

**Fine-Grained Fault-Tolerance :
Reliability as a Fungible Resource**

by

François Impens

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

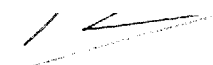
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2004

© Massachusetts Institute of Technology 2004. All rights reserved.

Author



...

Department of
Electrical Engineering and Computer Science
January 15, 2004

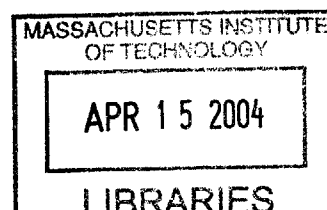
Certified by

.....

Isaac L. Chuang
Associate Professor of Media Arts and Sciences
and Department of Physics
Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Students



BARKER

Fine-Grained Fault-Tolerance :
Reliability as a Fungible Resource

by

François Impens

Submitted to the Department of
Electrical Engineering and Computer Science
on January 15, 2004, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

The traditional design of logic circuits, based on reliable components, is incompatible with the next generation of devices relying on fewer resources and subject to high rates of soft errors. These allow a trade-off between failure probability and their space and power consumption. Using this, we show that reliability can be a fungible resource, interconvertible with other physical resources in multiple, unusual ways, via fault-tolerant architectures. This thesis investigates the potentialities offered by a fault-tolerant design in devices whose reliability is limited by shrinking resources. Surprisingly, we find that an appropriate use of structured redundancy could lead to more efficient components. The performance of a fine-grained multiplexed design can indeed be systematically evaluated in terms of resource savings and reliability improvement. This analysis is applied to characterize technologies at the nano scale, such as molecular electronics, which may benefit enormously by fault-tolerant designs.

Thesis Supervisor: Isaac L. Chuang
Title: Associate Professor of Media Arts and Sciences
and Department of Physics

Acknowledgments

I thank my supervisor Prof. Isaac Chuang for his guidance and support. While Prof. Chuang always showed a great interest in the progress of my research, he was also patient enough to let me explore other fields providing a useful perspective on the subject.

Thanks to Pr. Sharpeskar for useful discussions on the noise model in CMOS transistors.

I would also like to thank my fellow labmates for their support and their help. Ken was always available to discuss research problems, to read a version of my thesis or to give technical advices, giving away generously a considerable amount of his time. Andrew, Setso, Aram, Rob and Zilong read different drafts of this work, providing me with a valuable feedback. Special thanks to Andrew and Setso for enjoyable companionship in long nights of labor, and for enduring in silence my musical tastes.

Thanks to my friends Shelly, Olivier, and to my parents for being very supportive.

Thanks to the groups Buena Vista Social Club and Madre Deus whose great music filled this room with an inspiring atmosphere as I was writing.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Contributions	2
1.3	Outline	3
2	Fault-Tolerance	
	in Combinational Systems	5
2.1	Introduction to Fault-Tolerant Computing	5
2.1.1	Summary of previous work	6
2.1.2	Purpose of fault-tolerance	7
2.1.3	Error correction techniques	9
2.2	Fault-Tolerant Design Guidelines	11
2.2.1	Computation on encoded codewords	11
2.2.2	Containment of error propagation	13
2.3	Mathematical Framework	17
2.3.1	Random combinational circuits	17
2.3.2	Fault-Tolerance	22
2.4	Algorithmic-based fault-tolerance: example of the encoded polynomial multiplication	25
2.5	Conclusion	30
3	Modular Redundancy-Based	
	Fault-Tolerant Architectures	31

3.1	Introduction	32
3.2	Triple Modular Redundancy with Perfect Majority Voting	34
3.2.1	Failure probability of a triplicated system	35
3.2.2	Cascaded Triple Modular Redundancy	37
3.2.3	Threshold Theorem	40
3.3	Modular Redundancy with imperfect majority voting	46
3.3.1	Reliability improvement through imperfect majority voting	47
3.3.2	Distributed Majority Voting	51
3.3.3	Recursive multiplexing	56
3.3.4	Reliability and resource consumption in recursively multiplexed circuits	61
3.4	Conclusion: Threshold Theorem with imperfect majority voting	64
4	Fined Grained Fault-Tolerance: Qualitative Approach	67
4.1	Resource-dependent reliability: Example of the CMOS transistor	68
4.1.1	Notion of noise	68
4.1.2	Resulting error probability	72
4.2	Problem Statement	74
4.2.1	Fundamental question and hypothesis	74
4.2.2	Two first examples	76
4.2.3	Framework	78
4.2.4	Two different points of view	81
4.3	Initial Solutions	83
4.3.1	Efficiency of a fault-tolerant design for reliability laws with residual error	83

4.3.2	Efficiency condition of a multiplexed triple modular redundant design	85
4.3.3	Equivalence of the optimization with limited resource availabil- ity and with reliability requirement	86
4.4	Reliability is a fungible resource	88
4.5	Conclusion	90
5	Fine-grained Fault-Tolerance: Quantitative Analysis	93
5.1	Systematic approach of fault-tolerant optimization	94
5.1.1	Identification of the optimal areas	94
5.1.2	Simplification of the Triple Modular Redundancy Efficiency Condition	97
5.1.3	Quantitative measures of efficiency: Definition and evaluation	98
5.2	Comparison of reliability-resource laws	99
5.2.1	Invariance of the efficiency towards dilatation of the reliability-resource law	100
5.2.2	Comparison of single parameter reliability-resource laws	101
5.2.3	Comparison of reliability laws with different resources	105
5.3	Treatment of specific reliability-resource laws	106
5.3.1	Representation of the efficiency of a Multiplexed Triple Modular Redundant design	107
5.3.2	Power laws: $p(A) = 1/A^\gamma$	108
5.3.3	Exponential laws: $p(A) = \frac{1}{2} \exp(-A^\gamma)$	111
5.4	CMOS Transistor	114
5.4.1	Reliability laws $p(A) = \frac{1}{2} \operatorname{erfc}(A^\gamma)$	114

5.4.2	Relation to the fault-tolerant implementability of a CMOS transistor	118
5.5	Conclusion	120
6	Conclusion	121
6.1	Summary and contributions	121
6.2	Open Problems	122
6.3	Relation to emerging technologies	123

List of Figures

2-1	Executive organ of a NAND gate encoded in the triple repetition code.	12
2-2	Action of an encoded gate on a set of words.	15
3-1	Simple majority voting system.	35
3-2	Double majority voting system.	38
3-3	External and internal modular design.	45
3-4	Recursive majority voting.	47
3-5	Distributed majority voting.	52
3-6	Sequence of two encoded NAND gates.	54
3-7	Distributed majority voting system taking imperfect inputs.	54
3-8	Multiplexed NAND of recursion level 2.	58
3-9	Recursive multiplexing of a NAND gate.	60
3-10	Structure of the code : ternary tree of depth k	60
4-1	Two alternative resource allocations for a NAND gate of layout area A_T .	75
4-2	Fault-tolerant design in steep and flat reliability laws.	77
4-3	General reliability-resource law.	79
4-4	Reliability-resource law with residual error.	84
4-5	Possible trajectories of the phase point.	89
4-6	Accessible phase space for a single resource reliability law.	90
4-7	Accessible phase space for a reliability law with several resources. . .	91
5-1	Comparison of two reliability-resource laws for a single reliability requirement ϵ	101

5-2	Critical exponent for the family of power laws $p(A) = A^{-\gamma}$ as a function of the reliability.	108
5-3	Maximum exponents compatible with a certain resource gain for the family of power laws.	109
5-4	Reliability obtained through optimized and direct allocation for the power law $p(A) = 1/A^2$	110
5-5	Critical exponent for the family of exponential laws $p(A) = \frac{1}{2} \exp(-A^\gamma)$ as a function of the reliability requirement.	111
5-6	Maximum exponents compatible with a certain resource gain for the family of exponential laws.	112
5-7	Reliability obtained through optimized and direct allocation for the exponential law $p(A) = \frac{1}{2} \exp(-A^{1/4})$	113
5-8	Critical exponent for the family of laws $p(A) = \frac{1}{2} \text{erfc}(A^\gamma)$	115
5-9	Maximum exponents compatible with a certain resource gain for the family of laws $p(A) = \frac{1}{2} \text{erfc}(A^\gamma)$	116
5-10	Reliability obtained through optimized and direct allocation for the law $p(A) = \frac{1}{2} \text{erfc}(A^{0.15})$	117
5-11	Range of efficiency for multiplexed triple modular redundancy	120

Chapter 1

Introduction

1.1 Motivation

If we are to face the coming challenges of information technology we need, today more than ever, new resource-efficient design methodologies.

Reliability and resource consumption are intimately connected in any real computing system. So far the strategy of manufacturers has been to allocate enough resources to each component until its failure probability becomes negligible for all practical purposes. Reliability comes at a certain price, and for each device there is a trade-off between deficiency and resource consumption. The current industrial choice, which is to saturate this relation for any part of the system, is likely based on the belief that the disorder arising from the elementary components at the small scale level would be amplified at the high scale level and lead to unreliable computations.

Unfortunately this policy is not compatible anymore with the coming technologies, which require that components rely always on fewer resources. As far as near-term technologies are concerned, the consumption of space and power is becoming a major issue. The electronics industry, in its struggle to keep up with the Moore's law predictions, is now engaged in a chip miniaturization process which will prove extremely challenging as the dimensions of the devices get close to those of molecules. As for power consumption, we live in a very energy-conscious era, and the increasing demand on electronic appliances with high autonomy strongly calls for low-power schemes. In

a longer term perspective, the continuous shrinking of chip layouts will make them extremely sensitive to external perturbations such as cosmic rays and highly subject to operational errors. Innovative designs capable of dealing with high rates of soft errors at a cost of a minimum overhead will then be extremely useful.

Furthermore, two elements strongly suggest that the policy of building systems exclusively out of reliable components is not optimal. The first point would be that natural computational systems work differently. The human brain, for instance, makes use of a great number of neurons whose answer is of statistical nature. The second argument is mathematical. By imposing a high reliability at any scale, although only the high scale matters, we do not take advantage of statistical properties predicted by the law of large numbers. From an information-theoretic point of view, we gain a great amount of unnecessary information. Indeed, if a device is perfectly reliable, then its behavior is predictable. In a circuit built out exclusively of ideally reliable components, the state of any device can be determined at any time on the basis of the system inputs. Such a knowledge is not useful, since the only relevant piece of information is the global answer to the computation.

1.2 Contributions

We investigate in this thesis an alternative design method, based on the introduction of a fault-tolerant structure at the fine-grain level in electronic circuits. In such a system, each elementary component exhibits a certain level of deficiency, and consumes in return less resources than if it were perfectly reliable. Our treatment differs from the previous work done on fault-tolerance, primarily concerned with the embedding of fixed faulty devices into reliable networks. It differs in nature, because the deficiency of the components is here conditioned by the mobilization of resources. The fault-tolerant architecture is internalized within the component itself and provides it with a structure optimizing its use of resources. It also differs in purpose, since the techniques we expose allow devices to rely on fewer resources, and may be relevant even in technologies with low failures rates.

The main contribution of this thesis is to systematically evaluate the efficiency of a modular redundant design operated at the fine-grain level in electronic components. The potentialities offered by fault-tolerance in terms of resource-efficient design could be largely underestimated. By showing that those constructions can potentially lead to better trades-off between area, power consumption and reliability than those currently used, we hope to help fill in this hole.

1.3 Outline

Chapter 2 and Chapter 3 review general material on fault-tolerance and modular redundancy. Chapter 4 and Chapter 5 present the original work accomplished in this thesis. The outline is the following:

Chapter 2 covers the general concepts and tools of fault-tolerance necessary to understand the strategy of the constructions presented afterwards. It also defines a mathematical framework for the encoded computation with faulty devices, appropriate to set the argumentation to come on a firm basis.

Chapter 3 exposes in detail a specific class of fault-tolerant architectures, based on modular redundancy. Their performance and cost for scalable circuits of encoded logic are evaluated.

Chapter 4 introduces the notion that reliability is a fungible resource and applies the fault-tolerant constructions of Chapter 3 in this new context. Initial results are derived, including a mathematical condition stating, for a given technology, whether modular redundancy can be efficiently introduced at the fine-grain level.

Chapter 5 defines a systematic procedure allowing one to evaluate the efficiency of a fine-grained modular redundant design. This method is applied to estimate the

gain brought by fault-tolerant schemes in specific instances including the case of a CMOS transistor.

An appendix gathers the definitions and acronyms used in this thesis.

Chapter 2

Fault-Tolerance in Combinational Systems

In this chapter we review the concepts of fault-tolerant computation and propose a mathematical framework for the treatment of probabilistic combinational circuits. Although we will focus in this thesis on the implementation of elementary logic components, which we can take as NAND gates without loss of generality, the techniques of fault-tolerance require the introduction of more general noisy circuits.

