIFO buffers. The data inputs to each FIFO are three 32-bit `x_scan_rsp` signals, which are the test-responses from each Xentium tile under test in the form of NoC flits passed to the DM-TRE via the DM-NI. Each valid response flit is accompanied by a `x_rsp_valid` signal which enables the FIFO shifting to the right side for one bit. The test-response data stream can flow to the three FIFO channels at a different data rate, depending on the number of NoC virtual channels which were allocated to each stream. The more NoC virtual channels one stream possesses, the faster it can deliver the test-response data to a FIFO channel and the faster this FIFO channel will shift its data. In case one stream is sending test-responses much faster than the other two, the FIFO connected to this stream can become full and will lose test-response data if the stream continues to flow. Hence, an `x_rsp_ac` signal is used by each channel to pause the test-response input in a similar way as the pace of the test-vector generation was adjusted.

The test-responses in the three FIFOs are compared in the comparator and

the result is passed to the DM-FSM by the `tre_result` signal. For three identical Xentium tiles, the test-response should be the same in case they are all fault free. If one Xentium tile is generating a different test-response than the other two, its unit number is reported to the DM-FSM via the `tre_result` signal. A rare but not impossible case would be if the test-responses from the three Xentiums tiles are *all* different from each other. In this case, the dependability test needs to regroup the Xentium tiles and perform more tests to determine which Xentium is really faulty. This also applies if only two Xentium tiles are having their responses compared and a difference is found.

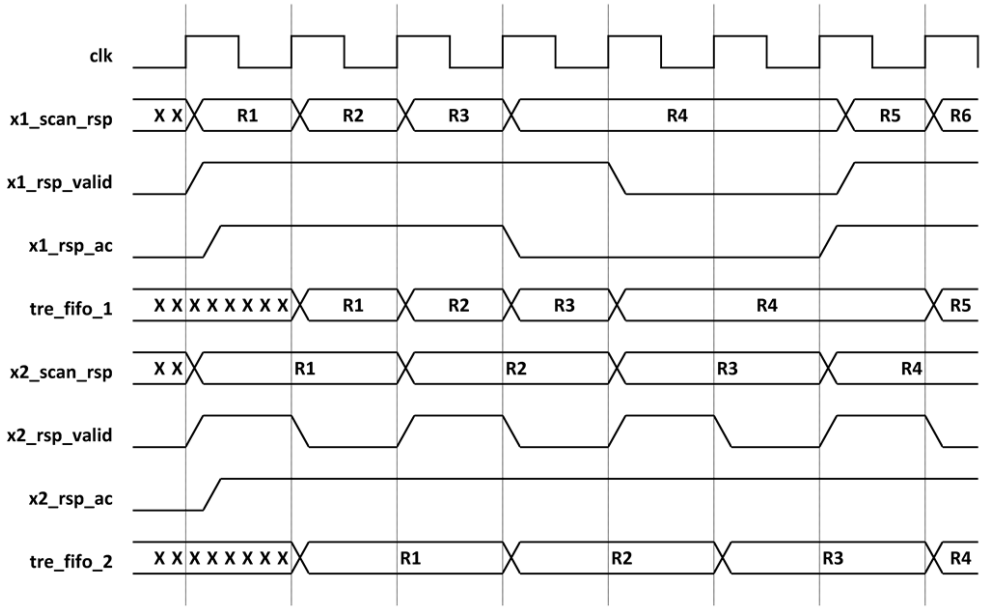


Figure 5.27: Timing diagram explaining the test-response collection process. Channel 1 and 2 have different NoC bandwidth ("x1_scan_rsp" transfers at double the data rate of "x2_scan_rsp".)

A timing diagram showing a test-response collection scenario in the DM-TRE is given in Figure 5.27. In this example, data signal `x_scan_rsp` represents the incoming test-response flit and data signal `tre_fifo` represents the input word of each FIFO buffer. It can be seen that every incoming response flit is shifted into the FIFO with a delay of one clock cycle. It can also be observed that the test-response data of channel 1 is arriving in the TRE at twice the rate of channel 2. In order to prevent the FIFO of channel 1 from overflowing, signal `x1_rsp_ac`

was set to "low" to function as a pause signal in order to halt the test-response flow `x1_scan_rsp`. If the same response flits (e.g. R1) in both channels have been compared with each other, they will be released to free FIFO buffer positions for more incoming responses. In this example, `x1_rsp_ac` was set to "high" again to allow more test-responses into FIFO 1.

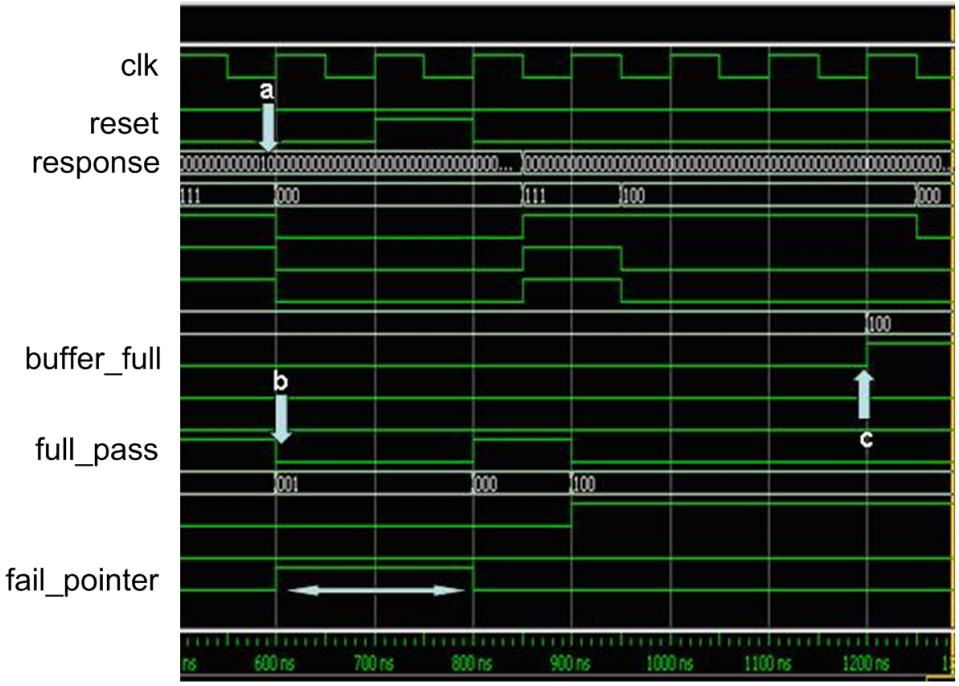


Figure 5.28: Simulation of the DM-TRE functionality

The DM-TRE block has been designed and implemented in synthesizable VHDL code and glued with the DM-FSM by a top-level entity. Functional simulation of the TRE behavior has been performed to verify whether the basic functional requirements have been met. Figure 5.28 shows one of the simulation results. In the simulation result, the first 2 signals on top are the clock and reset signals. They are followed by the 32-bit test-response data from the Xentium(s). Label a indicates that a faulty response has been emulated/injected. This causes signal `full_pass` at b to report that a fault has been spotted during the comparison and hence a Xentium core has failed. The signal `fail_pointer` using the bidirectional arrow indicates during which test-vector the comparison noticed a difference (caused by the fault injection). At label c, the `buffer_full` signal

indicates that the buffers in the TRE are full, and hence no new data can be read in. Post synthesis simulation suggests that the TRE circuit could operate beyond the required 200MHz speed.

5.7 Design of the DM Network Interface (DM-NI)

5.7.1 DM-NI overview

The Dependability Manager Network Interface (DM-NI) is a bridge between the NoC and the Dependability Manager [Kerk 10]. It takes care of the bidirectional communication between the DM-FSM, DM-TPG and DM-TRE at one side, and the NoC at the other side. The network interface has the following key functions:

1. **Control of the DM**: accept instructions from the GPD and communicate the status of the DM to the GPD via the NoC.

2. **NoC configuration**: at the request of the GPD, the DM-NI will configure the NoC, for which the GPD sends the proper header to construct the multicast connection for the dependability test. As discussed in Chapter 4, multicast is a dedicated NoC feature for the dependability test. It can broadcast the data from one source to multiple target destinations. After the dependability test, the DM-NI will perform a proper termination of connections, i.e. closing and freeing all the NoC connections from the DM to the Xentium tiles which were tested.

3. **Dependability test routing**: write the DM test-vectors to the XTWs through a NoC multicast connection. Read the Xentium tile test-responses via two or three separate NoC connections.

The DM-NI has two physical connection points (two routers) to the NoC. These two connections can be used simultaneously in the case of transporting the dependability data. The DM-NI itself has no knowledge of the topology of the NoC. The dependability software running in the GPD is responsible for

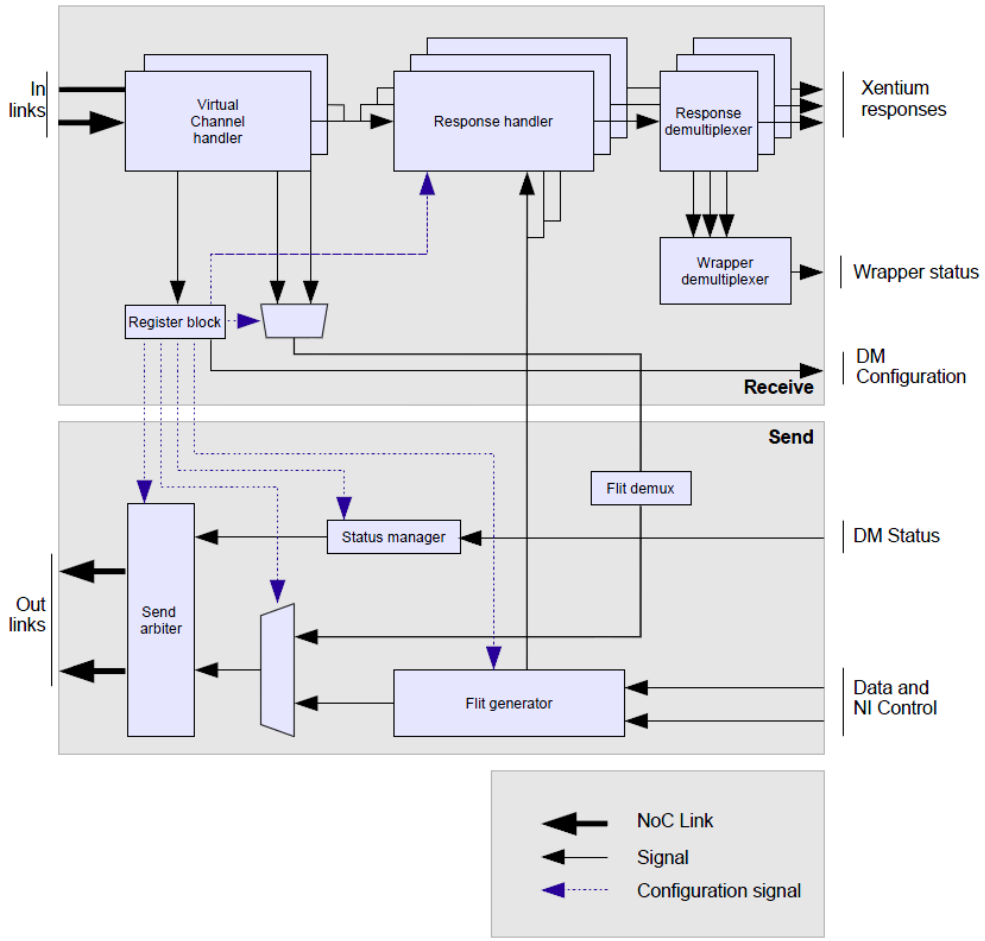


Figure 5.29: Block diagram of the DM-NI design [Kerk 10]

supplying the DM-NI with the headers that can setup the multicast connection to the Xentium tiles under test and the unicast return connections from the Xentium tiles. It is possible to configure how the DM-NI is connected to the NoC from the GPD. Both physical connections are able to handle all types of communication. It is also possible to run a dependability test using only one router. In that case, the complete test process will become slower than using two routers but the overall communication pressure on the NoC will be reduced.

5.7.2 DM-NI architecture

The basic scheme of the DM-NI, divided into both a sending and a receiving part, is shown in Figure 5.29. After an asynchronous reset, the DM-NI will generate a *tail* flit on all the virtual channels of both routers. This is necessary to ensure that any remaining NoC connections are terminated upon the reset.

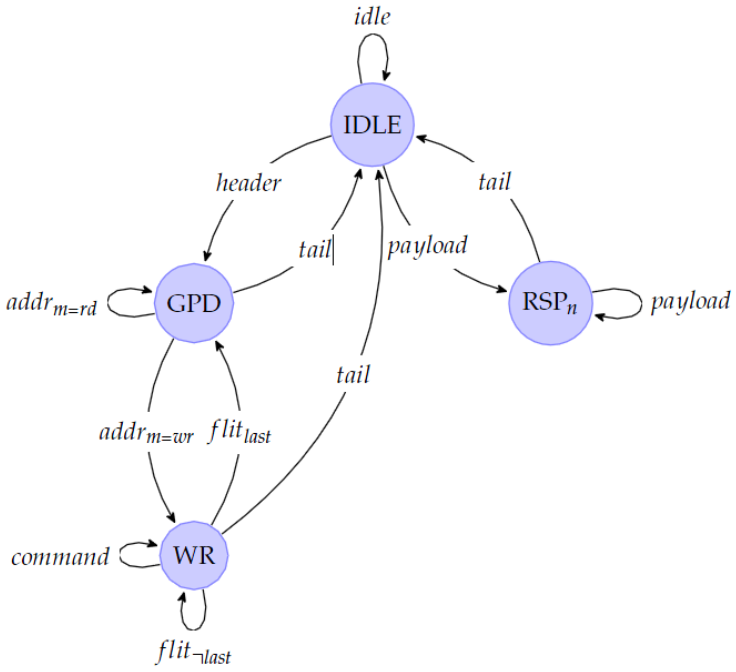


Figure 5.30: State-transition diagram of the Virtual Channel Handler

As shown in the top left corner of Figure 5.29, data flits delivered from the NoC arrive in the DM-NI via the *in-links*. The incoming data are subsequently handled by two *Virtual Channel Handlers* (VCH). Each VCH is connected to one NoC router. Figure 5.30 shows the state-transition graph of the VCH. A brief explanation of the VCH working flow will now be given. If new NoC data arrive, the VCH will leave the *Idle* state and deliver the input flits to proper output ports depending on the property of the input flits. There are in total three output ports, being the *Xentium responses* port, the *wrapper status* port and the *DM configuration* port as shown at the right side of Figure 5.29.

If the NoC flits are payloads containing the Xentium test-responses, the VCH will enter the *RSP* state (Figure 5.30). Subsequently, the payload data are

buffered in the *Response handlers* (Figure 5.29), multiplexed to the Xentium response port and sent to the scan response input `x_scan_rsp` of the DM. The data transport will be terminated if a tail flit arrives in the VCH.

If the NoC flit type is header, it will be further judged whether this data comes from the XTW or from the GPD by checking their source address. If the NoC flit comes from the XTW, they are sent to the wrapper status port and delivered to the `xw_rsp` input of the DM. If the flit comes from the GPD, it will be again checked whether it is DM configuration data or if it is NoC route configuration data. In the first case, the flit will be buffered in the *Register block* and sent to the DM configuration register. In the second case, the NoC route configuration data will be sent to the *Flit generator* (Figure 5.29) to form the header information (NoC routes) for the outgoing test-vector flits. A tail flit will bring the VCH back to the Idle state via polling (Figure 5.30).

If the GPD wants to read the status register of the DM, register content is read into the DM-NI *Status manager* and sent to the GPD via the *Out-links*. In this communication, the GPD is the master and the DM-NI is the slave. If the DM-FSM tries to write test-vectors to the Xentium tiles, the DM-NI packs the test-vector data into NoC flits using the Flit generator and sends them to the target Xentiums via the *Send arbitrator* (Figure 5.29). In this case, the DM-NI becomes the master of the communication and the XTWs are the slaves.

5.7.3 DM-NI simulation results

Figure 5.31 shows a QuestaSim simulation to show the communication between the GPD, the DM and the DM-NI. Due to space limitation, not all signals are treated; only important parts of the simulation are explained. At the left side, signal group `a` consists of the In-links and Out-links of the DM-NI. Group `b` includes the DM configuration and status signals. The test-responses of the three Xentiums under test are shown in group `c`. The last group of signals `d` shows the DM and DM-NI communications.

At stage (1) of the simulation (Figure 5.31), the GPD addresses the NI via the NoC, commanding the DM to start a test. As a result, the NoC Out-link connection is properly configured (2). The DM acknowledges the reception of

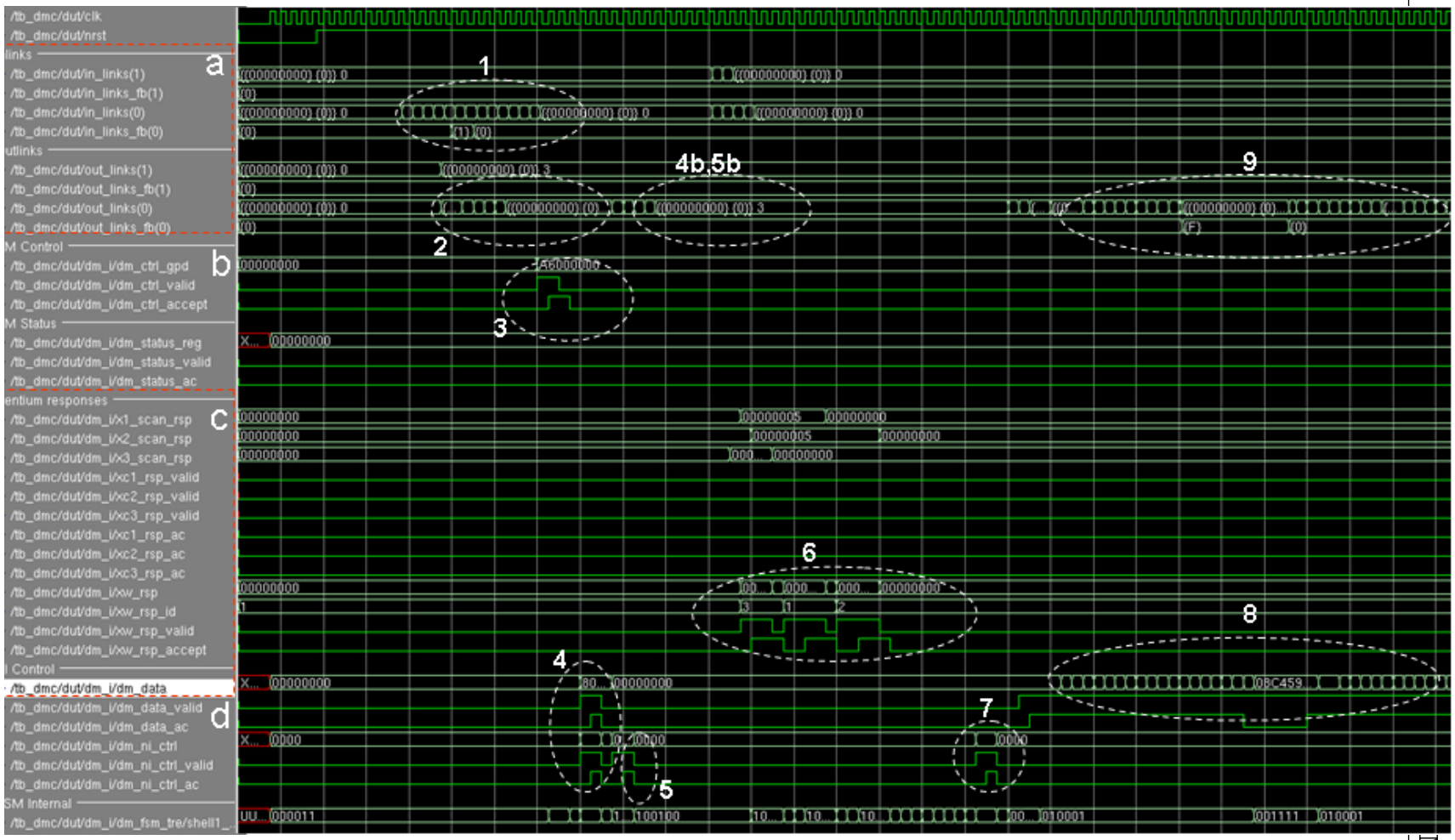


Figure 5.31: Simulation results of the DM-NI [Kerk 10]

the command and gets ready to start a test (3). In stage (4) and (5), the DM configures the related Xentium tile wrappers into test mode. The Xentiums under test send back their readiness signals at (6). In (7) , the confirmation is sent to the DM that all NoC routers are setup and all Xentiums under test are ready and standing by. Immediately the DM starts to generate test-vectors at (8). The test-vectors are subsequently sent over the NoC and multicasted to the Xentiums under test at (9).

5.8 A dependable DM

Because the DM is fabricated using the same semiconductor processing technology as used for other parts on the silicon die, any faults which can occur in the processor cores, can also occur within the DM. To ensure the effectiveness of our dependability approach, a "fault-free" DM as well as a "fault-free" NoC are the essential prerequisites so that all dependability tests can be performed correctly. Therefore, the dependability manager itself has been made dependable too.

