Teijo Lehtonen

On Fault Tolerance Methods for Networks-on-Chip

Turku Centre *for* Computer Science

TUCS Dissertations
No 122, October 2009

On Fault Tolerance Methods for Networks-on-Chip

Teijo Lehtonen

To be presented, with the permission of the Faculty of Mathematics and Natural Sciences of the University of Turku, for public criticism in Auditorium Lambda on November 13, 2009, at 12 noon.

University of Turku
Department of Information Technology
20014 Turun yliopisto

2009

Supervisors

Adjunct Professor Juha Plosila
Department of Information Technology
University of Turku
20014 Turun yliopisto
Finland

PhD Pasi Liljeberg
Department of Information Technology
University of Turku
20014 Turun yliopisto
Finland

Reviewers

Adjunct Professor Elena Troubitsyna
Department of Information Technologies
Åbo Akademi University
Joukahaisenkatu 3-5 A, 20520 Turku
Finland

PhD Gert Jervan
Department of Computer Engineering
Tallinn University of Technology
Raja 15, 12618 Tallinn
Estonia

Opponent

Professor Peeter Ellervee
Department of Computer Engineering
Tallinn University of Technology
Raja 15, 12618 Tallinn
Estonia

ISBN 978-952-12-2354-9 (PRINT)
ISBN 978-952-12-2355-6 (PDF)
ISSN 1239-1883

Abstract

Technology scaling has proceeded into dimensions in which the reliability of manufactured devices is becoming endangered. The reliability decrease is a consequence of physical limitations, relative increase of variations, and decreasing noise margins, among others. A promising solution for bringing the reliability of circuits back to a desired level is the use of design methods which introduce tolerance against possible faults in an integrated circuit.

This thesis studies and presents fault tolerance methods for network-on-chip (NoC) which is a design paradigm targeted for very large systems-on-chip. In a NoC resources, such as processors and memories, are connected to a communication network; comparable to the Internet. Fault tolerance in such a system can be achieved at many abstraction levels.

The thesis studies the origin of faults in modern technologies and explains the classification to transient, intermittent and permanent faults. A survey of fault tolerance methods is presented to demonstrate the diversity of available methods. Networks-on-chip are approached by exploring their main design choices: the selection of a topology, routing protocol, and flow control method. Fault tolerance methods for NoCs are studied at different layers of the OSI reference model.

The data link layer provides a reliable communication link over a physical channel. Error control coding is an efficient fault tolerance method especially against transient faults at this abstraction level. Error control coding methods suitable for on-chip communication are studied and their implementations presented. Error control coding loses its effectiveness in the presence of intermittent and permanent faults. Therefore, other solutions against them are presented. The introduction of spare wires and split transmissions are shown to provide good tolerance against intermittent and permanent errors and their combination to error control coding is illustrated.

At the network layer positioned above the data link layer, fault tolerance can be achieved with the design of fault tolerant network topologies and routing algorithms. Both of these approaches are presented in the thesis together with realizations in the both categories. The thesis concludes that an optimal fault tolerance solution contains carefully co-designed elements from different abstraction levels.

Acknowledgements

This work was carried out at the Department of Information Technology, University of Turku, during years 2005-2009. I would like to express my gratitude to Turku Centre for Computer Science (TUCS) Graduate School for the financial support.

I am truly grateful for all the support that my supervisors, Juha Plosila and Pasi Liljeberg, have given to me during these years. Despite of all the other commitments you have, I cannot remember a time when you would not have time for my questions. I would also like to thank Professor Paul Ampadu for inviting me to join his EdISon research group at the University of Rochester for summer 2008. I learned a lot during those months.

Adjunct Professor Elena Troubitsyna and PhD Gert Jervan are gratefully acknowledged for reviewing this thesis.

I would like to thank all my colleagues at the IT Department and at the EdISon research group for the enthusiastic atmosphere. I have had the pleasure to share a room with many exceptional individuals. I would like to thank you all for the scientific and also the non-scientific discussions we have had. Thank you for the good co-operation to everyone who have co-authored papers with me, especially David for writing together the paper also included in this thesis. Special thanks to Jarkko for the valuable comments regarding the coding related parts of the thesis and to Sami for the numerous times I have needed assistance with my computers.

I would like to thank my parents, Sirpa and Tauno, for all the help during different phases of my studies. And finally, my warmest thanks to my wife Sari. You pushed me to start the PhD studies in the first place, followed me to Rochester during my visiting period there, and kept me sane throughout these years by forcing me to every now and then also to put the research work to the background.

This work was financially supported by the Academy of Finland, the Varsinais-Suomi Regional Fund of the Finnish Cultural Foundation, the Finnish Foundation for Technology Promotion, the Ulla Tuominen Foundation, the Emil Aaltonen Foundation, the Nokia Foundation, and the Turku University Foundation.

Turku, October 2009
Teijo Lehtonen

List of original publications

I Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila. "Analysis of Forward Error Correction Methods for Nanoscale Networks-on-Chip", in *2nd International Conference on Nano-Networks, Nano-Net 2007*, Catania, Italy, September 2007, pages 1–5.

II Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila. "Online Reconfigurable Self-Timed Links for Fault Tolerant NoC", *VLSI Design*, vol. 2007, Article ID 94676, 2007, pages 1–13.

III Teijo Lehtonen, David Wolpert, Pasi Liljeberg, Juha Plosila, and Paul Ampadu. "Self-Adaptive System for Addressing Permanent Errors in On-Chip Interconnects", to appear in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

IV Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila. "Fault Tolerance Analysis of NoC Architectures", in *2007 IEEE International Symposium on Circuits and Systems, ISCAS 2007*, New Orleans, LA, May 2007, pages 361–364.

V Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila. "Fault Tolerant Distributed Algorithms for Mesh Networks-on-Chip", in *9th International Symposium on Signals, Circuits and Systems, ISSCS 2009*, Iasi, Romania, July 2009, pages 149–152.

VI Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila. "Analysis of Fault Tolerant Deadlock-free Routing Algorithms for Mesh NoCs", in *3rd Workshop on Diagnostic Services in Network-on-Chips, DSNOC '09*, Nice, France, April 2009, pages 54–57.

Contents

Chapter 1

Introduction

The ever continuing technology scaling further into the very-deep sub-micron or nanometer regime in CMOS chips results in increasing number of problems in the reliability of circuits. The shrinking geometries, smaller transistors, lower supply voltages, higher frequencies and denser integration, among others, cause faults in a chip. Although most of them can be detected at the manufacture test, larger and larger proportion of faults will not occur until at run-time. The only way to cope with run-time faults is to design the system to tolerate them. Furthermore, fault tolerance structures may provide better manufacturing yield when some faults can be allowed to exist in a chip.

Faults can be classified to permanent, intermittent and transient faults according to their duration [17]. Because there are different types of faults also the methods for tolerating them are different. In most cases, a single fault tolerance method is not an optimal solution for all types of faults. This gives rise to the idea of combining a set of fault tolerance methods which together can provide the desired fault tolerance. The different fault types are discussed further in Chapter 2 together with the typical fault sources in the modern technologies. A survey of available fault tolerance methods and classification for them is also presented.

Network-on-chip (NoC) is a design paradigm targeted for large systems-on-chip (SoC). The main principle in the NoC paradigm is to replace buses with a communication network somewhat similar to the Internet. The resources, processors, co-processors, memories, etc., are connected via network interfaces to the communication network created from point-to-point links and routers. The design of networks-on-chip concerns various aspects, such as the network topologies, routing protocols, and flow control methods.

When considering fault tolerance of a NoC, the first thing to do is to separate the communication infrastructure from the resources such as processing elements, memories, etc. According to the principles of the NoC

paradigm, the resources can be of a variety of types, which indicates that a variety of fault tolerance methods could be applied. Indeed, a fault tolerance method most feasible for a particular resource type should be selected. This work focuses on the fault tolerance methods for the communication infrastructure, although some methods presented in the survey of Chapter 2 can also be applied for the resources in a NoC.

The NoC communication infrastructure is commonly described as a pile of abstraction layers following the well known OSI reference model [91]. Therefore, it is logical to view also the fault tolerance methods at these layers. The fault tolerance methods presented in this thesis fall mainly to the data link and network layers, the second and third lowest layers of the OSI reference model. Above the mentioned layers are issues such as the software mapping to the underlying hardware platform, which are out of the scope of this thesis. On the other hand, below these layers is the physical layer, which consist of implementation-specific details such as the signaling method in the links and the selection between synchronous and asynchronous design style. In asynchronous design the clock signal, which schedules the system operation, is replaced with handshake signals between components. The use of an asynchronous design style makes it possible to get rid of the many clock-related problems present in many large synchronous systems. It has a positive impact on the robustness and reliability of systems.

The data link layer provides reliable transfer over a physical link. The error protection of communication channels is commonly achieved by the utilization of error control coding (ECC). Only a subset of the available ECC can be used in the error protection of on-chip signaling because of the strict area, latency and power-efficiency requirements typically set to the implementations of them. ECC has its strengths in tolerating transient faults. As already stated above, most of the fault tolerance methods are not at their best against all types of faults. The same stands for ECC, its effectiveness against permanent and intermittent faults is not that good as it is against transient faults. Spare wires and split transmissions are two methods proposed for the tolerance against intermittent and permanent faults. These methods are combined with ECC to achieve a system that tolerates all types of faults.

The links also include the transmitter and receiver circuitry, which in the presence of an ECC also consist of an encoder and decoder. The reliability of these circuits should be taken into consideration as it affects the reliability of the overall link system. In the papers included in this thesis the focus has been on the link structures and fault tolerance of the transmitter and receiver circuitry has been omitted. However, the survey of fault tolerance methods presented in Chapter 2 contains methods that could be used for this purpose.

Fault tolerance methods at the network layer include the selection of and possible modifications to a network topology so that it contains redundant routes, which increases the overall reliability. The procedure includes the identification of vulnerable parts in the topology and introduction of redundant links and components into such parts. Following the principle the overall reliability of the topology is increased.

Another network layer fault tolerance method is the development of fault tolerant routing algorithms. The fault tolerance a routing algorithm provides is either originated from redundant packets travelling in the network, i.e. multiple copies of each packet is transmitted, or from the adaptive selection of redundant routes in the topology. The design of fault tolerant algorithms includes important design choices such as the property to provide always the shortest route or resiliency against deadlocks, among others.

1.1 Organization

The thesis is divided into two parts. The first part is an introduction to the topics related to this research field, and the second part contains reprints of six published research papers related to the topic. The introductory part gives a motivation for the work and presents related work. It also explains the research field at a wider perspective which makes it easier to understand the design choices made in the works presented in the papers included in the second part of the thesis.

The introduction chapter has given an overview of the thesis. The rest of the thesis is organized as follows. Chapter 2 classifies the faults and discusses the most common sources for them in the modern technologies. It further provides a survey of fault tolerance methods. In Chapter 3 the concept of Network-on-Chip is introduced and its main design choices are addressed. At the end of the chapter an overview of fault tolerance in a NoC is given. Chapter 4 discusses the utilization of error control coding for the error protection of on-chip signaling. This is followed by an introduction of methods against permanent faults in Chapter 5. The network-level fault tolerance methods, topology and routing algorithms, are addressed in Chapter 6. A summary of the papers included in the second part is found in Chapter 7, which is followed by concluding remarks and discussion of future work in Chapter 8. The second part containing the included papers follows after the bibliography.

Paper I contains an analysis of error control codes suitable for on-chip interconnects. In Paper II error control codes are used for tolerating transient faults while spare wires and split transmissions are introduced for tackling intermittent and permanent faults. The paper introduces a self-timed reconfigurable link system and a permanent error detection method based on

evaluating consecutive error syndromes. The reconfigurable framework is further developed in Paper III, which enables the reconfiguration to proceed without stopping the normal link operation. The paper also introduces another method for detecting permanent errors, in which rotating tests are run for each wire in the link. Furthermore, an enhanced detection method based on evaluating consecutive syndromes is introduced.

Paper IV evaluates the reliability of a NoC at the topology level. The paper introduces additional network interfaces for each resource and presents modifications to the network structure. A number of approaches are analyzed in the paper. Papers V and VI introduce distributed fault tolerant routing algorithms and a comparison of them in the classes of deadlock-free, minimal and maximal fault tolerance.

1.2 Contributions

The main contributions of this thesis are:

1. Analysis of error control codes suitable for on-chip links in Paper I.
2. Combination of error control coding for tolerating transient faults and spare wires and split transmissions used to tolerate intermittent and permanent faults in Paper II.
3. Methods for detecting permanent faults in on-chip interconnects and a reconfigurable framework to be used with spare wires and split transmission in Papers II and III.
4. Introduction of additional network interfaces per resource in a NoC and modifications to the network to increase the overall reliability in Paper IV.
5. Distributed fault tolerant routing algorithms and an analysis of them in Papers V and VI.

Chapter 2

Faults and Fault Tolerance

A *fault* is a physical defect, imperfection, or flaw that occurs within some hardware or software component. A fault may or may not cause an *error*, deviation of accuracy or correctness. A *failure* occurs if the system performs one of its functions incorrectly due to an error. [39]

In this chapter the fault types are classified and the most common fault sources are discussed and connected to different fault types. The fault tolerance methods are divided into static and dynamic methods addressed at the end of the chapter.

The chapter provides a thorough view to the available fault tolerance methods. From this set of methods the suitable ones for network on chip realizations can be selected as will become evident in the later chapters of this thesis.

2.1 Fault Types

Faults can be divided into three main groups: *permanent*, *intermittent* and *transient faults* according to their duration and occurrence. In the following these main groups are briefly defined. [17]

2.1.1 Permanent Faults

Permanent faults are irreversible physical changes in a chip. The most common source for this kind of faults is the manufacturing process, but permanent faults also occur during the operation of the circuit, especially when the circuit is old and starts to wear out. Common to all permanent faults is that once they have occurred they will not vanish, and therefore the test to detect them can be easily repeated with the same results.

Manufacture testing is used to detect the permanent faults caused by the manufacturing processes and dismiss the circuits containing such faults. If a permanent fault emerges during the operation of the chip which has no fault

tolerance properties, the faulty circuit needs to be replaced. Fault tolerance methods can also provide a mean to achieve higher yield by accepting chips with some faults which are then masked by the fault tolerance properties.

2.1.2 Intermittent Faults

Intermittent faults are occasional fault bursts that usually repeat themselves every now and then but are not continuous as permanent faults, even though they may last for several clock cycles. The faults are caused by unstable hardware which can be operable under some environmental conditions, but an alteration in those conditions, such as temperature or voltage change, violates the operation. Intermittent faults often precede the occurrence of a permanent fault, for instance there may be an increased resistance in a wire before it totally breaks down creating an open circuit. These types of faults are commonly observed when a system operates most of the time correctly, but for some input instance it fails. The reason for this is that some path in a circuit may be slower than supposed to but not completely broken.

Intermittent faults are very hard to detect because they may occur only under certain environmental constraints or for some specific input combination. The way to repair these faults is to replace the faulty circuit or to bypass the faulty part of the circuit.

2.1.3 Transient Faults

Transient faults are momentary single malfunctions caused by some temporary environmental conditions which can be external phenomena such as radiation, or noise originating from the other parts of the chip. Transient faults do not make any permanent damage on the chip and therefore they are also called *soft errors*. A common impact of a transient fault is a change of value in a single bit. Another term *single-event upset* is used for a soft error, which describes the fact that malfunctions (upsets) are commonly caused by single events such as absorbed radiation.

The occurrence of transient faults is commonly random and therefore difficult to detect. Because of the random nature of these faults, a common measure for transient faults is the *soft error rate (SER)*, the probability of error occurrence. This rate describes both the tolerance of the circuit against variable phenomena causing soft errors and the amount of these in the environment where the circuit is operating. For example, the SER is much higher in space, because of the larger amount of background radiation than for the same chip operating in terrestrial conditions. The SER can be decreased e.g. by giving special concern to low-noise properties during the circuit design.

Transient faults are the most common fault type to cause system failures in nanoscale circuits; up to 80 % of all the system failures are associated with transient faults [6].

2.2 Fault Sources

Fault sources can be classified according to the phenomenon causing the fault. Such origins are for instance: *manufacturing process*, *physical changes during operation*, *internal noise* caused by other parts of the circuit and *external noise* originating from the chip's environment.

2.2.1 Manufacturing Process

Common defects in a chip are *spot defects* and *bridging faults* caused by silicon impurities, lithography variations and process deviations. These defects cause permanent faults in a circuit. The probability of these defects is likely to increase as a greater amount of transistors will be integrated in a single chip, and at the same time the devices and wires get smaller.

The move towards nanoscale circuits introduces also a set of new problems originating from the manufacturing process. As the dimensions shrink the relative extent of deviations becomes larger and their effects more severe. Lithography deviation is the main reason for gate length variations [26]. Doping profile fluctuations on the other hand cause changes in the threshold voltage [26, 84]. These together with an increasing number of resistive vias and contacts result in a large operation speed deviation [33]. At the same time the operation frequencies of the circuits are expected to increase rapidly. In the worst case scenario, series of "slow" devices may lead to timing violations and therefore to a malfunction of the circuit. This is considered an intermittent fault because the circuit might work correctly for most of the time, which would not be the case with a permanent fault.

Metal slivers are small pieces of metal between two metal wires demonstrated in Figure 2.1. In normal conditions this metal piece does not touch the wires but when the temperature increases it could create a short between wires due to the metal expansion. This is a typical intermittent fault but a high voltage may also cause the sliver to burn in and hence, lead to a permanent fault. [33]

The opposite of slivers are *cracks* in metal wires illustrated as well in Figure 2.1. They cause open circuits at low temperatures but do not alter the wire functionality at all or occur as an increased resistance at normal temperatures. Cracks are another example of intermittent fault sources.

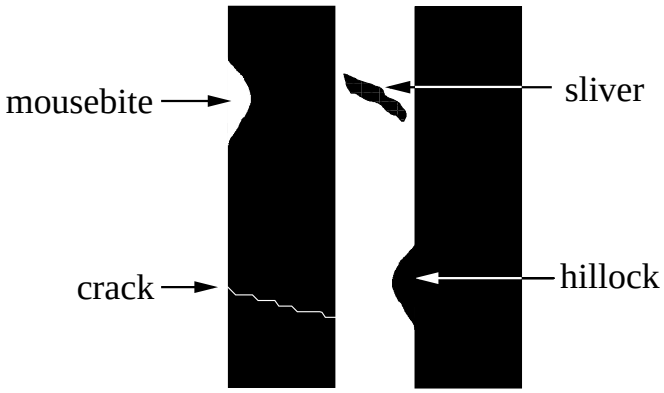


Figure 2.1: Manufacturing faults in wires.