Section 2.1 is an introduction to the area of fault-tolerant computing. Section 2.2 exposes the strategy underlying the design of reliable combinational systems constituted of faulty elements. Section 2.3 introduces the formalism relative to noisy and encoded computations. Section 2.4 illustrates the concepts of this chapter through the example of fault-tolerant polynomial multiplication. Section 2.5 concludes by highlighting the important points developed in this chapter.

2.1 Introduction to Fault-Tolerant Computing

This section reviews the basic concepts involved in fault-tolerance as well as the design guidelines of efficient fault-tolerant combinational circuits. The ideas exposed here will guide us through the later development of this thesis. The interested reader might refer to the references of the next paragraph for a deeper insight in the field.

2.1.1 Summary of previous work

The work of Von Neumann [Neu56] initiated to a large extent the search for a theory of computation through unreliable elements. Later on, Elias [Eli58], Winograd and Cowan [WC63], and Taylor [Tay90] tried to apply the techniques of information theory [Sha48a, Sha48b] to protect computing systems from hardware faults. An important concern was the efficiency of the existing fault-tolerant architectures: Winograd and Cowan underlined in [WC63] that arbitrarily high levels of reliability in noisy circuits could so far only be achieved at the cost of an arbitrary high number of components, whereas in information theory a constant overhead could yield ideally reliable communication through noisy channels [Sha48a, Sha48b]. Later on, Pippenger [Pip85, Pip89] addressed this point by exhibiting certain boolean functions for which only a constant multiplicative overhead is necessary. A more general result on the efficiency of multiplexing was also proven: Dobrushin and Ortuyov [DO77b] and Pippenger [Pip85] improved theoretically the result of Von Neumann by showing that the computation of any boolean function through a noisy circuit can be done with an overhead factor which is logarithmic in the failure probability. Therefore the multiplexing techniques of Von Neumann [Neu56] may work effectively with a practically acceptable overhead.

Lower bounds on the required amount of redundancy have also been established. Dobrushin and Ortuyov [DO77a], Pippenger, Stamoulis and Tsitsiklis [PST91], and Gál [G85] proved that a multiplication of the number of components by a logarithmic factor is necessary for the computation of certain boolean functions with noisy gates. Hajek and Weller [HW91], Evans and Pippenger [EP98] determined the maximum amount of noise within logic gates compatible with the performance of reliable computations.

Worth mentioning also is the work of Gács [G86], who investigated the different problem of the network dimensionality. He showed that arbitrarily large computations were possible with one-dimensional cellular automata.

2.1.2 Purpose of fault-tolerance

Notion of Fault-Tolerant System

Many applications require the performance of computations with an arbitrarily low risk of failure. Unfortunately, the method which consists in ensuring the reliable behavior of all the components turns out to be inefficient when the system is complex or when the failures are caused by uncontrollable environmental factors. Fault-tolerance is an alternative strategy which, in lieu of trying to prevent the occurrence of failures, aims at designing systems robust to a certain error rate. In the language of computer science, we may say that the circuit is able to catch a certain number of exceptions. In the most general sense, when evoking the notion of fault-tolerance we mean the following property:

Definition 2.1.1 : *We shall say that a system is fault-tolerant when it preserves a desirable behavior under the occurrence of a certain rate of internal faults.*

The ability to preserve a reliable behavior in spite of a partial corruption of the assigned process is sometimes referred to as graceful degradation. Fault-tolerant techniques are in fact not limited to the field of reliable computing. Manufacturing processes, for instance, may involve automatons whose erroneous behavior could physically endanger the whole production chain and thus need to be protected by self-checking procedures.

Desirable behavior

By desirable behavior, for an automata, we mean that the system should stay within a restricted set of states labelled as “acceptable.” For a combinational circuit, this term signifies that it should deliver a correct output with high probability. A correct output is a word whose decoding leads to the result of a fault-free computation, the decoding being a many-one transformation fixed before the computation and independent from the input sequence. We will have the opportunity to go back to this assertion and specify it in the next paragraphs.

Internal Faults

By internal faults we mean that one of the components, within the system, has not performed the task it was assigned to. In general, two kind of failures may occur in a circuit:

- Permanent or hard failures, affecting a processor for an unlimited amount of time. The defect in the processor remains during future operations.
- Soft or transient errors, denoting a temporary malfunction of the processor. These errors last for a finite amount of time, and typically affect only one isolated computation.

The error model we adopt in this thesis is, however, more restrictive: *we consider only the occurrence of soft errors* and neglect the possibility of hard failures in the circuits.

Hard failures typically originate from manufacturing defects and can be limited pending sufficient technical improvements. Recent engineering work shows that these errors may also be managed through reconfiguration techniques [CAC⁺96], [HKSW98], [MSST00].

Soft errors are a greater concern for emerging technologies. Inherent to the device operation or to environmental influences, these random faults happen on a much faster time-scale than hard errors and cannot be fought by the same techniques. Thanks to the very high level of reliability met by current electronic devices (failure probability on the order of 10^{-16}), soft errors have not been a major issue so far, except for the operation in exceptional environmental conditions encountered in space engineering. The situation will be different with the next generation of components, whose dimensions will be on the nano-scale. The failure probability of those devices is more likely to be on the order of 10^{-4} . Indeed, the unreliability of single electron devices [Lik99] is already considered as a serious threat for their integration into large-scale circuits. In the area of quantum computing, decoherence intrinsically limits the reliability of elementary quantum gates: they have not been performed so far with a failure probability lower than 10^{-5} . In molecular electronics, cosmic rays might also

be an important source of operational errors. Unlike hard failures, because they are attached to physical limitations, transient errors will not be circumventable. In these near or long-term technologies, computing devices will thus enter a regime in which soft errors prevail. Their management will be essential for the integration of those devices into high-scale systems. This is why we deliberately focus in this thesis on the design of resource-efficient architectures *robust to soft errors only*, without taking hard failures into consideration.

We should also point out three other important assumptions relative to our error model:

- *No component* in the computational system is assumed to be *perfectly reliable*.
- Errors occurring within *different processors* are *independent* events.
- Processors fail *independently* from their *inputs*.

2.1.3 Error correction techniques

Necessity of Redundancy

The concept of fault-tolerance relies on the ability to recover the data corrupted through a computation, or to distinguish between valid and invalid states of an automata. This implies the existence of error correction techniques capable of restoring integrally a signal partially corrupted. Any of those requires the introduction of a certain amount of redundancy within the system. This principle seems at first in contradiction with the achievement of resource-efficient computing. In fact it is *not* because, as we shall see, reliability itself comes at a certain price. Nonetheless, in order to minimize the incurring overhead, the added redundancy should exhibit an appropriate structure and be added in places where errors are either likely to arise or critical. To implement this strategy, we may give to the redundancy the form of an encoding, and choose the code according to the nature of the task performed and the features of potential errors.

The work exposed in this thesis is relative to the efficient fault-tolerant implementation of elementary logic gates suitable for any general boolean computation. We must therefore adopt an encoding compatible with generality requirement. Previous work has been done concerning the protection through encoding of very specific computational tasks [Bec92], [Had02], branch of fault-tolerance referred to as algorithmic-based. We will examine later an example of reliability improvement through such a an error correcting code.

The encoding of logical bits into codewords is the standard technique to protect against their alteration in a noisy environment. The simplest code, the triple repetition, maps 0 to 000 and 1 to 111. The most natural decoding associates to words of three bits containing at least two 0 and two 1 respectively the single bit 0 and the single bit 1. This type of decoding is called majority voting.

Certain class of codes group several logical bits (words) at a time within a single codeword. A very commonly used family of codes having this property is the class of linear codes [McE76]. Their main advantage is that codewords can be generated and checked easily through a simple matrix multiplication. Linear codes indeed work along the following principle: they map contiguous words of logical bits into a bigger space, in which each codeword is isolated from the others. The distance d between those determines the maximum number of errors, compatible with a correct decoding, that may simultaneously affect a codeword. More precisely, in a code of minimum distance d , each codeword is surrounded by a ball of radius $\lfloor d/2 \rfloor$ which does not intersect with the balls corresponding to other codewords. Words corrupted with less than $\lfloor d/2 \rfloor$ errors can thus be recovered without any ambiguity through a decoding scheme projecting each ball onto its associated codeword. Linear codes are classified by the length of the encoded word, by the number of logical bits encoded in a codeword, and by the distance between them, information which is usually summed up into the triplet of integers (n, k, d) . These classical linear codes have been integrated in the definition of powerful quantum error correction codes [Ste96] which were used to prove the existence of fault-tolerant computing architectures [Sho96].

Separate and Systematic Codes

The degree of separation between the informative content of the codewords and the redundancy added for fault detection and correction is an important indicator of the easiness with which the encoding or decoding operations can be performed. In certain codes, the distinction between the informative and redundant part of the codewords is such that this separation appears in the encoding of the logic operands. These considerations motivate their classification along the following criterions:

Definition 2.1.2 : *A systematic error correcting code is a code whose codewords are composed of two distinct parts: the original word, and parity check symbols derived from it. A systematic error correcting code is separate if the encoded operations are performed separately on the original data and on the parity check symbols.*

The decoding is trivial in systematic codes, since it is possible to read directly the data in the first components of the codeword. In separate codes, the computation does not mix the added redundancy with the original data. The dynamic of the latter stays unchanged in the encoded computation and is readable on the first components of the encoded codewords. This is an important property.

2.2 Fault-Tolerant Design Guidelines

In order for the protection to be effective, the circuit must meet two essential requirements. First, the data must stay encoded at all times through the computational process. Second, the propagation of errors during the computation must be contained, so that the level of signal alteration remains sufficiently low for the correction techniques to be applicable.

2.2.1 Computation on encoded codewords

It is necessary, in order to protect the data against dynamic alteration at every stage of the computation, that the logic gates of a fault-tolerant circuit perform the

computation directly on encoded bits. In the case of the triple repetition code, it is sufficient to operate the original gate separately three times on similar sets of inputs to execute the computation while preserving the code structure(see figure 2-1 for the computation of a NAND gate). Taking the terms of Von Neumann [Neu56], we may call this ensemble of gates the *executive* organ associated with the encoding of the gate through the triple repetition code. It will reveal useful to couple this organ to an other system, taking advantage of the distance between the codewords to perform some error correction. But the latter should not operate any decoding, which would make the data vulnerable. It should simply take the word delivered by the executive organ and match it, with high probability of success, with the codeword encoding the expected result of the original computation. In the denomination of Von Neumann, this part of the circuit would be the *restoring* organ. From now on, we will call the assembly of these two components - the *executive* and the *restoring* organ - the encoded gate.

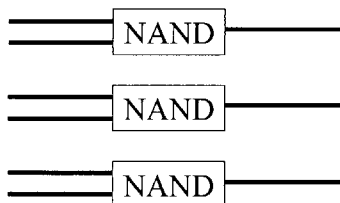


Figure 2-1: Executive organ of a NAND gate encoded in the triple repetition code. The data flows from the left to the right, each NAND gate takes two inputs and delivers one output. The six inputs correspond to two logical bits which have been previously encoded in the triple repetition code.

It is now time to attract the attention of the reader on an important assumption which will be made in the rest of this thesis. Since the operation of the boolean processor maps inputs encoded in a certain code to outputs similarly encoded, the encoding and decoding operations are only performed at the beginning and at the end of a chain of computations. In most applications, the depth of current logic circuits is such that the number of operations involved in this chain is several orders of magnitude greater than that required by a single encoding or decoding of the data.

Consequently, we assume that the cost associated with those operations is negligible. Since we focus on the resource-efficiency aspect of fault-tolerant computation, we will not detail the circuits performing the encoding and decoding. The encoding needs to be a *fixed* one to one transformation, mapping words of length k to words of length n . Otherwise, if it were to depend on the sequence of previous inputs, the encoder would have the ability to perform some computation. Since the encoder is a reliable device, this assumption would contradict our approach in which all components are subject to soft errors. The decoding is also *fixed*, but it is a many to one transformation. There are several possible decoding procedures associated with a single code. The choice of an appropriate decoding has an important impact on the reliability of the gate. When errors among bits are identically distributed, an optimal choice is to project words of length n onto the closest codeword - or one of the closest codewords if there are several - in the sense of the Hamming distance. We should mention, however, that this scheme is not necessarily the best if the occurrence of errors is more likely in certain parts of the codeword than in others. In this situation, a decoding taking into account the expected error locations may be more efficient.