If the implementation of the DM does not require too much silicon area overhead, a quick solution could be introducing spare DM(s) into the MPSoC. For example, assume the reliability of one DM block is 90%. It can be easily calculated that the introduction of a spare DM can increase their overall reliability to 99% by using the equations discussed in Chapter 3. As such, the DM itself becomes a 1-out-of-2 good system. Of course it is up to the user to determine whether this overhead is acceptable or not.

An alternative approach is to add debug/test infrastructures to the DM such that its internal states and correctness can be observed and verified as demanded. An example setup is shown in Figure 5.32, which mainly consists of wrapper cells and multiplexers. A customized wrapper has been designed, which is located in between the DM-NI and the data and control signals of the DM. The wrapper is transparent in normal functional mode, allowing the DM to communicate with the DM-NI as normal. If the wrapper is set to test mode, the DM then receives or sends data from/to the external DM debug pins. The wrapper is directly controlled via external DM test pins. Some important pins have been shown in Figure 5.32, such as external clock and reset pins, wrapper read/write pins and

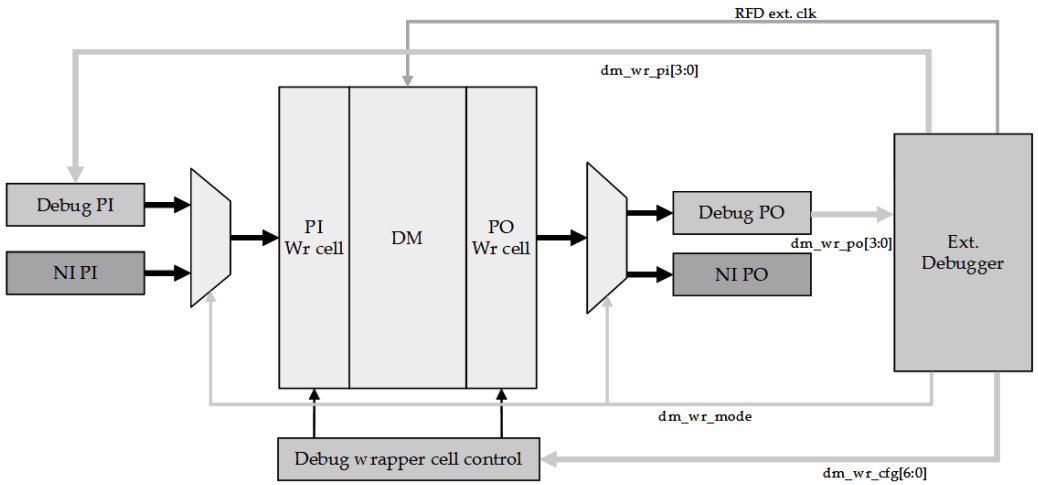


Figure 5.32: Basic setup of the DM debugging/test infrastructure

wrapper control pins. The primary inputs `dm_wr_pi` and outputs `dm_wr_po` are currently 4-bits wide, but they can be reduced in future to 1-bit wide to reduce the pin count at the cost of a lower test speed. Essentially, the DM test setup makes the functional behavior of the DM controllable and observable from outside. As such, one can verify the correctness of the DM externally.

While operating in the field, a periodic test of the DM can be carried out with the help of e.g. the general purpose processor device. For example, the DM can generate the test vectors and the GPD can broadcast and route the vectors back to the DM-TRE to make a comparison. In this way, a thorough test of the key components of the DM, being the DM-TPG and DM-TRE, is performed. The GPD can even alter the test vector in one virtual channel to emulate faults in the test responses. These approaches can easily test the functionality of the DM and guarantee its correctness during its mission time. A conventional scan-based structural test of the DM and other parts on the MPSoC is still performed after manufacturing as will be shown in the next chapter.

5.9 Conclusions

The Dependability Manager has been designed and implemented as a stand-alone Infrastructural IP for the dependability test of an MPSoC containing an array of Xentium processing tiles. The DM consists of a test-pattern Generator for test-vector generation, a test-response Evaluator for test-response evaluation and a Finite State Machine for internal control and communication with special dependability software running on the GPD. The reseeding technique has been adopted in the design of the DM-TPG to achieve test-vector compression. A Matlab software tool has been developed to automate the design process of the DM-TPG in a generic and fast manner. Synthesis results proved that the essential requirements for the DM-TPG design have been satisfied. The DM-FSM block has been developed using the StateCAD software from Xilinx. The functional behavior of the DM-FSM has been extensively simulated and synthesizable VHDL code has been generated. The DM-TRE has been designed using VHDL and a top level entity has glued the three blocks together to form a complete dependability manager. It could be available as a stand-alone IP.

A network interface for the DM has been developed. The DM-NI bridges the communication from the DM to the external environment. The control of the DM and the transportation of the dependability test data are all carried out via the DM-NI. Last but not least, a DM test setup containing DM wrappers and multiplexers has been designed to provide a possibility for DM functionality verification externally.

A dependable DM is the starting point of the dependability approach proposed in this research. Various methods such as the usage of a second DM and the introduction of debug and test infrastructures have been proposed to enhance the dependability of the DM. In addition, periodic test of the DM can also be carried out using a general purpose processor at application run-time to ensure its correctness during field operations.

Chapter 6

Implementation, Verification and Experimental Results

ABSTRACT - *In previous chapters, the concept of a dependable MPSoC has been introduced. Design for dependability approaches as well as the infrastructures required for performing dependability tests in an MPSoC have also been discussed. The essential part of the dependability approach is the hardware Dependability Manager (DM) built into the MPSoC to perform dependability tests and evaluate the test results. Details of the architecture of the envisioned DM have been given in Chapter 5. One of the research goals of this thesis is to investigate the impact of the proposed dependability approach on a real MPSoC. As important system parameters such as power dissipation, area and maximum clock frequency as well as our novel dependability test methods can only be evaluated on a hardware platform, this chapter is focused on the hardware implementation and verification of the DM. Measurement and experimental results are provided to validate the DM design as well as the proposed dependability approach in a real homogeneous MPSoC platform.*

Parts of this chapter have been presented at the IEEE 17th Pacific Rim International Symposium on Dependable Computing, Pasadena, CA, USA [Zhan 11], and at the IEEE 8th International Conference on Design & Technology of Integrated Systems in Nanoscale Era, Abu Dhabi, UAE [Zhao 13].

6. IMPLEMENTATION, VERIFICATION AND EXPERIMENTAL RESULTS

As a result of the limited man power and time budget within this project, little effort has been devoted into the optimization of the area and power consumption of the DM design. Instead, the primary goal is to prove the concept of our dependability approach and the feasibility of the dependability test method at application run-time. As a result the functional verification of the DM and its performance in a homogeneous MPSoC framework form the key contents of this chapter.

The complete design and verification flow of the DM is shown in Figure 6.1. The flow comprises three major stages, being DM design and verification on an FPGA, a tailored MPSoC verification on an FPGA and the full MPSoC implementation in a chip. Each stage is further introduced as follows.

In the first stage, three key components of the DM, being the DM-TPG, DM-FSM and DM-TRE, have been designed using the hardware description language VHDL. These components are first separately simulated in a VHDL testbench, then integrated into a higher level DM entity. After the integration of the DM, simulations to examine the overall functionality of the DM block are performed and then synthesizable VHDL codes of the DM are generated.

Before the DM is implemented into silicon, it is common practice to carry out extensive simulations and verifications in order to examine both the DM internal functional blocks and the DM behaviour when integrated into an MPSoC as an infrastructural IP. However, it has been observed that several hours (real time) were required for the DM to generate all the test-vectors in a basic cycle-accurate simulation using a commercial VHDL simulator and even days for more complex system-level experiments. Thus it was necessary to adopt a hardware-based platform (hardware prototyping) to accelerate the simulation and verification process. A Field-Programmable Gate Array (FPGA) device is usually used for rapid system prototyping due to its flexibility, ease of use and lower cost compared to an Application-Specific Integrated Circuit (ASIC). In addition to the acceleration of the simulation process, correctness of the DM design can be verified on the FPGA platform (DM block verification in Figure 6.1).

As the DM has been designed to enhance the dependability of an MPSoC, we cannot thoroughly evaluate the DM performance and our dependability approach without integrating the DM into an MPSoC framework. Thus in the second stage

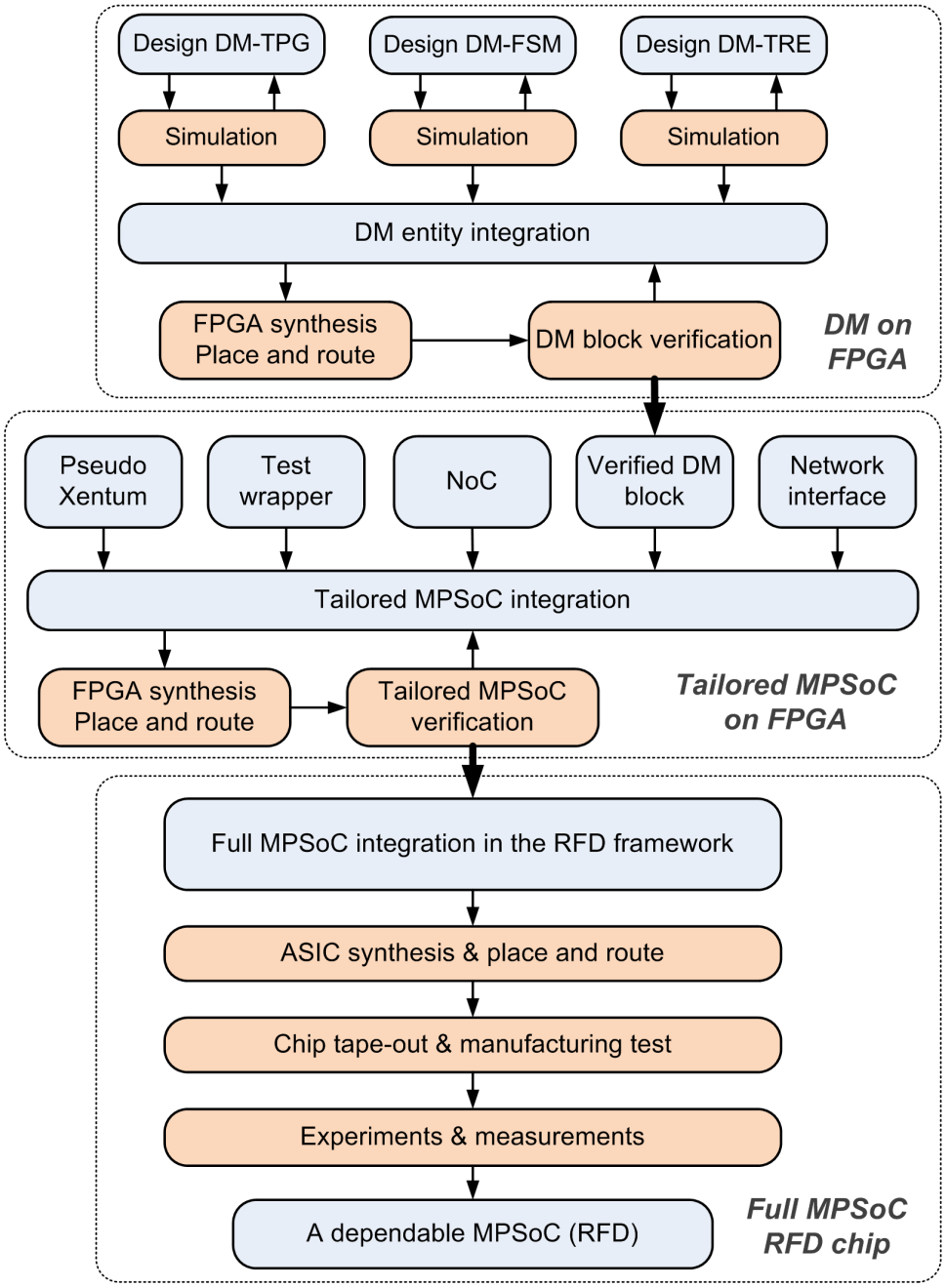


Figure 6.1: Design and verification flow of a dependable homogeneous MPSoC

of the flow, the target MPSoC has been tailored and implemented in the FPGA to provide a NoC-based many-processor SoC environment for the verification of the DM. In addition to the DM, the tailored MPSoC consists of other key dependability infrastructures such as a ring-topology NoC, DM network interface, Xentium tile wrapper and pseudo Xentium tiles, etc. Important issues to be examined on this platform include feasibility of the test-vector multicast mechanism, handling of test-responses from multiple sources in presence of NoC latency and DM-FSM behaviour given complex inputs, etc.

In the third stage, after the DM-IP has been fully verified in the tailored MPSoC environment, it was integrated into a homogeneous MPSoC with nine Xentium tile processors and a virtual channel NoC. This MPSoC has been fabricated using UMC 90nm CMOS technology and referred to as a Reconfigurable Fabric Device (RFD). Measurements and experiments have been carried out to test the important features of the DM in the RFD chip and the dependability improvement of the RFD has also been evaluated by calculations.

The remainder of the chapter is organized as follows. In section 6.1, we present the implementation and verification details of the DM in an FPGA platform thoroughly. Facts about the DM realization in an ASIC standard-cell design are given in section 6.2. Measurement and experimental results of the DM in the RFD ASIC are shown in section 6.3. Power dissipation of the dependability test has been analyzed in section 6.4. An actual power measurement has been carried out as well and the results are shown. In section 6.5, the system dependability improvement has been evaluated and discussed. In section 6.6, the conclusions will be drawn.

6.1 FPGA-based implementation and verification

6.1.1 Introduction

As already shown in Figure 6.1, FPGA-based simulation and verification of the DM design have been carried out in two stages. First, the DM has been im-

plemented in an FPGA device as a standalone IP and functional verifications have been performed to check the correctness of basic DM functions, such as test-response handling and FSM-state transitions. A testbench for the DM has been designed in VHDL and implemented in the FPGA. The testbench can directly generate the input signals of the DM on every clock cycle and collect its responses. Several communication protocols, such as RS232 or USB are available to communicate with the testbench via dedicated ports on the FPGA board. PC client software with a graphical user interface (GUI) has been developed using Visual Basic software. One can assign certain input signals to the DM and observe its output results using the PC client software. The primary goal in this phase is the functional verification of the DM itself.

In the second stage, some key dependability infrastructures such as the NoC, Xentium tile wrappers and DM network interface are integrated into an MPSoC framework with reduced size and functionality. This tailored MPSoC is also implemented into the FPGA to provide an environment for carrying out system-level experiments. As a series of system-level experiment cases have to be performed, it is difficult to generate all input stimuli manually. Thus a test stimuli generator (TSG) has been designed and instantiated in the FPGA. The TSG can bring the complete system to a predefined state after reset and provide appropriate test stimuli to trigger certain actions, such as invoking the DM to perform a Xentium memory test or a scan-based test. The DM test results can be collected and evaluated by an embedded logic analyzer program from the FPGA manufacturer (ChipScope) or an external Agilent logic analyzer.

A high-density and high-performance FPGA device, the Xilinx Virtex-4 LX200, has been chosen for the experiment as it is sufficiently large to implement the DM design and all the hardware components in the experimental framework. Table 6.1 shows a list of the total resources available on the Virtex-4 FPGA device.

All the software tools and equipment used in the experiment are listed below:

- QuestaSim for RTL Simulation;
- Precision RTL Synthesys and Xilinx ISE for design synthesis, placement and routing on the FPGA device;
- Visual Basic for PC client programing and GUI design;

Table 6.1: Resources available in the Xilinx Virtex-4 LX200 FPGA device. A slice is an elementary programmable logic block in a Xilinx FPGA.

Resources	Amount
Slices	89088
Registers	178176
LUTs	178176

- Matlab script for comparison of the generated test-vectors with the original ones in the ATPG test pattern file;
- ChipScope Pro for FPGA internal signal inspection;
- An Agilent 16823A logic analyzer for analyzing the FPGA output signals.

6.1.2 DM Verification

To verify its basic functions, the complete DM, including the TPG, TRE and FSM, has been synthesized using Mentor Graphics Precision RTL synthesis and implemented in the Xilinx Virtex-4 FPGA device. Table 6.2 lists the resource usage for the implementation of the basic system. The maximum clock frequency of the system should be able to reach 200MHz.

Table 6.2: Total resources used by the basic system, relative to the total resources.

Module	Slices		Registers		LUT	
	Amount	Percent	Amount	Percent	Amount	Percent
DM	17273	19.4	1898	1.1	27205	15.3
Testbench	370	0.4	688	0.4	546	0.3
Total	17643	19.8	2586	1.5	27751	15.6

A block diagram of the verification framework is shown in Figure 6.2. The DM block is directly connected to the testbench block while the clock manager has been implemented as a separate IP. The testbench block can communicate with client software running on a PC via the RS232 or USB protocol. The client software is programmed using Visual Basic and offers two Graphical User Interface (GUI) to verify separate parts of the DM.

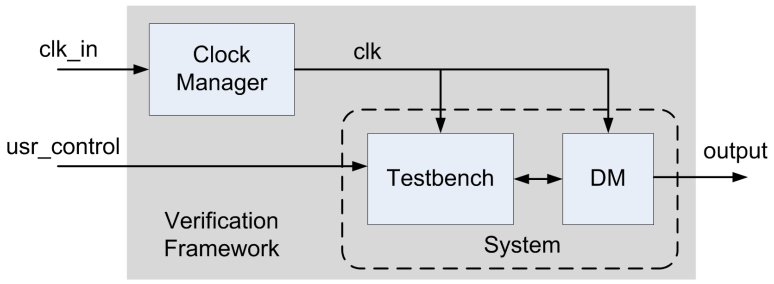


Figure 6.2: Overview of the DM verification framework

6.1.2.1 Verification of the DM-TRE

The Test Response Evaluator (TRE) of the DM has been designed to compare the test-responses returning from multiple Xentium tile processors under test. As all the Xentium tiles have the same structure, identical test-responses are expected given the same test-vectors are applied to each Xentium. If one Xentium generates a different test-response from other Xentiums, it can be considered as faulty. The DM-TRE can queue the test-responses in its local buffers in case of delayed arrival of test-responses from a certain Xentium caused by NoC latency. Figure 6.3 shows an example scenario if test-responses from Xentium 3 arrive later at the DM-TRE than the other two Xentiums due to e.g. limited bandwidth.

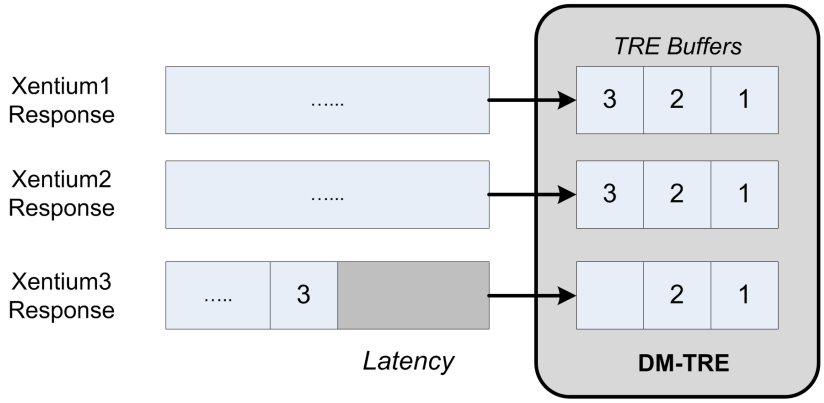


Figure 6.3: Test response evaluation with NoC latency. Each numbered square represents a 32-bit test-response data package.

A Visual Basic program *DM-Verify* has been developed to control the test-bench block connected to the DM. The client side GUI of the software is shown

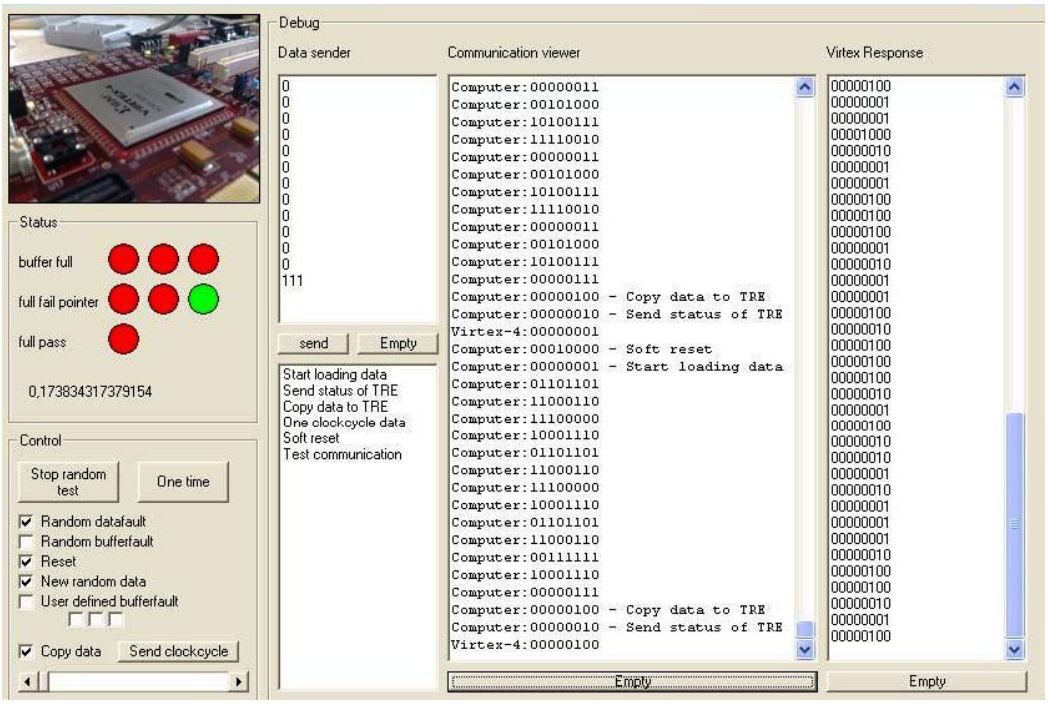


Figure 6.4: *DM-Verify* program user interface for the DM-TRE test

in Figure 6.4. On the top left corner, important TRE output signals such as *buffer full* or test *fail/pass* are shown in an intuitive way to indicate the status of the TRE under test. In the left column, operation instructions and related data signals can be specified by the user and pushed to the testbench. The communication between the client and the TRE and corresponding TRE responses are shown in the two columns at the right side. The testbench can use either pre-defined test-response stimuli or random response values to test the TRE.