2.2.2 Physical Changes During Operation

Electromigration means current-induced atom-level transport which is generated by collisions of electrons with metal atoms [65]. This phenomenon is especially critical if there are *mousebites* or *hillocks* in the metal wires, both of which are illustrated in Figure 2.1 [5]. In the place of a mousebite the wire is narrower, which means that the current density is higher and so the electromigration effect is stronger. Additionally, the wire is already narrower in that place so the impact of electromigration causes rapidly increasing resistivity and finally an open. In the place of a hillock material is accumulated and electromigration moves more material to such a place because now the current density is lower than in other parts of the wire. This can eventually result in a short to another wire. The electromigration impact is observed often first as an intermittent error, and later as a permanent error if an open or short circuit is formed.

Electromigration is becoming a more and more severe problem as the dimensions of wires and insulators between wires decrease at the same time as their relative deviations increase. Also the increasing operation temperature strengthens the effect of electromigration. One improvement that has reduced the electromigration problems has been the replacement of aluminium with copper in wires due to copper's higher electromigration threshold than that of aluminium [84].

The ever thinner gate oxides are prone to current tunneling resulting in a breakdown which means a permanent fault. On the other hand, a *soft breakdown (SBD)* in ultrathin gate oxides slows down transistors causing intermittent faults in the same way as other device deviations [33]. The soft breakdown differs from the traditional *hard breakdown* in that it causes current fluctuations while the hard breakdown brings about shorts and thus a measurable current. The SBD effects can be regarded as tiny breakdowns

that only partially violate the operation of the transistor. The SBD is also a typical source for an increased power consumption.

2.2.3 Internal Noise

When technology is scaled down also the supply voltages are lowered. In addition to the lower power consumption the benefit of the voltage scaling is the better durability of the components and wires. High voltages expose thin gate oxides for breakdown and high current densities in narrow wires accelerate electromigration. The drawback is a degraded noise tolerance, which is illustrated in Figure 2.2. The impact is further amplified by the large variation of the threshold voltage, which means that for some circuits or some parts of the circuit the noise margin is negligible. At the same time the amount of noise is increasing as the relative fluctuations in the manufacturing processes get larger.

The impact of crosstalk noise between signal lines is increasing because the height and width of the wires are not scaled by the same factor. The width of the smallest wires scales with the same factor as the length of the gate, but the technology scaling factor for the height of the wires is smaller

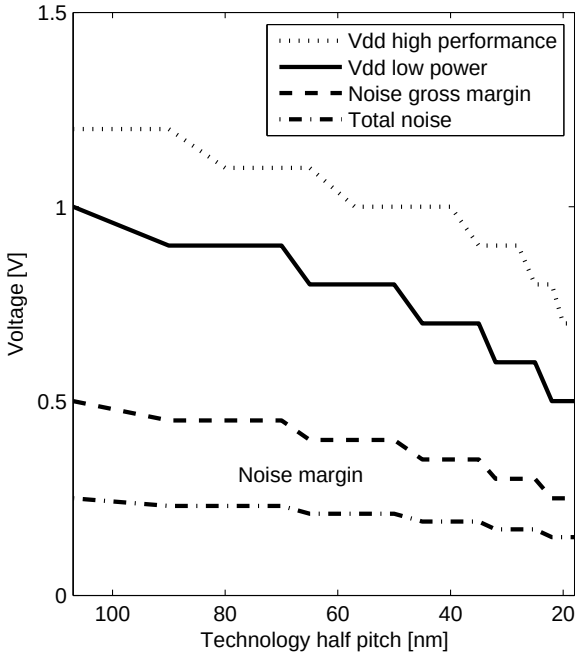


Figure 2.2: The noise margin decreases as the technology is scaled down [38]. The gross noise margin is calculated from the low power V_{dd} and the total noise with independent noise of 50 mV and relative noise of 0.2 [19].

in order to keep the resistance of the wires tolerable. This increases the capacitive area between adjacent wires. At the same time, the wire spacing and also the distance of adjacent layers get smaller, which additionally increases the capacitive coupling.

The higher frequency in nanoscale circuits gives rise also to inductance based noise called the *skin effect* [19]. When the current changes rapidly it flows near the wire surface, which means an increase in the resistance. Since the current flow proximity to surface depends on the frequency of current changes, also the wire resistance will vary with the frequency.

The timing uncertainty is another type of noise source. As the relative deviations of components and wires increase it will be impossible to get a signal to two or more distinct nodes in the circuit exactly at the same moment. The resistive vias and contacts worsen the situation even further. The increasing skew and jitter is a problem not only in synchronous systems but also in asynchronous designs that use delay elements, because the delay value cannot be set exactly. [19]

The impact of noise can usually be modeled as a transient or sometimes as an intermittent fault.

2.2.4 External Noise

Radiation has not been regarded as a severe noise source unless the circuit is to be used in space, aeroplanes, nuclear plant or similar places where the background radiation is higher than usual terrestrial amounts. As technology scales down the radiation should be taken into consideration also in other circuits, because the shrinking dimensions increase the probability that an α-particle, proton or neutron hitting the chip also causes a change in a bit value. The occurrence of an upset is more likely since a lower supply voltage together with smaller transistors mean that the charge the particle introduces is sufficient to flip a bit. The charge needed to flip a bit and cause a particle-induced transient is called the *critical charge* and it is dependent on the charge a transistor can hold. For instance for a 90 nm CMOS technology the charge a transistor holds is 1–10 fC, which is less than one tenth of the charge of over 100 fC delivered by an α-particle hitting the circuit [40].

Other external noise sources are *electromagnetic interference* and *electrostatic discharge*. A common source for the former are other devices especially those emitting high energy signals, and for the latter the users releasing static charge accumulated in their clothing. The faults are normally transient but in principle also permanent faults may result especially from an electrostatic discharge [17].

2.3 Fault Tolerance Metrics

Fault tolerance of a system can be described in many metrics. The most common of these is the *reliability* $R(t)$ which has been defined as the probability that a system operates correctly under given set of operating conditions over a given period of time $[t_0, t]$, given that the system was performing correctly at time t_0. Other metrics include the *availability* $A(t)$ which is the probability that a system can perform its task correctly at the instant of time t, and *safety* $S(t)$ which is the probability that a system performs its task correctly or discontinues its operation in a safe manner. Safety is defined over a period of time $[t_0, t]$ corresponding with the definition of reliability. *Dependability* is used to indicate all of these properties. [39, 81]

The occurrence of failures can be described with the *failure rate* λ, which is the expected number of failures per a given time period. Its reciprocal is the *mean time to failure (MTTF)*, $\text{MTTF} = \frac{1}{\lambda}$. The relation of the reliability and failure rate is $R(t) = e^{-\lambda t}$ for a constant λ. [39]

2.4 Static Fault Tolerance

A circuit is said to be utilizing static fault tolerance when it is built in such a way that a fault somewhere in the circuit will not violate the correct operation of the circuit. The word static stands for the fact that fault tolerance is built into the system structure and it effectively masks the fault effects. The method to create such fault masking properties is to use some kind of redundancy. Static fault tolerance can be categorized to hardware, information and time redundandy according to the resource that is used to create the redundancy. Also a combination of these can be used.

2.4.1 Hardware Redundancy

Hardware redundancy generally means making copies of the processing module and providing a voting circuit to decide the correct output value based on the outputs of the module and its copies. A higher reliability is gained because when a component fails, the voter can decide the correct output based on the results of the working copies. The method can be used at many different abstraction levels, the modules can be as simple as single gates but also as complex as whole processors or even larger constructs. The voter can be a simple bitwise hardware implementation or a software algorithm running on a processor. This versatility makes the method a good candidate for large systems such as networks-on-chip, where solutions for multiple abstraction levels is required. Common to all hardware redundancy realizations is the need for additional area. Therefore the methodology is also called physical, area or space redundancy.

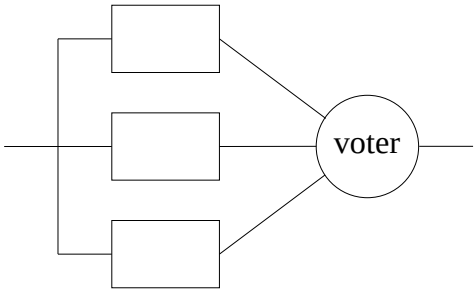


Figure 2.3: Triple modular redundancy.

The most common hardware redundancy realization is *triple modular redundancy (TMR)* which consists of three redundant modules and a voting circuit as illustrated in Figure 2.3. The voter normally performs majority voting, which means that the output is the same as the output of two-out-of-three of the modules. TMR is capable of masking a single error in the processing modules. The weak point of the circuit is the voting circuitry itself where an error can cause the whole system to fail. This has been tackled by making also three copies of the voter and connecting the module outputs to each of the three voters [39]. Other possibility is to modify the voting circuitry in such a way that possible errors can be detected. Approaches include e.g. a voter that is on-line self-testing for internal faults [14, 58] and a voter that can be checked based on the quiescent current (I_{DDQ}) [13]. Another important issue to consider is the mismatch and crosstalk in lines from module outputs to voter inputs, which can cause severe malfunctions [25]. For this reason synchronization between the voter inputs may be required. The simplest method for realizing synchronization is to insert registers to voter inputs [39].

A more generalized hardware redundancy realization is *n-modular redundancy (NMR)*, which means that there are n copies of a module and a voter. This structure is capable of masking $\lfloor (n-1)/2 \rfloor$ errors in different processing modules. The most common structures besides TMR are 5- and 7-modular redundancies capable of masking 2 and 3 errors, respectively.

The voting algorithms can be divided according to their functionality to *generic* and *hybrid voting algorithms* and to *purpose-built voters*. Generic voters use only the information of input signals to produce the output while hybrid voters have also some additional information such as the reliability of different modules or history of previous votes. The purpose-built voters are e.g. special microprocessor systems designed for space shuttles [48].

Generic voters produce the output according to the present output values of the modules. The most common algorithm is the *exact majority*

voting, which means that when the majority of the module outputs have the same value, this value is forwarded to the output. This is easily achieved in bitwise voting because the only possible values are logic '0' and '1'. If the module outputs are not just one bit wide but for instance integers, then it is possible that there is no majority agreement. In this case the voter can have an output "no result", which is an exception signal. The values of different modules can be slightly different because of noise or e.g. sensor elements cannot be physically at the exactly same place. Therefore *inexact majority voting* has been introduced, where the output is decided if the majority of the module outputs lie inside a certain threshold. A "no result" is output only if majority of the outputs are more than a threshold apart from each other. The threshold effect can be easily achieved by dropping a couple of least significant bits from the voting procedure [39]. In *plurality voting* the majority of the module outputs do not necessarily have the same value (exact) or values in threshold limits (inexact), only the number of the same value or values within a threshold is larger than the number of modules having some other value. For instance, if there are five modules, it is enough that two of them have the same value if each of the other three has a different value. In inexact voting the selection of the output value can be a random selection of one of the majority or plurality output values or it can be *mid-value selection*, where the output is calculated as the mid-value of the majority or plurality outputs. [41, 80]

Another voting scheme is *median voting*, where the median of all the module outputs is selected as the voter output. An efficient software realization is to sort the output values and then select $[(n+1)/2]$th value as the output, where n is the (odd) number of redundant modules. [45]

In *weighted average voting* a weight is given to each module output and the output is calculated as the average of the module outputs scaled by these weights. The output is scaled down by the sum of the weights in order to produce an output that is at the same scale as the inputs. In bitwise voting the output is returned to one of the logic states according to a threshold value which is adjusted to some level between the logic states.

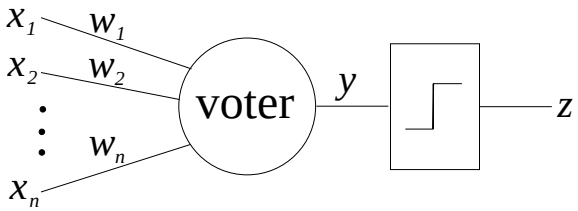


Figure 2.4: Weighted average voting with threshold: $y = \sum_{i=1}^{n} w_i x_i / \sum_{i=1}^{n} w_i$, z ='1' if $y \geq T$ and z ='0' if $y < T$, where T is the set threshold.

The operation is demonstrated in Figure 2.4. The weights are calculated based on the distance of the output value to the other output values. If a module's output value is far away from all the other output values, it is given a smaller weight than a module whose output is close to many other module's output values. Advanced methods to calculate these values have also been presented. These include e.g. a voter that uses the concept of the soft threshold for determining the closeness of all voter input pairs and a tunable roll-off parameter which gives the possibility to adjust the voter behaviour from a majority voter to an average voter [51], and a fuzzy voter using fuzzy set theory to adjust the weights [50]. Circuit realizations of voters include e.g. weighted bit-wise voters with a threshold [15, 64] and an analog weighted average voter [74] together with a threshold circuit using capacitive threshold logic [73].

The adjustment of the threshold is an essential task for the operation of the circuit. Threshold can be static or dynamic. A static threshold can be based on circuit realization or set after the manufacturing. Dynamic threshold, in contrast, adapts to the changes in the operation environment. For instance, the use of an artificial neural network learning algorithm in adjusting the thresholds has been suggested [72].

The selection of a voting method for a particular application depends on the required properties. In some applications even a single incorrect data bit may violate the operation of the whole system whereas for some other application few corrupted data bits make no harm. For the former ones high safety is demanded, which means that it is better to indicate even the small possibility of an incorrect result even though it may decrease the total number of correct answers. For the latter case the number of correct outputs is maximized without paying any concern to the potentially incorrect results.

Majority voting is a rather safe voting scheme because it is more likely to output a "no result" than a faulty result. The weighted average voting on the contrary gives more correct results but also the amount of incorrect results increases, thus the safety is not that good [49]. Above mentioned is true when module outputs are wider than just one bit. However, for bitwise voting the safety of a majority voter is poor because in the case of multiple errors the output is likely to be incorrect instead of "no result". The weighted average voting for bitwise voters is shown to result in a higher number of correct results in the presence of multiple defects [75].

Hybrid voters combine the information on present module outputs and some other information regarding the module circuits or output sequence. An example of the use of history data together with the information on present outputs is weighted average voting, where the weights are based on history records [47]. The voting procedure can be also totally based on the history, e.g. by choosing for the output of the voter the output of the module

that has been the best in the history. The best module is the one that has had output closer than a threshold to majority of the module outputs for the most times [46]. The use of history values can be also a backup system if no agreement can be found among the module outputs [46]. In addition to using the history data to detect the most reliable modules, it can also be used to predict the next output value. Many system outputs are somehow dependent on the previous outputs, which gives the justification for this procedure. One of the simplest ways to predict the value in the case of a module output disagreement, is to check if any of them is closer than a threshold to the previous voter output. If such an output is found, the one closest to the previous value is selected [49].

2.4.2 Time Redundancy

The principle in time redundancy is to use the same resource many times and compare the results obtained from different rounds of computation. The method saves area since no copies of the processing module are required. The drawback is the longer processing time because of the recalculations, which on the other hand may be acceptable for certain types of applications. The method of repeating the same calculation many times is effective in detecting transient errors but permanent and in many cases also intermittent errors occur at the same place during all calculations and cannot therefore be detected and corrected. This problem can be overcome by encoding the operands before processing and decoding afterwards. Commonly the operation is first performed with the original operand and after that with the coded ones.

In *alternating logic* method complementation is the used coding method. In order to be able to use this coding, the self-duality of the circuit is required or possibly an additional input is needed. *Recomputing with shifted operands (RESO)* means shifting the operands before calculation to left and back to right after calculation. This method makes arithmetic operations wider. Alternatively, cyclic shift can be used which in turn means complex logic for carry signals in adder circuits. One more coding approach is *recomputing with swapped operands (RESWO)*, where upper and lower parts of the operands are swapped before and after the calculation. The method needs no additional bits, and the logic for handling carry bits in adder circuits is more straightforward than in RESO. [39]

The error correcting properties are obtained by repeating the operation at least three times and performing voting for the three results. Different coding is used at different calculation rounds, e.g. no coding, shift 1 and shift 2. Bitwise majority voting can be problematic because the arithmetic operations commonly affect many bits. The different voting approaches were already discussed in the previous section. [39]

2.4.3 Information Redundancy

Information redundancy in general means appending additional bits to stored or transmitted data. *Error control codes (ECC)* can be used to detect and/or correct errors in a *codeword*, the data word into which check bits have been appended. The application of an ECC to extract the exact location of the error in order to correct it is called *forward error correction (FEC)*, to distinguish it from approaches where a code is used only for detection of an error in a codeword. The codes can be classified to *separable* and *non-separable codes* according to how the additional information is appended. If the data is left unchanged and check bits are appended for the correction, the code is separable. A non-separable code needs a decoding circuit to return the data to its normal form before further processing. Separable codes are also called *systematic codes* and non-separable *non-systematic*, respectively.

A simple separable ECC is the *parity code*, which means appending one bit to every word. The value of this appended bit is adjusted to make the number of bits with value '1' odd (odd-parity code) or even (even-parity code). The parity code itself can be used only for detecting single errors in a codeword, but when a parity is calculated separately for both rows and columns for instance in a memory block, the exact location of a single error can be extracted. [39, 44]

Error control codes are an important tool in designing fault tolerant signaling also in on-chip communication as well as for other communication. This usage of ECC will be discussed in detail in Chapter 4 and therefore it is omitted here.