2.2.2 Containment of error propagation

An important design requirement

A careful design of a fault-tolerant circuit should ensure that errors arising in some bits of the data through component failures do not propagate significantly to many other bits of the data stream. In order to maintain the noise level within the circuit under control, we require that it satisfy the property:

No single component failure, whether it happens in a wire or in a gate, should corrupt several bits within the same codeword at a time

This restriction will shape the form of the logic gate encoding. Indeed a random encoding of logic gates is likely to transfer an error occurring in a bit of a codeword to other bits of the same word.

Transversality

A very favorable situation in the respect of error propagation is obtained when the executive organ merely consists in a transversal replication of the original gate applied separately on the different bits of a codeword. Codes for which a universal set of operations is encodable in a bitwise manner have therefore a great advantage: not only do they minimize the overhead in complexity due to the encoding of logic operations (their executive organ, a transverse implementation of the original gate, is very simple), but they also provide a frame in which *error propagation is naturally contained*. The repetition codes fall into this category and thus deserve a particular attention.

Scaling of the output failure probability

While a simple logic gate may be described through an assignment of failures probabilities to each possible set of inputs, the behavior of an encoded gate is more complex. As an example, let us consider a code C mapping each bit to a word of length l distant of $d = 2t + 1$ from other words. According to our former analysis, C can therefore correct up to t errors. The output of an encoded gate simulating a boolean computation can be considered as correct as long as, when decoded, it yields the correct result of the non-encoded boolean computation. But the probability that the output falls in or out of the Hamming ball of radius t centered on the fault-free result is not sufficient to predict the effective reliability of such a gate embedded into real circuits. Let us see why.

During the performance of a long chain of boolean operations, the outputs of each encoded gate are sent into the next one. Each encoded gate contains an executive organ followed by a restoring organ. While a certain amount of error correction is performed within each encoded gate through the restoring organ, however, no error correction is performed *in between* two encoded gates. Therefore, in order to be consistent, we must accept that inputs have a certain probability to be distant

from perfectly encoded codewords in the ordinary operation of the encoded gate (see figure 2-2).

Furthermore, the probability that an encoded gate fails in the sense that “the decoding of the output does not lead to the result” is higher if its inputs are already corrupted. This fact implies that, in order to describe an encoded gate, one should know:

- The probability that the gate leads to a wrong result under imperfectly encoded inputs.
- The probability that the gate delivers a partially corrupted output, in the sense of correct but not within the codewords.

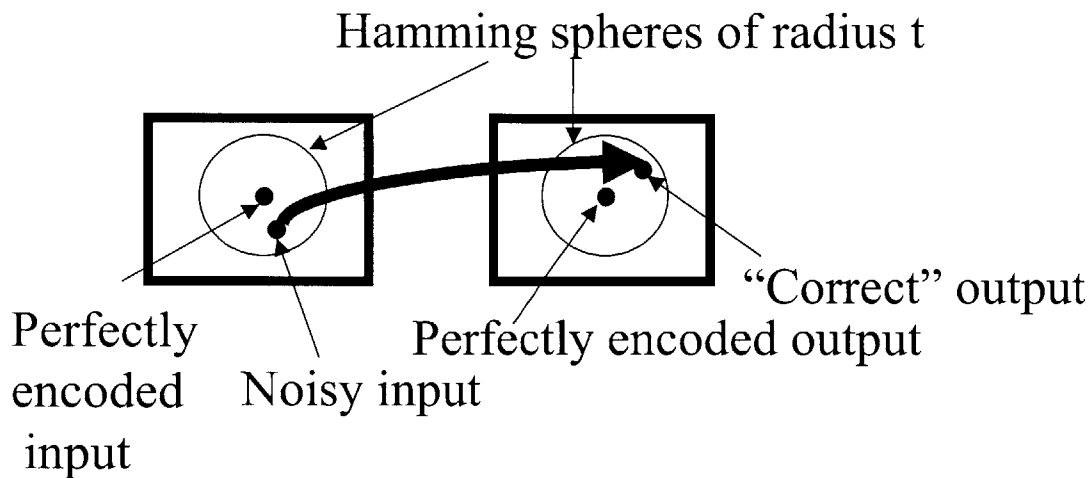


Figure 2-2: Action of an encoded gate on a set of words. A full description of an encoded gate must include its action on the set of words within a Hamming ball of radius $t = \lceil \frac{d}{2} \rceil$ centered on the perfectly encoded inputs. This set corresponds to partially corrupted inputs, which are also possible correct outputs of previous encoded gates.

We can now appreciate the benefits of the design requirement introduced in this section. Let us imagine a computation in which two inputs, each representing an encoded logical bit with at most u corrupted bits, are sent into an encoded NAND gate which delivers in output a possibly corrupted codeword. The encoded gate is

indeed composed of elementary gates taking single bits, and failing with a probability bounded by p . A decisive point is now: how many of these simple logical devices have to fail altogether in the computation in order to generate more than v errors in the output? Because of the condition on error propagation, each corrupted bit can be traced back to a unique component failure, and this quantity is therefore $v - u$. We obtain then the essential property:

The probability that v bits are corrupted within the output, knowing that u bits or less are corrupted in each input, is on the order of p^{v-u} .

In particular, the failure probability of the computation as a whole is on the order:

$$p_{fail} \leq c p^t = O(p^t) \tag{2.1}$$

where c is the number of possible locations where t failures may appear within the circuit. If the components are fairly reliable, so that $p \ll 1$, the gain in reliability through the encoding is great. Furthermore, in order to compute the effective reliability of the gate in a network of similar gates, it is sufficient to treat each of its inputs as components failing with a probability p .

Block Computation

In certain codes, such as linear codes, several logical bits are encoded within the same codeword. The operation of logic gates on such codewords is referred to as block computation [Eli58]. Let us consider such a computation involving only two inputs-one output boolean functions. We may then distinguish between two kind of operations:

- Operations on two logical bits encoded within the same codeword.
- Operations on two logical bits situated in different codewords.

A failure in the logic operation of the first kind, which interacts two bits of the same codeword, may endanger several bits within this single codeword. Besides, a

fixed encoding should allow one to perform both type of operations, which seems challenging. Indeed the encoding of logic gates performing block computation is not trivial at all. We will use in this thesis the simpler approach in which each codeword represents a single logical bit.

2.3 Mathematical Framework

We define in this section the essential terms necessary for the rigorous treatment of fault-tolerant systems. The language introduced here will be useful in the later developments of this thesis.

Our first task will be to define the logic gates or processors present in the circuit as mathematical objects. We then introduce the notion of combinational circuits. Finally we specify the notions of encoded computation and fault-tolerance in those circuits. Although we will try, whenever possible, to insist on the concrete implications of those definitions, some abstraction is necessary to perform a rigorous approach of the reliability within probabilistic boolean circuits.

2.3.1 Random combinational circuits

Notion of Processor

A natural framework for the treatment of faulty boolean circuits seems to be the theory of automata. The building blocks of such circuits, the processors, need to be to some extent unpredictable. A faulty processor could then be an automata whose output is a set of random variables correlated to an other set of random variables called input. This was the approach followed by Von Neumann [Neu56]. But the use of automata for the description of combinational circuits has the disadvantage of introducing the additional problem of synchronization, irrelevant here because the output may be entirely determined by the input of the circuit.

Von Neumann showed [Neu56] that the issue of synchronization maybe circumvented by using logic gates automata as delays in the circuit, leading to an overhead

in its complexity. In fact, the great majority of the architectures investigated in this thesis will be naturally synchronized: the path of each particular input to an outputs of the circuit will be of equal length in terms of number of operations. Nonetheless, for sake of simplicity, we will assume that any gate automatically waits for all the input signals to arrive before performing the logic operation. This suggests the following definition for a processor:

Definition 2.3.1 : *A processor P taking m inputs and delivering n outputs is an automata whose inputs and outputs are given respectively by the set of random vectors $\mathbf{X} = (X_1, \dots, X_m)$ and $\mathbf{Y} = (Y_1, \dots, Y_n)$ following the properties:*

- *These random vectors are functions of time may be sampled only at $t = 0, 1, 2, \dots$*
- *If t is the first instant in which all inputs have been sampled, the output vector is sampled at the time $t + 1$*
- *The distribution of \mathbf{Y} at $t+1$ is conditioned to that of X at t and is independent of t .*

In other words, processors are unit-delay deficient automaton, memorizing the inputs arriving at different times, and whose behavior is stationary during the computation. But we will consider only combinational circuits and focus merely on the result of the computation, without interest for the transient state of the different processors involved. From this perspective the use of automaton for their description seems superfluous and we may define processors directly in the frame of instantaneous random variables. Furthermore, we can specialize to single-output processors.

We shall note Ω the universe of events (with a measure p called probability) and $\mathbf{R}(\{0, 1\}^m)$ the space of random vectors of $\{0, 1\}^m$. We remind that random vectors X of this space are functions mapping Ω to $\{0, 1\}^m$ such that:

$$\forall \mathbf{x} \in \{0, 1\}^m \quad \mathbf{X} = \mathbf{x} \quad \text{is an event i.e.} \quad \mathbf{X}^{-1}(\mathbf{x}) \quad \text{is a measurable set of } \Omega \quad (2.2)$$

In the error model defined in section 2.1.2, errors occur independently in different processors and independently from their inputs. With these assumptions in hand, we can model the action of a noisy processor by the following relation between its inputs and its output:

Definition 2.3.2 : *A processor P computing $F : \{0, 1\}^m \rightarrow \{0, 1\}$ and delivering a single output is a map:*

$$G : \mathbf{R}(\{0, 1\}^m) \longrightarrow \mathbf{R}(\{0, 1\}) \quad (2.3)$$

$$(X_1, \dots, X_m) \longmapsto Y = F(X_1, \dots, X_m) + \delta \quad (2.4)$$

where δ is a random variable reflecting the errors occurring in the processor. δ is independent from the inputs (X_1, \dots, X_m) , and two variables δ associated with distinct processors are independent.

As we shall see in Chapter 4, the distribution of δ depends on the amount of resources allocated to the processor. We will also suppose that the error probability $p(\delta \neq 0)$ can be made arbitrarily small pending sufficient resource availability.

Definition 2.3.3 : *Two processors implement the same gate if they compute the same function F up to a composition with a permutation of their inputs. Two processors implementing the same gate are identical if the distribution of their respective random variables δ are equal.*

We now define the notion of deficient computation. Indeed we will distinguish between three regimes:

Definition 2.3.4 : *Let $\epsilon \in]0, 1/2[$ and P be a processor computing a function $F : \{0, 1\}^m \rightarrow \{0, 1\}$ in the sense of definition (2.3.2) .*

- *We shall say that a processor P weakly ϵ -computes F if and only if :*

$$\forall \mathbf{x} \in \{0, 1\}^m \quad p(\delta = 1 | \mathbf{X} = \mathbf{x}) \leq \epsilon \quad (2.5)$$

- We shall say that a processor P strictly ϵ -computes F if and only if :

$$\forall \mathbf{x} \in \{0, 1\}^m \quad p(\delta = 1 | \mathbf{X} = \mathbf{x}) = \epsilon \quad (2.6)$$

- A processor P is said to reliably compute F if it 0-computes f .

The “weak” computation represents a more general deficiency model than the “strict” computation: a processor P which strictly ϵ -computes a boolean function is *reliably unreliable* because its failure probability is known and constant in time, while the failure probability of a processor P which weakly ϵ -computes a function F might have a reliability varying over time. Besides, the exact reliability of a device can often not be determined but simply bounded. That is why the model of weak computation is more realistic. But it has the disadvantage of leading to looser bounds on the reliability of noisy circuits. For this reason we will use both approaches.

Combinational circuits

When combining different processors into a circuit, we have to impose certain restrictions in order to avoid inconsistencies with the definitions we adopted.

Two processors are *connected* when their sets of inputs and output share common random variables. Two different processors P_1 and P_2 may not have a common output, but P_1 and P_2 may have common inputs. Besides the output of P_1 may be a coordinate in the input vector of P_2 .

Depending on the assumptions concerning the communications within the circuit, certain kind of processors may or may not be connected. If we assume that the circuit has reliable wires, the signal is identical at both ends of the line and a single random variable is needed to represent the excitation level of the wire. In this case, all kind of processors may be connected along the guidelines mentioned. This is the model of circuit we adopt from now on, and the corresponding mathematical definition is:

Definition 2.3.5 :

A combinational C circuit made out of processors from S is a map from $\mathbf{R}(\{0,1\}^M)$ to $\mathbf{R}(\{0,1\}^N)$ obtained by composition of processors from S .