Experiments have been carried out to verify the DM behaviour in several typical scenarios. These scenarios and the experimental results have been summarized in Table 6.3. All important DM-TRE functions have been verified and the verification results confirm that the DM-TRE design is correct.

6.1.2.2 Verification of the DM-FSM and DM-TPG

A Finite State Machine (FSM) in the DM can receive control commands via its programmable interface, being a 32-bit control register. Hence the DM-FSM will perform proper actions according to the command received. One of the key

Table 6.3: DM-TRE experimental results

Experiment	Result
Same random test-responses broadcasted to each channel at a high speed (200MHz)	*no responses mismatch* reported
Pre-defined test-responses to emulation a fault in one channel	*one faulty channel* reported
Pre-defined test-responses to emulate all three channels receive different responses	*three channels all different* reported
NoC latency emulation: delay the test-response arriving at one channel for several clock cycles	earlier responses correctly buffered
Long latency emulation: delay the test-response arriving at one channel for a longer time	*buffer full* and *pause* signal correctly generated

dependability tests is the scan-based test of the Xentium tiles, in which structural test-vectors are generated by the DM Test Pattern Generator (DM-TPG). The *DM-Verify* program GUI for verifying the DM-FSM and DM-TPG is shown in Figure 6.5.

At the left side of the GUI are the input signals which will to be applied to the DM. For example, field (1) `dm_ctrl_gpd` emulates the control signals which will be sent to the DM from the General Purpose Device (GPD) in the system. Instead of using the GPD, the user can manually type in the control signals to determine the operations the DM-FSM will perform. In field (2), the three `x_scan_rsp` signals are used to emulate the test-responses returned from Xentium processors being tested. In field (3), `xw_rsp` represents the value returning by the Xentium tile wrapper status register. During the verification process, specific values can be filled into these input fields and they will be assigned to the DM with the press of the *Put Data* button (at (7)).

At the right side of the GUI, the output signals of the DM can be monitored at run-time. The (4) `dm_data` field is a 32-bit word which can be used for test-vector generation or Xentium tile wrapper control signals generation in different operation scenarios. Field (5) `dm_status_register` indicates the intermediate

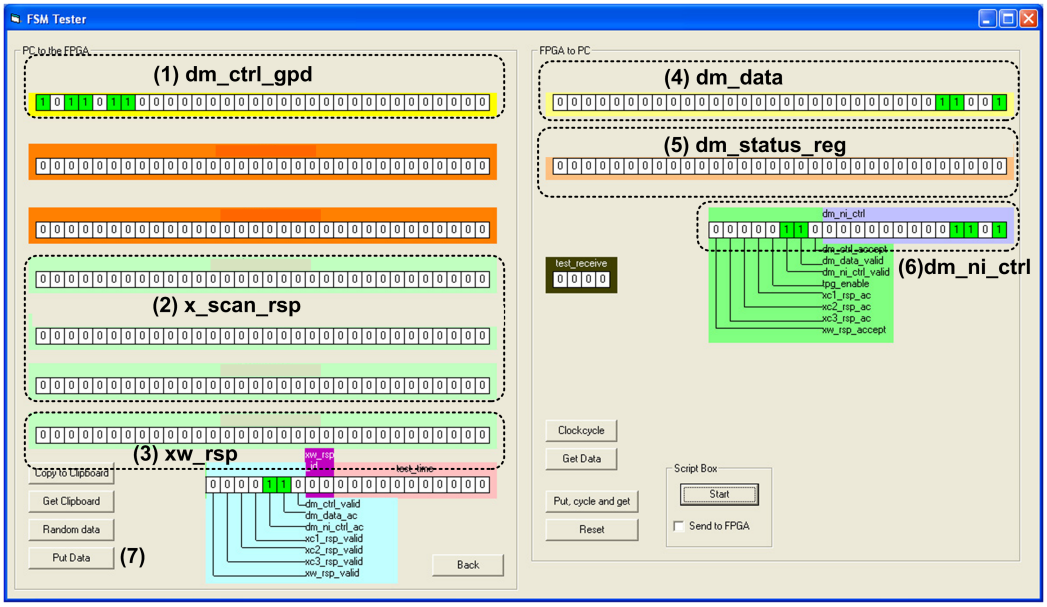


Figure 6.5: *DM-Verify* program user interface for DM-FSM and DM-TPG test

states of the DM-FSM as well as the final test report of a dependability test. In field (6), signal `dm_ni_ctrl` shows all the communication between the DM and DM-NI.

In summary, using the *DM-Verify* program, one can monitor the DM-FSM state transitions and verify its key functions. For example, one can verify whether the DM has generated the correct signals to configure the Xentium tile wrapper or whether the NoC time-out function has been correctly performed to avoid deadlocks. By storing the scan test-vectors generated by the DM into a local log file, one can also verify whether these vectors are the same as the ones generated in the ATPG test pattern file.

A list of experiments and their results is given in Table 6.4. The DM-FSM behaves correctly and the test-vector values generated by the DM-TPG are as expected. Both designs have met their functional specifications.

Table 6.4: DM-FSM and DM-TPG experimental results

Experiment	Result
Monitor the DM state register and go through all operational modes	all DM-FSM state transactions are correct
Monitor the `dm_ni_ctrl` register to check the handshake signals between the DM and DM-NI	DM and DM-NI communication follows the defined protocol
Long NoC latency emulation: delay the input signal to the DM for a longer time than the DM *wait threshold* time	*time-out* function correctly triggered, *error report* generated indicating a time-out error and DM is soft-reset
Compare the DM-TPG scan test-vectors with the reference from the original test pattern file	Generated test-vectors match the the reference

6.1.3 DM verification with a tailored MPSoC framework on FPGA

6.1.3.1 Experimental framework

In order to thoroughly verify the proposed dependability approach, the DM needs to be integrated into an MPSoC environment to examine some key features, such as the interaction between the DM and Xentium tile processors and the feasibility of online scan-based test using the multicast mechanism in the presence of random NoC latency. The block diagram of a tailored MPSoC framework which can be used for this purpose is shown in Figure 6.6. In this tailored MPSoC framework, the DM IP has been extended with a ring topology NoC. Four routers are arranged at the four corners with one Xentium tile connected to each router. As the complete Xentium tile processor is too large to be integrated into the FPGA device, a much smaller "pseudo Xentium" design has been used to emulate the behaviour of real Xentium tile processors in a dependability test. The pseudo Xentium has the same I/O interface and scan-chains as a complete Xentium tile processor. A test stimuli generator (TSG) has been designed and attached to the

system to emulate the behaviour of a general purpose processor. It can send the control signals to the DM and collect DM status and test report data via the NoC.

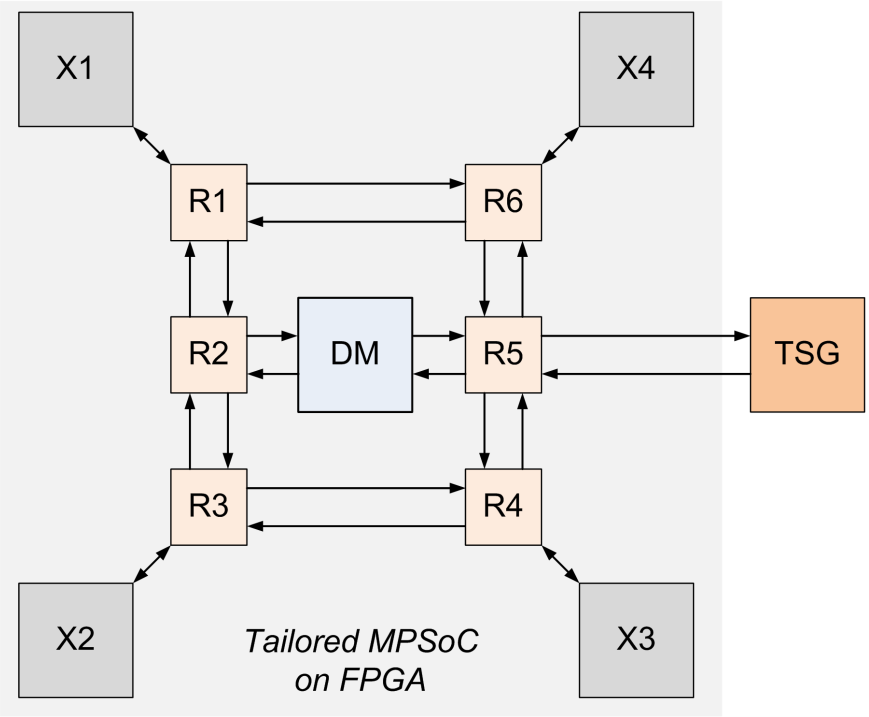


Figure 6.6: Overview of the tailored MPSoC framework for system-level DM verification (R = Router, X = Xentium tile, TSG = Test Stimuli Generator)

It should be noted that the wrapper structure and network interface for the DM and pseudo Xentium tiles are not drawn in Figure 6.6. Figure 6.7 shows the diagram of their actual implementation. The network interface acts as a bridge between the two IPs and the NoC. It can receive control signals from the TSG to configure a proper communication route for its attached block. It is also responsible for setting-up multicast routes in the NoC and collect the responses. In case of dependability tests, the network interface of the DM (DM-NI) always acts as a master and the network interface of the Xentiums tile processors (X-NI) are configured as slaves. This organization gives the DM full control over the dependability test process.

Details of the Xentium tile wrapper (XTW) have been treated in Chapter

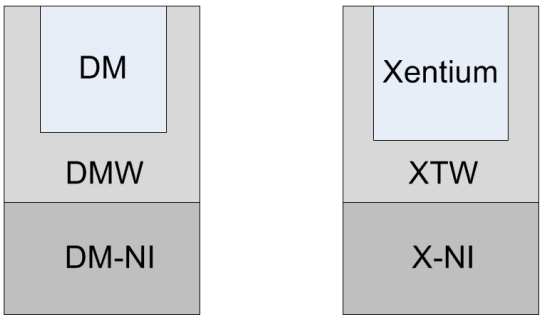


Figure 6.7: DM (left) and pseudo Xentium tile (right) with wrapper (W) and network interface (NI)

5. The XTW can be directly configured by the DM via the NoC into normal function mode or dependability test mode. In the dependability test mode, the XTW can load the input test-vector data into the scan-chains of Xentium tiles or start the memory BIST engine, depending on whether a scan-based test or a memory test has to be performed. Dependability test results are also gathered by the XTW into its status register which can be accessed by the DM. The DM wrapper (DMW) has been designed to support direct debug of the DM IP via a limited number of external input pins. As described in Chapter 5, this wrapper is bypassed if the DM is normally functioning by configuring the wrapper-mode selection pins.

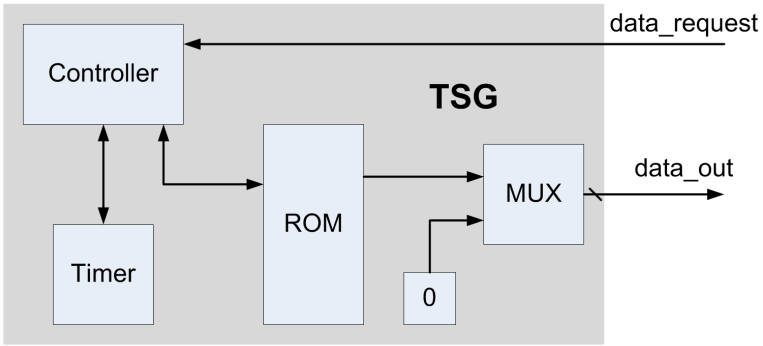


Figure 6.8: Block diagram of the Test Stimuli Generator (TSG) for FPGA-based DM verification. The TSG output is multiplexed between the ROM output and a dummy output(all zeros)

A test stimuli generator (TSG) is required to generate all the test stimuli to trigger each use case of the dependability test. Due to the complex and frequent

communication between the TSG and the system, it is no longer feasible to generate these stimuli manually with a PC client program as used in section 6.1.2.1. As a consequence a hardware TSG has been designed using synthesizable VHDL language and integrated into the FPGA-based MPSoC framework. The TSG interfaces with the system under test via the NoC. A block diagram of the internal components of the TSG has been shown in Figure 6.8.

The TSG mainly consists of a Read-Only Memory (ROM), a controller and a timer. Test stimuli and important parameters such as the number of stimuli looping times and delay cycles have been stored in the ROM. The controller coordinates the activity of the TSG, interprets the stimuli parameters and communicates with the main system via the NoC. The timer is used to insert delays between test stimuli to emulate the delay effects present in a real NoC.

As stated, emulated Xentium tiles have been used instead of the real Xentium design because of resource limitations on the FPGA device. These emulated Xentium tiles have been designed with the same I/O interface of a real Xentium tile to meet the specifications of the XTW and X-NI. Their main function is to ensure the DM can perform normal read and write operations as with a normal Xentium tile processor. Figure 6.9 shows an example finite-state machine inside the pseudo Xentium tile in the case it is used for a scan-based dependability test. The state machine is designed to receive scan test-vectors from the DM, to return the correct test-responses after the test and repeat this process for a proper number of times.

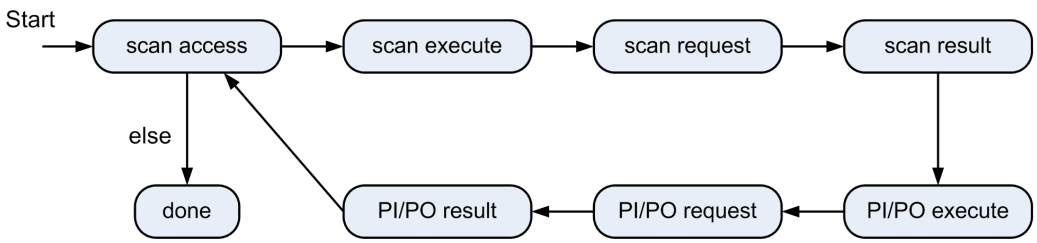


Figure 6.9: The finite-state machine of the emulated Xentium tile (PI = primary input, PO = primary output). The start signal is issued by the GPD when it sets up the virtual channels for the test patterns.

6.1.3.2 FPGA implementation and experimental results

The DM as well as the tailored MPSoC framework have been synthesized and implemented on the Xilinx Virtex-4 FPGA device. Table 6.5 shows a summary of the resource usage of each sub-component. Although the pseudo Xentium tiles have been designed to only retain the minimum necessary functions, the complete system still occupies 95% of the total slices in the FPGA device. As listed in Table 6.5, most of the FPGA resources have been used by the NoC and pseudo Xentium tiles. The maximum clock frequency of the complete system turned out to be dropped to around 50MHz from 200MHz (in the case only the DM-IP is synthesized and implemented). Extra delay of the datapath has been introduced because the emulated Xentium tiles were provided as a stand-alone IP by partners which was not fully optimized given the limited time.

Table 6.5: Resources used by the main components. The percentage equals FPGA usage (with emulated Xentiums)

Module	Slices		Registers		LUT	
	Amount	Percent	Amount	Percent	Amount	Percent
Dependability Manager	17233	19.2	1899	1.1	26968	15.1
DM Network Interface	1494	1.7	1057	0.6	1866	1.1
Network on Chip	33706	37.8	12820	7.2	49109	27.6
Pseudo Xentiums	31916	36.0	22300	12.5	55700	31.3
Test Stimuli Generator	285	0.3	157	0.1	459	0.3
Total	84634	95.0	38304	21.5	135212	75.9

The complete dependability test process has been verified step by step in the experiments. At this stage, no application runs on the platform, which means the complete NoC bandwidth is available for the dependability test.

At the beginning of the experiment, the TSG sets up the communication route from the DM to the Xentium tile processors. The communication between the DM and XTWs have been observed via the ChipScope software. ChipScope results show that the DM can configure the XTW into the proper operational modes, activate the Xentium memory BIST and confirms that the XTW has correctly updated its status register.

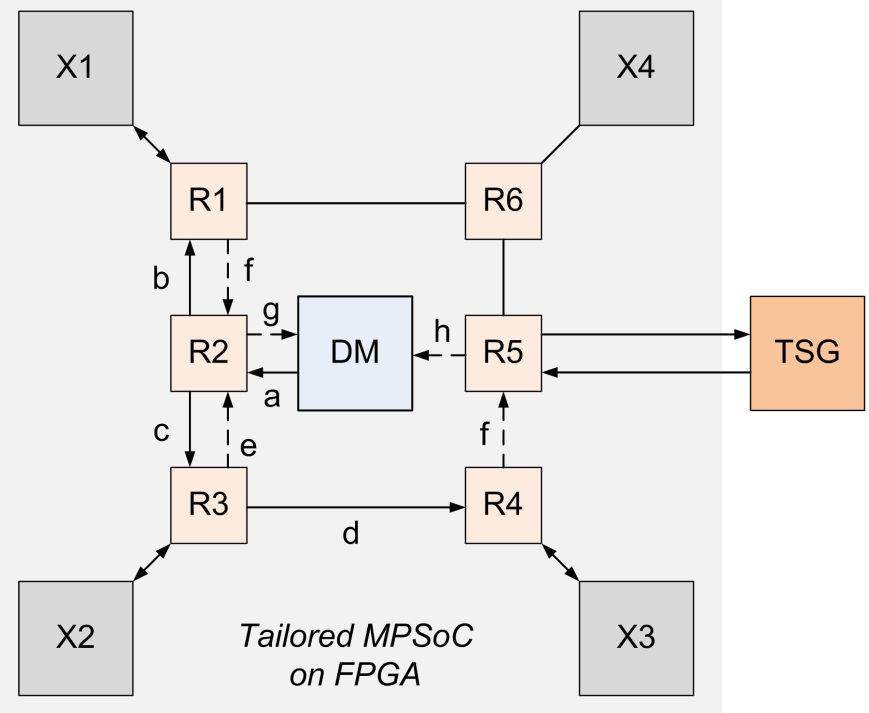


Figure 6.10: A scan-based dependability test process (R = Router, X = Xentium tile, solid arrow = test-vectors, dashed arrow = test-responses)

The multicast function of the NoC has also been verified. As shown in Figure 6.10, the DM sends the scan test-vectors to Xentium X1 along path "`a-b`", to Xentium X2 along path "`a-c`" and to Xentium X3 along path "`a-c-d`". Note that the test-vector data has been duplicated and split into two identical streams at router R2 and R3. In these tests, Xentium X4 was not used.

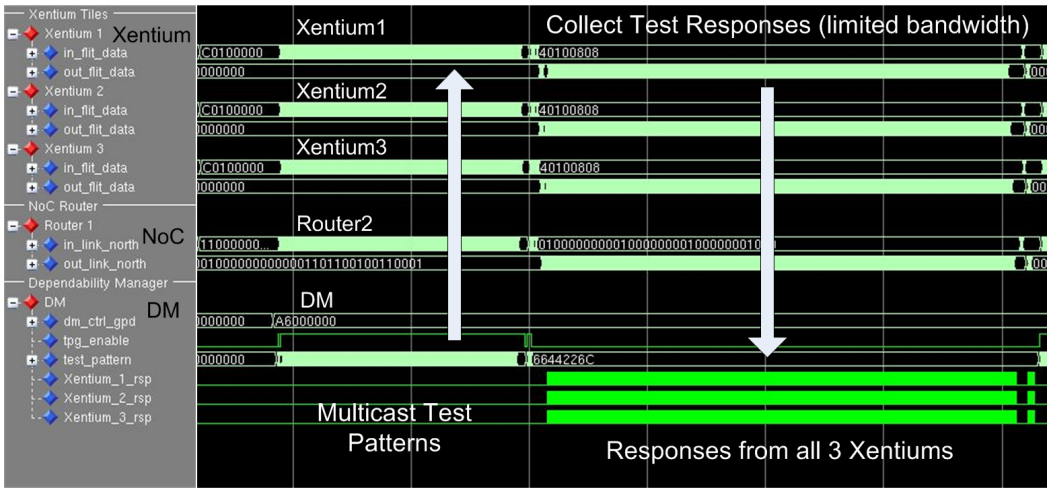


Figure 6.11: Post-synthesis simulation results of the complete scan-based dependability test

After one complete test-vector has been loaded into all Xentium tiles, test-responses are generated and collected by the DM as shown by the dashed arrows in Figure 6.10. Since the test-response data from X1 and X2 share the same router R2 to enter the DM, they each get half of the full bandwidth of NoC path `g`. While X3 can use the full bandwidth of path `h`, it needs to periodically pause and resume the test-response data to match the pace of the other two Xentiums (X1 and X2). An FPGA-based post-synthesis simulation of the complete scan-based test scenario is shown in Figure 6.11 before the design is downloaded and implemented into the FPGA.