The *redundant residue number system (RRNS)* is used for error correction in arithmetic operations such as addition, subtraction and multiplication. In residue number system each number is presented as the residues for a set of relatively prime moduli. For instance, if the moduli set is {3,5} then a 9_{10} is presented as 04_{RNS} because $9 \equiv 0 \pmod 3$ and $9 \equiv 4 \pmod 5$. The set {3,5} can be used to present numbers $0_{10} - 14_{10}$, which is determined by multiplying the moduli of the set ($3 \cdot 5 = 15$). The operations are performed bitwise as mod m_i, where m_i is the moduli used to create the numbers at this bit location. When additional moduli are inserted to the set, a redundant residue number system is obtained, which can be used to detect and correct errors. If r redundant moduli are added, the system can correct up to $\lfloor r/2 \rfloor$ errors. The use of RRNS has been proposed to be used e.g. in digital filters and in software-defined radio. [24, 35, 39]

2.5 Dynamic Fault Tolerance

Dynamic fault tolerance is based on active operations as opposite to the passive nature of static fault tolerance. It can be divided into four phases

the first of which is *fault detection*. After the detection of an erroneous behaviour the next thing to do is to *locate the fault*. The third phase is *fault containment* which means isolating the error source so that no new errors can occur. The final phase is *fault recovery* meaning usually reconfiguration of the system so that the erroneous part has been disabled. [39]

Dynamic fault tolerance typically requires special control circuitry and elements. Unfortunately, the design of these control parts is not always straightforward. The benefit of dynamic fault tolerance is better reliability especially in the occurrence of permanent and multiple errors, and quite often the reliability increase is obtained with a smaller area overhead than in the corresponding static fault tolerance approach. Many network-on-chip implementations contain means for controlling traffic flows and processing resources. The control required for dynamic fault tolerance could be integrated with such systems, which makes dynamic fault tolerance a promising method for networks-on-chip.

2.5.1 Fault Detection

The fault detection is an essential part of dynamic fault tolerance. If the fault is not detected the circuit can produce erroneous outputs and it can falsely be considered fault-free. The fault detection can be organized as *periodic tests*, *self-checking circuits* or *watchdog timers* [44].

The principle of the periodic tests is to stop the circuit operation every now and then and perform a self-test. The method cannot guarantee that every fault is detected because the test is run only occasionally. Furthermore, the time needed for testing is a drawback of this method.

Watchdog timers are used especially in multi-processor environments. A control circuit sets a timer when a processor starts to execute a certain task. At some predefined point of the procedure the processor resets the timer. If for some reason the processor halts during the operation, the control circuit detects it by observing the timer value exceeding some limit. To recover from this situation the control circuit can for instance reset the halted processor or start a reconfiguration process. [39]

There are numerous ways to create self-checking circuits. The most straightforward method is *duplication with comparison*, which means creating two identical modules and comparing their outputs as illustrated in Figure 2.5. Sometimes there can be small deviations between the two components although they are operating correctly. In these cases the comparison process can ignore least significant bits. The method can also be extended to tolerate common mode failures which are failures that affect both the modules in the same way. One such an enhancement is to replace the duplicate module with a module that calculates the complement function of the original module's function. The circuit works correctly when the output of the

original module and the output of the module calculating the complement function are complements of each other. [39]

Duplication can be done also in time domain. The information is sent twice and the results are compared. In order to gain better detection abilities, the data can be complemented or swapped at the second iteration and an inverse transformation is performed at the output. [39]

A number of error control codes can be used to detect errors in data. Possibly the most common error detecting code is the parity code which is extensively used e.g. in memory circuits. The code can detect single errors, but in the presence of multiple errors the parity bit may have a correct value, and therefore the error is not detected. More sophisticated error control codes are presented in Chapter 4, where coding for on-chip signaling is addressed in more detail.

Errors are commonly expected to occur randomly, independently of each other. This is not a realistic model for all cases. In some situations errors only in one direction can occur, meaning that '0' can turn to '1' but '1' cannot flip to '0' or vice versa. Errors occurring only in one direction are called *unidirectional* and errors that can occur in both directions are *bidirectional*. Errors can also occur as bursts which are common for instance in communication signaling, where burst errors originate from disturbances in the signaling medium. Intermittent faults in the signaling medium typically cause error bursts.

M-out-of-n codes are able to detect not only single errors but also multiple unidirectional errors. The basic principle of the code is that every codeword has m ones and $n - m$ zeros while the length of a codeword is n. The code is not separable except in some special cases. Such a special case is *k-out-of-2k code* where there are two bits per one data bit. The code is called *k-pair two-rail code*, when the check bits are bitwise complements of the information bits [39, 44]. An example of the use of m-out-of-n codes is an online error detecting adder, where 1-out-of-3 code has been used [82].

The *Berger code* requires the fewest number of checkbits of the available separable codes for detecting arbitrary number of unidirectional errors. The

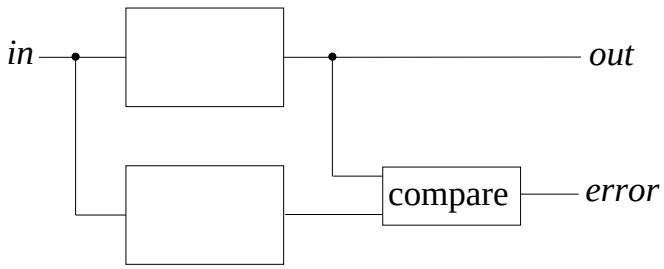


Figure 2.5: Duplication with comparison.

code is based on counting the number of '1's in the word and appending a complement of the count to the word. The number of checkbits is $\lceil \log_2(k+1) \rceil$, where k is the number of information bits [39, 44]. If it is not necessary to detect all the possible unidirectional errors, but only maximum t of them, then the amount of checkbits can be further decreased. Such codes are e.g. the *modified Berger code*, *Borden code* and *Bose-Lin codes* [44]. A set of codes for detecting unidirectional errors and error bursts of length t, the *unidirectional burst error detecting codes*, includes e.g. codes by Berger, Bose, and Blaum [44].

One way to detect an error occurred during signal transmission is to calculate a *checksum* of the data words and transmit it together with the data words. At the receiving end the checksum is recalculated and compared with the received one. Examples of checksums are *single- and double precision checksums* as well as *Honeywell* and *residue checksums*. The length of the single-precision and residue checksum is the same as the width of the data words, while the other two have the length of double the width of the data words. The mentioned checksums are calculated by summing up all the data words. They differ in the way the data words are organized and carry bits handled in the addition process [39].

Arithmetic codes are used to detect errors in arithmetic operations. One example of such a code is the *AN code* which can be used for addition and subtraction. Every operand is multiplied by A before the operation and the result should be evenly dividable by A, otherwise an error has occured. A commonly used AN code is the $3N$ code. Other arithmetic codes include *residue* and *inverse-residue codes*. In these codes, the operands are divided by integer m (e.g. by 3 for mod-3 residue), and the remainder is appended to the data, thus the code is separable. The arithmetic operation for the appended remainder is computed modulo m and the result is checked against the remainder of the operation result divided by m. The procedure is eligible for addition, multiplication and logical units [44]. Residue number systems, described in Section 2.4.3 along with their error correcting capabilities, can also be used for error detection. [39]

Checkers are circuits that are used to determine if an error has occured or not. A checker commonly compares two values and signals an error if they differ or are equal, depending on the used method. The robust design of checkers is very important because an error in the checker circuit may invalidate the fault tolerance of the whole system.

In the design of checker circuits the concepts of *fault-secure* and *self-testing* design are commonly used. The circuit is fault-secure if every valid input produces either a correct output or an output which can be easily observed to be incorrect. This kind of incorrect outputs are called *non-code* outputs, which indicates that they do not fulfill the requirements set

for the codewords. A self-testing circuit, in turn, is such where for every fault in some set of faults there is an input combination which produces a non-code output so that the fault can be observed. The circuit that is both fault-secure and self-testing is called *totally self-checking*. [44]

The fault-security in a circuit can be achieved by designing the circuit in a way that there is separate logic for separate outputs. This ensures that a single error affects only one output signal. The self-testability is achieved by non-redundant design.

A common checker is a two-rail checker, which has four inputs and two outputs. The inputs consist of two input pairs, both of which consist of two complementary signals. The circuit checks that the signals of an input pair are indeed complementary. The output signals are complementary when there are no errors, and in the presence of an error in the input or in the checker circuit the output signals are the same. A checker for a larger number of input pairs can be constructed by forming a tree connection from the two-rail checkers. Totally self-checking two-rail checkers have been presented, and a combination of these checkers forming a larger checker is also totally self-checking. [44]

2.5.2 Fault Location

After the detection of a fault situation the fault source has to be located, if it is not already known based on fault detection information. One way to accomplish this is to start a specific self-diagnostic procedure after the fault detection [39]. The self-diagnostics may consume too much time to be used in some time-critical applications, but for most applications the additional time is tolerable since it is needed reasonably seldom, i.e. only during a fault occurrence. Another way to locate a fault is to use two different detection methods. For example, if both duplication with comparison and parity check are used, the faulty module can be easily located. [3]

2.5.3 Fault Recovery

There are two common ways to accomplish fault recovery. The transmission or calculation repetition, *automatic repeat query (ARQ)*, needs additional time while the use of *spare modules* and *reconfiguration* causes area overhead.

ARQ is suitable for recovering from transient and in some cases also from intermittent errors. However, it cannot be used against permanent errors since the system will not work regardless of how many times the operation is repeated. Different approaches to ARQ are introduced at the end of Chapter 4.

The use of spare modules and reconfiguration is especially suitable for recovering from permanent and intermittent errors. On the other hand, it

is not very efficient to discard a whole module and replace it with a spare one if the error is only temporary. Therefore a combination of ARQ and reconfiguration might give the best result. The operation is first repeated and if the error remains, the reconfiguration process is commenced. [44]

The spare modules can be divided into *hot and cold spares*, depending on whether the spare modules are immediately ready for use (hot) or do they need to be initialized before usage (cold). Cold spares are also called *standby spares* and the use of hot spares is referred to as *reconfigurable duplication*. The standby spare system leads to higher reliability but the reconfigurable duplication provides higher safety. [39, 44]

Spares have also been combined with N-modular redundancy (NMR). In these systems N modules are part of the voting procedure and in addition to them, there are spare modules as illustrated in Figure 2.6. After voting the output is compared to the output of each individual module and in the case of a disagreement, the module is replaced with a spare. The combination leads to a higher reliablity than the normal NMR system. [39, 44]

A *self-purging system* is an NMR system which in the presence of a module failure reconfigures itself and the result is an (N-1)MR system. The isolation of modules is done similarly as in NMR with spares or it can be based on evaluation of the distances between different module output values. The method is also called *shift-out modular redundancy*. In conjunction with self-purging systems the use of weighted average voting with a dynamically adjustable threshold has been suggested. [15, 39, 64]

Yet another combination of dynamic redundancy and static NMR is the *triple-duplex architecture* presented in Figure 2.7. In this combination every module is duplicated and in the case of a disagreement the module is isolated

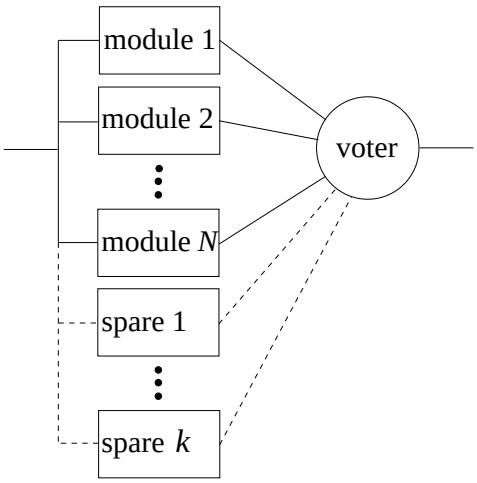


Figure 2.6: N-modular redundancy with k spares.

from the voting unit. One system therefore consists of six modules and a voter unit. [39]

The insertion of spare modules to a system causes easily a large area overhead, because for every kind of module a special spare has to be implemented. If the modules are similar, but not identical, *heterogeneous redundancy* can be used. Redundant blocks that can be programmed like field-programmable gate arrays (FPGA) to execute many different functions are inserted to a circuit. One such reconfigurable module can be used to replace different types of modules and therefore the overall number of spare modules can be decreased. The disadvantage is that these programmable modules are generally slower than fixed-logic blocks and also the reconfiguration takes more time. [43]

Another way for relaxing the need for additional area, is to take advantage of redundant structures inherently present in a system. For instance, there are parallel structures in many radio structures, where a single redundant module can act as a spare module for many modules in use [52].

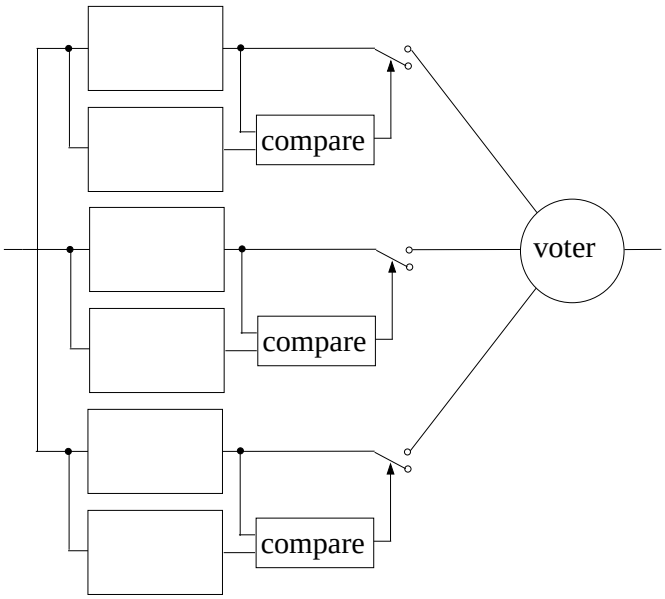


Figure 2.7: Triple-duplex architecture.

Chapter 3

Network-on-Chip

Network-on-Chip (NoC) is a communication infrastructure paradigm targeted to large on-chip systems consisting of tens or hundreds of resources such as processor cores, memories, etc. The idea in NoC is to replace buses with a communication network consisting of *routers* or *switches* and point-to-point *links* connecting the resources to routers as well as the routers to each other to form a network. The generic structure of a NoC is illustrated in Figure 3.1. NoC realizations borrow ideas from both circuit switching networks such as the landline phones as well as packet switching networks, the most known example of which is the Internet. Packet switching provides flexibility while circuit switching comes with a more predictable performance. *Network interfaces (NI)* connected between each resource and

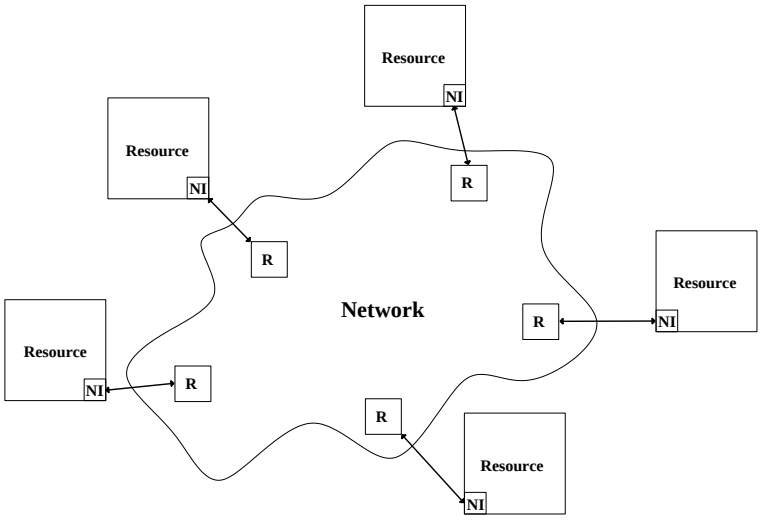


Figure 3.1: In a NoC resources are connected via network interfaces (NI) to routers (R). Routers are connected to each other to form a network.

the network are used to create and unpack packets or establish a connection between the source and destination resources. In packet switching network each packet contains information needed for routing of the packet to the correct destination.

The motivation for the NoC approach is the poor scalability of buses for systems consisting of a large number of resources. The longer the bus is the higher capacitive load it introduces to the drivers connected to it, and therefore results in a longer latency and higher power consumption. The same effect results from the increasing number of devices connected to the bus emphasized by the effect on the scheduling of the bus: the more resources connected to the bus, the less time is left for each of them to use the bus. Furthermore, the arbitration becomes more complex and time-consuming as the number of devices connected to a bus increases. NoC provides better scalability because as more resources are introduced to a system, also more routers and links are introduced to connect them to the network. The additional links and routers provide the communication capacity needed for the new resources. Many NoC realizations also inherently contain some redundancy in the communication media, which can be used to provide a higher reliability and traffic balancing. This is in contrast to bus structures, which rely on a single communication medium.

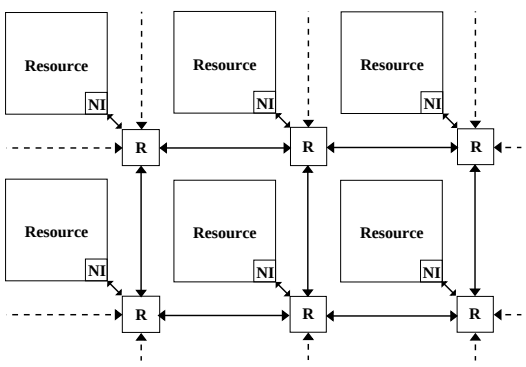
The NoC paradigm was first introduced at the beginning of this millennium by multiple independent research groups [6, 20, 31, 34, 76]. After that a lot of design efforts have been put to the topic, examples include Ætheral [28], MANGO [9], Nostrum [42], SPIN [1], and Xpipes [7]. Some approaches are already in commercial use, such as Arteris™ [4] and STNoC™ [79].

The main design choices in NoC are the way the routers are connected to each other, the *network topology*, the principles how the packets are routed from a source to a destination, the *routing protocol*, and the allocation of network resources to packets travelling in the network, the *flow control*. These are discussed in the following subsections. The network components, the routers, network interfaces and links, implement the topology, routing protocol, and flow control.

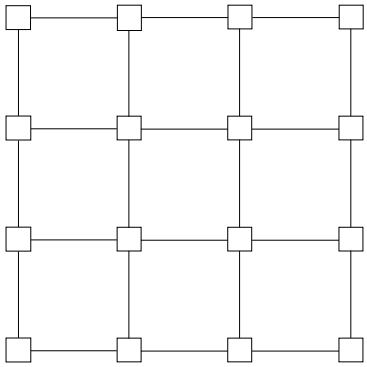
3.1 Topology

The network topology defines the way the routers are connected to each other to form a network. Most of the proposed topologies for NoC are borrowed from the world of inter-processor networks in supercomputers and computer networks.