Circuits are most adequately represented in terms of graphs, representation which has the advantage of suggesting their physical layout. A very important restriction on the schemes we consider is that none of their processors should have any feedback. This is the specific feature of the class of combinational circuits, which includes only loop-free automaton arrangements. Should this condition not be enforced, the output of the circuit may not be stationary any longer. This would lead to a breakdown of the “instantaneous time” approach and inconsistencies in the probability distributions associated with each processor. To treat such circuits, one would need to reformulate the processors in terms of automata and handle correctly the issue of synchronization. This is a much more complicated problem which is not relevant to this thesis addressing the resource-efficient implementation of boolean functions.

A combinational circuit with reliable wires may be represented through an *oriented* graph with the following properties:

- Each node of the graph represents a processor. Its sets of inputs and its output are given respectively by the lines terminating and originating from the node.
- Each line (called wire) is associated with a unique random variable. All the lines are oriented and connected to at least one node. A line connecting the output of the processor k with the processor l wears a number corresponding to the input label in the processor l . A wire which does not originate from any node carries an *input* of the circuit, and a wire which does not terminate in any node carries an *output* of the circuit.
- There is no sequence of processors P_{i_1}, \dots, P_{i_k} and wires L_{j_1}, \dots, L_{j_k} such that:

$$\forall u \in [1, k] \quad L_{j_u} \text{ originates from } P_{i_u} \text{ and terminates in } P_{i_{(u+1) \bmod k}} \quad (2.7)$$

In any of the coming figures representing combinational circuits, the wires are oriented from the left to the right.

2.3.2 Fault-Tolerance

Faults and Deviations

It is crucial, in order to state whether the global computation of the circuit is successful or not, to know which processors have failed during the computational process. To this purpose, we introduce the notion of fault-path:

Definition (Fault Path) 2.3.6 : *Let C be a combinational circuit consisting in the processors P_1, \dots, P_N . We shall call fault-path the random vector $\mathbf{E} = (E_1, \dots, E_n)$ where the coordinates E_i are binary random variables such that:*

$$E_i = 1 \quad \Leftrightarrow \quad P_i \text{ fails}$$

From this definition we can define the perfect computation in a circuit:

Definition 2.3.7 :

Let C be a combinational circuit consisting in the processors P_1, \dots, P_N , of inputs $\mathbf{X} = X_1, \dots, X_m$ and outputs $\mathbf{Y} = Y_1, \dots, Y_n$.

- *A perfect (or fault-free) computation of x_1, \dots, x_m is a computation yielding $\mathbf{X} = (x_1, \dots, x_m)$ and $\mathbf{E} = (E_1, \dots, E_n) = \mathbf{0}$*
- *The perfect result $r(x_1, \dots, x_m)$ the value taken by the vector $\mathbf{Y} = (y_1, \dots, y_n)$ in a perfect computation. The function $r(x_1, \dots, x_m)$ is the function represented by the circuit.*

While the notion of failure is valid for a processor, it is inappropriate for wires. Nonetheless, we will need in our analysis to identify when the inputs feeding a processor differ from what would be expected in a perfect computation. With this goal in mind we introduce the notion of deviation:

Definition 2.3.8 : Let C be a boolean circuit and W be the random variable associated with a wire of the circuit. We shall say that the wire W deviates during a computation when it takes a different value from that yielded by a perfect computation in C .

Encoded Computation

We may now translate the ideas of the previous section regarding the encoding of logic gates, focusing directly on the case of interest where each codeword contains one and only one logical bit. From now on, if T is a code of length l , we denote $T(\mathbf{u})$ the boolean vector corresponding to the bitwise encoding of \mathbf{u} :

$$T(u_1, \dots, u_m) = (T(u_1)_1, T(u_1)_2, \dots, T(u_1)_l, T(u_2)_1, \dots, T(u_m)_l) \quad (2.8)$$

Definition 2.3.9 : Let $T : \{0, 1\} \rightarrow \{0, 1\}^l$ be a function. Let $F : \{0, 1\}^m \rightarrow \{0, 1\}$ be a boolean function, and C be a combinational circuit with the inputs $\mathbf{X} = X_1, \dots, X_{m \times l}$ and outputs $\mathbf{Y} = Y_1, \dots, Y_{n \times l}$, built out of the processors P_1, \dots, P_N . We shall say that C represents the T -encoded implementation of F if and only if:

$$\forall (x_1, \dots, x_m) \in \{0, 1\}^m \quad r(T(\mathbf{x})) = T[F(\mathbf{x})] \quad (2.9)$$

Definition 2.3.10 : Let $T : \{0, 1\} \rightarrow \{0, 1\}^l$ and $D : \{0, 1\}^l \rightarrow \{0, 1\}$ be two functions such that $\mathbf{D} \circ \mathbf{T} = \text{Id}$ on $\{0, 1\}$. Let C be a combinational circuit implementing a T -encoded computation of $F : \{0, 1\}^m \rightarrow \{0, 1\}^n$, taking the inputs $\mathbf{X} = X_1, \dots, X_{m \times l}$ and delivering the outputs $\mathbf{Y} = Y_1, \dots, Y_{n \times l}$.

We shall say that a computation yielding $\mathbf{X} = (x_1, \dots, x_{m \times l})$ and $\mathbf{Y} = (y_1, \dots, y_{n \times l})$ is a correct (T, D) -computation of F if and only if:

$$(D[(y_1, \dots, y_l)], \dots, D[(y_{(n-1)l+1}, \dots, y_{nl})]) = F[D(x_1, \dots, x_l), \dots, D(x_{(m-1)l+1}, \dots, x_{ml})] \quad (2.10)$$

Our definitions consistently ensure that a perfect computation is correct. Conversely, it is sufficient to know the result of a correct computation in a processor representing a (T, D) -encoded computation of F with a set of inputs x_1, \dots, x_m in order to retrieve the result of the perfect computation on the same inputs.

Encoding and Fault-Tolerance

By relaxing the requirement that a computation be perfect and considering correct computations as satisfactory results, we have a considerably enlarged set of acceptable outputs (since the function D is many-one) and thus a significantly higher success probability for the T -encoded gate than if we were to consider only perfect computations as satisfactory. It does not mean, however, that a gate representing a T -encoded implementation of F is more reliable than a processor simply representing F : the multiplication of the acceptable outputs is counterbalanced by the higher complexity of the T -encoded gates. As we shall see in Chapter 4, the cost of this increased complexity is a reduced amount of resources for each component as well as a bigger number of possible error locations. We now understand precisely the notion of fault-tolerance: the failures tolerated by the system are exactly the fault-paths which do not affect the decoding of the output.

In a circuit of encoded logic gates such that no decoding-encoding is performed in between the encoded gates, consistency requires to treat the inputs and the outputs on the same footing. Indeed this assumption is vital not only because the cost associated with such operations after each logic operation would be prohibitive, but also because the data going through the computational process must stay encoded at all time in order to be protected. Therefore, we must tolerate a certain amount of noise in the inputs as a normal regime of the gate operation. The fault-tolerant architecture that we develop for NAND gates should attenuate this input noise and deliver a more stable output.

Let us consider an encoded NAND gate inserted in a large chip built out exclusively of this component. In order to determine the effective reliability of this gate for the performance of computations within the chip, we proceed as follows:

- We compute the separate failure probability p_i of each bit in the output of the encoded gate and take their maximum value p_m .
- We place in each input of the encoded NAND gate a random channel flipping the bit with a probability bounded by p_m , leading to input deviations.
- We compute with these noisy inputs the probability ϵ that the decoding of the output of the circuit yield a wrong result when the inputs are correct: we allow the inputs to deviate but they must be associated with previous correct computations.

If T and D represent respectively the encoding and the decoding, and if ϵ is computed according to the procedure above, we will say that the encoded processor performs an ϵ -reliable (T,D)-computation of a NAND gate.

2.4 Algorithmic-based fault-tolerance: example of the encoded polynomial multiplication

As an illustration of the ideas on fault-tolerant computation exposed above, we now turn to the description of a code leading to a non-trivial encoding of the computational tasks. This example is taken from prior work of Beckmann [Bec92], who applied in the context of fault-tolerance an algorithm of polynomial multiplication invented by Winograd [Win80]. The code considered here is arithmetic, and the failure protection it provides is suitable only for a specific type of computation, which is the multiplication of polynomials.

Convolution of two sequences : algebraic framework

Definition 2.4.1 : *The convolution of two sequences $a[n]$ and $b[n]$, which are non-zero for $0 \leq n \leq P - 1$, is defined as the sequence $c[n] = a[n] * b[n]$, non zero for*

$0 \leq n \leq Q = 2P - 1$ and such that:

$$c[n] = \sum_{i=0}^n a[i]b[n-i] \quad \text{for} \quad n = 0, \dots, Q - 1 \quad (2.11)$$

The convolution is nothing else than the usual formal polynomial multiplication. We note $F[x]$ the set of polynomials with coefficients in a field F . If $M[x]$ is an element of $F[x]$, we denote $F[x]/M[x]$ the set of polynomials modulo $M[x]$. This set is a ring with the usual multiplication and addition modulo a polynomial. If $M(x)$ has a sufficiently high degree ($\deg M(x) > \deg a(x) + \deg c(x)$), then the computation within $F[x]/M[x]$ and within $F[x]$ yields identical results. Let us assume that $M(x)$ is factorized in relatively prime, irreducible polynomials:

$$M(x) = m_1(x)m_2(x)\dots m_N(x) \quad (2.12)$$

It is a well known result of general algebra that the ring $F[x]/M[x]$ is isomorphic to the direct sum of the smaller rings $F[x]/m_1[x] \times F[x]/m_2[x] \times \dots \times F[x]/m_N[x]$. The mapping consists simply in computing the corresponding residues:

$$a(x) \sim (a_1(x), a_2(x), \dots, a_N(x)) \quad \text{with} \quad a_i(x) = \langle a(x) \rangle_{m_i(x)} \quad (2.13)$$

where we noted $\langle P(x) \rangle_{m(x)} = P(x) \bmod m(x)$.

The inverse of this mapping is given by the Chinese Remainder Theorem:

$$a(x) = \sum_{k=1}^N \langle a_k(x)D_k(x) \rangle_{m_k(x)} M_k(x) \quad \text{with} \quad M_k(x) = \frac{M(x)}{m_k(x)} \quad (2.14)$$

where $D_k(x)$ is one of the Bezout coefficients between $M_k(x)$ and $m_k(x)$ (the primality condition in the decomposition of $M(x)$ ensures the existence of such polynomials):

$$\langle M_k(x)D_k(x) \rangle_{m_k(x)} = 1 \quad (2.15)$$

The operations of addition and multiplication can then be done within the smaller rings:

$$\begin{aligned} a(x) \times b(x) &\sim (a_1(x)b_1(x), a_2(x)b_2(x), \dots, a_N(x)b_N(x)) \\ a(x) + b(x) &\sim (a_1(x) + b_1(x), a_2(x) + b_2(x), \dots, a_N(x) + b_N(x)) \end{aligned} \quad (2.16)$$

This fact enlightens an alternative way to compute the convolution than that given by the definition (2.4.1): we may compute each of the products in the smaller ring and put everything together thanks to the Chinese Remainder Theorem.

Encoding of the Computation

This preliminary algebraic framework set, it is now possible to explain how structured redundancy may be added. We start with a Q -point linear convolution using N moduli $m_1(x), \dots, m_N(x)$ which satisfy the primality condition. Q is chosen large enough so that the computation of the convolution is identical in the rings $F[x]$ and $F[x]/M[x]$.

We add to this set C extra moduli $m_{N+1}(x) \dots m_{N+C}(x)$ coprime to each other and to the other N extra moduli. The computation will thus be made in the larger ring $F[x]/M^+[x]$ where $M^+[x] = \prod_{k=1}^{N+C} m_k(x)$. It is possible, in principle, to compute convolution of length up to $Q^+ = \sum_{k=1}^{N+C} \deg m_k(x)$. But, in order to keep a room for redundancy, we will require that the convolutions be of length smaller than $Q = \sum_{k=1}^N \deg m_k(x)$. The quantity $Q^+ - Q$ represents the extra dimensions playing a role analogous to the parity check symbols for linear codes.