The rising arrow at the left side indicates the test-vectors from the DM have been multicasted to the three Xentiums under test via the NoC. The falling arrow at the right side shows the test-responses have been collected by the DM into its local registers and compared with one another. A detailed examination of the test-responses is shown in Figure 6.12, which clearly shows the test-response from

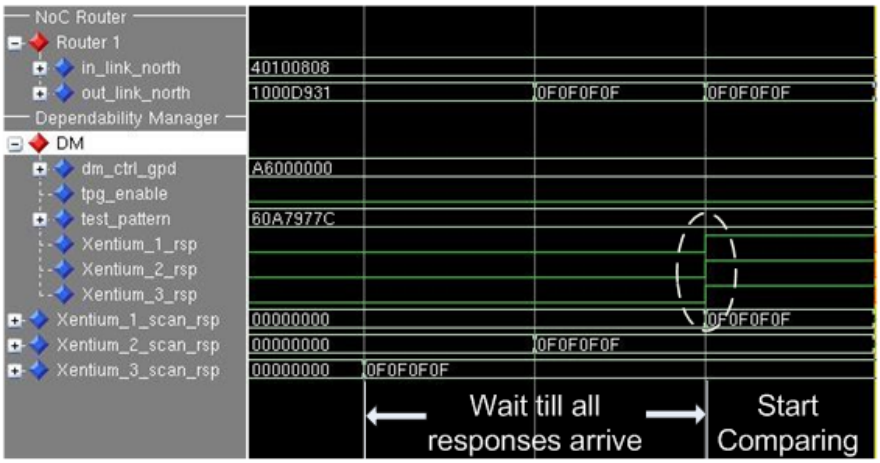


Figure 6.12: Post-synthesis simulation results of the handling of the test-responses from multiple Xentiums

Xentium 3 gets to the DM-TRE earlier than the other two Xentiums; however the evaluation process did not start until all test-responses have arrived.

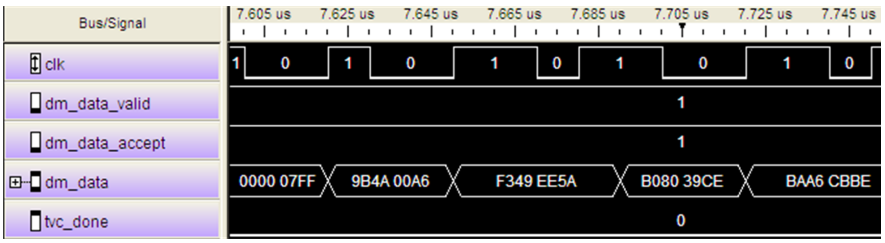
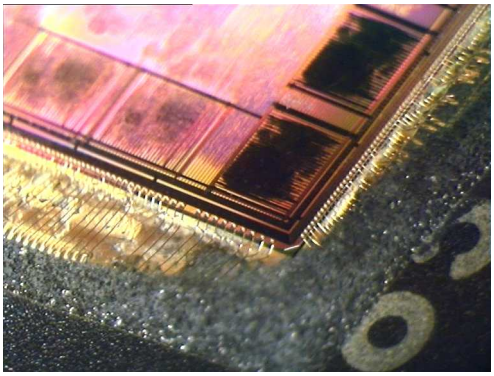


Figure 6.13: Measurement result of the test-vectors generated by the DM

To further verify the NoC multicast function, test-vectors entering each Xentium tile processor have been redirected to the external I/O pins of the FPGA board. A measurement is subsequently performed using an Agilent logic analyzer on these I/O signals. Measurement results show that identical test-vectors have entered each Xentium, which confirms the correct behaviour of the multicast function. Figure 6.13 shows an example measurement result of the test-vectors. The main clock frequency has been set to 33.3MHz to obtain a stable output due to speed limitations of the I/O pins on the FPGA board. The measured test-vector values (shown in signal `dm_data`) correspond to the ones obtained in previous simulations.

(a) RFD chip in the BGA package



(b) Zoomed-in view of part of the RFD chip

Figure 6.14: The reconfigurable fabric device implemented in UMC 90nm CMOS technology

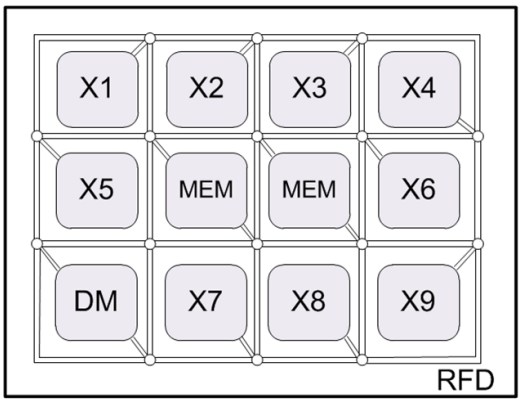
6.2 ASIC Realization of a Dependable Homogeneous MPSoC

6.2.1 DM in the Reconfigurable Fabric Device (RFD)

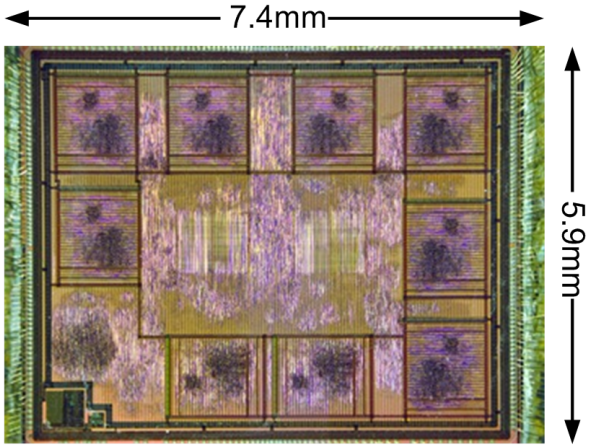
In the framework of the CRISP project, we aim to provide a highly scalable, dependable and reconfigurable platform for a wide range of streaming data applications. The dependability feature of this platform is realized by integrating the fully verified DM hardware into the Reconfigurable Fabric Device (RFD) as shown in Figure 6.14. Major functional blocks in each RFD chip include:

- 9 Xentium processing tiles
- 2 Smart Memory Tiles (SMT)
- Dependability Manager tile (DM) as an Infrastructural IP (IIP)
- 6 Multi-Channel Ports (MCP) + Die Link Interfaces (DLI)
- 2D-mesh virtual channel Network-on-Chip (NoC)

The block diagram of one RFD is shown in Figure 6.15(a). As all the nine Xentium tile processors in the RFD have identical structures, the RFD can be



(a) RFD block diagram, X = Xentium tile processor, MEM = embedded SRAM, DM = Dependability Manager



(b) Microphotograph of the RFD silicon

Figure 6.15: The reconfigurable fabric device (RFD)

regarded as a good example of a homogeneous MPSoC. As mentioned in Section 4.3, a 32-bit virtual channel 2D mesh network-on-chip (NoC) has been implemented as the main communication method for Xentium processing tiles, smart memory tiles and the DM. In addition, the NoC is also reused as a test access mechanism (TAM) for the dependability test.

A microphotograph of the RFD silicon is shown in Figure 6.15(b). The RFD chip has been fabricated using standard-cell UMC 90nm CMOS technology. The dimensions of the chip are about 7.4mm by 5.9mm. The RFD is a complete SoC with clock and reset management hardware and has passed a structural

manufacturing test by Atmel Automotive GmbH. As listed in Table 6.6, the area of the RFD chip is about 43.8mm^2. The area of a single Xentium hard macro in the RFD is 1.88mm^2. The RFD main clock frequency is designed to be 200MHz.

Table 6.6: Silicon area of the RFD, the Xentium tile and the DM (UMC 90nm CMOS technology)

Name	**Area(mm^2)**
RFD	43.8
Xentium hardmacro	1.88
DM	0.4

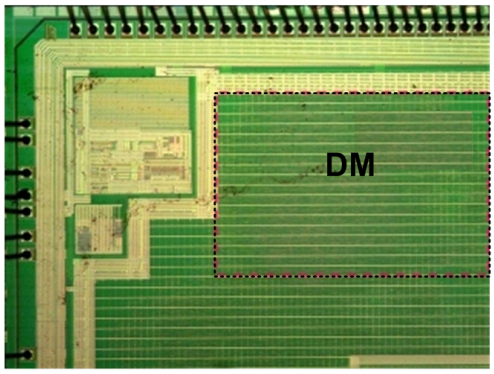


Figure 6.16: Microphotograph of the DM IP in the RFD

A microphotograph of the DM IP in the RFD chip is shown in Figure 6.16. As can be observed, the DM has been implemented as glue logic instead of a hard macro in the RFD. It consists of around 41,000 logic gates and occupies about 0.43mm^2 area. The silicon area of the DM is about 23% of one Xentium hard macro and less than 1% of the entire RFD. The maximum clock frequency of the DM is 500MHz, according to post-synthesis simulations (using Synopsis Design Compiler). On the left upper corner of the figure, the phase-locked loop (PLL) clock circuit is shown.

6.2.2 The General Stream Processor platform

Originally the CRISP platform was envisioned to be a single chip, the General Stream Processor (GSP). For the sake of reduced risk and cost, the GSP prototype has currently been implemented on a Printing Circuit Board (PCB). A simplified block diagram of the GSP is shown in Figure 6.17(a) and a photograph of the manufactured GSP on a PCB is shown in Figure 6.17(b). Two major building blocks of the GSP platform include a General Purpose Device (GPD) and five Reconfigurable Fabric Devices (RFD), individually assembled in a 400 Ball Grid Array (BGA) package and interconnected on the PCB. The reconfigurable many-core processor architecture of the GSP is implemented in its five RFDs.

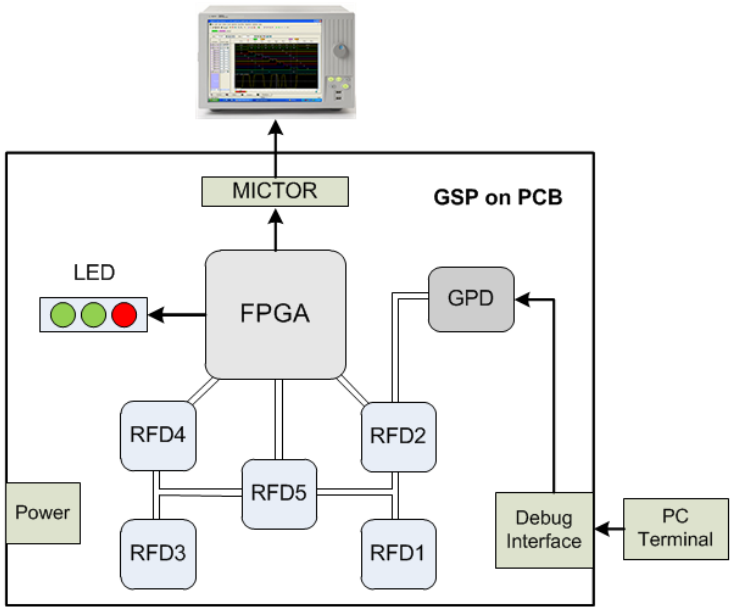
The GPD consists of an ARM926 general purpose processor, SRAM and various peripheral devices. The main clock frequency of the GPD is 200MHz. An embedded Linux operating system is adopted to run system applications such as the run-time mapping software for run-time resource management and the dependability software for coordinating the dependability test [Burg 11].

High-speed chip-to-chip connections (C2C) and Multi-Channel Ports (MCP) form the backbone of the inter-chip communication in the GSP. A Xilinx Virtex-4 FPGA device has been incorporated on the PCB for user-defined applications and system debugging. The data stream in the NoC of each RFD can be looped-back via the MCP in the FPGA device. This data can subsequently be accessed from the on-board MICTOR connector by external devices, for example a logic analyzer. Figure 6.18 shows the setup to test the DM in the RFD chip in the GSP board. The MICTOR connector has in total 34 output pins, of which one pin is used for the measurement of the MCP port clock signal `mcp_clock`, one pin for the NoC write enable signal `noc_write` and the remaining 32 pins for the 32-bit data package `noc_flit_data[0:31]` transported by the NoC.

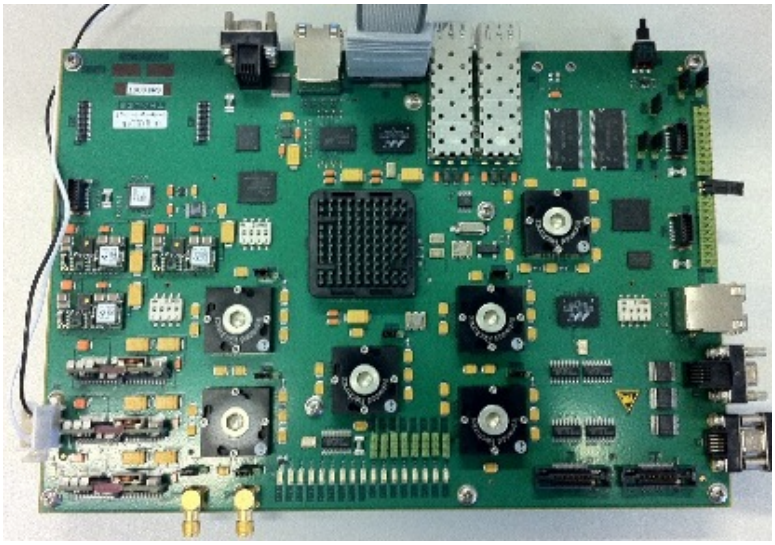
6.3 Measurement results of the GSP platform

6.3.1 The dependability software

The DM hardware has been designed to perform dependability tests such as scan-based structural test on the Xentium tile logic part or memory BIST on the



(a) The GSP block diagram



(b) Photo of the GSP on a PCB

Figure 6.17: The GSP platform

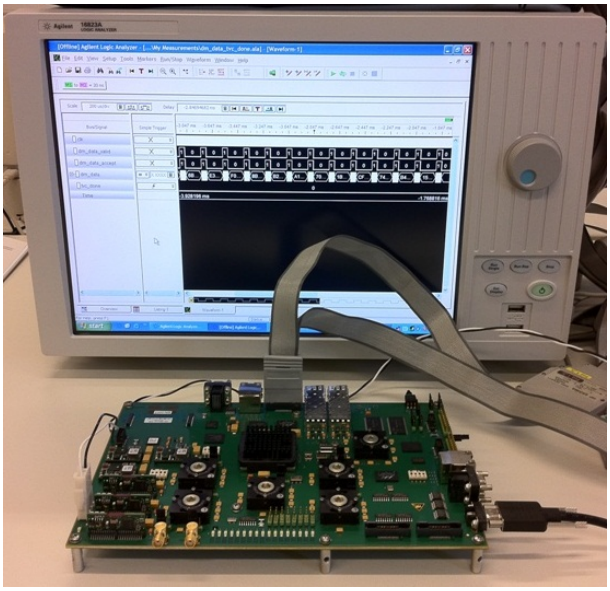


Figure 6.18: Setup for the GSP platform experiments

Xentium tile memory in the RFD. Moreover, dependability software has been developed to control the process of the dependability tests. Figure 6.19 shows the relation between the dependability software and other software and functional blocks involved in the dependability test.

All the software programs shown in Figure 6.19 have been developed using the C programming language. They are hosted by the embedded Linux OS running on the ARM926 processor in the GPD. The resource management software treats the dependability test as a special application and allocates resources such as Xentium tiles and NoC virtual channels for it. The dependability software will configure the NoC route from the DM to the Xentiums under test and command the DM to start dependability tests by writing proper commands into the configuration register of the DM via the NoC. The DM will perform the tests as requested, generate a test report and store it in the DM status register. If a faulty Xentium tile is detected by the DM, the dependability software will notify the resource management software to isolate the faulty tile. It will also request the run-time mapping software to remap the user application to fault-free resources [Ter 11]. The cooperation of these three software programs ensures that the RFD can still be considered as a fault-free platform in the presence of faulty Xentium tiles. The

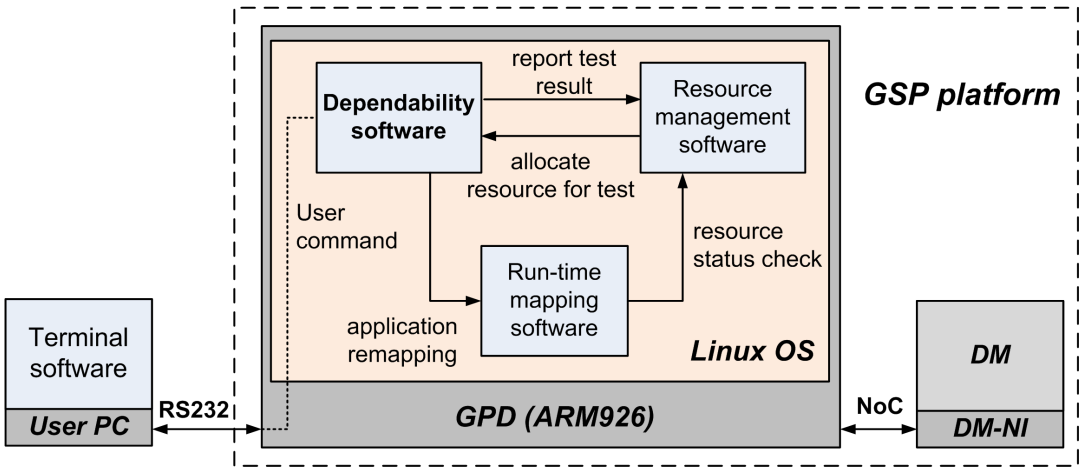


Figure 6.19: Dependability software interaction with other parts

dependability test can be invoked manually or performed periodically depending on the actual application requirements. One can interact with the dependability software using standard terminal software on a client machine via the serial port of the GSP platform (shown at the bottom left corner of Figure 6.19).

6.3.2 Test of the DM operations in the RFD

At the beginning of a dependability test, the dependability software sets up a communication channel (a NoC virtual channel) between the GPD and the DM-NI and configures the multicast route between the DM-NI and the Xentium tiles under test. Subsequently, the dependability software writes a "start the test" command to the DM configuration register, requesting the DM to perform proper dependability tests. Figure 6.20 shows the measurement result of this process. In this measurement, command `9600 0000` was written into the DM configuration register via the NoC (signal `noc_flit_data`). The command was requesting the DM to start up the embedded memory BIST on selected Xentium tiles. As the maximum speed of the MCP in the FPGA device is only 100MHz, the clock period is thus 10ns in this measurement.

Similar measurement procedures have been carried out to capture the structural test-vectors generated by the DM. The DM has been designed to generate 413 groups of test-vectors in one scan-based dependability test as discussed in

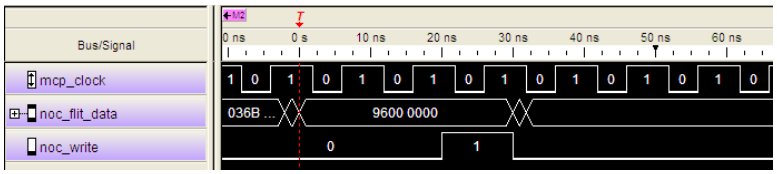
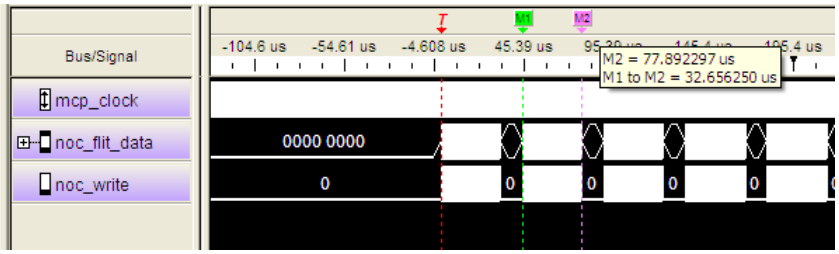
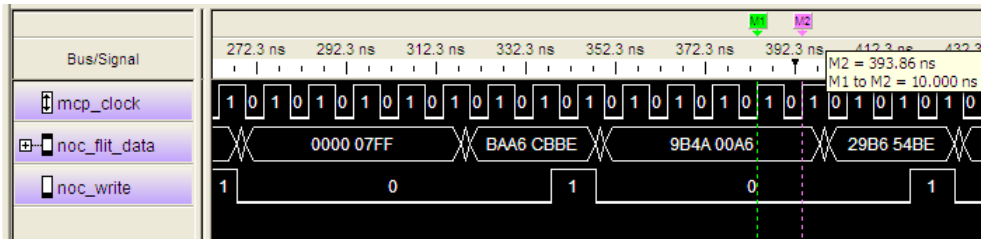


Figure 6.20: Measurement result of the GPD command



(a) Five DM test-vectors (zoomed out)



(b) Four 32-bit DM test-vector words (zoomed in)

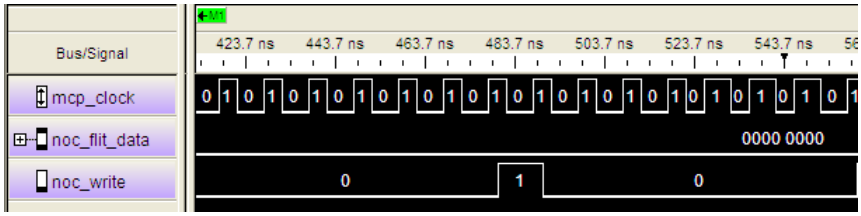
Figure 6.21: Measurement results of the test-vectors generated by the DM

the previous chapter. Each test-vector can be divided into 404 32-bit words. In Figure 6.21(a), a zoom-out view of the test-vectors is shown in the signal `noc_flit_data`. For example, the data block between M1 and M2 is one complete test-vector which took about 32.7μs to generate with a 100MHz clock signal. As a complete structural dependability test consists of 413 such test-vectors, the actual time used to generate all the test-vectors is $413 \times 32.7\mu s \approx 13.5ms$ at a 100MHz clock frequency.