Probably the most often proposed topology for NoC is the *mesh* presented in Figure 3.2. Resources are organized into a two-dimensional array and each resource is connected to one router assigned to it. Each router is

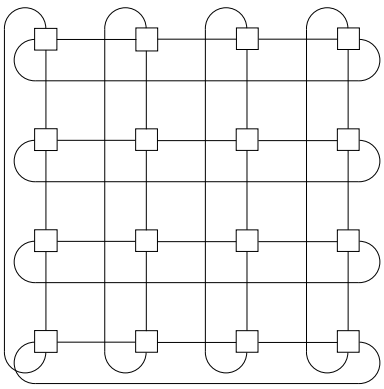


(a) A section of a mesh topology.

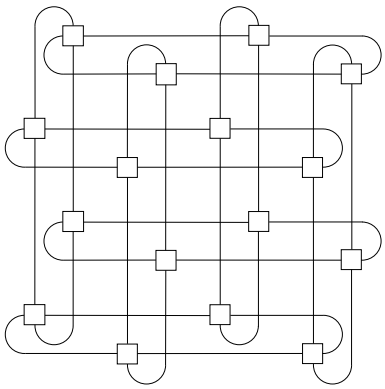


(b) 4 x 4 mesh, showing only the routers.

Figure 3.2: Mesh network topology.



(a) 4 x 4 torus.

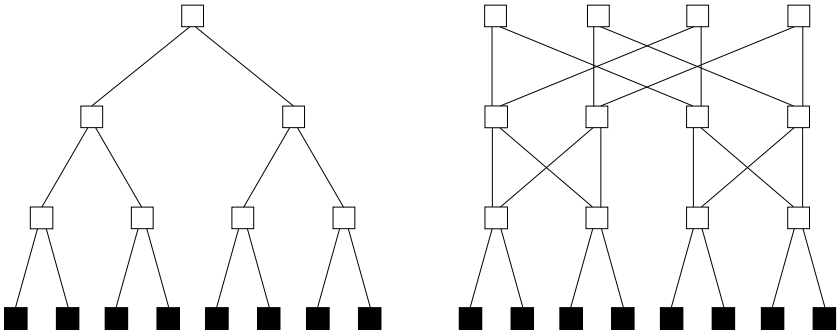


(b) 4 x 4 folded torus.

Figure 3.3: Torus and folded torus network topologies.

connected to its neighbors in four directions, typically named East, South, West and North, excluding the ones at the edges of the structure having only 2–3 neighbors. The topology is highly symmetric which makes the addressing and routing straightforward. Furthermore, the inherently horizontal and vertical links organized in an array shape between the resources simplify the creation of the chip layout.

The *torus* topology, presented in Figure 3.3(a), fixes one of the drawbacks of the mesh topology, namely the long distance between opposite edges. In a torus the routers at the edges are directly connected to the routers at the opposite edge of the system, thus decreasing the hop count on a route. The drawback of this approach is the uneven length of links, the edge-to-edge connections are longer than the links inside the mesh. To overcome this, a



(a) Binary tree. (b) Fat tree.

Figure 3.4: Binary and fat tree network topologies (the black squares represent resources).

structure called *folded torus* has been presented [21]. In a folded torus all the links are of the same length, illustrated in Figure 3.3(b).

Besides the mesh, most commonly used NoC topologies are variations from the *tree* topology, where the resources are the leaves and the routers form the nodes of the tree. Figure 3.4 presents two tree structures, the *binary tree* and *fat tree*. In the binary tree topology each router is connected to two resources or routers below it (children, descendants) and to one router above it (parent, ancestor). Fat tree [53] provides a more balanced capacity by having more links closer to the root of the tree. When also the number of routers is increased the fat tree topology provides the possibility to balance traffic and therefore decrease the possibility for congestion. The drawback of the fat tree topology is its complexity for on-chip implementation. While a binary tree can be easily created with an *H-structure*, the layout of the fat tree topology is not that obvious.

Other NoC topologies include *ring*, *star* and *spidergon*. The ring and star are very descriptive names: in a ring topology the routers are connected so that a ring is formed and in a star topology all routers are connected to one central router. Spidergon is a modification of the ring with an even number routers. In addition to the ring connections, each router is connected to its counterpart at the other side of the ring [18].

The regular structures explained above are suitable for general-purpose systems. However, they may loose some effectiveness when used with application specific systems. For this kind of systems custom topologies have been presented [7]. Their design starts from a thorough analysis of the traffic between the system components. A custom topology may contain links of different lengths and widths, odd-sized resources and it may have hybrid communication media combining buses, point-to-point links and networks.

3.2 Routing

The routing protocol defines the way a message is transmitted from the source to the destination. For routing a header is introduced to the packet at the network interface. The header contains information about the route to the destination. In *source routing* this information includes routing directives for each router in the route, i.e. should the packet be forwarded straight at the particular router or is a turn required. In this case the network interfaces contain a *routing table* which provides information about every route in the network. Another approach is to give an address to each router in the network and include the destination address in the packet header. The routers then read this address, compare it to their own address and make the decision about the direction in which to forward the packet. In this approach the routing tables or decision logic is in the routers.

The selection of the route through the network from a source to a destination is defined by the routing algorithm, which can be either *oblivious* or *adaptive*. *Oblivious* routing algorithms do not take into consideration any possible changes in the state of the network. An important subset of the oblivious routing algorithms are the *deterministic* routing algorithms, which always provide the same route through the network. Adaptive routing algorithms, in contrast to oblivious routing algorithms, change the route based on the circumstances in the network. For instance a congested part of the network may be bypassed through other routes, and therefore better traffic balance may be achieved. The drawback of adaptive routing is the higher complexity and the possibility that packets arrive at the receiver in different order than they were sent. *In-order delivery* is one of the strengths of deterministic routing algorithms.

The chosen network topology sets guidelines for the routing algorithm. The simplest deterministic routing algorithm for the mesh topology is *dimension order routing*, better known as *XY routing*. In XY routing a packet is first routed along the x-dimension to the correct column and then along the y-dimension to the correct row. XY routing algorithm is *minimal* because it provides the shortest route from the source to the destination. The minimality is achieved by routing packets only in progressive directions in each routing stage.

An example of an oblivious routing algorithm which is not deterministic is the *Valiant's randomized routing algorithm* [21]. A packet sent from some source s to a destination d is first routed to a random intermediate terminal node x and from there to the destination d. The routing from s to x and from x to d can be done e.g. using dimension order routing. The algorithm effectively distributes the traffic in the network, thus minimizing congestion.

Adaptive routing can be minimal or fully adaptive. In minimal adaptive routing the direction of routing is selected among the possible progressive

routes. A variant of XY routing called *dynamic XY routing* [55] is an example of minimal adaptive routing. In that algorithm, the progressive direction with less traffic is selected if there are multiple options. Otherwise the algorithm follows the principles of XY routing.

Fully adaptive routing algorithms do not restrict the packets to the minimal routes. Some algorithms even allow U-turns. When using non-minimal routing algorithms, there is a possibility for a *livelock*, a situation where a packet travels in the network without ever reaching the destination. A livelock can be detected with the inclusion of a *hop counter* in the message header. The counter is set to some initial value at the source and in each router the hop count is decreased by one. If the counter value equals zero before it reaches the destination, the packet is probably in a livelock, and as a solution it is dropped from the network. The dropping should be recognized by the flow control, and a retransmission of the packet commenced.

Besides livelock, other exceptions in the routing include *deadlock* and *starvation*. Deadlock means a situation, where two, or more, messages wait for a resource reserved by the other, and at the same time reserve a resource the other one is acquiring. Therefore, neither of these messages cannot proceed and the network is halted. Deadlock is a severe malfunction which should be either prohibited in advance (*deadlock avoidance*) or handled when it occurs (*deadlock recovery*). Starvation may occur if packets have different priorities and a lower priority packet must wait its routing turn forever.

Deadlocks can be prevented at the routing algorithm level by admitting only such turns and routes that will not lead to a deadlock. Basically this means the prevention of cyclic dependencies between packets. XY routing is an example of a deadlock-free routing algorithm, because it admits only four of the eight possible turns (East-North EN, East-South ES, West-North WN and West-South WS, see Figure 3.5), which guarantees that no cyclic dependencies are possible.

For gaining deadlock-freedom it is necessary to prevent only two turns from the eight possible. This indicates that XY routing wastes some of the adaptivity. *West-first*, *north-last* and *negative-first* are turn models that prohibit only the minimum number of turns [27]. The difference in between them is which turns are not allowed. In west-first turn model the packet is first routed West if necessary and it cannot be routed West later, thus the turns NW and SW are not allowed. In north-last turn model the routing to North is done last if it is required, the turns NW and NE are prohibited. According to the negative-first turn model the packets are first routed to West and South if required and after that to East and North. The turns ES and NW are not allowed. The allowed and prohibited turns are illustrated in Figure 3.5.

Another, a bit more complex model is the *odd-even turn model* [16]. In that model different turns are prohibited in odd and even columns in order

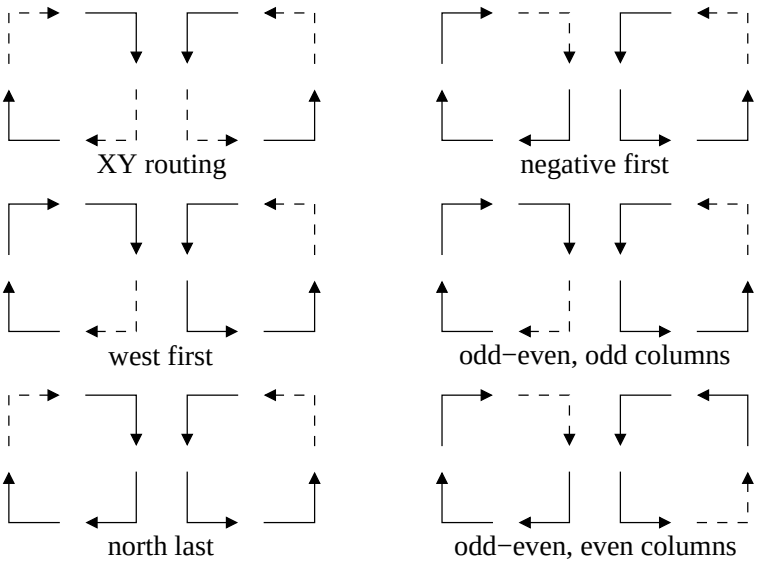


Figure 3.5: The allowed (solid line) and forbidden (dashed line) turns in each turn model.

to achieve a more balanced adaptivity. Routers at odd columns cannot make turns NW and SW, while the turns EN and ES are prohibited from routers at even columns (see Figure 3.5).

The routing algorithms obeying one of the turn models are deadlock-free. They are especially useful for creating adaptive deadlock-free algorithms. Their use in designing fault-tolerant routing algorithms is addressed in Chapter 6 and in Papers V and VI which propose fault tolerant routing algorithms based on turn models. A survey of routing algorithms proposed to be used in NoCs is presented in [66].

3.3 Flow Control

Flow control manages the allocation of network resources to packets as they proceed along their route [21]. If two packets arrive at the same time to a router and are to be forwarded to a same output, the flow control defines what happens to the other packet that cannot be processed instantly. In *bufferless flow control* there is no buffering capacity in the routers and therefore the other packet must be dropped or misrouted. Routing, where packets that cannot be forwarded into the desired direction are misrouted to other directions, is called *deflection routing* or *hot-potato routing*.

When there is buffering capacity at each router the flow control, also called the routing mode, defines how it is used in storing packets. In the *store-and-forward* routing mode each packet is completely stored in the

router's buffer before it is forwarded. Thus, the buffer size must be at least of the size of the largest packet. The *virtual cut-through* routing mode is an optimization of the store-and-forward routing mode. In virtual cut-through the forwarding of the packet can be started while still receiving it. In the *wormhole* routing mode the packets are divided into equal sized *flow control digits*, *flits*. The first flit is routed similarly as in virtual cut-through and at the same time the route is reserved for the flits following the first flit. This reserved route is called a wormhole. After the transmission of the last flit of the packet, the reservation of the routers is released and the wormhole no longer exists. The wormhole routing mode requires less buffering than the other two, because only one flit needs to be stored at a time.

The drawback of wormhole routing is an increased possibility for deadlocks. *Virtual channels* [21] are an efficient way to avoid deadlocks besides the routing algorithm level approaches presented in the previous subsection. The use of virtual channels means that a physical channel is divided into multiple virtual channels. Parallel buffers are introduced to the inputs and outputs of the routers, one for each virtual channel. This makes it possible that although some traffic is blocked, the other virtual channels can still be used and the probability for a deadlock is minimized or removed completely depending on the chosen routing algorithm.

Virtual channels are also an important resource in providing *quality of service, QoS*, which in general means the classification of the traffic into *best effort* and *guaranteed service*. Virtual channels are used to separate these two traffic classes, but they do not alone provide guaranteed services. One way of actually guaranteeing services such as throughput and latency, is the use of *time division multiplexing*, where resources are allocated in space and time [21, 28]. In addition to the guaranteed throughput and latency, an important property of a NoC is to provide *fairness* between traffic flows inside a traffic class. The packet hop count or live time can be used in creating fair arbitration, so that the oldest packets are served first [21, 62].

3.4 Fault Tolerant NoC

Fault tolerance issues can be considered in a NoC at many abstraction levels. The well known network OSI reference model [91], illustrated in Figure 3.6, has been applied to NoCs as well [22, 59]. The four bottommost layers, the *physical*, *data link*, *network* and *transport layer*, are part of the implementation of the hardware platform. The lowest layer, the physical layer, in NoC defines the wiring and signaling method. At this level the fault tolerance means the selection of appropriate signaling methods and fitting them into the noise margin.

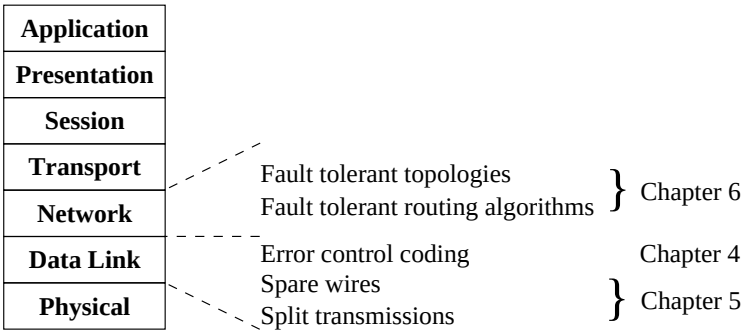


Figure 3.6: The seven layers OSI architecture [91] and the abstraction levels of the fault tolerance methods presented in the thesis.

The data link layer provides reliable transfer of information across the physical link. A number of fault tolerance methods are available to be used at this level. Common is the introduction of error control coding to the channel the link implements. Error control coding for on-chip communication is addressed in Chapter 4. ECC is not an optimal solution for tackling permanent errors in interconnects. Alternative methods are proposed in Chapter 5.

The network layer is responsible for providing the delivery of packets from the source to the destination. This layer includes most of the design choices of a NoC, containing the topology and routing algorithm. Adaptive routing algorithms are a way to overcome faults in a NoC. However, they require redundant routes in the topology. These issues are discussed in Chapter 6. Also end-to-end error control coding is an approach that can be considered to be realized at the network layer.

The transport layer handles the establishment of communication, i.e. it addresses congestion and flow control issues. The fault tolerance realized at this level is mainly originated from the solutions at the underlying layers. They just need to be taken into consideration when implementing the services of this layer.

An optimal fault tolerance solution should include parts from several abstraction layers. A fault should be tackled at the level where it is most cost-efficient in terms of power consumption increase and performance decline. A lower layer may handle some faults and inform the upper layer if it detects more faults than it is capable of correcting. To have an efficient system level approach, the methods used in each layer should fit into the whole, i.e. the methods should provide means for inter-layer information passing.

Chapter 4

Error Control Coding for On-chip Signaling

Error control coding (ECC) is a prominent fault tolerance method for on-chip signaling. In this chapter the terminology and principles of coding are presented together with promising coding approaches for on-chip signaling.

In error control coding a data word of length k is encoded to form a *codeword* of length $n \geq k$. The added $n - k$ bits are called *check bits* and in a *separable*, also called *systematic*, code they are appended to the end of the data word to form the code word. Separable codes are beneficial to be used in on-chip signaling, since the data bits are directly accessible in the code word. In on-chip signaling typically parallel transmissions are used in contrast to many data networks and wireless transmissions, where the common form of transmission is serial. In on-chip signaling a set of parallel wires usually forms a link, thus the check bits mean additional wires. A codeword at a time is transmitted over the link.

The efficiency of a code can be measured by its error detection and/or correction capability and the relation of data word and codeword widths, the *code rate* $R = \frac{k}{n}$. *Code distance* d is the minimum number of distinct bits in two codewords; to put it in another way, the minimum number of bits that must be flipped to get another codeword. The code's error detection capability can be calculated from the distance by $t_d = d - 1$, and the correction capability by $t_c = \lfloor (d-1)/2 \rfloor$. If a single code is used to correct some number of errors and detect some more errors, the relation to distance is $d = 2t_c + t_d + 1$. Because the on-chip links are commonly a set of parallel wires, the code rate indicates the number of additional wires required. The lower the code rate the more additional wires are required, which mean area and power overhead.

Burst errors affect multiple adjacent bits in a codeword. In parallel transmissions typical to on-chip signaling, a burst error therefore means an

error affecting two or more adjacent wires. This kind of error situation can be a result of many phenomena in a chip. For instance different kinds of couplings between wires are likely to cause burst errors.

Interleaving is an efficient way to cope with burst errors [90]. It means partitioning the data word into parts and encoding each of them separately. The final codeword is formed by taking one bit at a time from each encoded part. The length of burst errors the code obtained by interleaving can detect or correct is the number of interleaving section times the detection/correction capability of the code used to encode the sections. For instance, if we name the inputs as $i_{0..k-1}$, the check bits as $c_{0..n-k-1}$ and there are four interleaving sections, the coding proceeds as follows: the check bits $c_0, c_4, c_8, \ldots$ are calculated from inputs $i_0, i_4, i_8, \ldots$, the check bits $c_1, c_5, c_9, \ldots$ are calculated from inputs $i_1, i_5, i_9, \ldots$, and the other two interleaving sections correspondingly. Since the number of interleaving sections in this example is four, the obtained code has burst error detection capability $t_d^{burst} = 4t_d$ and correction capability $t_c^{burst} = 4t_c$, where t_d and t_c are the error detection and correction capabilities of the underlying code, respectively. Interleaving in parallel transmission links practically means reordering of the wires, which is very cost-efficient. Therefore, it suits very well for on-chip realizations.