The computation will be distributed among different processors in such a way that each processor failure causes a residue and only one to be incorrect. We assume that we have at our disposal $N + C$ processors, each of those taking as inputs $a(x)$ and $b(x)$, and computing :

$$a_k(x) = \langle a(x) \rangle_{m_k(x)}, \quad b_k(x) = \langle b(x) \rangle_{m_k(x)}, \quad c_k(x) = \langle a(x)b(x) \rangle_{m_k(x)} \quad (2.17)$$

The result might be decoded after the computation, using precomputed Bezout polynomials and the Chinese Remainder Theorem:

$$z(x) = \sum_{k=1}^N \langle z_k(x) D_k^+(x) \rangle_{m_k(x)} M_k^+(x) \quad \text{with} \quad M_k^+(x) = \frac{M^+(x)}{m_k(x)} \quad (2.18)$$

where the $D_k(x)$ have been computed once for all prior to the computation:

$$\langle M_k^+(x) D_k^+(x) \rangle_{m_k(x)} = 1 \quad (2.19)$$

We use the following additive error model:

$$z_k(x) = c_k(x) + \Phi_k(x) \quad (2.20)$$

where $\Phi_k(x)$ is non zero for at most λ different values of k . Then, recombining the result yields the following error syndrome:

$$z(x) = c(x) + \sum_{k=1}^N \langle \Phi_k(x) D_k^+(x) \rangle_{m_k(x)} M_k^+(x) \quad (2.21)$$

The subset of valid results is identified as:

$$H_V(x) = \{ \quad c(x) \in F[x]/M^+[x] \quad | \quad \deg c(x) < Q \quad \} \quad (2.22)$$

Error Correction

A way to proceed is to introduce in the polynomial ring the concepts that led to successful error correction for linear codes. With this goal in mind we define, as an analog of the Hamming distance, the quantity D :

$$D \quad \text{largest integer s.t.} \quad \deg \left(\frac{M^+(x)}{m_{l_1}(x) \dots m_{l_D}(x)} \right) \geq Q$$

for every set of D unique moduli $\{m_{l_1}(x), \dots, m_{l_D}(x)\}$ (2.23)

To change a valid set of residues into an other valid set, at least $D+1$ residues must be modified. In that respect, D is really a relevant measure of the amount of redundancy, and the natural translation in the polynomial ring of the Hamming distance. Note that if the redundant moduli $m_{N+1}(x)\dots m_{N+C}(x)$ are chosen to have a degree equal or greater than those of the moduli $m_1(x)\dots m_C(x)$, then $D = C$. We can now mention two claims which are essential in the interpretation of the error syndrome:

Claim 2.4.1 :

Let D be the polynomial distance of the code verifying the last property. Then:

- *If no failure occur, then $\deg z(x) < Q$ and the correct convolution is $c(x) = z(x)$*
- *If between 1 and D failures occur then $\deg z(x) \geq Q$*

Claim 2.4.2 :

Let D be the polynomial distance of the code and two integers α and β such that $D = \alpha + \beta$. We assume that no more than α processors can fail at a time. Then:

- *We can reliably distinguish between the following two cases: β or fewer errors occur, between β and $\alpha + 1$ errors occur.*
- *If β or fewer errors occur, we can correct them using the erroneous result $z(x)$*

Performance of the code

We have thus seen a non-trivial example of a non-systematic code for which the basic operations, the addition and the multiplication, have been changed a lot in the encoding process: we have replaced a computation in a big ring by a sequence of smaller computation performed in rings of smaller dimensions. The encoding and the decoding of the data, which consist respectively in the computation of residues and application of the Chinese Remainder Theorem, are also non-trivial. Our description indeed encompasses not only a single code but a large *family of codes*, because for each choice of the additional set of moduli, one obtains a different code.

Considering a choice of the extra moduli enabling single error correction, it is of interest to estimate the *overhead* implied by the encoding and relate it to that

resulting from a mere triplication of the computation. Indeed, if the latter turned out to be smaller, it would not be worth using the moduli-based encoded computation. To this purpose, Beckmann [Bec92] compares the unprotected computation of an NR-point convolution with the equivalent computation embedded in a ring of dimension $N + 2$. For some values of N ($N=14$), the overhead due to the embedding in a bigger polynomial ring is as low as 65%, while 90% of the computational process is protected. This is a significant advantage over the triple modular redundancy scheme, for which the overhead is 200%.

2.5 Conclusion

In this chapter, we have exposed the problematic of fault-tolerance as well as the cornerstone of a fault-tolerant design: the encoding of the data and of the operational gates in such a way that error do not propagate significantly. Repetition codes appear as natural candidates because they lead to transversal implementation of gates in which this error propagation is naturally contained. Nonetheless, we have seen on an example that other forms of encoding may be considered (section 2.4), especially for the protection of failures in specialized computations. We have also developed a rigorous approach of the encoded computation within noisy circuits which sets the future developments of this thesis on firm foundations.

Chapter 3

Modular Redundancy-Based Fault-Tolerant Architectures

We reviewed in Chapter 2 the general techniques enabling to provide combinational circuits with a robustness to soft errors. We now specialize to the specific class of fault-tolerant architectures on which we will base, in Chapter 4 and in Chapter 5, an alternative design of a logic gate. Introduced by Von Neumann in 1956 [Neu56], these architectures are referred to as being modular redundant. This chapter describes their theory and their performance for circuits of encoded logic.

The outline is the following. Section 3.1 gives a brief introduction to the work of Von Neumann and positions our approach relative to his own. Section 3.2 covers the modular redundant design with perfect majority voting and presents in this simplified framework the threshold theorem. In section 3.3 the assumption that majority gates be perfect is released, and the technique of multiplexing is introduced. The relation between resource consumption and reliability is carefully investigated. Section 3.4 concludes this chapter by reconsidering the threshold theorem in the more general context of imperfect majority voting.

3.1 Introduction

Perspective on Von Neumann's work

Expanding on lectures given by R.S. Pierce in the early 1950's, Von Neumann exposed the modular redundant design in a celebrated paper [Neu56], at a time where the absence of reliable electronic devices strongly motivated the development of new highly fault-tolerant architectures. While Shannon had already laid out a theory of reliable data transmission through noisy channels [Sha48a, Sha48b], a systematic treatment of fault-tolerant computing was still to be constructed. The paper of Von Neumann [Neu56] was an important step in this direction, showing in particular that arbitrarily high levels of reliability may be achieved in a computing system made of components failing with a probability below a certain threshold. Von Neumann also gave an heuristic argument allowing to upper-bound the number of faulty components needed for the achievement of an ϵ -reliable computation (definition (2.3.4)): any circuit made of N gates may be simulated with a high probability of success with an equivalent circuit consisting of $O\left(N \log\left(\frac{N}{\epsilon}\right)\right)$ gates whose unreliability is below the threshold. Dobrushin and Ortuyov [DO77b, DO77a] later turned his argument into a rigorous proof. Winograd and Cowan [WC63] improved his schemes by distributing single bits on different units of the network.

The bound suggested by Von Neumann shows that a general computation may be simulated on a noisy circuit with a high probability of success not only in principle, but also *efficiently*. This precision is of the uppermost importance: the control of the additional complexity overhead incurring through a fault-tolerant design is a *sine-qua-none* condition for the its introduction in the manufacturing of real circuits. In fact this overhead, which implied the multiplication of the number of wires by a factor of several thousands, was still beyond the reach of component technologies at that time. That is why the techniques exposed by Von Neumann were not immediately involved in chip design. With the fast progresses in the manufacturing of reliable logic gates, engineers were able to produce devices so reliable that it was no longer obviously useful to embed them in a fault-tolerant network, even for the realization of fairly

long computations. For this reason, the techniques of modular redundancy have not been put to use so far in the industrial production of computers. Nonetheless, they have been successfully implemented in several architectures exploring the boundaries of defect-tolerance such as Teramac [CAC⁺96]. We should also mention that fault-tolerance shed an interesting light on neuronal sciences: in his original paper [Neu56], Von Neumann used the theory of multiplexing to estimate the amount of parallelism in the computations performed in the brain.

Model of the computational system

Inspired by the model of neural networks developed by McCulloch and Pitts [MP43], Von Neumann set his work in the frame of cellular automata. Building blocks of his constructions, these automata are entirely defined by an array of probabilities giving the output distribution for each possible set of inputs and a processing time. Von Neumann showed that the set of automata generating universal computation may be reduced to a single organ performing triple majority voting or a NAND. He considered afterwards fault-tolerant circuits implying exclusively one of these two gates and a “black-box” interconnecting bundles of wires at random, organ for which he did not provide an explicit construction. Our approach will differ from his own on two points:

- We address only the issue of implementing combinational circuits and not general arrangements of cellular automata. As mentioned at the beginning of Chapter 2, we specialize to the fault-tolerant implementation of a universal gate such as the NAND. This does not reduce the scope of our discussion, since arrangements of encoded NAND can simulate any general computation. Indeed, in section 2.3.2 we carefully defined the reliability of an encoded NAND in order to account for its embedding in noisy circuits of similar gates. Our treatment then encompasses indirectly any general combinational circuits. Nonetheless, the approach of Von Neumann did not exclude arrangements of cellular automata with loops, restriction which we impose here.

- We will not assume the existence of an organ performing randomized interconnections. While this assumption of is suitable for neural networks, which was the ground on which Von Neumann set his analysis, it is not well adapted to the design of electronic circuits with a fixed layout. We will avoid the use of such a mixing device by introducing some diversity in the components used within our circuits: instead of working with a strictly minimal set of components as Von Neumann did, we shall use simultaneously NAND and 3-majority gates. This diversity could come at the price of having to take into account several different technologies for each class of components while optimizing the reliability of the circuit as a whole. But since the 3-majority and the NAND gates are of similar complexity, we will assume that these components are ruled by identical technological constraints. We will specify the full meaning of this assumption in Chapter 4.

3.2 Triple Modular Redundancy with Perfect Majority Voting

In this section we present the simplest possible process improving the reliability of a noisy boolean computation: its triplication followed by a majority voting, which will be taken - in this section only - as perfect. This assumption stands apparently in contradiction with the objective set in Chapter 2 to design fault-tolerant systems without any ideally reliable component. This assumption of perfect 3-majority gates is, however, a suitable frame for an introduction to the main concepts of recursive modular design. In section 3.3, we will combine the schemes presented here with an additional technique addressing the issue of imperfect majority voting. As we shall see, the performances of those circuits will be essentially preserved in the framework of imperfect majority voting.

3.2.1 Failure probability of a triplicated system

Description of the system

Let P be a processor p -computing (see definition (2.3.4)) the boolean function:

$$F : \{0, 1\}^m \longrightarrow \{0, 1\}$$
$$(x_1, \dots, x_m) \mapsto F(x_1, \dots, x_m)$$

Instead of using this single processor, we consider the circuit exposed on figure 3-1, composed of three identical processors P_1, P_2, P_3 and a majority voting gate. In lieu of the m original inputs, we now have m bundles of three wires, each of which feeding a processor P_i whose output is directed toward a majority-voting gate. It is straightforward to see that, during error-free operation, this circuit is simply equivalent to the original processor P .

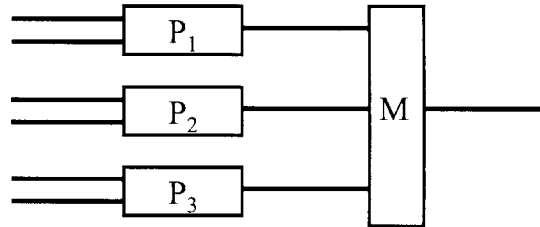


Figure 3-1: Simple majority voting system. The signal is processing from left to right: the circuit is taking $m = 2$ bundles of 3 wires in input and delivering a single bundle in output. The inputs correspond to $m = 2$ logical bits which have been previously encoded in the triple repetition code. The processors P_i might fail, but the majority gate is assumed to be perfectly reliable.

As mentioned in introduction, we assume that the 3-majority gate is perfect. This assumption may be justified if the computation occurring within the processors P_i is several orders of magnitude more complex than a 3-majority voting, leading to a failure probability for each processor P_i significantly higher than that of a 3-majority gate. The majority voting plays a role analogous to amplifiers in electrical circuits, since it suppresses the noise and amplifies the signal part of the bitstream in the bundle of 3 wires with high probability. Taking again the denominations of

section 2.1.3, the three replicas of P and the 3-majority gate are respectively the executive and the restoring organ.