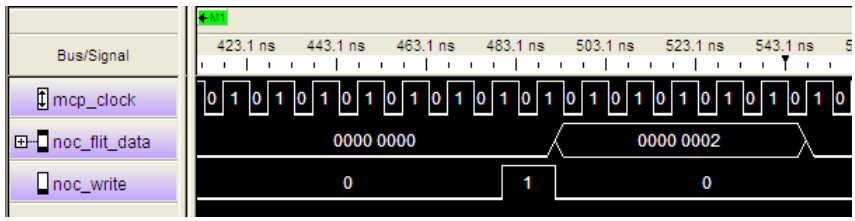
The test-vector words have been packed into 32-bit data flits by the DM-NI and are multicasted to the Xentium tile processors under test via the NoC. In Figure 6.21(b), test-vector data flits such as `0000 07FF` and `9B4A 00A6` are passing a NoC router as is shown in signal `noc_flit_data`. It takes about 80ns to transport one test-vector word and one NoC control flit. Thus it takes approximately 32μs to generate all the 404 combinations, which matches the time of one complete test-vector (32.7μs) shown in Figure 6.21(a). Due to the internal memory limit of the logic analyzer, the test-vector words cannot be all captured in a single run. Thus they have been captured in separate runs and the results have been stored in local data files. All the recorded test-vector values have been compared with the reference test-vectors in the original ATPG test pattern file and it can be concluded that the correct test-vectors have been generated by the DM-TPG hardware.

6.3.3 Test of the XTW fault emulation function

After the test-vectors have been applied to the Xentium tile processors, test-responses are subsequently collected by the DM. Then the DM will examine the test results and generate a test report accordingly. For the Xentium tiles we have experimented with, no faults have been identified. This experimental result is reasonable if all the Xentium tiles in the RFD are fault-free. However, one can hardly test all the DM functions if there is no faulty Xentium tile sample in the RFD. One of the possible solutions is to create real structural stuck-at faults in some Xentium tile within the RFD chip. This is achievable by e.g. laser fault injection, which is quite difficult given our experimental facilities. A simpler method is to emulate the behaviour of a faulty Xentium tile, which means to



(a) Test response of a fault-free Xentium tile



(b) Test response of a Xentium tile with fault emulation

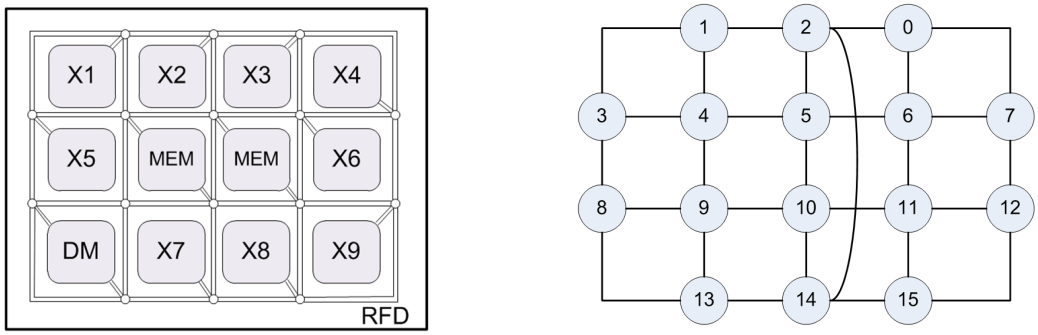
Figure 6.22: Measurement results of the fault injection experiment

modify the structural test result of a certain Xentium.

As a result, a faulty response emulation block has been developed and included in the Xentium tile wrapper during the design phase. This block can be activated by the dependability software in the GPD, which communicates with the XTW via the NoC. If the fault emulation block in a certain XTW is enabled, it can flip some bits of the produced test-responses, thus making the target Xentium generate a different test result from the fault-free ones. Experiments have been performed to validate this function and the measurement results are shown in Figure 6.22. Figure 6.22(a) shows that the test-response of a fault-free Xentium tile is `0000 0000` (in signal `noc_flit_data`). Figure 6.22(b) shows that, with fault emulation activated, the test-response has become `0000 0002`. This proves that the fault emulation hardware in the XTW has been correctly implemented.

6.3.4 Full dependability test without applications

In this part, the features mentioned in previous experiments are combined to perform a full dependability test within the RFD. Three random Xentium tile processors are allocated for the test by the resource management software. A NoC multicast route has been automatically calculated by the dependability software



(a) RFD block diagram

(b) Numbering of the NoC routers in the RFD

Figure 6.23: Block diagram of the RFD internal blocks

for broadcasting test-vectors from the DM to target Xentium tiles.

To help understanding the dependability test report, Figure 6.23(a) shows the major functional blocks in the RFD which have been connected to the NoC. Small circles in the figure represent the NoC routers. Figure 6.23(b) shows the numbering of the 16 routers in the RFD. The full dependability test result is compiled and stored in the DM status register and will be collected by the dependability software via the NoC. Important test result information is decoded and directed to the user terminal in an understandable format.

Figure 6.24(a) shows one of these test reports. The NoC route allocated for the dependability test is shown in the upper part of the test result. For example, the text `DM - R13 - R09 - R04` means the test-vector data depart from DM and go through router 13, 9 and 4. These vectors enter Xentium 4 (note that the Xentium numbering starts from "0" in the report and hence Xentium 4 denotes "X5" in Figure 6.23) via router 3, and the subsequent test-responses return to the DM via router 3, 8 and 13. The first dependability test which is carried out is a memory BIST on the cache memory blocks of the three chosen Xentiums. The test result loaded into the DM status register is a hexadecimal number `7000 03E8` which is interpreted in our scheme as "no error in the communication, and no fault has been identified". Some of the important fields of the test result have been decoded and shown in the report. For example, "is error" and "error type" fields indicate whether there is an operational error (not Xentium fault) during the test

```
Test scenario 2:
DM - R13 - R09 - R04
                   |- R03 - XE4 :: R03 - R08 - R13 - DM
                   |- R01
                         |- XE0 :: R01 - R02 - R14 - R13 - DM
                         |- R02 - XE1 :: R02 - R14 - R13 - DM

DM BIST test returned: 0x700003e8
---------------------------------
   is error: 0
 error type: No error
 core fault: 0
  mem fault: 0
faulty DUTs: 000

DM SCAN test returned: 0x700003e8
---------------------------------
   is error: 0
 error type: No error
 core fault: 0
  mem fault: 0
faulty DUTs: 000

DM FULL test returned: 0x700003e8
---------------------------------
   is error: 0
 error type: No error
 core fault: 0
  mem fault: 0
faulty DUTs: 000
#
```

(a) Test report: no error in communication (error type: No error) and all Xentium tiles tested are determined to be fault-free (core fault: 0)

```
Test scenario 2:
DM - R13 - R09 - R04
                   |- R03 - XE4 :: R03 - R08 - R13 - DM
                   |- R01
                         |- XE0 :: R01 - R02 - R14 - R13 - DM
                         |- R02 - XE1 :: R02 - R14 - R13 - DM

DM BIST test returned: 0x700003e8
---------------------------------
   is error: 0
 error type: No error
 core fault: 0
  mem fault: 0
faulty DUTs: 000

DM SCAN test returned: 0x703023e8
---------------------------------
   is error: 0
 error type: No error
 core fault: 1
  mem fault: 0
faulty DUTs: 100

DM FULL test returned: 0x703023e8
---------------------------------
   is error: 0
 error type: No error
 core fault: 1
  mem fault: 0
faulty DUTs: 100
#
```

(b) Test report: no error in communication but one faulty Xentium tile is detected (core fault: 1)

Figure 6.24: Full dependability test report as printed to the user terminal

process. Field "core fault" and "mem fault" indicate whether the dependability test has detected any faulty part, and the faulty part ID ("faulty DUTs").

The second dependability test in Figure 6.24(a) is a scan-based structural test which tests the logic part of the three Xentiums tile processors. The same test result `7000 03E8` is reported which means no faults in the three Xentiums are found. The last test is an overall test which combines Xentium memory BIST and the scan-based test; the test result conform to the previous two tests.

Next the same dependability test is repeated with the fault emulation function activated in Xentium 4. The test results are shown in Figure 6.24(b). The Xentium memory BIST result is not affected by the fault insertion. But the scan-based test and the overall tile dependability test have both reported a *core fault* has been found with the faulty unit identified (`100` denotes Xentium 4). The result of this experiment proves that the DM has been correctly implemented in the RFD chip and that our dependability approach works properly. Moreover, the fault emulation block in the XTW has also been correctly realized.

In addition, the dependability test has been successfully performed on Xentium tiles from multiple RFDs. For example, the DM in RFD 1 can test one Xentium tile in RFD 1 and two other Xentium tiles from RFD 2 at the same time. The test data are communicated between the two different RFDs by their MCP. This method adds to the flexibility of the dependability test although the speed of the test will decrease because the MCP communication rate is lower than that of the NoC in one RFD. It also implies that the absolute minimum number of Xentium tiles reserved for dependability test is one in each RFD.

6.3.5 Full dependability test at application run-time

One important advantage to use the NoC as a test access mechanism (TAM) is that the dependability tests can be carried out at application run-time. This means the test-vector or test-response data can share the NoC bandwidth with normal application data. This is achieved by assigning time-multiplexed virtual channels of the same NoC fabric to both the dependability tests as well as the application data. Virtual channels assigned to different applications carry different weight, indicating the priority of their mission. In case there is a conflict on the

usage of the same NoC resource between the dependability test and higher priority applications, the dependability test data stream can be paused and resumed to ensure its continuity and correctness as explained in Section 4.4.

A simple application referred to as the "mailbox application" (a token message is delivered to a processor like a mail to a mailbox) has been developed to work jointly with the dependability test. The basic principle of the application is demonstrated in Figure 6.25. At the moment the application starts up, software in the GPD sends a mail message to one RFD and this mail will loop through several Xentium tiles via the NoC. Each Xentium tile processor is associated with an LED on the GSP board, which lights up to indicate the receipt of the mail message. If one of the Xentium tiles becomes faulty or has a hardware fault emulated, the whole application will crash and the dependability software will start the DM to test the RFD and locate the faulty Xentium tile. The user can easily observe the status of the application by monitoring the LEDs on the board.

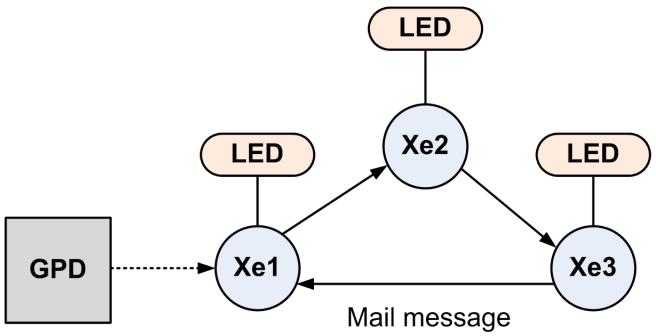


Figure 6.25: The mailbox application using three Xentium tile processors

The following experiments have been performed with the mailbox application:

- Within one RFD, run the mailbox application and dependability test on separate groups of Xentium tile processors.
- Within one RFD, emulate a fault in one of the Xentium tiles used for the mailbox application.

In the first experiment, the mailbox application can run correctly and remains stable, without being disturbed by the dependability test. This means the dependability test has no influence on the Xentium tiles and NoC parts used by

the application. This is because the "mail message" passed around the three Xentium tiles occupies little NoC bandwidth, which causes no conflicts with the dependability test.

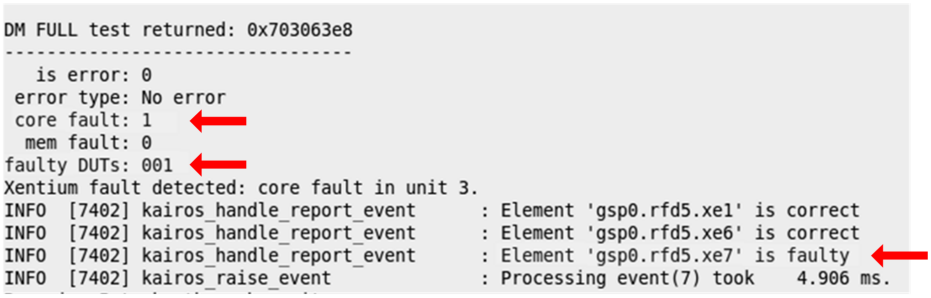


Figure 6.26: Experimental result of the mailbox application: faulty tile detected.

In the second experiment, the dependability software should invoke the DM to find out the fault and notify the run-time mapping software to remap the application on fault-free resources. In the actual experiment, Xentium number 1, 6 and 7 in RFD 5 have been adopted for the mailbox application. Meanwhile, the dependability software orders the DM to perform periodic dependability tests. At a certain moment, the user triggers the fault emulation in Xentium 7. Subsequently the mailbox application breaks down as expected. The DM immediately identified the faulty Xentium tile; its test result is summarized and shown in the screen shot of the terminal in Figure 6.26. It reports that one faulty Xentium tile, Xentium 7 (`gsp0.rfd5.xe7`), has been detected and the total time used for the test is about 4.9ms. The dependability software running in the GPD acquires this test result from the DM and notifies the resource management software and the run-time mapping software subsequently to remap existing applications to fault-free Xentium tiles. The time required for application remapping is about 90ms and the complete system unavailability time is thus around 95ms (time used for dependability test and remapping combined). Note that the dependability test time can vary depending on the sequence in the complete test-vector set of the test-vector which detects the fault.

These two previous experiments prove that the dependability test and normal applications can co-exist in one RFD. The application will not be interrupted by the dependability test. In the case an application crashes as a result of a

faulty Xentium tile, our dependability approach successfully manages to replace the faulty tile and resumes the application on fault-free resources without the intervention from the user.

6.4 Dependability test power evaluation

6.4.1 CMOS circuit power dissipation

The power dissipation of an integrated circuit can be classified into two main categories, being the dynamic power and static power dissipation. Complementary metal-oxide-semiconductor (CMOS) is an integrated circuit fabrication technology which combines the p-channel and n-channel metal-oxide-semiconductor field-effect transistors (MOSFET). One of the important advantages of the CMOS logic is its very low static power consumption compared to NMOS or PMOS logic.

Figure 6.27 shows the scheme of a simple CMOS inverter. The switching current (`I_switch`) which charges and discharges the load capacitor contributes to the dynamic power consumption. It has been known that the dynamic power dissipation is mainly determined by the supply voltage, the load and the frequency of the switching activities. Equation 6.1 shows a typical formula which can be used to calculate the dynamic power consumption P_d of CMOS logic circuit. F is the operating frequency of the circuit. V_{dd} is the supply voltage and $\sum C_{load}$ is the accumulated capacitive load.

$$P_d = F \times V_{dd}^2 \times \sum C_{load} \tag{6.1}$$

As shown in Figure 6.27, leakage power dissipation is caused by the leakage current (`I_leakage`). The leakage power is static power dissipation which exists regardless of the switching activity of the circuit. Equation 6.2 shows a formula which can be used to calculate the static power dissipation P_s. The total power dissipation of a circuit is the sum of P_d and P_s.

$$P_s = \sum I_{leakage} \times V_{dd} \tag{6.2}$$

Given the processing technology, one can estimate the power dissipation of

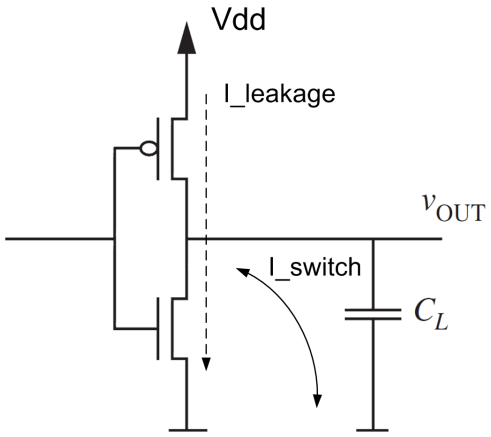


Figure 6.27: CMOS inverter current flow. $I\_leakage$ and I_switch represent the leakage current and switch current respectively. C_L represents the load capacitance.

a target circuit. For the ease of calculation, Equations 6.3 and 6.4 can be used [Taiw 06].

$$P_d = P_{comb} \times F \times TR \times N_{gate} \tag{6.3}$$

P_{comb} is the power dissipation per MHz per logic gate count (nW/MHz/gate), F is the operating frequency of the circuit (MHz), TR is the toggle rate and N_{gate} is the number of logic gates. The toggle rate reflects how often an output changes relative to a given input and can be represented as a percentage between 0 and 100%. If one takes a synchronous circuit as an example, the maximum data toggle rate of 100% means that the output toggles at every active clock edge.

$$P_s = P_{leakage} \times N_{gate} \tag{6.4}$$

$P_{leakage}$ is the average leakage power per gate and N_{gate} is the gate count of the circuit.

The calculation of the power dissipation of an example circuit implemented using the TSMC 90nm CMOS technology is given as follows. A typical logic gate (2-input AND gate) has a P_{comb} of 5.442 nW/MHz/gate and $P_{leakage}$ of 4.111 nW/gate [Taiw 06]. Let the example circuit be the DM which comprises in total 41,000 logic gates. Suppose it operates at 100MHz with a toggle rate of 20%,

then its dynamic power dissipation is estimated to be:

$$P_{dynamic} = 5.442nW/MHz/gate \times 100MHz \times 20\% \times 41000 \approx 4.46mW$$

Its static power dissipation is:

$$P_{static} = 4.111nW/gate \times 41000 \approx 0.17mW$$

Hence the total power dissipation of the DM operating at 100MHz with 20% toggle rate can be estimated to be around 4.63 mW. Note that the actual toggle rate depends on the way in which the DM is used, as explained in the next section.

6.4.2 Estimation of the dependability test power dissipation

Several major power consuming entities can be identified during a dependability test, being the DM, the Xentium tiles under test, the GPD (ARM), the related network interfaces and the NoC [Zhao 13]. In order to be consistent with subsequent measurements, it is assumed that the ARM GPD operates at 200MHz and all other blocks work at 100MHz (the test application ran at 100MHz by the time of the experiment). Only the dynamic power consumption is considered since the static power dissipation is negligible as calculated in the previous part. The power dissipation of each block can be estimated as follows.

It is known that throughout a dependability test, the DM is extremely busy generating the test-vectors (assume 100% toggle rate), but less occupied while evaluating the test-responses (assume 20% toggle rate). Hence, it is assumed that the overall toggle rate of the DM is 60% on average. The DM-NI consumes about 5% of the power of the DM according to its size and activities. The Xentium processing tiles under test are known to be extremely busy flipping their internal scan-chain cells throughout the entire test. Thus a maximum 100% toggle rate is assumed. The same case holds for the Xentium tile wrappers and the Xentium network interfaces. On the other hand, the power consumption of the Xentium cache memory BIST is ignored as it is a burst operation performed within a much shorter time compared to the scan-based tests. The toggle rate and estimated

power dissipation of the aforementioned blocks are summarized in Table 6.7.

Table 6.7: Power estimation of some functional blocks in a dependability test.

Block name	Toggle rate	Power dissipation (mW)
DM	60%	13.39
DM-NI	60%	0.67
3 Xentiums (with XTW)	100%	140.2
3 Xentium-NIs	100%	8.9

The power dissipation of the selected NoC type (packet-switched NoC with virtual channel flow control) has been extensively researched in terms of link and router energy being 150pJ for a 256-bit packet [Wolk 09]. In the dependability test, scan-test-vectors are transported from the DM to three Xentium tiles over the NoC and back. The test-data volume involved is 26 Mbit. Assumed that on average 50% of the routers is active (8 in total), a power dissipation of 0.4mW results for a dependability test which lasts for 300ms (see Section 6.4.3).

It is known that an ARM926EJ-S based processor will dissipate around 70-80mW power at full speed when fully loaded, and about 34mW power when staying at idle state with only some I/O control and timer hardware active [Atme 09]. The involvement of the GPD (ARM) in the dependability test is two fold. First, it needs to start the dependability test. Second, it has to perform a periodic polling (e.g. frequency is 100KHz) on the DM to check whether the dependability test has been completed and to collect the test results. It is obvious that these operations do not dissipate much power in the ARM processor throughout the entire dependability test period. Hence, a close to idle power (35mW) is taken as the average power dissipation of the GPD during the dependability test. Note that the updating of the usable resource table and remapping of applications will only take place in case a fault occurs, thus the power dissipation of this part will not be included in the normal dependability test power estimation.

Besides the ARM itself, there are other PCB peripherals and SRAM chips associated to it. Hence it is difficult to calculate the exact power dissipation of these parts at the printed circuit board level. Thus when comparing the calculated and measured power dissipation, the PCB peripherals and SRAM power measurement results are used in both cases.