The interleaving affects mainly the burst error tolerance but it also has an effect on tolerance against multiple single errors affecting the same codeword. Multiple single errors can be corrected if they affect separate interleaving sections. The efficiency of interleaving in tolerating multiple single errors has been analyzed in Paper I.

4.1 Linear Block Codes

Linear block codes are a well known and widely used set of codes. The name indicates that a set of bits, a block, data word, is encoded at a time. A codeword is always a linear combination of the other codewords. The encoding and decoding of these codes is in most cases straightforward and fast, which makes them well suited for on-chip realizations.

Linear block codes can be presented by their *generator matrix* G, which is of size $k \times n$, where k is the data word length and n the codeword length. The encoding of a linear block code can be done with matrix multiplication $\vec{c} = \vec{a}G$, where $\vec{a}$ is a data word vector of length k, G is the $k \times n$ generator matrix and $\vec{c}$ is a codeword vector of length n. The code is systematic if the generator matrix contains an $k \times k$ identity matrix, i.e. the codeword includes the data word unaltered. Hardware realizations of systematic (separable) code encoders can be simplified, because only $n - k$ codeword symbols have to be calculated and the rest are just the symbols of the data word. Binary

matrix multiplication is simply calculating parity bits, which in hardware means trees of *xor* gates.

Syndrome decoding is a common technique for decoding linear block codes. A *syndrome* can be considered to contain information on the errors of a transmitted codeword in a compressed form. It is calculated by matrix multiplication $\vec{s} = \vec{u}H^T$, where $\vec{u}$ is a received code word vector of length n ($\vec{u} = \vec{c} + \vec{e}$, where $\vec{c}$ is a transmitted codeword and $\vec{e}$ an *error vector*, both of length n), H^T is the transpose of the $(n-k) \times n$ *parity check matrix* and $\vec{s}$ is a syndrome vector of length $n-k$. The parity check matrix can be constructed from the generator matrix and vice versa. The syndrome gives the index to the table of minimum weight error vectors, so the error vector $\vec{e}$ can be easily determined. The correction is done by $\vec{c} = \vec{u} + \vec{e}$, eliminating the error from the received data word.

4.1.1 Hamming Codes

The most common linear block codes are the *Hamming codes*. A Hamming code fulfills the rule $2^{n-k} \geq n + 1$, where $n - k$ is the number of check bits and n is the length of the codeword. The minimum distance of a Hamming code is 3, so it can correct a single error in each codeword ($t_c = 1$) or it can be used to detect double errors ($t_d = 2$). The Hamming code is also described as using the concept of overlapping parity where there are multiple parity bits and every data bit is involved in calculating several of them. The overlapping parity concept is a direct result of the matrix multiplication encoding presented above. A modified Hamming code for both correcting single errors and detecting double errors can be achieved by adding one more check bit, which is used as the parity bit of the whole codeword [39, 44]. The method of adding one more parity bit to each codeword is called *extending* and it can be applied to other linear block codes as well. The opposite to extending is *shortening*, where a number of data bits is set always to zero and therefore can be left out of the codeword. Thus, shortening with a factor s results in a code which has data word length $k - s$ and codeword length $n - s$. Shortening decreases the rate since $\frac{k-s}{n-s} < \frac{k}{n}$, for $s > 0$. Shortening is extensively used in on-chip error control coding because a bus width seldom directly matches the data word width of the wanted code.

Hamming codes are the most widely used codes in the research on interconnect link error protection [8, 54, 60, 69, 78, 90]. For instance in [54] parity and Hamming coding have been used in adaptive error detection system which is built to obtain a more energy efficient design without affecting the error detection capabilities. The system monitors the noise level of the transmission channel and dynamically changes to a code that has better error detection capabilities in the case of an increased noise level, and respectively changes to a weaker code when the noise level is lower. In

the design, parity, Hamming double error detection and extended Hamming triple error detection codes are used.

A set of codes similar to Hamming codes are the *Hsiao codes*. They differ from Hamming codes in the way the generator and parity check matrices are constructed. [44]

4.1.2 Cyclic Codes

Cyclic codes are a set of codes where a cyclic shift of a codeword generates another codeword. Because of this property, efficient realizations of these codes can be achieved using *linear feedback shift registers (LFSR)*. In the standard form this creates non-separable codes, but the codes can also be made separable by small changes in the generation process [39]. Cyclic codes are normally presented with their *generator polynomial* $g(x)$ instead of the generator matrix. The generator polynomial is of degree $n - k$ and the encoding is proceed by $c(x) = a(x)g(x)$, where $a(x)$ is the data word and $c(x)$ the codeword, both in their polynomial forms.

A class of cyclic codes, the *cyclic redundancy check codes (CRC)* are often used for detecting errors. These codes are able to detect single errors and adjacent multiple errors, which makes them extremely suitable for detecting burst errors. The number of adjacent errors that can be detected is $n-k-1$, where k is the number of data bits and the coded word contains n bits. A generator polynomial of degree $n - k$ is used. For instance, CRC-8 (8 for the degree of the generator polynomial) is used in a self-calibrating design to detect errors on the transmission channel, where self-calibration means that the voltage-swing for the transmission channel is scaled dynamically to obtain minimum energy consumption [85].

4.1.3 BCH Codes

As the probability for multiple errors increases when scaling further into the nanometer regime, error correcting codes capable of correcting several errors are needed. Popular linear block codes for multiple error correction are the *Bose-Chaudhuri-Hocquenghem (BCH)* codes [83], which are also cyclic codes. They can be easily constructed according to specifications for correcting as many errors as is required. The BCH code is similar to Hamming code when used as a single error correcting code, actually Hamming codes are BCH codes with $t_c = 1$, but commonly codes with $t_c \geq 2$ are called BCH codes. In on-chip signaling, where only two logic states are possible, only binary BCH codes are of interest.

The syndrome decoding method explained above is limited by the size of the minimum error vector table. When the number of check bits $n - k$ is high the table is impractically large, and therefore alternative decoding

methods should be used. The decoding of BCH codes can be done using the *Berlekamp-Massey* (B-M) algorithm [10]. The algorithm is iterative requiring $2t_c$ iterations, where t_c is the error correction capability of the code. The calculations are done in Galois Field (GF) 2^m, where m depends on the length of the code ($2^m \geq n$). In addition to the actual algorithm a pre-processing circuit is needed. Using *Fourier transform* in $GF(2^m)$ $2t_c$ syndromes are calculated. For binary BCH codes, this basically means quite similar trees of *xor* gates in hardware as for the syndrome calculation explained above. The error vector $\vec{e}$ is extracted by using a method called *Chien search* to find the zeros of the *error-locator polynomial* $\Lambda(x)$ obtained from the B-M algorithm. [10]

4.1.4 Reed-Solomon Codes

The *Reed-Solomon codes* are other cyclic codes that can be used to correct multiple errors. The codes are nonbinary, which means that instead of bits, groups of m bits (e.g. $m = 8$, a byte) are used as symbols for the codes. The Reed-Solomon codes are optimal meaning that they provide the maximum distance at the used number of check symbols. If a word contains k groups of data and its length is n groups, at most $\lfloor (n-k)/2 \rfloor$ errors can be corrected. Because on-chip signals are binary, each symbol must be coded by binary bits. These binary-coded nonbinary codes are especially effective in correcting burst errors, since the detection and correction is based on symbols which consist of many adjacent bits. On the other hand, they are not very effective in single error tolerance because a single bit fault in a binary-coded symbol takes the whole correction capability of that symbol. Binary codes, such as binary BCH, provide the same tolerance against single faults with a lower number of check bits. [10, 44]

Decoding of Reed-Solomon codes is also based on the Berlekamp-Massey algorithm. The main difference is that in addition to the error vector $\vec{e}$ also the error values are needed to perform the error correction. The error vector $\vec{e}$ points the erroneous GF(2^m) symbol and the actual correction is done by adding the error value of that particular symbol and the transmitted symbol itself. The error values can be extracted using the *Forney* algorithm with inputs obtained from the B-M algorithm. Also the Fourier transform requires slight changes compared to the one used for BCH decoding. In Reed-Solomon decoding all the calculations are done with GF(2^m) symbols while for binary BCH codes the syndrome calculation is just calculating parities of different sets of bits. [10]

A special feature of Reed-Solomon decoding is that it can be constructed so that it provides information on correctness of the decoding. This kind of information could be very important in safety-critical designs, where it is better to stop the system than give a false result.

Paper I presents hardware realizations of encoders and decoders for a set of Hamming, BCH and Reed-Solomon codes. Syndrome decoding is used for the simpler codes and the algorithmic approach for the more complex BCH and Reed-Solomon codes. The paper presents a comparison of the complexities of the realizations together with an analysis of the differences in fault tolerance of each code.

4.2 Other Coding Approaches

Besides linear block codes, also other coding approaches have been presented for on-chip signaling. One example is the *Dual rail code (DRC)*, where every bit is doubled quite similar to the well-known *repetition code*. The resulting code is practically separable because the data word can be found from the codeword by simply reordering the wires. In addition to the repetition of bits, a parity bit is introduced, thus the code needs $2k + 1$ bits, where k is the number of information bits [71]. In an extended version of the dual rail code also the parity bit is doubled so $2(k+1)$ bits are needed [70]. The DRC is able to correct single bit errors like the Hamming code but the number of check bits is much higher. The code rate is less than a half, $R = \frac{k}{2k+1} < 0.5$. The benefit of the DRC is its ability for crosstalk minimization, which simply results from the fact that a signal wire and its duplicate always have equal transactions. The DRC and especially the extended DRC perform better than the Hamming code when transmission delay or energy consumption is regarded in the presence of crosstalk noise and when the wires are placed optimally [70, 71]. In nanoscale circuits crosstalk capacitance is expected to be dominant compared to other wire capacitance and therefore the benefits of DRC methods are important.

4.3 Error Recovery Methods

Error control coding can be applied to detect errors in a message transmission. In general, two recovery techniques are used when an error has been detected. In *automatic repeat query* (ARQ) a retransmission is requested, while in *forward error correction* (FEC) check bits transmitted together with the data are used to correct errors without the need for retransmission [89]. Hybrid approaches try to combine the best properties of ARQ and FEC [8, 23, 78].

As explained above, a code's error detection capability t_d is larger than its error correction capability t_c. Thus, the same code's detection capabilities can be used for worse error conditions than its correction capabilities if the same detection/correction rate is targeted. Under uniform noise distribution the probability of a link error depends on the link voltage. Therefore, a

lower link voltage can be used to detect errors than is required to correct them. As a result, error detection combined with ARQ can be more power efficient than FEC [8], especially when dynamic voltage scaling is applied [77]. The drawback of ARQ is the retransmission latency. Furthermore, the number of retransmissions depends on error conditions. In persistent noise environments, a large number of retransmissions may result, making ARQ less energy-efficient than FEC. The FEC approach has a fixed, guaranteed throughput, which is important for many applications.

More importantly, the ARQ method fails in the presence of permanent errors (retransmission is only useful for avoiding transient errors). FEC codes can detect and correct permanent errors, but each permanent error will reduce the code's capability to tolerate transient or intermittent errors. FEC codes which can detect and correct multiple errors (e.g. BCH codes) have large power and area overheads.

The realization of retransmission request and the retransmission has its overheads. In the *stop-and-wait* method [89] the receiver informs the transmitter after each transmission about the transmission success and if a retransmission is required. This results in latency overhead and throughput decline. The *go-back-N* method [89] does not stop after each transmission but continues until an error is detected. At this point the receiver informs the transmitter which starts the transmission again from the word where the error occurred. The overhead of this method is the additional area and power for storing past data words in the transmitter as well as the latency and throughput penalty of starting the transmission again. The third ARQ method is *selective repeat* [89] which is similar to *go-back-N*, but now only the erroneous word is retransmitted. This approach has area and power overhead for not only storing words at the transmitter but also at the receiver. On the other hand, the delay and throughput penalties are the lowest of the ARQ approaches.

ARQ fits well with asynchronous signaling, where handshaking is used to inform a successful transmission. Since an acknowledgment signal will in all cases be transmitted from receiver to transmitter, it can be extended to contain also information about the status of the transmission. This is achieved for instance by sending either an acknowledgement or a negative acknowledgement. The former means a correct transmission while the latter is a request for a retransmission.

The timing signals in asynchronous links are crucial for the correct operation of the link, and therefore they have to be protected against errors. For this purpose triple modular redundancy (TMR) can be used as proposed in [60]. Furthermore, the three instances of each control signal can be physically dispersed to maintain the tolerance against burst errors. TMR in on-chip signaling means actually the utilization of a repetition code, where the voter is the decoder.

FEC, on the other hand, suits nicely for synchronous signaling. Both of them target for a high throughput without any backward signaling. Synchronous signaling is based on tight timing constraints, and the usage of forward error correction can be sometimes used to loosen these constraints. The code can correct the errors caused by some signals arriving late in the worst case conditions for which the clock frequency would need to be otherwise matched.

Chapter 5

Protection Against Permanent Faults in On-chip Links

Error control coding (ECC) is an efficient way of tackling transient errors in on-chip interconnects. However, its usefulness against permanent faults is not that obvious. A single permanent fault can drastically reduce or even eliminate the correction capabilities of the commonly used codes. Codes with a higher correction capability, on the other hand, are too complex for on-chip realizations which often have strict performance and cost requirements. More importantly, the automatic repeat query (ARQ) method fails in the presence of permanent errors, since a retransmission is only useful for avoiding transient errors. Most of the failures (80%) are caused by transient faults [22]. The other way around, up to one fifth of all the failures are originating from permanent or intermittent faults. Thus, the fault tolerance approach must contain elements to not only tolerate the temporary faults but also the ones of more permanent nature.

The idea presented in Papers II and III is to combine two different fault tolerance methods to achieve a system that is efficient in tolerating transient, intermittent and permanent errors. For transient faults error control coding combined with interleaving is used. In Paper II ARQ is used while in Paper III the recovery method is forward error correction (FEC). For tolerating permanent and intermittent errors two methods are introduced, one that uses hardware redundancy and the other based on time redundancy. In the first approach spare wires are introduced together with reconfiguration circuitry. In the second approach, the data is split into two transmissions and in both of them the transmitted data is replicated. In the receiver the fault-free copy is chosen and the data of the two transmissions are again combined into a complete word.

Permanent fault correction in on-chip links is a two-step process. First, the permanent fault must be detected; then, the link must be reconfigured to avoid transmitting over the faulty wire. The detection of permanent faults is discussed in Section 5.1 and the spare wire and split transmission approaches are addressed in Sections 5.2 and 5.3, respectively.

5.1 Permanent Fault Detection

To detect permanent errors in on-chip interconnects, two distinct methods have been proposed in Papers II and III. In *in-line test* (ILT) method each pair of adjacent wires in a link is tested for opens and shorts. These tests can be run periodically to ensure that each link's ECC capability is not crippled by permanent errors. By testing every wire in the link, the ILT method also recovers resources from intermittent errors that were incorrectly flagged as permanent.

The *syndrome storing-based error detection* (SSD) method is based on evaluation of consecutive error syndromes at the receiver. Syndromes are calculated during the decoding procedure. They contain information on errors in the received words as was described in the previous chapter. If a number of consecutive error syndromes are equal, it may indicate a permanent fault in the link.

5.1.1 In-Line Test Method

The in-line test (ILT) method proposed in Paper III sequentially routes data from each pair of adjacent wires to a set of available spare wires, allowing each pair to be tested for intermittent and permanent faults. The system requires a test pattern generator at the transmitter side of the link and a detection block at the receiver. The wires under test are connected to these modules.

The test pattern for the ILT test has been derived in Paper III. A test consisting of bit patterns "01" and "10" is sufficient for finding all shorts and opens in or in between the wires. The paper presents also test procedures for situations where only one or no spare wires are available. In these cases a single wire test or a test only for the wires that are marked erroneous is run.

The whole test procedure can be run during normal operation, without interrupting data transmission. This is enabled by the reconfiguration system described in Paper III.

To protect against runtime permanent errors, the ILT is run periodically, with a period that can be shortened to improve error resilience or increased for energy efficiency. In addition to this periodic testing, the ILT can be triggered when an error is detected beyond the error correction capability

of the code protecting the link. The trigger can use a simple timer or be adaptively controlled by an upper protocol layer to save energy during idle periods.

5.1.2 Syndrome Storing-Based Detection Method

The syndrome storing-based detection (SSD) method for detecting permanent faults was first introduced in Paper II and further developed in Paper III. The basic idea behind SSD is that the error syndrome of a linear block code contains information about the errors in a received codeword. If the syndromes of a number of consecutively received codewords are the same, it can be concluded that there is a permanent error in the link (limitations described momentarily). The error location can be extracted from the syndrome using the normal decoding procedure. The effectiveness of this approach comes from the fact that it takes advantage of the code and decoder already present at the system. Hence, the amount of additional logic can be minimized.

If there are more errors in the link than the error correction code is capable of handling, the syndrome will be decoded incorrectly, thus giving a wrong error location or no location at all. For example, shortened codes have more possible syndromes than there are error patterns in the code. These remaining syndromes can be set to produce an all-zero error vector, which is a safe approach, or one of the error vectors having the minimal weight above the error correction capability is chosen. Since there are typically many error vectors having the same minimal weight above the error correction capability there is a possibility that a wrong one is chosen, and thus a chance for a decoding error. Nevertheless, there is also a possibility to catch an error pattern beyond the basic error correction capability of the code.

An important design decision for the SSD method is to determine how many syndromes to consider before deciding that an error is permanent. This number of cycles is referred as the *observation period* t_{op}. If an intermittent error is misdiagnosed as a permanent error, a spare wire is consumed. In SSD, there is no method for recovering spare wires once they have been assigned, thus if the error observation period is set too short the result can be wasted wire resources. On the other hand, too long an observation period may result in a large number of cycles before the error is detected or may even leave errors undetected. This is because the detection of stuck-at faults is data dependent; in order to be detected, the error must occur in all datawords during the observation period. For example, a stuck-at-1 fault can only be detected if all the data bits passing through that wire over t_{op} cycles are 0. The upper limit for the number of cycles before the permanent error should be detected can be derived from the transient *bit error rate (BER)*. Since a permanent fault in a link may prohibit the detection and

correction of a transient fault, the number of cycles to detect a permanent fault should be much smaller than the mean time between transient faults.