Estimation of the failure probability

We now want to estimate the failure probability of the triplicated circuit. In order to do this, we must use an important assumption regarding the error model, already mentioned in section 2.1.2: we suppose that the events “the processor P_i delivers a wrong result” are *independent*. This is not necessarily the case and it depends indeed on the mechanism producing the errors. If the errors were caused by radiation of particles, for instance, the same flux may contaminate several neighboring processors at a time, generating burst of processor failures instead of isolated errors [RCM⁺97]. Nonetheless, in many situations it is sensible to assume that the processes responsible for the emergence of soft errors are micro or mesoscopic and thus have a smaller extension than the dimensions of a processor. Consequently, the failure of a component does not allow one to infer anything about the behavior of its neighbors. In the case of cosmic rays, this assumption is legitimate if the flux of particles has a cross section much smaller than the typical size of a processor. Furthermore, in the present circuit, we assume that the inputs of the processors do not deviate. Our argument could still be valid in the case of faulty inputs, provided that the errors within each bundle of 3 wires are independent of each other. Under these conditions, the probability of a wrong 3-majority gate output is:

$$p_{fail} = f(p) = 3p^2(1 - p) + p^3 = 3p^2 - 2p^3 \quad (3.1)$$

Furthermore, even if the processors P_1, P_2, P_3 have different failures probabilities, it is still possible to bound the failure probability of the system:

$$\forall i \quad p_i \leq p \quad \Rightarrow \quad p_{fail} \leq f(p) \quad (3.2)$$

Necessity of independent processor failures

Had we not assumed the independence of the errors happening in each processor output, we would have simply been able to say:

$$p_{fail} \leq 3p \quad (3.3)$$

This bound does not guarantee any improvement through majority voting, and we cannot get a tighter bound without any knowledge of the correlation of errors in the inputs. That is why it is important to preserve the “failure independence” of the inputs before each majority voting. This may be achieved through a device doing interconnections of wires *at random* between the executive stage and the restoring stage [Neu56]. This can also be done by using majority voting gates taking as many inputs as wires in the bundle. Only the latter option is suitable for the application of modular redundancy in the layout of electronic components.

3.2.2 Cascaded Triple Modular Redundancy

Motivation and Definition

We would now like to see how far one can go in the reliability amplification under iteration the majority voting process. For instance, one could start with m bundles of 9 wires feeding 9 processors P_i computing F , followed by two stages of majority voting as on figure 3-2. We shall call this circuit a triple modular implementation of F of recursion level 2. Under the same error model as in the last paragraph, the failure probability would then be: $p_{fail}^{(2)} = f^{(2)}(p)$. Of course this process may be iterated further:

Definition 3.2.1 (Recursive Triple Modular Implementation):

Let P be a deficient processor computing a boolean function F with a single output. A triple modular redundant (TMR) implementation of the processor P of level k (noted $P^{(k)}$) is:

- for $k = 0$ the processor P .

- for $k \geq 1$ the scheme obtained by embedding the outputs of three TMR implementation of P of level $k - 1$ into a 3-majority gate.

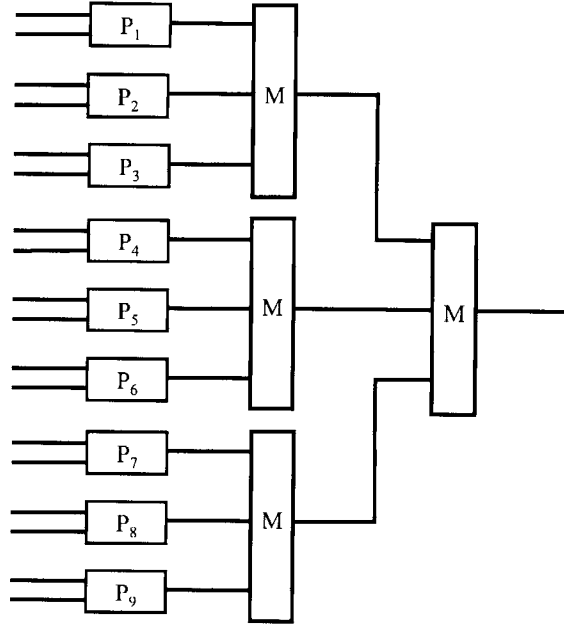


Figure 3-2: Double majority voting system. Two logical bits are sent in input which have been doubly encoded into the triple repetition code: each logical bit is present in the inputs through 9 copies. The gate outputs one single bit after two stages of majority voting. The processors P_i might fail, but the majority gates are assumed to be perfectly reliable.

Remark 1: The latter definition may be extended to the computation of boolean functions F with n outputs by embedding each of the n triplets of outputs delivered by the TMR implementation of level $k - 1$ in separate 3-majority gates.

Remark 2: It is possible to similarly define the d -modular redundant implementation of level k of a processor P (d being an odd integer) by using majority gates taking d inputs instead of 3.

Asymptotic reliability of recursive TMR implementation with perfect majority voting

It is of interest to study what levels of reliability may be achieved when TMR implementing a deficient processor P at a high recursion level. The reliability asymptotically obtained is reflected in the behavior of the recursive sequence $p^{(k)} = f^{(k)}(p)$, giving the failure probability associated with the TMR implementation of level k . This leads us to the following claim:

Claim 3.2.1 : *Let P be a processor strictly ϵ -computing (with $\epsilon < 1/2$) a single-output boolean function F in the sense of the definition (2.3.4). By TMR implementing P at a sufficiently high level, we obtain a scheme computing F with an arbitrarily high reliability.*

Proof: In other words we want to show:

$$\forall \epsilon > 0 \quad \exists k_0 \quad s.t. \quad \forall k > k_0 \quad P^{(k)} \quad \epsilon - \text{computes} \quad F \quad (3.4)$$

We need to study the recursive sequence $p^{(k)} = f^{(k)}(p)$. By direct inspection, we see that the function f has only three fixed points: $x = 0, x = 1/2, x = 1$. Also f is an increasing function on the interval $[0, 1]$, therefore the sequence $p^{(k)}$ is monotonous: if $f(p) < p$ it is strictly decreasing, while if $f(p) > p$ it is strictly increasing. The function $g(x) = f(x) - x$ satisfies $g(x) < 0$ for $x < \frac{1}{2}$ and $g(x) > 0$ for $x > \frac{1}{2}$. Hence, if the failure probability of the processor P is such that $0 < p < \frac{1}{2}$, the sequence $p^{(k)}$ satisfies:

$$\forall k > 1 \quad 0 < p^{(k+1)} < p^{(k)} < \frac{1}{2} \quad (3.5)$$

and it converges then towards 0, because there is no other fixed point between 0 and $\frac{1}{2}$. Asymptotically, and assuming we have at our disposal perfect 3-majority gates, it is possible to reach obtain any level of reliability by iterating the process of triple modular implementation a sufficient number of times.

The case $p = \frac{1}{2}$ might be discarded because the output of the processor P is then totally irrelevant. The possibility $p > \frac{1}{2}$ may be also eliminated by definition of

the computation of a boolean function with a noisy processor (2.3.4). We have now proven the desired result.

This claim is significant: it shows that the amplification by recursive majority voting allows one to use any processor P computing F for the reliable implementation of this function, whatever the level of deficiency of the processor. But we should emphasize that this result is only valid when a very restrictive error model applies to the processors P_i : not only did we assume that each of them failed independently, but we also took these processors as “reliably unreliable” with identical probability of failure p . Without this assumption, the failure probability of the recursive circuit is not as easy to derive.

3.2.3 Threshold Theorem

So far we have proven that the cascaded modular redundant design is a technique which allows one, in a rather specific framework - existence of perfect 3-majority gate and errors among processors independent and identically distributed -, to achieve arbitrarily high levels of reliability in the computation of boolean functions with arbitrarily highly deficient processors. But we have no clue yet on the efficiency of this process, information which is naturally crucial for real design problems. It is indeed entirely reflected in the convergence speed of the sequence $p^{(k)}$.

We now fill in this hole and bound the minimum number of majority votes necessary to attain a certain reliability. But this comes at the cost of an additional assumption: we also require that each processor must have a probability of failure bounded above by a certain threshold which we will evaluate later on:

$$\forall i \ P_i \text{ fails with probability } p_i \text{ satisfying } p_i < p < p_{th} < \frac{1}{2} \quad (3.6)$$

We thus no longer consider arbitrarily deficient processors. As a compensation, the error model is more general than before: we have released the assumption that they

be identically distributed. In the sense of definition (2.3.4), the different processors P_i now *weakly* p -compute their assigned boolean functions.

This restriction to processors more reliable than the threshold guarantees the efficiency of the majority voting process. The key point is that the failure probability of the system changes from $O(p)$ to $O(p^2)$ through the triplication followed by perfect majority voting. If the elementary processors are not strongly deficient, that is if $p \ll 1/2$, then the gain in reliability is great.

Relation between the number of gates and reliability

The number of gates involved scales exponentially with the recursion level k of the TMR implementation. Indeed, noting $N_1(k)$ and $N_2(k)$ the number of processors P and 3-majority gates M :

$$N_1(k) = 3^k \quad \text{and} \quad N_2(k) = \sum_{l=0}^{k-1} 3^l = \frac{3^k - 1}{2} \quad (3.7)$$

The total number of gates is therefore:

$$N(k) = 3^k + \frac{3^k - 1}{2} \simeq \frac{3}{2} 3^k \quad (3.8)$$

In fact, this exponential scaling of the resources does not compromise the efficiency of the reliability amplification: under a threshold condition on the processor reliability, it is compensated by a much faster decrease of the failure probability with the recursion level k . Precisely, assuming that three processors satisfy the condition (3.6), the failure probability of the corresponding simple majority system of figure 3-1 is upper-bounded by $f(p)$ (equation (3.2)). This function can be in turn bounded above by a monomial:

$$f(p) \leq 3p^2 \quad (3.9)$$

Let us now consider a TMR implementation of level k whose processors P_i fail with probability $p_i < p$. An immediate recursion shows that the failure probability

p_{fail} of this system satisfies:

$$p_{fail} \leq f^{(k)}(p) \leq \frac{1}{c} (cp)^{2^k} \quad \text{with } c = 3 \quad (3.10)$$

If p is such that $p < \frac{1}{c}$ then the failure probability p_{fail} decreases *doubly exponentially* with k . For circuits *with perfect majority voting*, the threshold probability of condition (3.6) is thus:

$$p_{th} = \frac{1}{c} = \frac{1}{3} \quad (3.11)$$

Efficiency of the cascaded TMR design

From the relations (3.8) and (3.10), we can estimate the overhead incurring in the computation of a boolean function through a noisy circuit with components satisfying the threshold condition (3.6). Let $F(x_1, x_2, \dots, x_m)$ be a boolean function computable with a combinational circuit C of N reliable gates g_1, \dots, g_N . We have at our disposal the devices G_1, \dots, G_N weakly p_G -computing (in the sense of the definition (2.3.4)) the gates g_1, \dots, g_N , and failing independently from each other. We wish to ϵ -compute F with those components. Using the principles exposed so far, we may think of two different strategies to implement this circuit.

External modular design

We may refer to this scheme as an “external triple modular redundant implementation.” It consists in the following. We replace each of the gates g_i involved in the original circuit by the component G_i , yielding a noisy circuit C . This circuit may be considered as a processor P with an associated failure probability bounded above by p_C . Following the definition (3.2.1), we duplicate P in 3^k independent circuits on which we perform different stages of majority voting. This scheme is presented on figure 3-3 for $k=1$. Assuming that $p_C < \frac{1}{3}$, we use the failure probability bound (3.10) to obtain a recursion level k_0 , sufficient for the ϵ -computation of F :

$$k_0 = \left\lceil \log_2 \left(\frac{\log(3\epsilon)}{\log(3p_C)} \right) \right\rceil \quad (3.12)$$

Of course, any TMR implementation of level $k \geq k_0$ will also ϵ -compute F . Anyway, it is sufficient to use $N_1(k_0)$ replicas of the original circuit and $N_2(k_0)$ majority gates to build the desired fault-tolerant implementation. The total number of gates required is then upper-bounded by:

$$\begin{aligned} N_{\epsilon, p_C}^{ext} &\leq N 3^{k_0} + \frac{1}{2} 3^{k_0} \leq \left(N + \frac{1}{2} \right) \exp \left(\log(3) \left(1 + \log_2 \left(\frac{\log(3\epsilon)}{\log(3p_C)} \right) \right) \right) + k_0 \\ &\leq 3 \left(N + \frac{1}{2} \right) \left(\frac{\log(3\epsilon)}{\log(3p_C)} \right)^{\frac{\log(3)}{\log(2)}} \end{aligned} \quad (3.13)$$

Unfortunately, in order to use this design method, we need to have a noisy circuit C such that $p_C < \frac{1}{3}$. This is a much more stringent restriction than $p < \frac{1}{3}$ for the individual components, especially if the reliable circuit computing F has a fair level of complexity.

Internal modular design

We now turn to an other implementation which, by inserting a fault-tolerant design directly at the component level, circumvents the latter problem. This second scheme may be called an “internal modular redundant implementation.” We choose here to replace each gate g_i by the component $G_i^{(k_i)}$, triple modular implementation of G_i of level k_i . We will introduce in the next chapter the hypothesis that reliability varies continuously with area. The internal fault-tolerant design will then appear as a partition of a continuous pool of resources.