In summary, the total power dissipation of the major functional blocks (DM, ARM, NoC and Xentium) during the dependability test is about 198mW by adding the power consumption of each part together. Adding the measured PCB peripherals and memory chips dissipation of 115 mW, this results in an estimation of 313mW total power dissipation for a dependability test.

6.4.3 Power measurement result and discussion

Actual power measurements have been carried out using the GSP board (as shown in Figure 6.17b) with a 5V power supply and an Agilent N6705B power analyzer. Due to interface limitations, the measurement was performed at board level instead of at chip level. To monitor the test data stream, the test data was routed to the FPGA device on the board which caused longer test time than a usual test without monitoring. The system operates at 100MHz and the measurement result is shown in Figure 6.28.

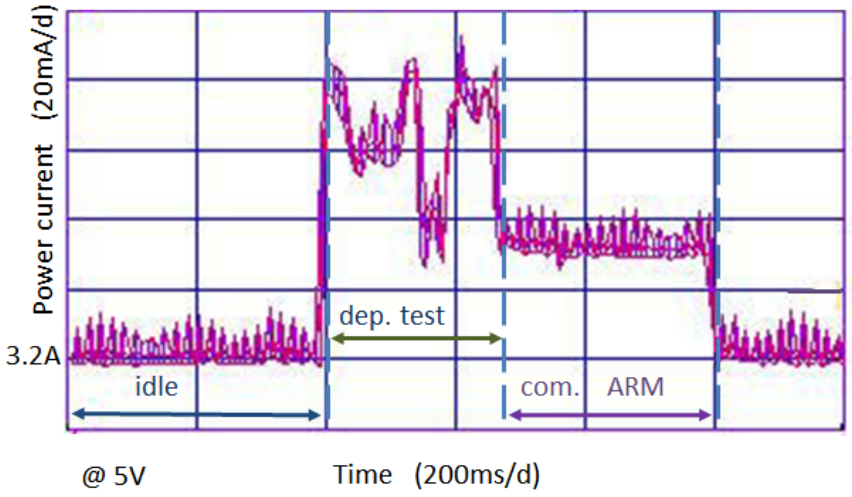


Figure 6.28: GSP board power dissipation measurement result during the dependability test. Horizontal axis 200ms/div, vertical axis 20mA/div.

The first 400ms in Figure 6.28 reflects the power dissipation of the board in the absence of a dependability test. The peaks during the next 600ms are due to the dependability test in a single RFD MPSoC. At the end of the measurement when the dependability test has been completed, the system power dissipation

drops to the original level.

The 600ms current peak can be divide into two parts. The dependability test itself takes place in the first 300ms. The averaged power increase of this part (the higher peak) is about 300mW, which is close to the estimated power dissipation (313mW). After the dependability test has been carried out, the ARM and some other board peripherals and the SRAM chips on the board continue to be active for some 300ms. The lower peak represents their activities and the average power of this part is about 150mW. Since the current peak at 600ms is caused by the dependability test and related activities, it can be concluded that the overall measured power increase related to the dependability test (without repair) is about 225mW (average of 300mW and 150mW).

As the dependability test is not always active during system operation, the actual power dissipation of the dependability test depends on the frequency (number of occurrences) at which the test takes place. For example, if one tenth of the time is used for this test, then in the case of the structural scan-based dependability test, the total dependability power dissipation will then be 22.5mW.

It is obvious that more frequent dependability tests will result in a highly dependable (e.g. low MDT) SoC, as any repair actions can only take place after starting up a dependability session, where subsequently detection and remapping decisions can take place. Hence, the degree of dependability is directly (proportional) related to the added power dissipation: a high dependability results in high added power dissipation.

6.5 Dependability improvement

In this section, the RFD chip and the GSP platform have been used as an example to demonstrate the improvement of system dependability as a result of our dependability approach [Zhan 11]. Important dependability attributes introduced in Chapter 3 such as reliability, availability and maintainability have been evaluated and the system resource cost for the dependability approach have been assessed as well.

6.5.1 Reliability improvement

As discussed earlier, the essential functional part of the RFD consists of nine identical Xentium tile processors; as such it can be considered as a good example of a homogeneous MPSoC. Suppose the RFD has no dependability feature at all, then it will not be possible to detect a structural fault at the tile level. A fault can only be detected at the RFD level based on e.g. an application malfunction. After that, the best one can do is to label the entire RFD chip as *faulty* and exclude it from the GSP platform as there is no maintainability either. In contrast, the RFD chip can still function with some reduced performance if the dependability infrastructures are employed.

In Chapter 3, an MPSoC has been modeled as a K-out-of-N: good system and the formula for system reliability estimation have been introduced. Now assume the RFD is configured as a 7-out-of-9 system, which means a minimum of 7 tiles are needed for the user application. In this case the system reliability distribution with and without the dependability infrastructures is shown in Figure 6.29. It is obvious that if the dependability approach is used, the dependability test fault coverage also plays an important role with regard to the overall system reliability improvement. Note that the FIT value (1,000) here is an exaggerated one in order to highlight the reliability improvement.

As analyzed in Chapter 3, the more processor cores are involved in an MPSoC, the more system reliability improvement can be achieved by our dependability approach. Hence much more system reliability enhancement can be realized if the reliability analysis is performed at the entire GSP platform level (45 cores).

6.5.2 Availability and maintainability improvement

As a result of the dependability infrastructural IP (the DM), the spare Xentium tiles and the run-time mapping software, the RFD becomes a maintainable system. Its availability attribute is characterized by the system mean down time (MDT). For the RFD, its MDT consists of two major parts: the time spent for the dependability test after a fault occurs and the time for reconfiguration and resource remapping using the run-time mapping software. Timer software has been developed and runs in the GPD to record the real time spent for these

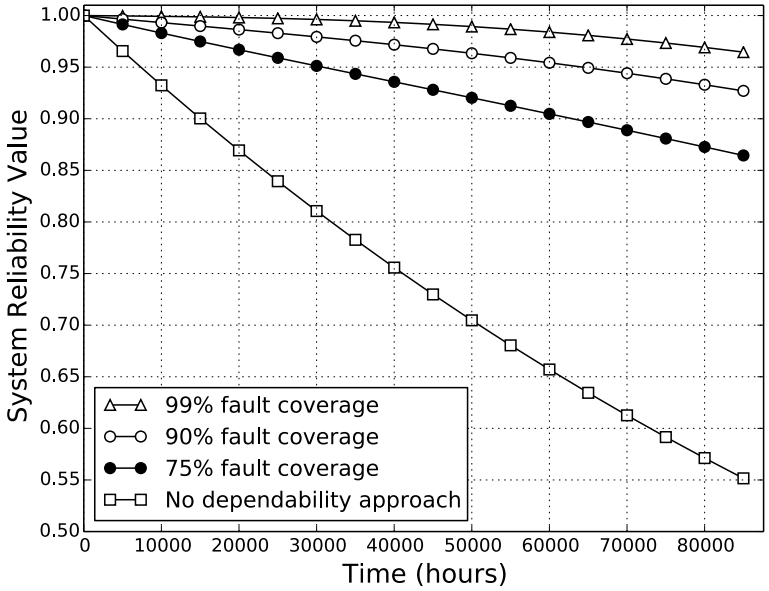


Figure 6.29: Reliability distribution of a nine-tile RFD in 10 years with different dependability test fault coverages; processor FIT=1,000.

operations.

In the case of the mailbox application, the measured time spent for structural test and memory BIST for 3 Xentiums is about 7ms with a 200MHz clock. And a complete dependability test on all nine Xentiums can be carried out in three groups, which takes 21ms in total. The time spent for system reconfiguration and application remapping is measured at roughly 90ms. In addition, a software-based NoC test is performed to test the complete NoC each time the GSP is reset and this test takes about 175ms. Including the NoC software test time, the total MDT of the RFD is about $21 + 90 + 175 = 286ms$. Given a more complex application such as the beamforming application, the time spent for reconfiguration and remapping will be several times longer than the mailbox application (90nm). It can be concluded that the MDT of the RFD is related to the complexity of the application which needs to be remapped. The actual time spent for the dependability test (MTTD) is much shorter than the reconfiguration and remapping time (MTTR). To minimize the system MDT, more efforts should be spent on optimizing tile reconfiguration and application remapping time. This

will not be further addressed in this thesis.

6.5.3 Cost of dependability

The dependability features in the RFD chip come at a cost of extra silicon area. Note that the scan chains for Xentium tiles are basic DfT infrastructure and the NoC is the basic communication fabric so they do not count as dependability cost. Thus the real dependability overhead is the area for the DM and Xentium tile wrappers. The area of the DM is around 0.43mm^2; the area of 9 Xentium tile wrappers is about 0.8mm^2 and the area of the RFD chip is 43.8mm^2. Thus the dependability infrastructure overhead is about 3% in terms of silicon area. This is in the acceptable range specified by the end user (<10%).

In addition to the silicon area used by the dependability infrastructures, spare Xentium tiles in the RFD need to be reserved for the improvement of its reliability. In the example shown in Figure 6.29, one needs to find out the maximum number of Xentium tile processors that can be used for normal applications while still meeting the system reliability requirements. Moreover, dependability testing requires NoC bandwidth for transporting dependability test patterns. In order to have a minimum impact on the applications using the same NoC fabric, the NoC virtual channels used for dependability test are usually given lower priority. When combined with the pause/resume function of the DM, the dependability test can be performed without interrupting the on-going application in the RFD.

Extra power dissipation is also part of the cost for a dependable MPSoC. The actual power spent during a dependability test is related to the system operation frequency, the complexity of the dependability infrastructures and the technology in which these infrastructures are implemented. Moreover, the frequency at which the test is carried out also directly impact the total dependability-test power dissipation. A high degree of dependability requires high added power dissipation.

6.6 Conclusions

This chapter has covered details of the implementation and verification of the DM. The research goal is to prove the concept of our dependability approach

and the feasibility of the dependability test method in a homogeneous MPSoC platform. Before its realization in ASIC, the DM has first been implemented on an Xilinx Virtex-4 FPGA device. Thorough verifications have been carried out and experimental results indicate that the functional specifications of the DM have all been met.

Next, the DM infrastructural IP has been integrated into an General Stream Processor (GSP) platform, more specifically, in its Reconfigurable Fabric Devices (RFD) to provide dependability support. The RFD has been fabricated using UMC 90nm CMOS technology. The area of the RFD is 43.8mm^2 and the area of the DM is about 0.4mm^2. The overall dependability infrastructure overhead in terms of silicon area is about 3% which is below the 10% maximum area overhead demanded by the end user.

Dedicated dependability software has been designed to work with the DM to facilitate the dependability test in the RFD. The NoC data in the RFD has been directed to an on-board MICTOR connector and made accessible for an external logic analyzer. Measurement results of the test-vectors generated by the DM for scan-based dependability test have been gathered and proved to be consistent with the original ATPG test patterns. Test on the Xentium tile memory and logic part have been separately carried out and the DM can detect faulty Xentium tiles if a fault is emulated by the fault emulation block in the Xentium tile wrapper. An example application has been developed to run with the dependability test in the same RFD. Experimental results show that the dependability test can be carried out at application run-time without interrupting the function of other applications. Hence it is shown that our dependability approach is feasible and can be performed with little intrusion on user applications.

Dependability attributes such as reliability, availability and maintainability of the target MPSoC have all been improved as a result of our dependability approach. In the example shown in Section 6.5, when two Xentium tiles are used as spare, the reliability of the RFD is raised by about 40% by usage of our dependability approach. The inclusion of the DM into the RFD makes it a maintainable MPSoC with very short stuck-at and memory fault detection time (21ms) and very short MDT (hundreds of milliseconds). The cost of the dependability approach and its relation with system performance and power dissipation has been

analyzed. If a high degree of dependability, specifically less MDT, is required by the end user, then more frequent dependability tests have to be carried out and hence more power will be consumed.

In conclusion, the design and implementation of the dependability hardware (DM) has been proven to be successful. Our proposed dependability approach and dependability test methods have shown to be feasible and efficient to be used in an MPSoC device for overall dependability improvement.

Chapter 7

Conclusion

7.1 General conclusions

The CMOS down scaling trend has been negatively impacting the dependability of semiconductor devices. Thinner oxide layers as well as new MOSFET structures, such as the FinFET [Boko 00, Ahme 02, Cour 11], make it a major challenge to sustain the current IC dependability level with future devices, especially those complex ICs with multiple processor cores. This research was set out to explore the dependability attributes of an MPSoC and methods for their improvements. We have proposed a set of dependability enhancement approaches for a target MPSoC, designed hardware and software architecture accordingly and realized an implementation on silicon. The study aimed to answer the three research questions raised in the beginning of this thesis:

- What is a dependable MPSoC? What are the measures of its important attributes?
- How can one enhance the dependability of an MPSoC? What are the subsequent costs?
- How to implement and evaluate the proposed dependability approach in silicon?

7.1.1 MPSoC dependability attributes and measures

An MPSoC can be regarded as a complex computing system integrated into one silicon chip. For modern computing systems, dependability can be defined as the ability of a system to deliver expected services under given conditions. The three important dependability attributes, reliability, availability and maintainability of an MPSoC are examined in detail. Reliability denotes the probability that the system will fail after a certain period of time. Because physical repair of an embedded IP block in a chip is quite difficult, thus in the context of this thesis, maintainability refers to the isolation/bypass of faulty components and reconfiguration of the fault-free spare parts to maintain system functionality. Availability denotes the readiness of the system to provide correct service. Sometimes unavailability is used to indicate the time period when the system is unavailable for service. Unavailability can be measured by time varying from days, hours to milliseconds.

7.1.2 MPSoC dependability enhancement and costs

As a result of the dependability issues caused by the aggressive CMOS technology advances, the problem of how to make a dependable MPSoC boils down to the question how to make a dependable system out of undependable components. A common method to enhance the reliability of a system is to introduce spare parts. The major building blocks of an MPSoC are the processor cores. By defining these cores as working and spare cores, one can model the MPSoC as a K-out-of-N:G system. In Chapter 3, equations have been presented to evaluate the system reliability in a quantative way assuming a constant failure rate of individual cores. The variation of MPSoC reliability has been calculated given different number of working and spare cores. Theoretically, system reliability dramatically increases as more cores are used as spares. At the same time, the area overhead also increases. A such, one needs to find a balance between reliability improvements and resource overhead depending on the actual requirements of the application running on the MPSoC [Zhan 11, Kerk 08].

Meanwhile, maintainability can be realized by incorporating fault detection and self-repair features into an MPSoC. By dynamically detecting faults and

reconfiguring the system to circumvent them, the system can be regarded as functionally correct, though probably at the cost of a drop in performance. The time spent for fault detection and system repair is combined as system down time if the system cannot provide specified services. Faster fault detection and repair operations will decrease system down time and enable a highly available MPSoC.

7.1.3 Our dependability approach and implementation

The dependability approach proposed in this thesis consists of two major parts, the self-test and self-repair of a target MPSoC [Kuik 08]. Non-concurrent on-line testing holds the combined advantage of fault detection in terms of system availability and DfT overhead. The self-test of an MPSoC for the sake of dependability is defined as dependability test and is performed at processing-core level. Stuck-at fault models are used to find the permanent faults which could be caused by the degradation and wearout of the MPSoC.

By treating the basic elements in an MPSoC such as cores and NoC segments as resources, so-called resource management software can maintain a table of all the fault-free resources in the MPSoC. Faulty resources, being a core or a segment of NoC, can be deleted from the fault-free list. Reconfiguration and run-time mapping software can reallocate applications to fault-free resources [Ter 10].

In summary, the innovation of our dependability approach proposed in this thesis is to perform a dependability test at application run-time for fault detection and to make use of resource management software and run-time mapping software for core-level system repair by means of resource reconfiguration.

In order to show the concept of our dependability approach and to validate the feasibility of the proposed dependability test method, a homogeneous MPSoC platform with multiple Xentium processing tiles for e.g. a beam-forming application was adopted as the vehicle of our experiment [Zhan 09b]. A stand-alone infrastructural IP block, namely the Dependability Manager (DM), has been designed and integrated into the MPSoC platform. Test parallelism can be achieved by broadcasting the same test stimuli to multiple cores simultaneously. Test response evaluation can be achieved by comparing the test responses of several identical cores. Any faulty core can be detected by majority-voting the test

responses from cores under test.

The DM consists of a Test Pattern Generator (DM-TPG) for test vector generation, a Test Response Evaluator (DM-TRE) for test response evaluation and a Finite State Machine (DM-FSM) for internal control and communication with the dependability software running on the GPD. The reseeding technique has been adopted in the design of the DM-TPG to achieve test vector compression. A Matlab software tool has been developed to automate the design process of the DM-TPG. By synthesizing the generated TPG design with CMOS 90nm technology library, its timing and area status can be analyzed. Depending on the actual fault coverage requirement and the silicon area limit, one can generate a DM-TPG design with a specific set of test patterns. The DM-FSM block has been developed using the StateCAD software from Xilinx, which can generate synthesizable VHDL codes for the DM-FSM. The CAD methods used in the DM design greatly enhanced its possibility for reuse in similar applications.

In addition, dedicated test wrappers and NoC (reused as a TAM) were included into the platform MPSoC as well. A modified scan-based test scheme was used for a back-pressure style test data flow control by pausing and resuming the test data in the NoC. The test of the NoC was performed via software on a periodic basis [Kerk 10].

The MPSoC platform was fabricated as a Reconfigurable Fabric Device (RFD) using UMC 90nm CMOS technology. The area of the RFD is 43.8mm^2 and the area of the DM is about 0.4mm^2. The dependability overhead in terms of silicon area is about 1%. Dedicated dependability software has been designed to work with the DM to facilitate the dependability test in the RFD. The NoC data in the RFD has been directed to an on-board MICTOR connector and made accessible for an external logic analyzer. Thorough functional tests have been carried out and experimental results indicate that the functional specifications of the DM have all been met.

Measurement results of the test vectors generated by the DM for scan-based dependability test have been gathered and proved to be consistent with the original ATPG test patterns. The DM can detect a faulty Xentium tile when a fault is emulated by the fault emulation block in the Xentium tile wrapper. An example application has been developed to run together with the dependability test in the

same RFD. Experimental results show that the dependability test can be carried out at application run-time without interrupting the function of other applications. The inclusion of the DM into the RFD makes it a maintainable MPSoC with very short stuck-at and memory fault detection time (21ms) and reasonable MDT (hundreds of milliseconds) [Zhan 11].

In conclusion, all the research questions raised at the beginning of this thesis have been answered. The design and implementation of the dependability hardware (DM) has proven to be successful. Our proposed dependability approach and dependability test methods have proven to be feasible and efficient. The successful integration of the DM into the RFD and its correct operation indicate that our dependability approach can be applied to similar homogeneous MPSoC platforms to bring dependability improvement.

However, recent research has pointed out a limitation of our test-based dependability approach. Due to the continuous scan-based structural tests, the power dissipation of our approach is higher than the recently developed health monitoring scheme which uses various sensors to monitor dependability parameters [Zhao 13].

7.2 Future work

The research work described in thesis can be extended in several ways. First, the core of the dependability manager, the test-pattern generator, has room for improvement. The reseeding, bit-flipping and compression method can be all incorporated to achieve the best test-pattern compression ratio. The dependability approach proposed in this thesis can be slightly modified to deal with more types of faults. For example, by executing the dependability test for multiple times, it is possible to detect intermittent or transient faults. It may also require a totally different test approach to test the new type of faults and new test infrastructures may be needed as well.

Second, it is suggested to extend the current DM design into an embedded instrument IP. A better tool chain can be made to fully customize the DM to deal with different type of faults and accommodate various processor platforms. In general, the DM can be made into a universal dependability IP for MPSoCs

with different processing tiles.

Third, the current method of the NoC dependability test is based on a software functional test. One should also consider the possibility to structurally test the NoC segments at application run-time, e.g. use test wrappers to separate the parts under test. The existing DM can be used for the structural test of the NoC with little or even no modification. As such, the dependability test of the complete MPSoC platform can all be done within the DM and the general purpose processor can be freed for other tasks.

Last but not least, smart system software can be made to better coordinate the dependability test. A balance of the frequency of the dependability test and application activities should be achieved.