In Paper II the observation period t_{op} was set to three transmissions. Paper III presents a more detailed analysis of the trade-offs between t_{op} and reliability, and provide guidelines for selecting t_{op}. Based on that analysis $t_{op} = 9$ was found to be optimal for implementation perspective and to allow the transient BER to be as high as 10^{-6} while ensuring that the probability of a transient fault occurring before the permanent fault is detected is less than 1 %.

When combining SSD with forward error correction as done in Paper III, the error extraction circuitry already existing in the FEC decoder can be used. Error extraction is the process of obtaining the error vector $\vec{e}$ from the syndrome $\vec{s}$ explained in the previous chapter. The error extraction circuit only needs to be expanded to extract also possible check bit errors. Certainly, if there is a permanent error in a wire used for a check bit, it should be corrected as well, since it affects the overall error correction capability of the code. This is generally not implemented in FEC decoders, since only the data bits are part of the output of the link. In Paper II the SSD is combined with a system using ARQ, which means that the error extraction has to be completely implemented in the SSD unit.

5.2 Spare Wires

The use of spare modules to replace erroneous ones, especially in array structures, is a long known fault tolerance approach [39]. Spare cells and wires have been used in field programmable gate arrays (FPGA) to bypass defective components [32, 86]. Refan, *et al.* use spare wires to recover from a switch failure by connecting each processing element to two switches in a network-on-chip (NoC) [67]; if a permanent fault occurs in one switch, processing elements share the working switch, and the system reroutes its data accordingly. Grecu, *et al.* have analyzed the use of spare wires in NoCs [29] to increase manufacture yield; reconfiguration of the links utilized crossbar switches with redundant channels. Unfortunately, the authors did not discuss the error detection procedure. In another work, Grecu, *et al.* presented a built-in self-test methodology for NoC interconnects [30] and thoroughly discussed manufacture testing methods for NoCs, but they did not address runtime failures. Reick, *et al.* discuss dynamic I/O bitline repair using spare wires [68], but their detection and correction processes are not specified.

Spare wires to tackle permanent faults in on-chip links are introduced in Papers II and III. The number of spare wires to be inserted into the system depends on the probability of a permanent error in a wire, the number

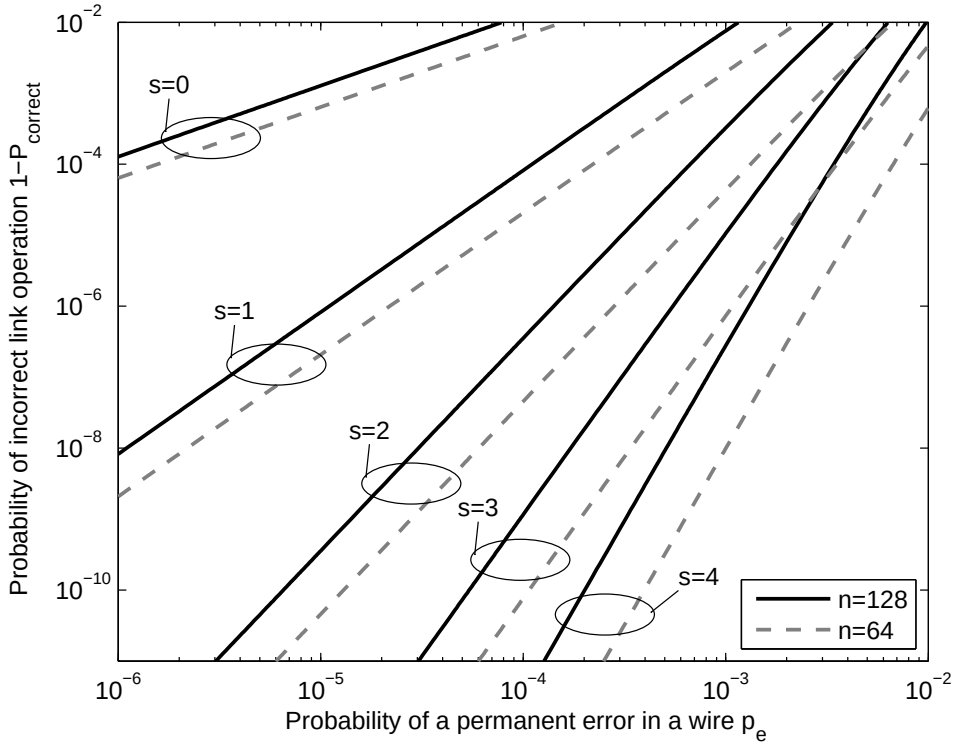


Figure 5.1: The effect of number of spare wires s on the probability of incorrect link operation.

of wires and the desired probability for correct operation of the link. Assuming that permanent errors are independent and that the detection and reconfiguration can be done in all cases, this relation can be described by

$$P_{correct} = \sum_{i=0}^{s} \binom{n+s}{i} (1-p_e)^{n+s-i} p_e^i \tag{5.1}$$

where $P_{correct}$ is the probability for correct link operation, n is the number of wires in a link, s is the number of spare wires and p_e is the probability of a permanent error in a wire. The probability of an incorrect link operation, $1-P_{correct}$, is shown in Figure 5.1 as a function of p_e for 0 to 4 spare wires and two different values of n. Figure 5.1 can be used to determine the required number of spare wires to meet the reliability target given the estimated value of p_e. For example, if the probability of incorrect link operation must be less than 10^{-10} and the probability of a permanent error in a wire is 10^{-5}, three spare wires are needed if $n = 128$ while two spare wires are enough for $n = 64$. Interconnect error probability has been analyzed in [57].

The number of spare wires s has an effect not only on the permanent fault tolerance but also on the complexity of the reconfiguration units. Each

additional spare wire adds certain amount of logic. The amount of additional logic can be decreased if the usage of spares is restricted to only some parts of the link. In Paper II interleaving is used as part of the underlying coding approach and the utilization of spares is connected to the interleaving sections: one spare for each of the four interleaving sections. This has a remarkable effect on the complexity of the reconfiguration logic as compared to the approach in Paper III, where there is no limitations for the utilization of spare wires. Naturally, restrictions in the spare wire utilization have an influence to the overall reliability of the link.

The spare wires have also another purpose besides replacing faulty wires when ILT detection method is used. In ILT the spare wires are used to temporarily replace in-use wires to allow them to be tested, or to enable further testing of wires that have been declared erroneous and therefore already bypassed. The retest property allows bypassed wires to be restored if their errors turn out to be intermittent rather than permanent. Both properties are incorporated in the ILT detection method presented in Paper III.

The introduction of spare wires requires reconfiguration control and logic for bypassing erroneous wires, and a protocol for synchronizing information between receiver and transmitter. The reconfiguration procedure can be done by stopping the normal operation of the link or without interrupting the data flow. An example of the former is proposed in Paper II while the latter is used in Paper III.

5.2.1 Reconfiguration Control and Logic

The reconfiguration unit routes the data around the erroneous wires or wires under test. To balance the delay within routed wires, the reconfiguration ripples through the bus instead of rerouting erroneous wires to a bank of spare wires. The alternative method would require a large selection logic for spare wires as well as longer wires, both of which produce large delays, potentially causing timing errors.

The error detection circuits using either ILT or SSD method provide the location of the erroneous wire, which then should be bypassed. The reconfiguration units at each end of the link are used to transmit the reconfiguration information from the receiver to the transmitter, as well as synchronize the reconfiguration procedure so that both the transmitter and receiver do the reconfiguration in the same cycle. The transmission protocol is presented in the following section.

The reconfiguration itself is accomplished with a number of multiplexers connected to each wire. The state of these multiplexers is stored in a bank of control registers which are updated after the reconfiguration information transmission. Depending on the approach chosen for the utilization of spare wires, the control registers can simply store the location of the error when

only one spare wire for each section is introduced, or it can have separate control signals for each wire to indicate the number of bypassed wires at that current location. The former approach is explained in detail in Paper II and the latter in Paper III.

The reconfiguration control unit has two modes of operation. It can mark a wire erroneous, or remove the mark from a wire. The latter is needed for periodic testing or reallocating a wire to normal operation if the error on it turns out to be intermittent instead of permanent. For systems where an error mark cannot be removed (e.g. when using the SSD method), only the first mode is needed.

5.2.2 Transmission of the Reconfiguration Information

To minimize the link area and energy, the transmission of control data from receiver to transmitter is serial, using only a one-bit signal. Another signal is required for synchronization and to initialize the transaction. Both of these signals can be protected with TMR and spatial separation. If the link is asynchronous it already has a feedback channel for acknowledging the arrival of each message. This channel can be also used to transmit the reconfiguration data if the link operation is stopped during the reconfiguration. This approach is presented in Paper II.

The transmission protocol can be divided into three phases: (1) The initialization of a transmission, (2) the transmission of the error location one bit at a time, and (3) the end of the transmission. The reconfiguration procedure follows immediately after the transmission. Papers II and III propose two transmissions protocols. The main differences between them are that the one presented in Paper II is asynchronous while the one proposed in Paper III is synchronous. Also the reconfiguration information transmitted differ between these two approaches because of the different utilization of the spares (restrictions or not) and the fault detection method.

5.3 Split Transmissions

In the split transmission approach proposed in Paper II two transmissions are used to transmit one data word. The data word is split into two parts, and two copies of the same half is transmitted in each of the two transmissions. The approach gives tolerance against single permanent faults, and against multiple faults if they affect only one of the copies. The split transmission approach is also extended to take into account links that use interleaving for providing tolerance against permanent faults occurring as bursts.

The drawback of the split transmission approach is a significant impact on the latency and throughput. Its use can be motivated by considering a

typical NoC system. An erroneous link could be bypassed through neighboring routers, which would increase traffic in other links and may cause congestion. The total latency of links and routers on a bypassing route set an objective to the split transmission link design.

The split transmission system is combined with SSD method in Paper II. The required SSD implementation is a simplification of the one required for the spare wire approach since for the split transmission there is no need for extracting the exact error location. Also the transmission protocol simplifies to only synchronizing the change of operation modes between normal and split transmissions. The reconfiguration logic at the transmitter and receiver consist mainly of registers to store the halved data words and multiplexers selecting between them.

Chapter 6

Network-level Fault Tolerance in NoCs

The principle in network-level fault tolerance is that a packet can have multiple alternative routes from a source to a destination. To accomplish this, the topology must provide alternative routes and the routing algorithm must have adaptivity to choose between those alternative routes. These two aspects are addressed in the following subsections.

6.1 Fault Tolerant NoC Topology

A NoC topology provides fault tolerance if it has an alternative route or routes to bypass erroneous links and/or routers. Some topologies such as the star and binary tree do not have such properties while many others, including e.g. mesh and fat tree have inherently redundant routes. Refer to the figures in Chapter 3 to easily verify this.

The fault tolerance a topology provides can be further increased by a careful analysis and by inserting additional network components and links to selected positions. Paper IV presents modifications to increase the fault tolerance of the mesh topology repeated in Figure 6.1a. The weak point of the mesh topology is the connection to a resource. Each resource has only one network interface, which is connected via one link to one router. From that router on there are multiple connections, but a fault in a network interface, in a link connecting a network interface to a router, or in a router will always lead to a disconnected resource.

Paper IV suggests the addition of more network interfaces to each resource and modifications to the interconnection structure correspondingly. The most obvious way to do this is to increase the number of ports in routers. In Figure 6.1b a structure is presented where there are two network interfaces in each resource and the routers have six ports. A structure having

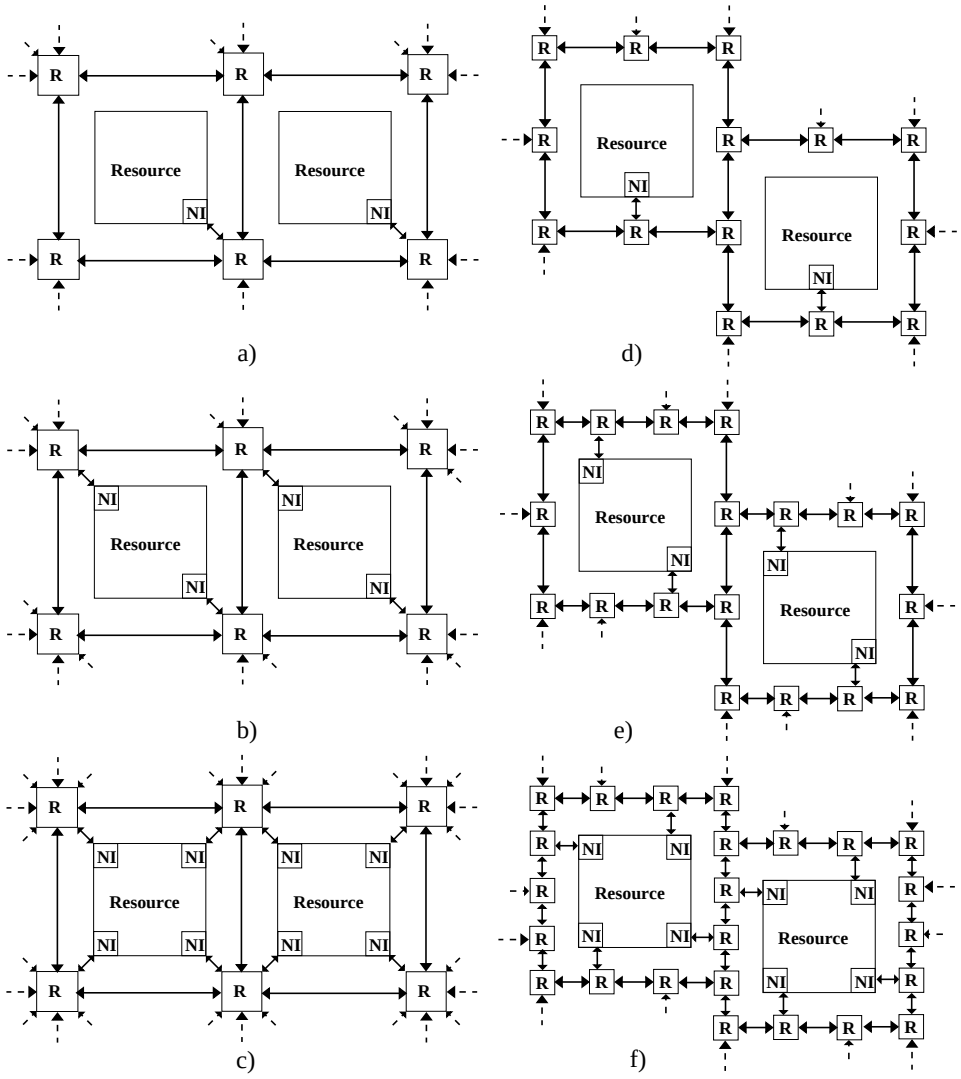


Figure 6.1: NoC architectures: a) Resources with one network interface (NI) and 5-port routers (R), b) 2 NIs and 6-port routers, c) 4 NIs and 8-port routers, d) 3-port routers and one NI per resource, e) 2 NIs, and f) 4 NIs.

four network interfaces in a resource and eight ports in each router is shown in Figure 6.1c.

The larger the area of a circuit the greater is the probability of faults in it. Therefore also structures with minimum size routers, i.e. routers with 3 ports, are presented in Figure 6.1d–f for one, two, and four network interfaces per resource, correspondingly. All six different architectures have been analyzed in Paper IV for their overall reliability and compared in area,

throughput and latency. In the analysis the impact of port count to the router performance and area is evaluated. For this purpose a scalable example router architecture is proposed and its realizations with three, five, six, and eight ports are analyzed.

The additional resources increase also the power consumption of the system. Therefore, a lot of design effort should be given to the power optimization of these circuits. One possibility is to use asynchronous design style because of its many advantages in distributed environments and with redundant components [56]. Asynchronous routers consume power only when data is transferred over the ports of it. Furthermore, if only some of the ports are needed for a transaction, the other ports will remain inactive and hence do not consume power. In a large NoC, there may be parts that are not used at all for a while, or only few packets traverse through that part, and therefore the reduction in power consumption can be remarkable. Obviously, one could build different clock gating mechanisms to reduce power consumption in a synchronous router, but it increases complexity and could be impractical, if the inactive periods occur often and their duration is short. The mentioned power reduction is further amplified when the system contains redundant routers which are implemented to increase reliability. As long as these asynchronous routers are not needed, they do not consume any power. Hence, the asynchronous redundant routers can be seen as cold spares [39]. Paper IV presents asynchronous realizations of routers and a network interface.

Reliability $R(t)$ is used as a measure in the fault tolerance analysis. It was introduced as one of the fault tolerance metrics in Section 2.3. In Paper IV a method for comparing analytically reliability of systems under uniform fault distribution is presented. The systems can consist of different-sized components, i.e. routers with varying number of ports. The topologies shown in Figure 6.1 are compared using the method, showing the reliability increase when the number of network interfaces per resource is increased.

6.2 Fault Tolerant Routing

The fault tolerance a routing algorithm introduces is originated either from redundant packets in the network or redundant routes in the network topology that are used to form a route for a single packet. Algorithms based on multiple copies of each packet include *flooding*, *gossip*, *stochastic communication* [12], *N-random walk* [63], and *multi-path routing with in-order delivery* [61].

In flooding the packet is sent to all possible directions from the source router and each router receiving the packet forwards it to all neighbors. The packet eventually reaches the destination after which the redundant copies

can be removed from the network. *Probabilistic flooding* [63] is a variant of flooding where only some of the routers forward received packets to all neighbors. In gossip algorithms the packet is forwarded to only a subset of the neighbors mimicking the behaviour of rumour spreading, the rumours are not necessarily shared with every neighbor. The selection in which directions to forward the packet can be randomized or based on an algorithm such as in *directed flooding* [63], where a higher probability is given to directions that move the packet towards the destination at the beginning of routing and more dispersion is allowed when the destination gets closer. Stochastic communication [12] is an implementation of a randomized gossip algorithm.

N-random walk decreases the number of copies of a packet in the network by sending only a fixed number N copies to the network. Each of these copies of a packet travels through different routes towards the destination. The algorithm for selecting these routes borrows principles from directed flooding. Multi-path routing with in-order delivery also operates with a fixed number of copies of a packet, but in this approach the routes are pre-calculated to find the optimal ones and preserving the in-order delivery.

Algorithms based on multiple copies of each packet generally provide shortest routes and good tolerance against faults. However, they do this with an extensive power consumption and a low resiliency to congestion. The power consumption can be lowered by decreasing the number of copies of a packet in the network. The drawback is the more complex routing algorithm, and thus a larger hardware-realization, or the demand for pre-calculated routes, i.e. large routing tables. Furthermore, the fault tolerance properties and ability to provide the shortest possible route may suffer from the decreased number of copies of a packet.