To achieve the reliability requirement ϵ for the encoded computation of the function F , it is sufficient to ensure that the failure probability of each TMR implemented component $G_i^{(k_i)}$ is bounded by:

$$p_{fail}^i \leq \frac{\epsilon}{N} \quad (3.14)$$

With a more careful investigation on the noiseless circuit computing F , we may impose a less stringent reliability requirement for each component G_i . But this simple condition, guaranteeing the reliability of the encoded circuit, is sufficient for the present discussion. Besides, a variation in p_{fail}^i would in fact only lead to a very

small change in the overhead for the fault-tolerant circuit. A simple substitution in equation (3.12) gives a recursion level k_0 sufficient to implement any component G_i and satisfy the condition (3.14):

$$k_0 = \left\lceil \log_2 \left(\frac{\log(3 \frac{\epsilon}{N})}{\log(3p)} \right) \right\rceil \quad (3.15)$$

The corresponding modular implementation requires the number of components:

$$N(k_0) = N_1(k_0) + N_2(k_0) \leq \frac{3}{2} 3^{k_0} = \frac{9}{2} \left(\frac{\log(3 \frac{\epsilon}{N})}{\log(3p)} \right)^{\frac{\log(3)}{\log(2)}} \quad (3.16)$$

We can then upper-bound the total number of devices necessary to ϵ -compute F :

$$N_{\epsilon,p}^{int} \leq \frac{9}{2} N \left(\frac{\log(3 \frac{\epsilon}{N})}{\log(3p)} \right)^{\frac{\log(3)}{\log(2)}} \quad (3.17)$$

The overhead factor is slightly higher for the internal fault-tolerant design: it is not strictly linear anymore in the number of components, there is now a term in $N \log(N)$, reflecting the dependence of the reliability requirement (3.14) on the total number of components. Nonetheless, this additional logarithm is indeed a finite quantity for all practical circuits and the equation (3.17) still shows that reliable computation can be achieved in *efficiently* with components satisfying the threshold condition (3.6). This very important result is known as the threshold theorem:

Theorem 3.2.1 (Threshold): *Let $S_G = \{g_1, \dots, g_r\}$ be a set of logic gates generating universal computation and $S_P = \{G_1, \dots, G_R\}$ be a set of processors weakly p -computing these gates with $p < \frac{1}{3}$. Given a boolean function F implemented by a circuit of N logic gates of S_G , there exists an arrangement of $O(N \log(\frac{N}{\epsilon}))$ units made of processors of S_P and perfect 3-majority gates which ϵ -computes F .*

Stated differently, the overhead imposed by the deficiency, below threshold, of the components G_1, \dots, G_r is only *polylogarithmic* in:

- The number of gates involved in the original circuit of reliable components.

- The reliability requirement set for the computation.

At the scale of any practical problem, logarithms are bounded by constants. The threshold theorem is thus equivalent to saying that any boolean function may be computed with arbitrary reliability at the cost of a *constant* overhead factor. This is remarkable.

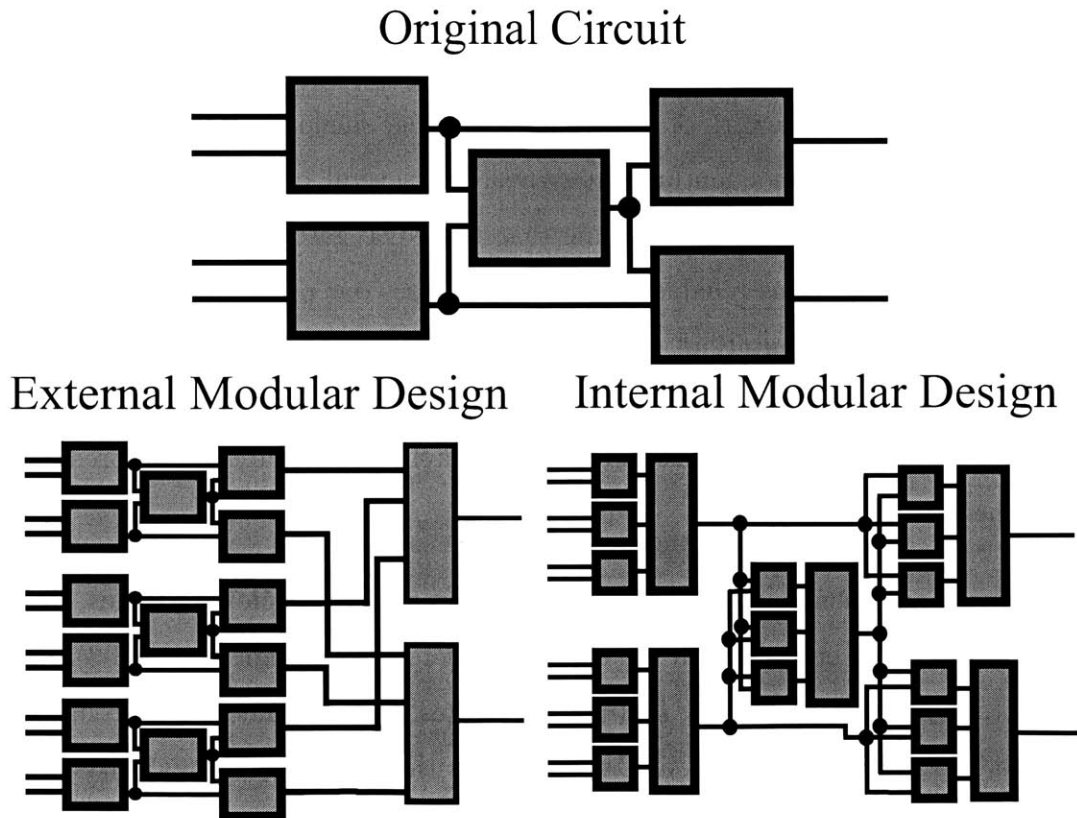


Figure 3-3: External and internal modular design. In the external modular design, a majority voting is taken on the output of three replications of the original circuit. In the internal modular design, each gate of the circuit is replaced by the corresponding simple majority voting scheme. Redundancy is present at the elementary level in the internal modular design.

3.3 Modular Redundancy with imperfect majority voting

As advertised earlier, we now release the assumption that 3-majority gates be perfect, which is indeed not legitimate for several reasons.

In Chapter 2, we set ourselves the goal to design fault-tolerant circuits in which no component is assumed to be ideally reliable. Even if we do not take this approach, the design of boolean schemes in which a subclass of logic operations is reliable could be of interest: for technological reasons some operations may be more reliably performed than others. In the framework of quantum computing, single qubit gates are much easier to implement than quantum gates involving several qubits at a time. For instance, it would make sense to assume that the NOT gate never fails. But the choice of the 3-majority as reliable operation trivializes our problem since this gate generates universal computation! This point seems to spoil the threshold theorem previously stated, since any boolean function can indeed be *reliably* implemented with perfect 3-majority voting systems and with maybe less resources than predicted by this theorem.

Our main argument was the following: in the case of complex and thus likely to fail processors P , embedded in an external fault-tolerant design (figure 3-3), it is possible to neglect the deficiency of 3-majority gates. This argument is also flawed. What we implicitly assumed in our discussion was that any of the 3-majority gates in the recursive construction was much more reliable than its inputs. While this assumption might be justified in the first stages of majority voting, it inevitably breaks down after a certain number of majority votes. Precisely, if we take for the 3-majority a failure probability $\epsilon \ll p$, after k_0 (equation (3.15)) stages of majority voting, the inputs become more reliable than the coming 3-majority gates (figure 3-4). The failure probability of the majority gates is then ultimately the limiting factor in the reliability amplification.

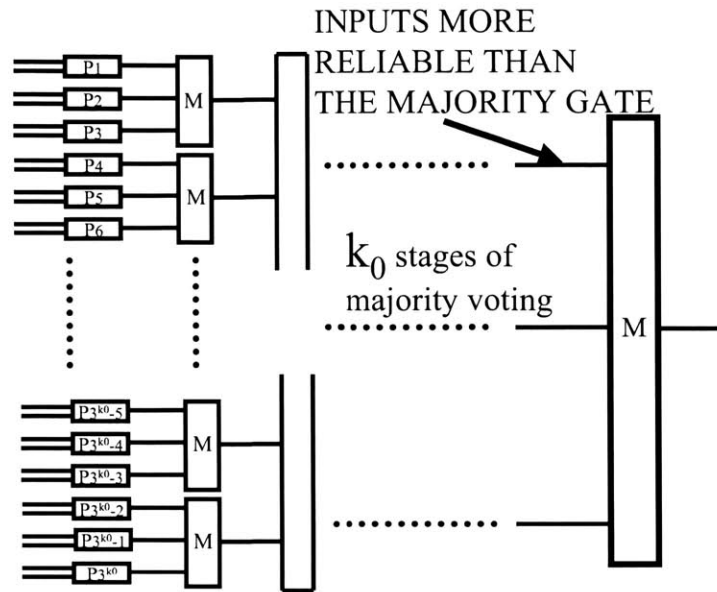


Figure 3-4: Recursive majority voting. If the majority gates have a non zero failure probability, after a sufficient number of stages of majority voting, the assumption that majority gates be less likely to fail than its inputs to deviate becomes invalid.

3.3.1 Reliability improvement through imperfect majority voting

We would now like to study quantitatively the degradation occurring in the previous schemes when 3-majority gates become imperfect. We will do so by answering the following questions:

1. What is the deficiency threshold of the 3-majority gates, that is the maximum failure probability of these components compatible with reliability improvement through majority voting ?
2. What is the reliability asymptotically obtained when TMR implementing a noisy processor with imperfect 3-majority gates at arbitrarily high recursion levels ?

We follow the same line of thought as in the proof of the claim (3.2.1). The conclusions are summed up in the following result, established by Von Neumann [Neu56]:

Claim 3.3.1 (Asymptotic reliability with faulty majority voting):

Let P and M be two processors respectively strictly p -computing a boolean function $F : \{0, 1\}^m \rightarrow \{0, 1\}$ and strictly ϵ -computing a 3-majority gate in the sense of definition (2.3.4).

- If $\epsilon \geq \frac{1}{6}$ the majority voting with M cannot bring any reliability improvement, and recursive TMR implementations of P lead asymptotically to irrelevant results.
- If $\epsilon < \frac{1}{6}$ majority voting with M is useful only if $p > p_\infty$, asymptotic failure probability of recursive TMR implementations of P :

$$p_\infty = \frac{1}{2} \left(1 - \sqrt{\frac{1-6\epsilon}{1-2\epsilon}} \right) \quad (3.18)$$

Proof: Following the definition (2.3.4), we assume that $p < \frac{1}{2}$. The failure probability of the TMR-implementation of level k is then $p^{(k)} = f_\epsilon^{(k)}(p)$, where:

$$\begin{aligned} f_\epsilon(p) &= \epsilon(1 - f(p)) + (1 - \epsilon)f(p) \quad \text{where} \quad f(p) = 3p^2 - 2p^3 \\ f_\epsilon(p) &= \epsilon + (1 - 2\epsilon)(3p^2 - 2p^3) \end{aligned} \quad (3.19)$$

As in the proof of the claim (3.2.1), we need to study the behavior of the recursive sequence $p^{(k)}$. f_ϵ is again increasing, so the sequence $p^{(k)}$ is monotonous. The sign of the polynomial of degree 3 $g_\epsilon(p) = f_\epsilon(p) - p$ determines whether the sequence $p^{(k)}$ is increasing or decreasing. Its possible limits are given by the roots of this polynomial. $\frac{1}{2}$ is a physically obvious fixed point of the function f_ϵ , and indeed one easily verifies that $p - \frac{1}{2}$ is a divider of $g_\epsilon(p)$. After factorizing this polynomial, we are left over with a polynomial of degree 2 whose roots can be analytically expressed. All the fixed points of f_ϵ are then identified. Precisely:

$$g_\epsilon(p) = \epsilon + (1 - 2\epsilon)(3p^2 - 2p^3) - p = 2 \left(\frac{1}{2} - p \right) [(1 - 2\epsilon)p^2 - (1 - 2\epsilon)p + \epsilon] \quad (3.20)$$

The discriminant corresponding to the second factor is:

$$\Delta = (1 - 2\epsilon)^2 - 4\epsilon(1 - 2\epsilon) = (1 - 2\epsilon)(1 - 6\epsilon) \quad (3.21)$$

We may then distinguish between three regimes.