Appendix A

DM-FSM Design Using the StateCAD Software

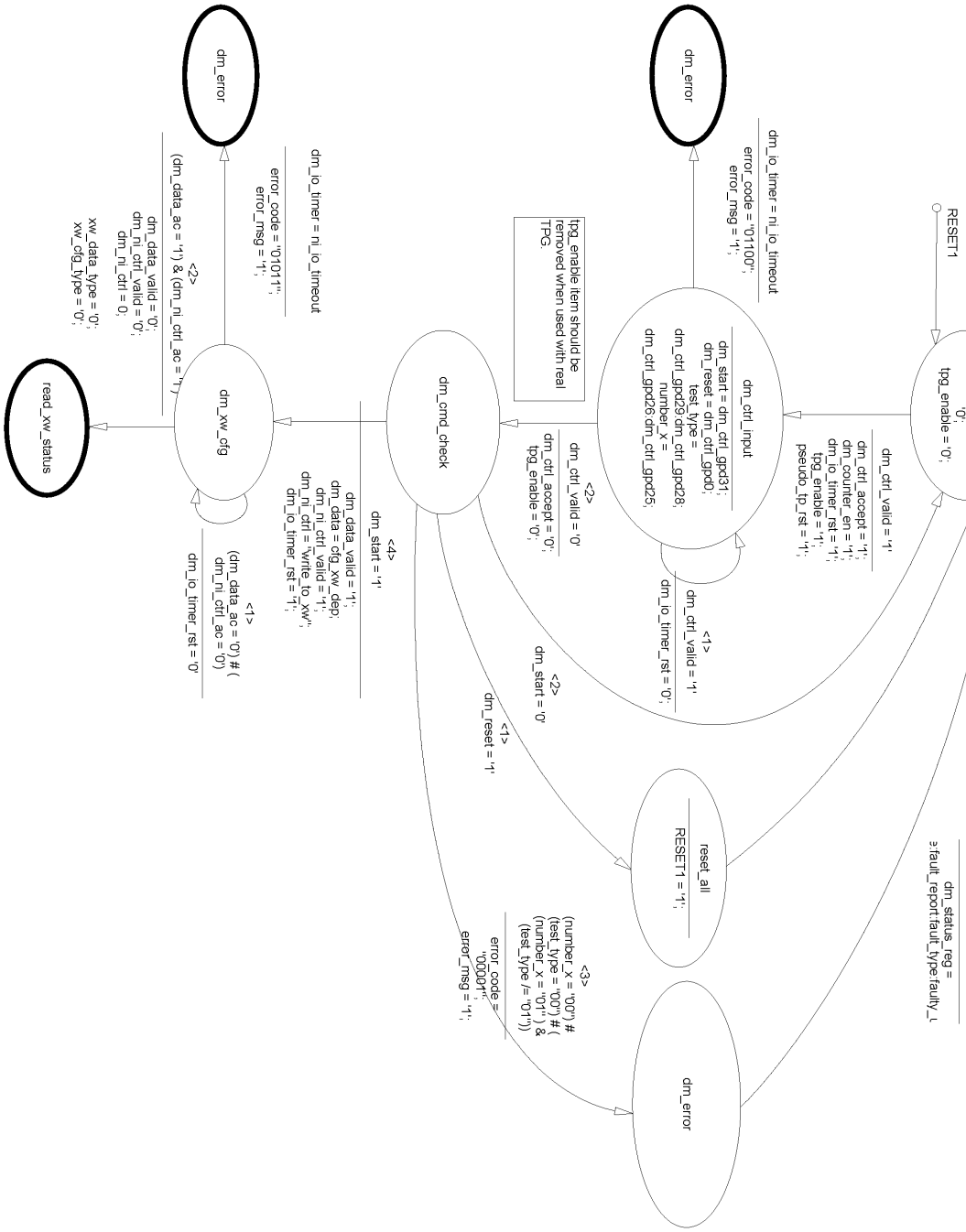


Figure A.1: DM-FSM design part 1

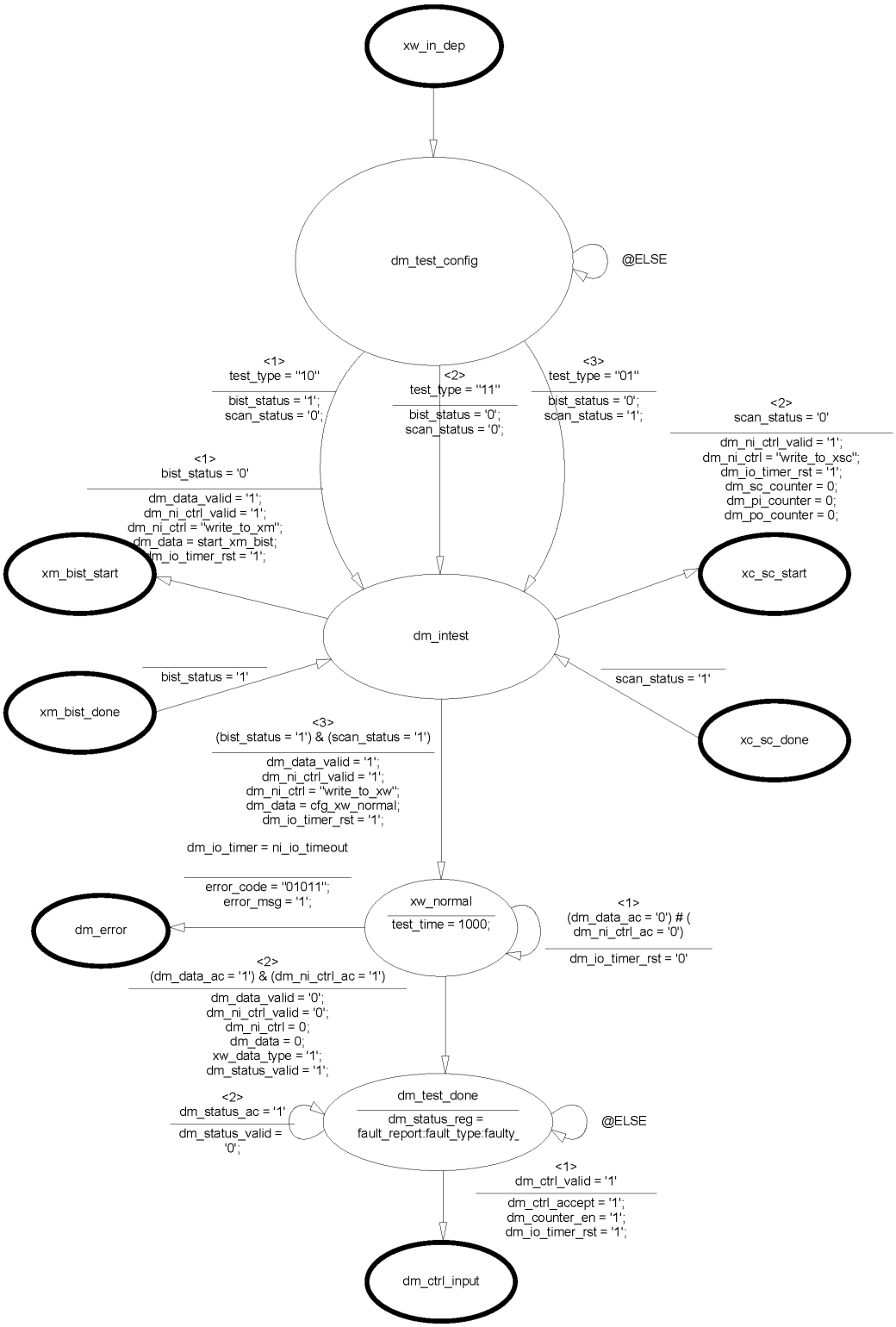


Figure A.2: DM-FSM design part 2

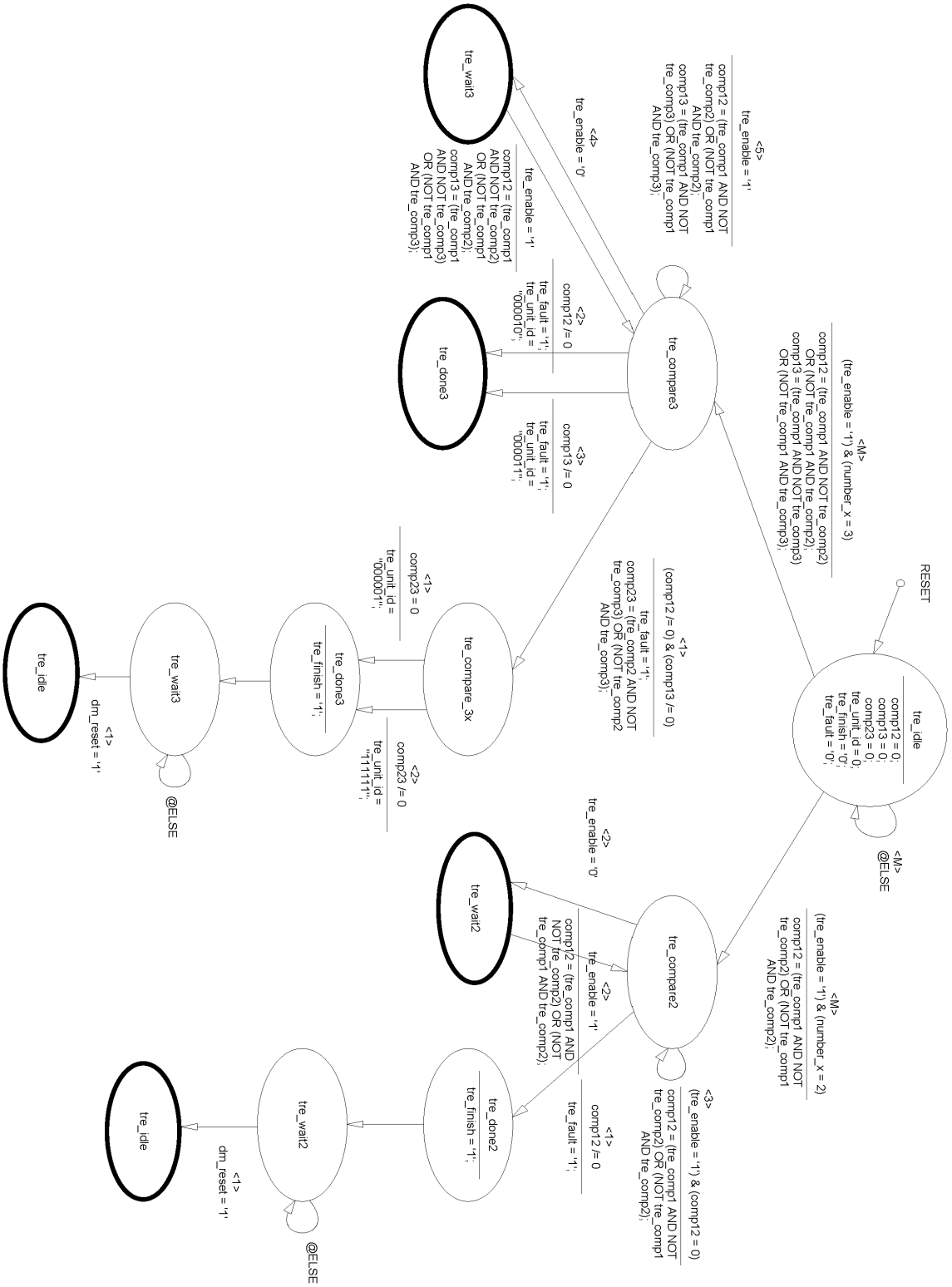


Figure A.3: DM-FSM design part 3

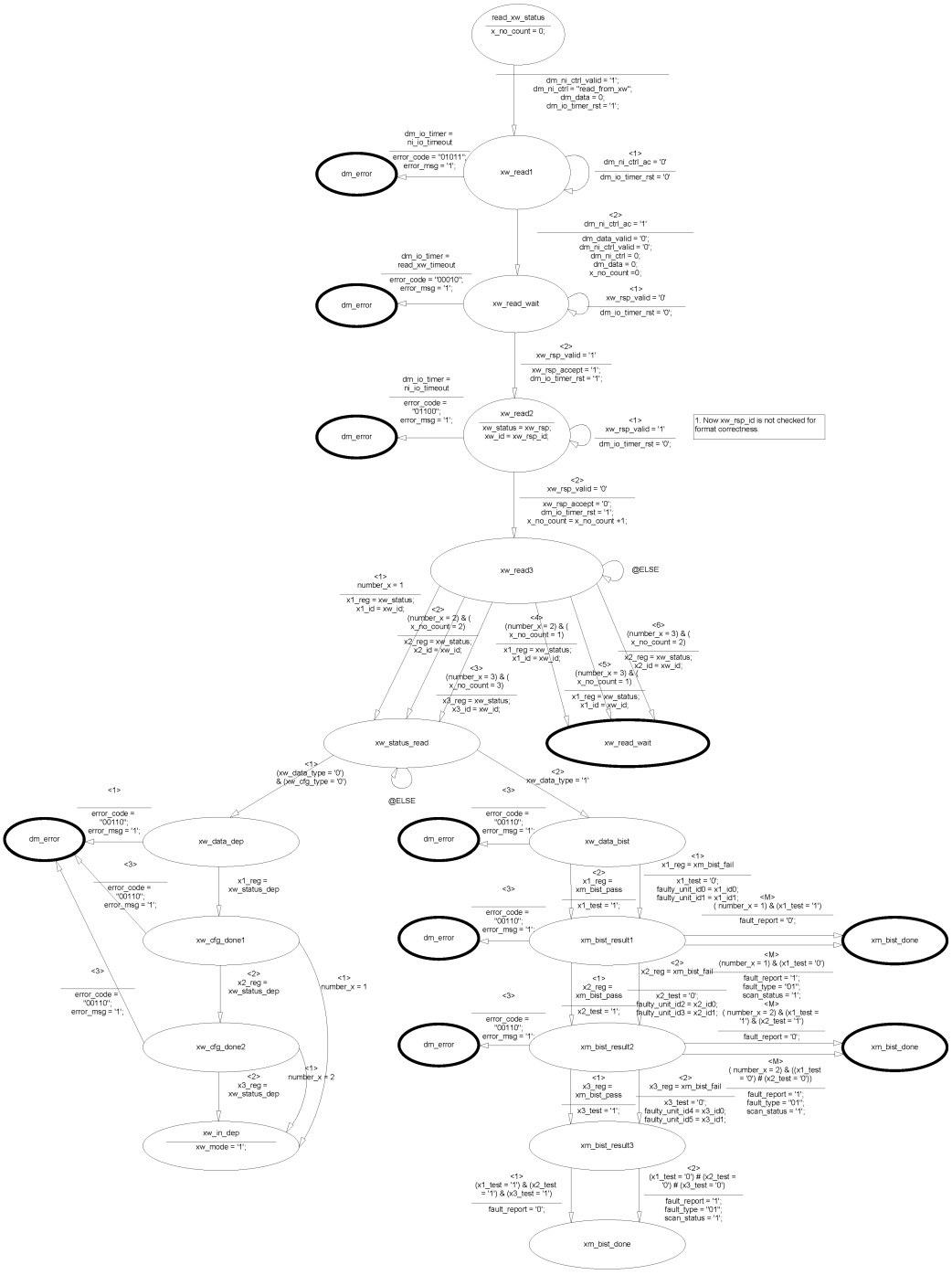


Figure A.4: DM-FSM design part 4

A. DM-FSM DESIGN USING THE STATECAD SOFTWARE

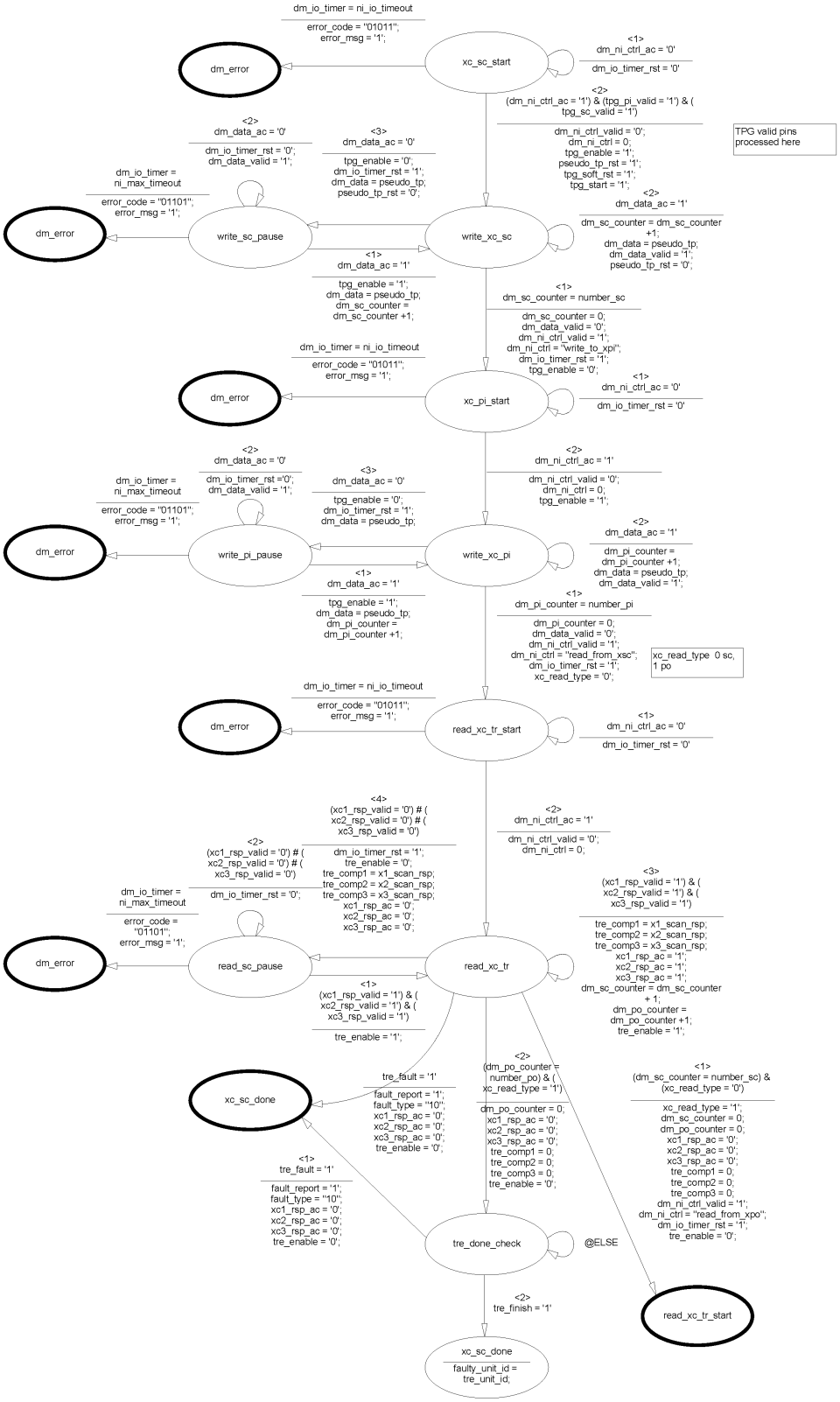


Figure A.5: DM-FSM design part 5

Bibliography

[Ahme 02] S. Ahmed, S. Bell, C. Tabery, J. Bokor, and D. Kyser. "FinFET scaling to 10 nm gate length". In: *Digest. International Electron Devices Meeting*, pp. 251–254, IEEE, 2002.

[AlYa 03] A. AlYamani and E. McCluskey. "Built-in reseeding for serial BIST". In: *Proceedings of 21st VLSI Test Symposium (VTS)*, pp. 63–68, 2003.

[Amor 05] A. Amory, E. Briao, E. Cota, M. Lubaszewski, and F. Moraes. "A scalable test strategy for network-on-chip routers". In: *Proceedings International Test Conference*, pp. 591–599, 2005.

[Anth 12] M. Anthony and M. Harvey. *Linear Algebra: Concepts and Methods*. Cambridge University Press, 2012.

[Asch 84] H. Ascher and H. Feingold. *Repairable Systems Reliability: Modeling, Inference, Misconceptions and Their Causes*. CRC Press/Marcel Dekker, Inc., 1984.

[ASIC 07] "What is the difference between FPGA and ASIC". `http://goo.gl/6T6sK`, 2007.

[Atme 09] Atmel Corporation. "AT91 ARM Thumb Microcontrollers datasheet". 2009.

[Aviz 04] 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. 11–33, Jan. 2004.

[Axer 11] P. Axer, M. Sebastian, and R. Ernst. "Reliability analysis for MPSoCs with mixed-critical, hard real-time constraints". In: *9th International Conference on Hardware/Software Codesign and System Synthesis*, pp. 149–158, 2011.

[Bala 06] K. Balakrishnan and S. Chakradhar. "PIDISC: pattern independent design independent seed compression technique". In: *19th International Conference on VLSI Design (VLSID 06)*, pp. 811–817, 2006.

[Bazo 04] I. Bazovsky. *Reliability Theory and Practice*. Dover Publications, 2004.

BIBLIOGRAPHY

[Beni 02] L. Benini and G. De Micheli. "Networks on chips: a new SoC paradigm". *Computer*, Vol. 35, No. 1, pp. 70–78, 2002.

[Bern 06] J. B. Bernstein, M. Gurfinkel, X. Li, J. Walters, Y. Shapira, and M. Talmor. "Electronic circuit reliability modeling". *Microelectronics Reliability*, Vol. 46, No. 12, pp. 1957–1979, Dec. 2006.

[Blac 69] J. Black. "Electromigration–A brief survey and some recent results". *IEEE Transactions on Electron Devices*, Vol. 16, No. 4, pp. 338–347, Apr. 1969.

[Bloo 12] Bloomberg News. "Tokyo Stock Exchange Computer Outage Sparks Biggest Trading Halt Since 2006". `http://goo.gl/tF0lj`, 2012.

[Boko 00] J. Bokor, E. Anderson, C. Kuo, K. Asano, H. Takeuchi, J. Kedzierski, and D. Hisamoto. "FinFET-a self-aligned double-gate MOSFET scalable to 20 nm". *IEEE Transactions on Electron Devices*, Vol. 47, No. 12, pp. 2320–2325, 2000.

[Bork 07] S. Borkar. "Thousand core chips: a technology perspective". In: *Proceedings of the 44th Annual Design Automation Conference*, pp. 746–749, ACM, 2007.

[Burg 10] S. Burgess, T. Ahonen, and J. Nurmi. "Software Based Approach to Fault Diagnosis for Multi-Die Networks-on-Chip". In: *2010 Conference on System, Software, SoC and Silicon Debug (S4D 10)*, 2010.