Algorithms taking advantage of redundant routes in a topology, but operating with only a single packet, can be divided into those that require knowledge about the state of the network and to the ones that use completely distributed control. Using global control provides means for finding the optimal route, but it introduces another aspect of complexity in realizing the control. Typically it means separate control channels or packets and routing tables in routers and/or network interfaces. All this increases area and power consumption. Examples of routing algorithms using global or partly global control include *distance vector routing*, *link state routing* [2], *DyNoC* [11] and reconfigurable routing based on *DSPIN* [87].

In distance vector routing each router maintains a routing table which contains information on directions which provide the shortest route to each destination. The route length is defined as the number of hops to the destination. Routers share this information with each others in a flooding manner so that each router can update its table and in case of changes in the network, such as faulty links and routers, the new shortest route is found. In link state routing each router finds out the state or speed of each link connected

to it with a special handshaking protocol. It then broadcasts this information to the rest of the network. Based on this information, each router can calculate the shortest route to each destination using e.g. Dijkstra's shortest path first algorithm [2].

Both DyNoC and reconfigurable routing based on DSPIN provide a way to bypass obstacles or erroneous regions in a NoC. In DyNoC each router next to an obstacle is provided with an information about into which direction the packets should be forwarded so that the obstacle is bypassed along a minimum length route. In DSPIN-based approach, on the other hand, each router has information on its neighboring routers' states as well as the state of the routers at the corner positions to it. Based on the states of these eight routers, each router chooses one of its nine possible routing algorithms, providing a deadlock-free way of bypassing the faulty region.

In completely distributed control the routers make the decision in which direction the packet should be routed without any knowledge of the state of the network outside the links connected directly to the router. The routing decision is therefore based only on the location of the router in the network and the destination address of the packet. Distributed control is ideal from scaling perspective. The routing decisions are very simple, thus the realizations are small and power-efficient. The drawback is that the routes are not always optimal.

The most known distributed routing algorithm is *dimension order routing*, better known as *XY routing* (see Chapter 3). An extension to XY routing, called *dynamic xy routing* is presented in [55]. According to the algorithm packets are routed into a progressive x or y-direction, whichever has the smaller amount of other traffic. The algorithm is minimal, since only progressive routes are used. A similar approach is presented in [36], where also non-progressive routes are allowed.

Turn models are proposed for creating deadlock-free distributed routing algorithms [16, 27]. Performance of these algorithms has been primarily analyzed in the sense of handling congestion, but in that context the fault tolerance properties have been partially addressed. In [37] a method combining a deterministic and an adaptive routing algorithm based on *odd-even turn model* has been presented. The performance is once again analyzed mainly for handling congestion. Zhu *et al.* present a fault tolerant routing algorithm based on the *negative-first turn model* in [88], where the algorithm has been compared to XY routing and N-random walk.

Papers V and VI contain an analysis and comparison of the fault tolerance properties of distributed routing algorithms for a mesh NoC. Faults in a NoC communication infrastructure are modelled as faulty links, which can be done without any loss of generality since a faulty component, a router or a network interface, can be modelled by a set of faulty links. The number of erroneous links is altered and the location of errors is randomized in the sim-

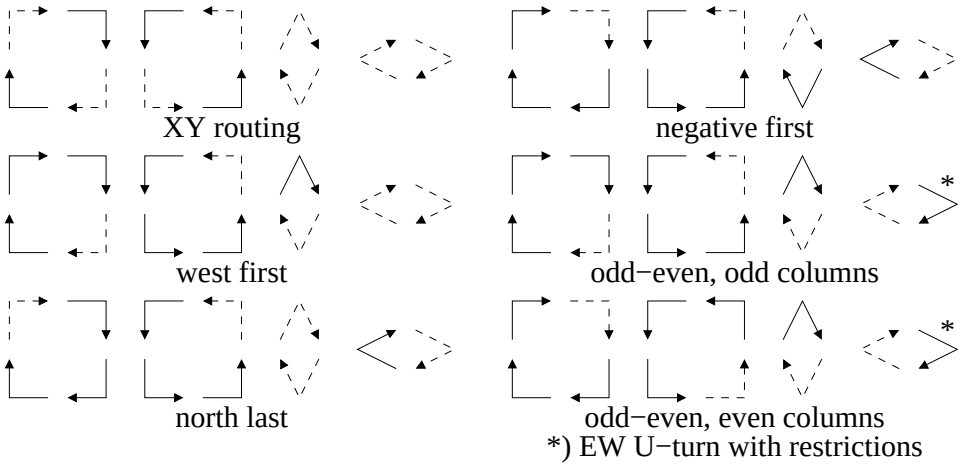


Figure 6.2: The allowed (solid line) and forbidden (dashed line) turns in each turn model.

ulations to obtain probabilities for correct transmission under different error scenarios. Furthermore, the changes in average hop count are monitored.

The analysis consist of classes of algorithms: minimal, deadlock-free, and non-deadlock-free. From the distributed algorithms mentioned above XY routing falls into minimal and deadlock-free classes and dynamic XY routing into minimal and non-deadlock-free algorithms. Obeying the turn models, four minimal and deadlock-free algorithms are constructed.

Paper V proposes a fully adaptive routing algorithm which targets for the maximal tolerance against link errors. The algorithm consists of a list of choices, in each of which a routing direction is selected. If a direction does not exist (first and last row and column) or the link is broken, the next choice in the list is tried. If no possible route is found, the packet is dropped. To achieve the best possible fault tolerance the requirement for minimal routing is relaxed.

The presented algorithm is modified to follow the turn models to produce fault tolerant deadlock-free algorithms. The algorithm cannot be always followed as such because of the special features of the turn models. The modifications are explained in detail in Paper VI. When non-minimal routes are considered also a U-turn is possible in some cases. In Figure 6.2 the allowed and forbidden turns of each turn model are repeated accompanied by the U-turns allowed in each turn model. Based on the proposed fully adaptive algorithm and respecting a turn model, four different routing algorithms were realized. Each of them maintains deadlock-freedom while striving for maximal fault tolerance.

None of the algorithms analyzed in Papers V and VI is superior in all properties. It is a matter of setting priorities to different aspects. While

the fully adaptive algorithm provides the best fault tolerance it does it with an increased hop count and without a guarantee for deadlock-freedom. Deadlock-free and minimal algorithms, on the other hand, fall behind in the fault tolerance. A good approach might be to handle some errors at the link level so that the requirements for fault tolerance at the network level can be relaxed. The fact that the link errors have to be somehow detected might guide the designer to this direction, since the same error control coding may be used for both link level protection and signaling to the network level about a faulty link. The deadlock avoidance or recovery methods besides turn models require buffering capacity and control structures. Both of them mean area and power overhead. The overheads introduced by the link level error protection may be smaller, which would justify the usage of deadlock-free algorithms instead of the others. Minimal routing algorithms are the only way to guarantee certain maximum hop-count. However, they need some additional fault tolerance structures for achieving acceptable reliability.

Chapter 7

Summary of Papers

7.1 Paper I: Analysis of Forward Error Correction Methods for Nanoscale Networks-on-Chip

The paper analyzes eight forward error correction approaches selected according to a set of requirements. The requirements include minimum error correction capability of two single errors and burst errors of length three, and a limit for the code rate to two thirds. Furthermore, all approaches must be suitable for an efficient on-chip realization. The analyzed approaches are based on Hamming, BCH, and Reed-Solomon codes and some of them also utilize interleaving. The approaches are implemented with VHDL, synthesized, and simulated to get performance values. Implementation details are included in the paper.

The analysis contains comparison of the coding approaches based on their error correction capability against multiple independent errors, burst errors, and combinations of them. The areas of the implementations are presented as well as the throughput and latency values for each approach. The combination of error correction capability and throughput, the effective throughput, reveals interesting details about the approaches, especially when scaled to a speculative nanoscale technology.

The power consumption values presented in the paper for the realizations are too optimistic because of an error in the used tool, which was noticed after the publication of the paper. However, the power values in the paper do not take into consideration the power consumed in signaling itself, so their usefulness in comparisons is anyway limited. This limitation has already been discussed in the paper.

7.2 Paper II: Online Reconfigurable Self-Timed Links for Fault Tolerant NoC

The paper proposes the utilization of spare wires and split transmissions for tackling intermittent and permanent faults in NoC links. These methods are further combined to error control coding which provides tolerance against transient faults. The error control coding approach is based on Hamming coding and ARQ. The detection of permanent faults by comparing consecutive error syndromes is proposed in this paper accompanied by a detailed description of its implementation. Furthermore, implementations of reconfiguration circuitry as well as a procedure for changing information between the receiver and transmitter is presented.

Asynchronous design style has been used in the realizations. Two-phase bundled data signaling is used in the link while the transmitter and receiver circuits are realized using a four-phase design style. The paper presents protocol converters to connect the link to the transmitter and receiver. In addition, it is shown how the transmission of reconfiguration information can be carried out delay-insensitively and by re-using the handshake signals already present in the system.

The spare wire and split transmission approaches are compared to two reference structures in a case study. One of the references does not have any fault tolerance properties while the other uses Hamming coding with ARQ. Comparisons are made in area, latency, throughput and power consumption.

The paper introduces also an enhanced fault model for links containing burst errors as well as intermittent and permanents errors in addition to single transient errors the earlier models were limited to. The implemented systems are compared using this fault model to find out their effectiveness against different types of errors.

Figure 5 at the fifth page of the paper has a small error. A corrected figure is included at the errata page found right after the paper in the second part of the thesis.

7.3 Paper III: Self-Adaptive System for Addressing Permanent Errors in On-Chip Interconnects

An adaptive system for detecting and bypassing permanent errors in on-chip interconnects is presented in the paper. The tolerance against permanent errors is achieved by introducing spare wires. Two alternative methods for detecting permanent errors are presented and analyzed. In the in-line test method periodic tests are run for each wire in a link. The paper derives the required test set and a look-up table for making a decision about the

faultiness of a wire based on the test results. The other detection method is the syndrome storing-based detection method, where the detection is based on the evaluation of consecutive error syndromes at the receiver. A mathematical method for finding the necessary design parameters for an implementation of syndrome storing-based detection is presented in the paper.

The paper proposes a link system that is capable of reconfiguring itself without stopping the normal operation of the link. Details about the reconfiguration logic, the reconfiguration control and the protocol for transmitting reconfiguration data between receiver and transmitter as well as synchronizing the reconfiguration are presented.

A case study analyzes the detection methods and two reference designs, one based on Hamming coding and the other on BCH coding. The paper presents comparisons in area, throughput, latency, power consumption and reliability. The research shows that protection against permanent errors can be achieved using spare wires with smaller penalties than using complex coding schemes.

7.4 Paper IV: Fault Tolerance Analysis of NoC Architectures

The paper explores the possibility to increase reliability by modifying the network topology. The number of network interfaces per resource is increased from one to two or four in a mesh topology, and the network is modified correspondingly. The modifications to the network are done in two distinct ways: by increasing the number of ports in each router and by utilizing minimum-size routers but increasing their number in the network. The modifications lead to total of six different mesh structures that are analyzed and compared.

The analysis is accomplished by counting the number of correctly operating cases of all the possible combinations with different number of erroneous components in a test structure consisting of two resources and the network around them. The reliability of different-sized components is normalized to the one of a minimum-size router by using an equation derived in the paper. Example routers with different number of ports and a network interface are implemented for obtaining the required area values. Asynchronous design style is used in the realizations.

The results of the analysis are further generalized to a larger NoC. According to the conclusions, the insertion of another network interface for each router increases significantly the reliability with a reasonable overhead. Incrementing the number of network interfaces further does not seem rational due to the small increase in reliability in contrast to large overheads. The structures created from minimum-size routers provide better performance.

7.5 Paper V: Fault Tolerant Distributed Algorithms for Mesh Networks-on-Chip

A fully adaptive routing algorithm for mesh NoCs is presented in the paper. The routing decisions following the proposed algorithm are based only on the location of the router in the network, the destination address of a packet, and the states of the links connected directly to the router. Thus, the algorithm is completely distributed.

In addition to the presented fully adaptive algorithm, ten other distributed algorithms are analyzed in fault tolerance and in hop count which gives an estimate about the latency. For the analysis a dedicated in-house simulator FANSI is used. The simulator was developed specifically for this purpose. The analyzed algorithms are divided to minimal and non-minimal as well as to deadlock-free and non-deadlock-free algorithms.

The paper concludes that only the presented fully adaptive algorithm performs acceptably when more than just a few faults occur in the system. The drawback is the larger hop count as the number of faults increases.

7.6 Paper VI: Analysis of Fault Tolerant Deadlock-free Routing Algorithms for Mesh NoCs

The paper presents four fault-tolerant deadlock-free routing algorithms with completely distributed control. A fully adaptive routing algorithm is combined with turn models to achieve the deadlock-freedom. Details of this combination with west-first, north-last, negative-first, and odd-even turn models are explained.

The developed algorithms are presented in a table form which can be directly used for hardware implementations. The analysis of the algorithms reveals that they provide clearly higher fault tolerance than XY routing which is used as a reference. The algorithm that uses north-last turn model achieves the best fault tolerance from the four developed fault tolerant deadlock-free algorithms.

Chapter 8

Conclusions

This thesis described fault tolerance methods for networks-on-chip. Presented methods include error control coding for on-chip signaling and solutions for tackling permanent and intermittent errors by the utilization of spare wires and split transmissions. Furthermore, fault tolerant network topologies and routing algorithms were studied. As background information, fault sources, fault classification and survey of fault tolerance methods were presented, along with the main points in the design of networks-on-chip.

Fault tolerance in a network-on-chip can be achieved via several methods at different abstraction levels or layers of the OSI reference model. Although a solution at some layer is designed to work with no restrictions to other layers, a good co-operation between layers may enhance the overall result. More importantly, a better overall solution can be achieved when the fault tolerance tasks are distributed among different layers.

Error control coding is an efficient fault tolerance method at the data link layer. An error control coding solution can utilize one of the two possible fault recovery methods: automatic repeat query or forward error correction. The selection between these two should be made with a careful understanding of the design choices made at the underlying physical layer. An important decision made at the physical layer is the selection between synchronous and asynchronous design styles. The asynchronous design style is motivated by its robustness mainly due to the lack of problems related to clock distribution. However, it may have a negative impact to throughput because every message needs to be acknowledged; a signal travels through a link twice per each message. Automatic repeat query requires an acknowledgement channel as does the asynchronous design style, and therefore combining those two is beneficial. Correspondingly, forward error correction suits well for the synchronous design style because neither of them requires backwards signaling. It is evident, that if the solution at the data link layer was selected without any consideration about the physical layer design choices,

the overall performance could be remarkably lower than achievable with a careful co-design of these two layers.

The data link layer fault tolerance solutions presented in this thesis combine multiple methods to achieve tolerance against all types of faults. Error control coding is proposed for tolerating transient faults, and two alternative methods, spare wires and split transmissions, are introduced for tackling the permanent and intermittent faults. The effectiveness of these approaches has been shown in the papers included in the thesis. However, as the number of simultaneous faults increases, the overhead in terms of area and power consumption becomes notable. This stands for both the coding approach as well as the reconfiguration circuitry complexity as the number of spare wires increases. This indicates that there is a limit of how much fault tolerance is efficient to accomplish at the data link layer before turning into the solutions available at the network layer.

Network layer solutions, on the other hand, are not efficient if the number of faults is high as was concluded in the papers included in the thesis. At the network layer larger parts of the NoC communication infrastructure are considered faulty at a time, hence a large number of such parts means that the overall performance of the system is endangered. This motivates the co-operation of the approaches at the data link and network layers. When most of the faults are handled at the data link layer, only few faults are left to be tolerated at the network layer. However, these faults are such that tolerance against them at the data link layer would be overly expensive.

Considering the presented approaches and facing the conclusions drawn above, there are a number of design choices that become evident. The fault detection required for the network layer approaches is provided by the solutions at the data link layer. Furthermore, the approaches at the network layer can be designed to tolerate a small number of errors. This enables e.g. the utilization of deadlock-free routing algorithms, and thus savings in the area of the router implementations.

Methods based on the modification of the network topology can also benefit from co-design of data link and network layer approaches. Some of the vulnerabilities of a topology can be hardened with the help of data link layer methods instead of inserting redundant parts to the topology. E.g. a stronger code and more spare wires could be used in the links connecting network interfaces to routers in a mesh topology if the introduction of redundant links is considered unpractical.

To summarize, an optimal fault tolerance solution is a carefully selected set of individual methods. The set consists of methods from different layers and methods targeted for all fault types. Furthermore, the selection of the methods is influenced by other design choices besides the reliability issues.

8.1 Future Work

As it is evident from the conclusions above, seamless co-operation of the fault tolerance methods introduced at different abstraction levels is essential for achieving the best performance for the overall system. Continuing the efforts on fitting together the approaches presented in the papers included in this thesis and the implementation details of such an overall system is an interesting topic for future work. Challenges include the trade-offs between reliability, performance, and power consumption. These requirements also vary greatly between applications.

The approaches presented in the included papers mostly omit the reliability of the circuitry introduced to accomplish the desired properties. As the technology scaling proceeds even further into the nanometer regime, these circuits can no longer be assumed fault-free. Therefore, solutions for the fault tolerance of encoders and decoders as well as controllers and routers are needed. This is another future work topic. The fault tolerance method chosen for these circuits should be optimally fitted to the fault tolerance approach the circuit is used for. An interesting aspect is for instance the possibility to use the same code that is used for the link error protection to the fault detection at the decoder as well.

Bibliography

[1] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino, "SPIN: a scalable, packet switched, on-chip micro-network," in *Design, Automation and Test in Europe Conference and Exhibition, DATE '03*, Munich, Germany, Mar. 2003, pp. 70–73.

[2] M. Ali, M. Welzl, M. Zwicknagl, and S. Hellebrand, "Considerations for fault-tolerant network on chips," in *17th International Conference on Microelectronics, ICM '05*, Islamabad, Pakistan, Dec. 2005, pp. 1–5.

[3] S. Almukhaizim and Y. Makris, "Fault tolerant design of combinational and sequential logic based on a parity check code," in *18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT '03*, Boston, MA, Nov. 2003, pp. 563–570.

[4] Arteris. [Online]. Available: http://www.arteris.com/

[5] R. Barsky and I. A. Wagner, "Reliability and yield: a joint defect-oriented approach," in *19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT '04*, Cannes, France, Oct. 2004, pp. 2–10.