[Burg 11] S. T. Burgess, H. Kerkhoff, X. Zhang, *et al.* *Reconfigurable Computing*, Chap. CRISP: Cutting Edge Reconfigurable ICs for Stream Processing, pp. 211–237. Springer New York, 2011.

[Bush 05] M. Bushnell and V. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer, 2005.

[Caze 04] J. Cazeaux, D. Rossi, and C. Metra. "New high speed CMOS self-checking voter". In: *10th IEEE International On-Line Testing Symposium, (IOLTS 2004)*, pp. 58–63, 2004.

[Chan 03] A. Chandra and K. Chakrabarty. "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes". *IEEE Transactions on Computers*, Vol. 52, No. 8, pp. 1076–1088, Aug. 2003.

[Chen 03] L. Chen, S. Ravi, A. Raghunathan, and S. Dey. "A scalable software-based self-test methodology for programmable processors". In: *Design Automation Conference, 2003. Proceedings*, pp. 548–553, June 2003.

[Chin 11] China State Administration of Work Safety. "Jul. 23 Yongwen Line Train Collision Accident Investigation Report". http://www.chinasafety.gov.cn/newpage/Contents/Channel_5498/2011/1228/160577/content_160577.htm, 2011.

[Clar 94] D. Clark and L.-J. Weng. "Maximal and near-maximal shift register sequences: efficient event counters and easy discrete logarithms". *IEEE Transactions on Computers*, Vol. 43, No. 5, pp. 560–568, May 1994.

[Cnet 11] "Intel's Sandy Bridge chipset flaw: The fallout". http://www.cnet.com/news/intels-sandy-bridge-chipset-flaw-the-fallout/, 2011.

[Cota 03] E. Cota, M. Kreutz, C. Zeferino, L. Carro, M. Lubaszewski, and A. Susin. "The impact of NoC reuse on the testing of core-based systems". In: *Proceedings 21st VLSI Test Symposium (VTS)*, pp. 128–133, 2003.

[Cota 04] E. Cota, L. Carro, and M. Lubaszewski. "Reusing an on-chip network for the test of core-based systems". *ACM Transactions on Design Automation of Electronic Systems*, Vol. 9, No. 4, pp. 471–499, Oct. 2004.

[Cota 06] E. Cota and C. Liu. "Constraint-Driven Test Scheduling for NoC-Based Systems". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25, No. 11, pp. 2465–2478, Nov. 2006.

[Cota 08] E. Cota, F. L. Kastensmidt, M. Cassel, *et al.* "A High-Fault-Coverage Approach for the Test of Data, Control and Handshake Interconnects in Mesh Networks-on-Chip". *IEEE Transactions on Computers*, Vol. 57, No. 9, pp. 1202–1215, Sep. 2008.

[Cour 11] R. Courtland. "The Origins of Intel's New Transistor, and Its Future". http://spectrum.ieee.org/semiconductors/design/the-origins-of-intels-new-transistor-and-its-future, 2011.

[CRISP 07] "The CRISP Project". http://www.crisp-project.eu/, 2007.

[Dall 87] W. J. Dally. "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks". *IEEE Transactions on Computers*, Vol. C-36, No. 5, pp. 547–553, May 1987.

[Edmo 90] P. Edmond, A. Gupta, D. Siewiorek, and A. Brennan. "ASSURE: automated design for dependability". In: *Design Automation Conference, 1990. Proceedings, 27th ACM/IEEE*, pp. 555–560, Jun 1990.

BIBLIOGRAPHY

[Eich 83] E. B. Eichelberger and E. Lindbloom. "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test". *IBM Journal of Research and Development*, Vol. 27, No. 3, pp. 265–272, May 1983.

[Gonc 03] P. Gonciari, B. Al-Hashimi, and N. Nicolici. "Variable-length input Huffman coding for system-on-a-chip test". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 22, No. 6, pp. 783–796, June 2003.

[Gras 09] T. Grasser and B. Kaczer. "Evidence That Two Tightly Coupled Mechanisms Are Responsible for Negative Bias Temperature Instability in Oxynitride MOSFETs". *IEEE Transactions on Electron Devices*, Vol. 56, No. 5, pp. 1056–1062, May 2009.

[Grec 05] C. Grecu, P. Pande, A. Ivanov, and R. Saleh. "Methodologies and algorithms for testing switch-based NoC interconnects". In: *20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT05)*, pp. 238–246, 2005.

[Guo 07] F. Guo, Y. Solihin, L. Zhao, and R. Iyer. "A Framework for Providing Quality of Service in Chip Multi-Processors". In: *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pp. 343–355, 2007.

[Hakm 07] A.-W. Hakmi, H.-J. Wunderlich, C. G. Zoellin, A. Glowatz, and F. Hapke. "Programmable deterministic Built-In Self-Test". In: *Proceedings International Test Conference (ITC 2007)*, pp. 1–9, 2007.

[Hari 11] N. Haridas and M. N. Devi. "Efficient linear feedback shift register design for pseudo exhaustive test generation in BIST". In: *2011 3rd International Conference on Electronics Computer Technology*, pp. 350–354, Apr. 2011.

[Harr 99] P. Harrod. "Testing reusable IP - a case study". In: *Proceedings International Test Conference (ITC)*, pp. 493–498, 1999.

[Haye 98] B. Hayes. "How to Avoid Yourself". *American Scientist*, Vol. 86, No. 4, p. 314, 1998.

[Hell 92] S. Hellebrand, S. Tarnick, J. Rajski, and B. Courtois. "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers". In: *Proceedings International Test Conference 1992 (ITC 1992)*, p. 120, 1992.

[Hell 95] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois. "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers". *IEEE Transactions on Computers*, Vol. 44, No. 2, pp. 223–233, 1995.

[Here 88] P. Heremans, R. Bellens, G. Groeseneken, and H. Maes. "Consistent model for the hot-carrier degradation in n-channel and p-channel MOSFETs". *IEEE Transactions on Electron Devices*, Vol. 35, No. 12, pp. 2194–2209, 1988.

[Heys 04] P. Heysters. *Coarse-Grained Reconfigurable Processors - Flexibility meets Efficiency*. PhD thesis, University of Twente, 2004.

[Hinc 10] M. Hinchey and L. Coyle. "Evolving Critical Systems: A Research Agenda for Computer-Based Systems". In: *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, pp. 430–435, IEEE, 2010.

[Hu 12] C. Hu. "3D FinFET and other sub-22nm transistors". In: *2012 19th IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits*, pp. 1–5, IEEE, July 2012.

[Huan 08] L. Huang and Q. Xu. "On Modeling the Lifetime Reliability of Homogeneous Manycore Systems". In: *2008 14th IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 87–94, Dec. 2008.

[IEC 07] "Application guide to the specification of dependability requirements". IEC standard 60300-3-4, Sep. 2007.

[IEC 12] "IEC Dependability Definitions". `http://tc56.iec.ch/about/definitions.htm#Dependability`, 2012.

[IEEE 05] *IEEE 1500 Standard for Embedded Core Test*, 2005. `http://grouper.ieee.org/groups/1500/index.html`.

[IFIP 69] "IFIP WG 10.4 on Dependable Computing and Fault Tolerance". `http://www.dependability.org/wg10.4/`, 1969.

[ITRS 11] "International Technology Roadmap for Semiconductors". `http://www.itrs.net/Links/2011ITRS/Home2011.htm`, 2011.

[Iyer 07] R. Iyer, L. Zhao, F. Guo, Y. Solihin, S. Markineni, D. Newell, R. Illikkal, L. Hsu, and S. Reinhardt. "QoS Policy and Architecture for Cache/Memory in CMP Platforms". In: *ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pp. 25–36, IEEE, 2007.

[Jas 03] A. Jas, J. Ghosh-Dastidar, and N. Touba. "An efficient test vector compression scheme using selective Huffman coding". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 22, No. 6, pp. 797–806, June 2003.

BIBLIOGRAPHY

[Jas 99] A. Jas, J. Ghosh-Dastidar, and N. Touba. "Scan vector compression/decompression using statistical coding". In: *Proceedings 17th IEEE VLSI Test Symposium*, pp. 114–120, 1999.

[Kerk 08] H. Kerkhoff, O. J. Kuiken, and X. Zhang. "Increasing SoC Dependability via Known Good Tile NoC Testing". In: *FastAbs Track of The 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN2008)*, pp. 4–5, 2008.

[Kerk 10] H. Kerkhoff and X. Zhang. "Design of an Infrastructural IP Dependability Manager for a Dependable Reconfigurable Many-Core Processor". *2010 Fifth IEEE International Symposium on Electronic Design, Test & Applications (DELTA)*, pp. 270–275, 2010.

[Kief 98] G. Kiefer and H.-J. Wunderlich. "Deterministic BIST with multiple scan chains". In: *Proceedings International Test Conference 1998 (ITC 1998)*, pp. 1057–1064, 1998.

[Kim 09] K. O. Kim and W. Kuo. "Optimal burn-in for maximizing reliability of repairable non-series systems". *European Journal of Operational Research*, Vol. 193, No. 1, pp. 140–151, Feb. 2009.

[Kone 91] B. Könemann. "LFSR-coded test patterns for scan designs". In: *Proc. of European Test Conference (ETS)*, pp. 237–242, 1991.

[Kora 02] S. Koranne. "On test scheduling for core-based SOCs". In: *Proceedings of the 15th International Conference on VLSI Design (VLSID02)*, pp. 505–510, 2002.

[Kran 03] N. Kranitis, G. Xenoulis, D. Gizopoulos, A. Paschalis, and Y. Zorian. "Low-cost software-based self-testing of RISC processor cores". *IEE Proceedings - Computers and Digital Techniques*, Vol. 150, No. 5, p. 355, 2003.

[Kris 01] C. Krishna, A. Jas, and N. Touba. "Test vector encoding using partial LFSR reseeding". In: *Proceedings International Test Conference (ITC 2001)*, pp. 885–893, 2001.

[Kris 02] C. Krishna and N. Touba. "Reducing test data volume using LFSR reseeding with seed compression". In: *Proceedings International Test Conference (ITC 2002)*, pp. 321–330, 2002.

[Kuik 08] O. Kuiken. "Improvement of dependability of an SoC via self-diagnosis based on the use of identical cores". Master's Thesis, University of Twente, 2008.

[Kuo 84] W. Kuo. "Reliability Enhancement Through Optimal Burn-In". *IEEE Transactions on Reliability*, Vol. R-33, No. 2, pp. 145–156, June 1984.

[Lay 11] D. C. Lay. *Linear Algebra and Its Applications, 4th Edition*. Pearson, 2011.

[Liu 05] C. Liu, V. Iyengar, J. Shi, and E. Cota. "Power-Aware Test Scheduling in Network-on-Chip Using Variable-Rate On-Chip Clocking". In: *23rd IEEE VLSI Test Symposium (VTS'05)*, pp. 349–354, 2005.

[Liu 98] H. Liu and M. Ieee. "Reliability of a load-sharing k-out-of-n:G system: non-iid components with arbitrary distributions". *IEEE Transactions on Reliability*, Vol. 47, No. 3, pp. 279–284, 1998.

[Lu 06] Z. Lu and W. Liu. "Reliability Evaluation of STATCOM Based on the k-out-of-n: G Model". In: *2006 International Conference on Power System Technology*, pp. 1–6, Oct. 2006.

[Maha 00] S. Mahapatra, C. Parikh, V. Rao, C. Viswanathan, and J. Vasi. "Device scaling effects on hot-carrier induced interface and oxide-trapped charge distributions in MOSFETs". *IEEE Transactions on Electron Devices*, Vol. 47, No. 4, pp. 789–796, Apr. 2000.

[Maka 07] S. Makar, T. Altinis, N. Patkar, and J. Wu. "Testing of Vega2, a chip multiprocessor with spare processors.". In: *Proceedings International Test Conference (ITC)*, pp. 1–10, 2007.

[Mari 98] E. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters. "A structured and scalable mechanism for test access to embedded reusable cores". In: *Proceedings International Test Conference (ITC)*, pp. 284–293, 1998.

[McCl 85] E. McCluskey. "Built-In Self-Test Techniques". *IEEE Design & Test of Computers*, Vol. 2, No. 2, pp. 21–28, 1985.

[McPh 06] J. McPherson. "Reliability challenges for 45nm and beyond". In: *2006 43rd ACM/IEEE Design Automation Conference*, pp. 176–181, 2006.

[Mumt 11] A. Mumtaz, M. E. Imhof, S. Holst, and H.-J. Wunderlich. "Embedded Test for Highly Accurate Defect Localization". In: *2011 Asian Test Symposium*, pp. 213–218, Nov. 2011.

[Nico 98] M. Nicolaidis, Y. Zorian, and S. Jose. "On-Line Testing for VLSI - A Compendium of Approaches". *Journal of Electronic Testing: Theory and Applications*, Vol. 12, pp. 7–20, 1998.

BIBLIOGRAPHY

[Parh 88] B. Parhami. "From defects to failures: a view of dependable computing". *ACM SIGARCH Computer Architecture News*, pp. 157–168, 1988.

[Paru 02] I. Parulkar, T. Ziaja, R. Pendurkar, A. D'Souza, and A. Majumdar. "A scalable, low cost design-for-test architecture for UltraSPARC chip multi-processors". In: *Proceedings. International Test Conference (ITC)*, pp. 726–735, 2002.

[Rabb 10] C. A. Rabbath and N. Lechevin. *Safety and Reliability in Cooperating Unmanned Aerial Systems*. World Scientific Publishing Company, 2010.

[Rade 13] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch. "Methods for fault tolerance in networks-on-chip". *ACM Computing Surveys*, Vol. 46, No. 1, pp. 1–38, Oct. 2013.

[Rajs 00] J. Rajski, N. Tamarapalli, and J. Tyszer. "Automated synthesis of phase shifters for built-in self-test applications". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 19, No. 10, pp. 1175–1188, 2000.

[Rajs 98] J. Rajski, J. Tyszer, and N. Zacharia. "Test data decompression for multiple scan designs with boundary scan". *IEEE Transactions on Computers*, Vol. 47, No. 11, pp. 1188–1200, 1998.

[Recore 13] "Xentium VLIW DSP". `http://www.recoresystems.com/products/xentium-vliw-dsp-ip/`, 2013.

[Rhee 04] C.-E. Rhee, H.-Y. Jeong, and S. Ha. "Many-to-many core-switch mapping in 2-D mesh NoC architectures". In: *Proceedings Computer Design: VLSI in Computers and Processors*, pp. 438–443, Oct 2004.

[Sava 60] Y. Savaria, M. Youssef, B. Kaminska, and M. Koudil. "Automatic test point insertion for pseudo-random testing". In: *IEEE International Symposium on Circuits and Systems*, pp. 1960–1963, 1960.

[Schr 03] D. K. Schroder and J. A. Babcock. "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing". *Journal of Applied Physics*, Vol. 94, No. 1, p. 1, 2003.

[Schu 05] E. Schuchman and T. Vijaykumar. "Rescue: A Microarchitecture for Testability and Defect Tolerance". In: *32nd International Symposium on Computer Architecture (ISCA'05)*, pp. 160–171, 2005.

[Shao 91] J. Shao and L. Lamberson. "Modeling a shared-load k-out-of-n:G system". *IEEE Transactions on Reliability*, Vol. 40, No. 2, pp. 205–209, June 1991.

[Stat 02] J. H. Stathis. "Reliability limits for the gate insulator in CMOS technology". *IBM Journal of Research and Development*, Vol. 46, No. 2.3, pp. 265–286, March 2002.

[Stat 10] J. Stathis, M. Wang, and K. Zhao. "Reliability of advanced high-k/metal-gate n-FET devices". *Microelectronics Reliability*, Vol. 50, No. 9-11, pp. 1199–1202, Sep. 2010.

[Stew 06] K. Stewart and S. Tragoudas. "Interconnect Testing for Networks on Chips". In: *24th IEEE VLSI Test Symposium (VTS)*, pp. 100–107, 2006.

[Taiw 06] Taiwan Semiconductor Manufacturing Company Limited (TSMC). "TSMC 90nm Core Library Application Note". 2006.

[Ter 10] T. D. Ter Braak, S. Burgess, H. Hurskainen, H. Kerkhoff, B. Vermeulen, and X. Zhang. "On-line dependability enhancement of multiprocessor SoCs by resource management". In: *2010 International Symposium on System on Chips*, pp. 103–110, Sep. 2010.

[Ter 11] T. D. Ter Braak, H. A. Toersche, A. B. J. Kokkeler, and G. J. M. Smit. "Adaptive resource allocation for streaming applications". In: *2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, pp. 388–395, July 2011.

[Terr 85] K. Terrill. "Hot-Electron-Induced MOSFET Degradation - Model, Monitor, and Improvement". *IEEE Journal of Solid-State Circuits*, Vol. 20, No. 1, pp. 295–305, Feb. 1985.

[Toms 12] "Intel Releases Itanium 9500, Packing Up to 8 Cores". `http://goo.gl/AJrSY`, 2012.

[Toub 06] N. Touba. "Survey of Test Vector Compression Techniques". *IEEE Design & Test of Computers*, Vol. 23, No. 4, pp. 294–303, Apr. 2006.

[Toub 96] N. Touba and E. McCluskey. "Test point insertion based on path tracing". In: *Proceedings of 14th VLSI Test Symposium (VTS)*, pp. 2–8, 1996.

[Tsai 11] C.-C. Tsai, S.-T. Tseng, and N. Balakrishnan. "Optimal Burn-In Policy for Highly Reliable Products Using Gamma Degradation Process". *IEEE Transactions on Reliability*, Vol. 60, No. 1, pp. 234–245, March 2011.

[Varm 98] P. Varma and B. Bhatia. "A structured test re-use methodology for core-based system chips". In: *Proceedings International Test Conference (ITC)*, pp. 294–302, 1998.

BIBLIOGRAPHY

[Volk 03] E. Volkerink and S. Mitra. "Efficient seed utilization for reseeding based compression". In: *Proceeding of 21st VLSI Test Symposium (VTS)*, pp. 232–237, IEEE Comput. Soc, 2003.

[Wats 12] J. Watson and C. Castro. "High-Temperature Electronics Pose Design and Reliability Challenges". *Analog Dialogue*, pp. 46–04, 2012.

[Whit 08a] M. White and J. B. Bernstein. "Microelectronics Reliability: Physics-of-Failure Based Modeling and Lifetime Evaluation". *NASA Electronic Parts and Packaging (NEPP) Program JPL Publication 08-5 2/08*, 2008.

[Whit 08b] M. White and Y. Chen. "Scaled CMOS Technology Reliability Users Guide". *NASA Electronic Parts and Packaging (NEPP) Program JPL Publication 08-14 3/08*, p. 15, 2008.

[Wolk 09] P. T. Wolkotte. *Exploration within the Network-on-Chip Paradigm*. PhD thesis, University of Twente, 2009.

[Wund 96] H.-J. Wunderlich and G. Kiefer. "Bit-flipping BIST". In: *Proceedings of International Conference on Computer Aided Design (ICCAD)*, pp. 337–343, 1996.

[Xeno 03] G. Xenoulis, D. Gizopoulos, N. Kranitis, and A. Paschalis. "Low-cost, on-line software-based self-testing of embedded processor cores". In: *9th IEEE On-Line Testing Symposium (IOLTS 2003)*, pp. 149–154, 2003.

[Xilinx 07] "StateCAD Introduction". `http://www.xilinx.com/itp/xilinx10/help/iseguide/mergedProjects/state/whnjs.htm`, 2007.

[Zhan 09a] L. Zhang, Y. Han, Q. Xu, and X. Li. "On Topology Reconfiguration for Defect-Tolerant NoC-Based Homogeneous Manycore Systems". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 17, No. 9, pp. 1173–1186, Sep. 2009.

[Zhan 09b] X. Zhang and H. Kerkhoff. "Design of a Highly Dependable Beamforming Chip". In: *2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, pp. 729–735, Aug. 2009.

[Zhan 10a] L. Zhang, Y. Yu, J. Dong, Y. Han, S. Ren, and X. Li. "Performance-asymmetry-aware topology virtualization for defect-tolerant NoC-based many-core processors". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*, pp. 1566–1571, 2010.

[Zhan 10b] X. Zhang, H. Kerkhoff, and B. Vermeulen. "On-chip Scan-Based Test Strategy for a Dependable Many-Core Processor Using a NoC as a Test Access Mechanism". *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, pp. 531–537, Sep. 2010.

[Zhan 11] X. Zhang and H. G. Kerkhoff. "A Dependability Solution for Homogeneous MP-SoCs". In: *2011 IEEE 17th Pacific Rim International Symposium on Dependable Computing*, pp. 53–62, Dec. 2011.

[Zhao 03] D. Zhao and S. Upadhyaya. "Power constrained test scheduling with dynamically varied TAM". In: *Proceedings. 21st VLSI Test Symposium (VTS)*, pp. 273–278, 2003.

[Zhao 13] Y. Zhao, X. Zhang, and H. G. Kerkhoff. "Power-dissipation comparison of two dependability approaches for multi-processor systems". In: *IEEE 8th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, pp. 56–61, March 2013.

[Zhu 09] J.-J. Zhu, W.-C. Lin, J.-H. Ye, and M.-D. Shieh. "Efficient Software-Based Self-Test Methods for Embedded Digital Signal Processors". In: *Asian Test Symposium (ATS), 2009*, pp. 206–211, Nov 2009.