[6] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[7] D. Bertozzi and L. Benini, "Xpipes: a network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits and Systems Magazine*, vol. 4, no. 2, pp. 18–31, Apr.–Jun. 2004.

[8] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: The energy-reliability tradeoff," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 818–831, Jun. 2005.

[9] T. Bjerregaard and J. Sparsø, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip,"

in *Design, Automation and Test in Europe Conference and Exhibition, DATE '05*, Munich, Germany, Mar. 2005, pp. 1226–1231.

[10] R. E. Blahut, *Algebraic Codes for Data Transmission*. Cambridge, UK: Cambridge University Press, 2003.

[11] C. Bobda *et al.*, "DyNoC: A dynamic infrastructure for communication in dynamically reconfugurable devices." in *International Conference on Field Programmable Logic and Applications*, Tampere, Finland, Aug. 2005, pp. 153–158.

[12] P. Bogdan, T. Dumitraş, and R. Marculescu, "Stochastic communication: A new paradigm for fault-tolerant networks-on-chip," *VLSI Design*, vol. 2007, pp. 1–17, 2007, article ID 95348.

[13] A. Bogliolo, M. Favalli, and M. Damiani, "Enabling testability of fault-tolerant circuits by means of I_{DDQ}-checkable voters," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 4, pp. 415–419, Aug. 2000.

[14] J. M. Cazeaux, D. Rossi, and C. Metra, "New high speed CMOS self-checking voter," in *10th IEEE International On-Line Testing Symposium, IOLTS '04*, Funchal, Madeira Island, Portugal, Jul. 2004, pp. 58–63.

[15] C. W. Chiou and T. C. Yang, "Self-purging redundancy with adjustable threshold for tolerating multiple module failures," *Electronics Letters*, vol. 31, no. 11, pp. 930–931, May 1995.

[16] G.-M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729–738, Jul. 2000.

[17] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, Jul./Aug. 2003.

[18] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: a novel on-chip communication network," in *2004 International Symposium on System-on-Chip*, Tampere, Finland, Nov. 2004, p. 15.

[19] W. J. Dally and J. W. Poulton, *Digital Systems Engineering*. Cambridge, UK: Cambridge University Press, 1998.

[20] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *38th Design Automation Conference, DAC 2001*, Las Vegas, NV, Jun. 2001, pp. 684–689.

[21] ——, *Principles and Practices of Interconnection Networks*. San Fransisco, CA: Morgan Kaufmann Publishers, 2004.

[22] G. De Micheli and L. Benini, *Networks on Chips: Technology and Tools (Systems on Silicon)*. San Fransisco, CA: Morgan Kaufmann, 2006.

[23] A. Ejlali, B. M. Al-Hashimi, P. Rosinger, and S. G. Miremadi, "Joint consideration of fault-tolerance, energy-efficiency and performance in on-chip networks," in *Design, Automation and Test in Europe Conference and Exhibition, DATE '07*, Nice, France, Apr. 2007, pp. 1–6.

[24] M. H. Etzel and W. K. Jenkins, "Redundant residue number systems for error detection and correction in digital filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 5, pp. 538–545, Oct. 1980.

[25] M. Favalli and C. Metra, "TMR voting in the presence of crosstalk faults at the voter inputs," *IEEE Transactions on Reliability*, vol. 53, no. 3, pp. 342–348, Sep. 2004.

[26] D. J. Frank *et al.*, "Device scaling limits of Si MOSFETs and their application dependencies," in *Emerging Nanoelectronics: Life with and after CMOS, vol. 1*, A. M. Ionescu and K. Banerjee, Eds. Norwell, MA: Kluwer Academic Publishers, 2005.

[27] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *Journal of ACM*, vol. 41, no. 5, pp. 874–902, Sep. 1994.

[28] K. Goossens, J. Dielissen, and A. Rădulescu, "Ætheral network on chip: Concepts, architectures, and implementations," *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 414–421, Sep.–Oct. 2005.

[29] C. Grecu, A. Ivanov, R. Saleh, and P. P. Pande, "NoC interconnect yield improvement using crosspoint redundancy," in *21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT '06*, Arlington, VA, Oct. 2006, pp. 457–465.

[30] ——, "Testing network-on-chip communication fabrics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 12, pp. 2201–2214, Dec. 2007.

[31] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Design, Automation and Test in Europe Conference and Exhibition, DATE '00*, Paris, France, Mar. 2000, pp. 250–256.

[32] F. Hanchek and S. Dutt, "Methodologies for tolerating cell and interconnect faults in FPGAs," *IEEE Transactions on Computers*, vol. 47, no. 1, pp. 15–33, Jan. 1998.

[33] C. Hawkins, A. Keshavarzi, and J. Segura, "A view from the bottom: nanometer technology AC parametric failures—why, where, and how to detect," in *18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT '03*, Boston, MA, Nov. 2003, pp. 267–276.

[34] A. Hemani *et al.*, "Network on a chip: An architecture for billion transistor era," in *18th NORCHIP Conference*, Turku, Finland, Nov. 2000, pp. 166–173.

[35] M. Honda, H. Harada, and M. Fujise, "Design of fault-tolerant digital filters based on redundant residue number arithmetic for over-the-air reconfiguration in software radio communication systems," in *IEEE 55th Vehicular Technology Conference*, vol. 1, Birmingham, AL, May 2002, pp. 280–284.

[36] A. Hosseini, T. Ragheb, and Y. Massoud, "A fault-aware dynamic routing algorithm for on-chip networks," in *IEEE International Symposium on Circuits and Systems, ISCAS 2008*, Seattle, WA, May 2008, pp. 2653–2656.

[37] J. Hu and R. Marculescu, "DyAD—smart routing for networks-on-chip," in *41st Design Automation Conference, DAC 2004*, San Diego, CA, Jun. 2004, pp. 260–263.

[38] International technology roadmap for semiconductors, 2006. [Online]. Available: http://public.itrs.net/

[39] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*. Boston, MA: Addison-Wesley, 1989.

[40] T. Karnik, P. Hazucha, and J. Patel, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, pp. 128–143, Apr.-Jun. 2004.

[41] M. D. Krstic, M. K. Stojcev, G. L. Djordjevic, and I. D. Andrejic, "A mid-value select voter," *Microelectronics Reliability*, vol. 45, no. 3-4, pp. 733–738, Mar./Apr. 2005.

[42] S. Kumar *et al.*, "A network on chip architecture and design methodology," in *IEEE Computer Society Annual Symposium on VLSI, ISVLSI '02*, Pittsburgh, PA, Apr. 2002, pp. 105–112.

[43] V. V. Kumar and J. Lach, "Heterogeneous redundancy for fault and defect tolerance with complexity independent area overhead," in *18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT '03*, Boston, MA, Nov. 2003, pp. 571–578.

[44] P. K. Lala, *Self-Checking and Fault-Tolerant Digital Design*. San Fransisco, CA: Morgan Kaufmann Publishers, 2001.

[45] G. Latif-Shabgahi, J. M. Bass, and S. Bennett, "Efficient implementation of inexact majority and median voters," *Electronics Letters*, vol. 36, no. 15, pp. 1326–1328, Jul. 2000.

[46] ——, "Component-oriented voter model for dependable control applications," *Microprocessors and Microsystems*, vol. 25, no. 3, pp. 167–176, May 2001.

[47] ——, "History-based weighted average voter: a novel software voting algorithm for fault-tolerant computer systems," in *9th Euromicro Workshop on Parallel and Distributed Processing*, Mantova, Italy, Feb. 2001, pp. 402–409.

[48] ——, "A taxonomy for software voting algorithms used in safety-critical systems," *IEEE Transactions on Reliability*, vol. 53, no. 3, pp. 319–328, Sep. 2004.

[49] G. Latif-Shabgahi, S. Bennett, and J. M. Bass, "Smoothing voter: a novel voting algorithm for handling multiple errors in fault-tolerant control systems," *Microprocessors and Microsystems*, vol. 27, no. 7, pp. 303–313, Aug. 2003.

[50] G. Latif-Shabgahi and A. J. Hirst, "A fuzzy voting scheme for hardware and software fault tolerant systems," *Fuzzy Sets and Systems*, vol. 150, no. 3, pp. 579–598, Mar. 2005.

[51] G. R. Latif-Shabgahi, "A novel algorithm for weighted average voting used in fault tolerant computing systems," *Microprocessors and Microsystems*, vol. 28, no. 7, pp. 357–361, Sep. 2004.

[52] T. Lehtonen, P. Rantala, P. Isomäki, J. Plosila, and J. Isoaho, "An approach for analysing and improving fault tolerance in radio architectures," in *IEEE International Symposium on Circuits and Systems, ISCAS 2006*, Kos, Greece, May 2006, pp. 3414–3417.

[53] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. 34, no. 10, pp. 892–901, Oct. 1985.

[54] L. Li, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Adaptive error protection for energy efficiency," in *International Conference on Computer Aided Design, ICCAD '03*, San Jose, CA, Nov. 2003, pp. 2–7.

[55] M. Li, Q.-A. Zeng, and W.-B. Jone, "DyXY—a proximity congestion-aware deadlock-free dynamic routing method for network on chip," in *43rd ACM/IEEE Design Automation Conference, DAC 2006*, San Francisco, CA, Jul. 2006, pp. 849–852.

[56] P. Liljeberg, J. Plosila, and J. Isoaho, "Self-timed communication platform for implementing high-performance systems-on-chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 43–67, Oct. 2004.

[57] Z. Lu, W. Huang, M. R. Stan, K. Skadron, and J. Lach, "Interconnect lifetime prediction for reliability-aware systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 2, pp. 159–172, Feb. 2007.

[58] C. Metra, M. Favalli, and B. Riccò, "Compact and low power on-line self-testing voting scheme," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT '97*, Paris, France, Oct. 1997, pp. 137–145.

[59] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The Nostrum backbone—a communication protocol stack for networks on chip," in *17th International Conference on VLSI Design, VLSID '04*, Mumbai, India, Jan. 2004, pp. 693–696.

[60] S. Murali *et al.*, "Analysis of error recovery schemes for networks on chips," *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 434–442, Sep.–Oct. 2005.

[61] S. Murali, D. Atienza, L. Benini, and G. De Micheli, "A method for routing packets across multiple paths in NoCs with in-order delivery and fault-tolerance gaurantees." *VLSI Design*, vol. 2007, pp. 1–11, 2007, article ID 37627.

[62] E. Nilsson, "Design and implementation of a hot-potato switch in a network on chip," Master's thesis, Royal Institute of Technology, Stockholm, Sweden, Jun. 2002.

[63] M. Pirretti *et al.*, "Fault tolerant algorithms for network-on-chip interconnect." in *IEEE Computer Society Annual Symposium on VLSI, ISVLSI '04*, Lafayette, LA, Feb. 2004, pp. 46–51.

[64] J. M. Quintana, M. J. Avedillo, E. Rodríguez-Villegas, and A. Rueda, "Efficient νMOS realization of threshold voters for self-purging redundancy," in *13th Symposium on Integrated Circuits and Systems Design*, Manaus, Brazil, Sep. 2000, pp. 321–326.

[65] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*. Upper Saddle River, NJ: Prentice Hall, Inc., 1996.

[66] V. Rantala, T. Lehtonen, and J. Plosila, "Network on chip routing algorithms," Turku Centre for Computer Science (TUCS), Turku, Finland, Tech. Rep. 779, Aug. 2006.

[67] F. Refan, H. Alemzadeh, S. Safari, P. Prinetto, and Z. Navabi, "Reliability in application specific mesh-based NoC architectures," in *14th IEEE International On-Line Testing Symposium, IOLTS '08*, Rhodes, Greece, Jul. 2008, pp. 207–212.

[68] K. Reick *et al.*, "Fault-tolerant design of the IBM Power6 microprocessor," *IEEE Micro*, vol. 28, no. 2, pp. 30–38, Mar.–Apr. 2008.

[69] D. Rossi, P. Angelini, and C. Metra, "Configurable error control scheme for NoC signal integrity," in *13th IEEE International On-Line Testing Symposium, IOLTS '07*, Crete, Greece, Jul. 2007, pp. 43–48.

[70] D. Rossi, S. Cavallotti, and C. Metra, "Error correcting codes for crosstalk effect minimization," in *18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT '03*, Boston, MA, Nov. 2003, pp. 257–264.

[71] D. Rossi, C. Metra, A. K. Nieuwland, and A. Katoch, "Exploiting ECC redundancy to minimize crosstalk impact," *IEEE Design and Test of Computers*, vol. 22, no. 1, pp. 59–70, Jan.–Feb. 2005.

[72] A. Schmid and Y. Leblebici, "A highly fault tolerant PLA architecture for failure-prone nanometer CMOS and novel quantum device technologies," in *19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT '04*, Cannes, France, Oct. 2004, pp. 39–47.

[73] ——, "Realisation of multiple-valued functions using the capacitive threshold logic gate," *IEE Proceedings - Computers and Digital Techniques*, vol. 151, no. 6, pp. 435–447, Nov. 2004.

[74] ——, "Regular array of nanometer-scale devices performing logic operations with fault-tolerance capability," in *4th IEEE Conference on Nanotechnology*, Munich, Germany, Aug. 2004, pp. 399–401.

[75] ——, "Robust circuit and system design methodologies for nanometer-scale devices and single-electron transistors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 11, pp. 1156–1166, Nov. 2004.

[76] M. Sgroi *et al.*, "Addressing the system-on-a-chip interconnect woes through communication-based design," in *38th Design Automation Conference, DAC 2001*, Las Vegas, NV, Jun. 2001, pp. 667–672.

[77] L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *9th International Symposium on High-Performance Computer Architecture, HPCA '03*, Anaheim, CA, Feb. 2003, pp. 91–102.

[78] S. R. Sridhara and N. R. Shanbhag, "Coding for system-on-chip networks: A unified framework," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 6, pp. 655–667, Jun. 2005.

[79] STMicroelectronics. [Online]. Available: http://www.st.com/

[80] M. K. Stojcev, G. L. Djordjevic, and M. D. Krstic, "A hardware mid-value select voter architecture," *Microelectronics Journal*, vol. 32, no. 2, pp. 149–162, Feb. 2001.

[81] N. Storey, *Safety-Critical Computer Systems*. Harlow, UK: Prentice Hall, 1996.

[82] W. J. Townsend, J. A. Abraham, and P. K. Lala, "On-line error detecting constant delay adder," in *9th IEEE International On-Line Testing Symposium, IOLTS '03*, Kos Island, Greece, Jul. 2003, pp. 17–22.

[83] R. B. Wells, *Applied Coding and Information Theory for Engineers*. Upper Saddle River, NJ: Prentice Hall, Inc., 1999.

[84] H.-S. P. Wong, D. J. Frank, P. M. Solomon, C. H. J. Wann, and J. J. Welser, "Nanoscale CMOS," in *Emerging Nanoelectronics: Life with and after CMOS, vol. 1*, A. M. Ionescu and K. Banerjee, Eds. Norwell, MA: Kluwer Academic Publishers, 2005.

[85] F. Worm, P. Ienne, P. Thiran, and G. De Micheli, "A robust self-calibrating transmission scheme for on-chip networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 1, pp. 126–139, Jan. 2005.

[86] A. J. Yu and G. G. F. Lemieux, "Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement," in *International Conference on Field Programmable Logic and Applications*, Tampere, Finland, Aug. 2005, pp. 255–262.

[87] Z. Zhang, A. Greiner, and S. Taktak, "A reconfigurable routing algorithm for a fault-tolerant 2D-mesh network-on-chip," in *45th ACM/IEEE Design Automation Conference, DAC 2008*, Anaheim, CA, Jun. 2008, pp. 441–446.

[88] H. Zhu, P. P. Pande, and C. Grecu, "Performance evaluation of adaptive routing algorithms for achieving fault tolerance in NoC fabrics," in *IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2007*, Montreal, Que., Jul. 2007, pp. 42–47.

[89] R. E. Ziemer and R. L. Peterson, *Introduction to Digital Communication*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2001.

[90] H. Zimmer and A. Jantch, "A fault model notation and error-control scheme for switch-to-switch buses in a network-on-chip," in *First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS '03*, Newport Beach, CA, Oct. 2003, pp. 188–193.

[91] H. Zimmermann, "OSI reference model—the ISO model of architecture for open systems interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, Apr. 1980.

Included papers

I

Paper I

Analysis of Forward Error Correction Methods for Nanoscale Networks-on-Chip

Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila

Published in: *Proceedings of the 2nd International Conference on Nano-Networks, Nano-Net 2007*, Catania, Italy, September 2007, pages 1–5.

Paper II

II

Online Reconfigurable Self-Timed Links for Fault Tolerant NoC

Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila

Published in: *VLSI Design*, vol. 2007, Article ID 94676, 2007, pages 1–13.

Paper III

III

Self-Adaptive System for Addressing Permanent Errors in On-Chip Interconnects

Teijo Lehtonen, David Wolpert, Pasi Liljeberg, Juha Plosila, and Paul Ampadu

To appear in: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Digital Object Identifier 10.1109/TVLSI.2009.2013711.

Paper IV

IV

Fault Tolerance Analysis of NoC Architectures

Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila

Published in: *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems, ISCAS 2007*, New Orleans, LA, May 2007, pages 361–364.

Paper V

Fault Tolerant Distributed Algorithms for Mesh Networks-on-Chip

V

Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila

Published in: *Proceedings of the 9th International Symposium on Signals, Circuits and Systems, ISSCS 2009*, Iasi, Romania, July 2009, pages 149–152.

Paper VI

Analysis of Fault Tolerant Deadlock-free Routing Algorithms for Mesh NoCs

Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila

VI

Published in: *Digest of the 3rd Workshop on Diagnostic Services in Network-on-Chips, DSNOC '09*, Nice, France, April 2009, pages 54–57.

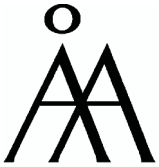
Turku Centre for Computer Science
TUCS Dissertations

87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Mehtods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics



Åbo Akademi University
- Department of Information Technologies

Turku School of Economics
- Institute of Information Systems Sciences

ISBN 978-952-12-2355-6
ISSN 1239-1883

Teijo Lehtonen

On Fault Tolerance Methods for Networks-on-Chip

Teijo Lehtonen

On Fault Tolerance Methods for Networks-on-Chip