- $\frac{1}{2} < \epsilon$: In this range, $\Delta > 0$ so the second factor in g_ϵ has two real roots:

$$q_{\frac{1}{2}} = \frac{1}{2} \left(1 \mp \sqrt{\frac{1 - 6\epsilon}{1 - 2\epsilon}} \right) \quad (3.22)$$

These roots are symmetric with respect to $\frac{1}{2}$, which is consistent since a processor taking (x_1, \dots, x_m) as inputs and whose output is $F(x_1, \dots, x_m)$ with probability $p > \frac{1}{2}$ also computes $\bar{F}(x_1, \dots, x_m)$ with failure probability $1 - p$. By inspection if the roots:

$$q_1 < 0 < 1 < q_2 \quad \Rightarrow \quad (1 - 2\epsilon)p^2 - (1 - 2\epsilon)p + \epsilon > 0 \quad \text{on} \quad [0, 1] \quad (3.23)$$

Therefore the sign of $g_\epsilon(p) = f_\epsilon(p) - p$ is that of $(\frac{1}{2} - p)$ on this interval. Consequently:

$$p < \frac{1}{2} \quad \Rightarrow \quad f_\epsilon(p) > p \quad \Rightarrow \quad \forall k > 0 \quad p^{(k+1)} > p^{(k)} \quad (3.24)$$

The sequence $p^{(k)}$ converges towards the fixed point $\frac{1}{2}$: $\lim_{k \rightarrow +\infty} p^{(k)} = \frac{1}{2}$

The outputs become less and less reliable after each majority voting and eventually irrelevant.

- $\frac{1}{6} < \epsilon < \frac{1}{2}$: In this range, $\Delta < 0$ so the polynomial $g_\epsilon(p)$ has only one real root: $p = \frac{1}{2}$. Also:

$$\forall p \in [0, 1] \quad (1 - 2\epsilon)p^2 - (1 - 2\epsilon)p + \epsilon > 0 \quad (3.25)$$

Thus the sign of g_ϵ is that of $\frac{1}{2} - p$. The conclusion of the previous case still applies: the majority voting does not bring any reliability improvement. Since

the sequence $p^{(k)}$ is bounded, it converges towards $p = \frac{1}{2}$: the outputs are eventually irrelevant.

- $0 < \epsilon < \frac{1}{6}$: In this range $\Delta > 0$, so the polynomial $g_\epsilon(p)$ has again three real roots, which satisfy: $0 < q_1 < \frac{1}{2} < q_2 < 1$. Consequently:

$$\begin{aligned} (1 - 2\epsilon)p^2 - (1 - 2\epsilon)p + \epsilon &\geq 0 && \text{on } [0, q_1] \\ &\leq 0 && \text{on } [q_1, \frac{1}{2}] \end{aligned} \quad (3.26)$$

Since $p < \frac{1}{2}$, $g_\epsilon(p)$ and this polynomial are of the same sign.

The sequence $p^{(k)}$ is thus:

- increasing for $0 < p < q_1$
- decreasing for $q_1 < p < \frac{1}{2}$

Hence the recursive majority voting improves the reliability of the processor P iff $q_1 < p < \frac{1}{2}$, and recursive circuits fail asymptotically with probability q_1 .

This ends up the proof of the claim.

Remark 1: The asymptotic failure probability depends exclusively on the reliability of the majority voting system and not of the level of deficiency of the processor P . In the case of fairly reliable 3-majority gates, for $\epsilon \ll \frac{1}{2}$, the reliability obtained asymptotically is indeed very close to that of the 3-majority gate itself:

$$p_\infty = \epsilon + 3\epsilon^2 + O(\epsilon^3) \quad (3.27)$$

We have thus shown that the recursive triple modular redundant implementation of P as defined in (3.2.1) leads to schemes whose reliability is ultimately limited by that of the 3-majority gates. From this point of view, the external modular design is more advisable than the internal one. In the external design, this limitation intervenes only at the final stage of the computation, yielding a circuit whose

deficiency is close to that of a 3-majority gate. In the internal design, this limitation is present at the level of any gate g_i of the original combinational circuit, resulting in a greater failure probability of the internal modular circuit. Independently from this consideration, none of these schemes enables arbitrarily reliable computation with imperfect 3-majority gates.

Remark 2: This result allows us to upper-bound the threshold p_{th} (condition (3.6)) compatible with the obtention of reliable circuits through circuits involving 3-majority components: we have proven that the use of 3-majority gates worsens the reliability of a computation if their failure probability is above $\epsilon_0 = \frac{1}{6}$. But this value is specific to these devices: Evans and Pippenger [EP98] showed that it is indeed possible to use arbitrarily deficient d -majority gates to perform reliable computations, provided that d is large enough.

3.3.2 Distributed Majority Voting

Motivation

Before moving onto the description of this design technique, let us investigate to what extent the constructions exposed so far respected the guidelines of fault-tolerant design exposed in section 2.2. From now on, the boolean function F computed will simply be a NAND gate, consistently with the goal set in Chapter 2.

All the circuits presented so far in this chapter were taking inputs recursively encoded in the triple repetition code and delivering a single bit output, thus decoding enroute the data. This consideration shows that we have omitted in those schemes a cornerstone of the fault-tolerant design, which is the computation on encoded data: the decoding operated by the 3-majority gates made the data highly sensitive to failures of those components. We should transform the “restoring organ” in such a way that this decoding does not take place.

Diverse encodings may be considered for the inputs. Let us focus on the simplest class of codes consisting in the d -repetition of a logical bit. We also require that d be

an odd integer. The purpose of this restriction is to avoid an asymmetric decoding scheme, in which an erroneous word containing an even number of 1 and 0 would be projected onto a codeword constituted only of 1 or 0. We could also consider a decoding declaring a failure (error detection) in the latter case, but we choose to simply avoid this complication by imposing that d be odd and using d -majority voting gates.

We then realize a construction using d identical NAND and d -majority gates as presented on figure 3-5 for $d = 3$. As advertised, this scheme no longer performs any decoding on the computed data. Taking the convention of the chapter 2, we shall say that this circuit (T, D) -computes a NAND gate where T is a d -replication encoding and D the decoding procedure “majority voting.”

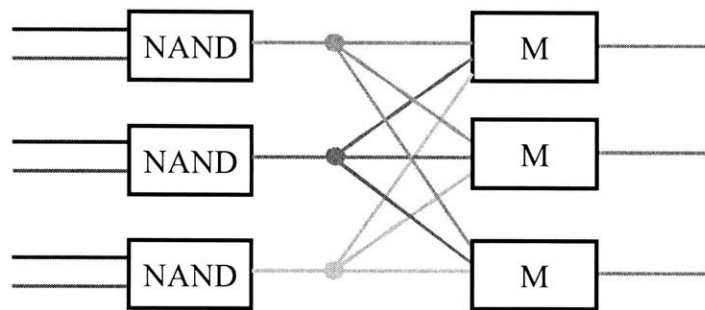


Figure 3-5: Distributed majority voting. The restoration of the codeword is now performed by three 3-majority gates instead of one.

Encoded Reliability of an encoded NAND gate taking perfect inputs

We assume here that the system on figure 3-5 receives perfectly encoded inputs. The computation within this circuit is now immune to $\lceil \frac{d}{2} \rceil$ component failures. More precisely, a fault-path causing a failure of the encoded computation must include:

- Either $\lceil \frac{d}{2} \rceil$ NAND gate failures. There are $\binom{d}{\lceil \frac{d}{2} \rceil}$ possible choices.
- Or $\lceil \frac{d}{2} \rceil$ 3-majority gate failures. Again, there are $\binom{d}{\lceil \frac{d}{2} \rceil}$ possible choices for the location of these errors.

The probability that the encoded NAND fails is then bounded by:

$$p_{fail} \leq \binom{d}{t} p_1^t + \binom{d}{t} p_2^t \quad \text{with} \quad t = \lceil \frac{d}{2} \rceil \quad (3.28)$$

where p_1 and p_2 are the failure probabilities associated respectively with a NAND and with a d -majority gate. When d is high, the probability p_2 might be considerably higher than p_1 for similar resource allocation to both components. In the coming chapters we will assume that the reliability of all components depends equally on the resources, and that those resources are equally divided among the devices. We will take therefore $p_1 = p_2 = p$, considering consistently systems involving exclusively 3-majority and NAND gates. Under this assumption, the failure probability of an encoded NAND is upper-bounded by:

$$p_{fail} \leq 6 p^2 \quad (3.29)$$

Therefore the threshold probability of an encoded NAND (through the triple repetition code) taking perfect inputs is: $p_{th} = \frac{1}{6} \simeq 17\%$

Reliability of an encoded NAND gate in a noisy circuit

In Chapter 2 we were concerned about the fact that, even if the output of an encoded gate is correct in the sense of decodable (definition (2.3.10)), it might be corrupted and lead to computations erroneous with high probability in the next encoded gates. This motivates the study of an encoded gate taking inputs which might *deviate* (definition (2.3.8)), but which are still correct. On figure 3-6, the inputs of the second encoded NAND gate might be altered if the first one does not deliver a perfectly encoded output.

The structure of the encoding makes this partial corruption of the output quite simple to model. An inspection of the encoded NAND on the figure 3-5 reveals that:

- Either $\lceil \frac{d}{2} \rceil = 2$ or more wires feeding the d -majority gates deviate. Then the output is wrong (in the sense of definition (2.3.10)) with high probability.

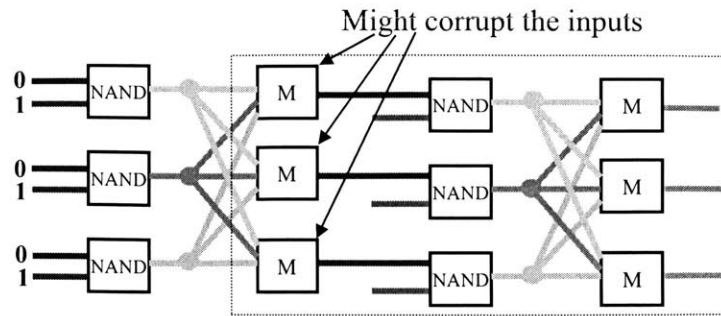


Figure 3-6: Sequence of two encoded NAND gates. A failure in a 3 majority gate of the first encoded NAND may corrupt an input of the second one.

- Or $\lfloor \frac{d}{2} \rfloor = 1$ or less wires feeding the d -majority gates deviate. In this case the corruption of each bit in the output is caused by an error in a d -majority gate.

In other words:

When the computation is correct, the corruption of each output bit of an encoded NAND is equivalent to the failure of a d -majority gate.

In order to consider the behavior of a NAND in a circuit of encoded logic, we need to account for the previous majority gates as possible source of errors in the inputs, as on figure 3-7.

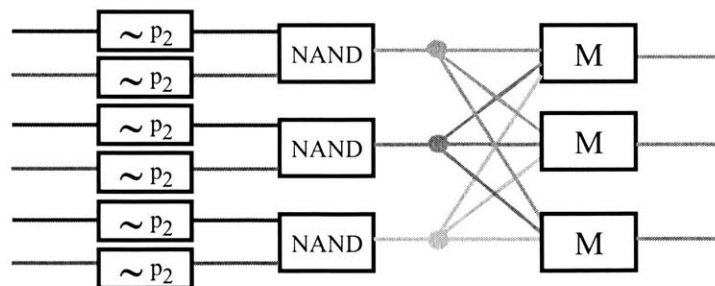


Figure 3-7: Distributed majority voting system taking imperfect inputs. The boxes before the NAND gates correspond to the sources of errors in the inputs introduced by previous encoded gates.

We consider now an encoded NAND taking two inputs issued by previous similar gates. We assume that these inputs are correct: their decoding through majority

voting leads to the right logical bit. Again, the encoded computation is immune to $\lfloor \frac{d}{2} \rfloor$ or less processors failures or input deviations. Let us count the possible events with $t = \lceil \frac{d}{2} \rceil$ failures or input deviations and leading to a wrong output:

- t failures in the d -majority gates. There are $\binom{d}{t}$ such possible events.
- k failures in the NAND gates and $t - k$ deviations in the inputs for $k = 1, \dots, \lceil \frac{d}{2} \rceil$. In order for the output to be wrong, each deviating input must be attached to a different NAND, chosen among the $d - k$ NAND which have not failed. There are $\binom{d}{k} 2^{t-k} \binom{d-k}{t-k}$ such possible events.
- t deviations in the inputs. In order for the output to be wrong, each deviating input must again feed a different NAND. Furthermore, since the inputs were assumed to be correct in the first place, these t deviations cannot affect simultaneously only one of the two input codewords. There are $2^t \binom{d}{t} - 2 \binom{d}{t}$ such possible events.

Any event spoiling the encoded computation must have a subset of t processors and inputs which fail or deviate according to one of the enumerated cases. An input deviation is equivalent to a previous d -majority gate failure. Since the error probabilities in the NAND and d -majority gates are respectively bounded by p_1 and p_2 , the failure probability of the encoded NAND is at most:

$$p_{fail} \leq \binom{d}{t} p_2^t + \sum_{k=1}^t 2^{t-k} \binom{d}{k} \binom{d-k}{t-k} p_1^k p_2^{t-k} + (2^t - 2) \binom{d}{t} p_2^t \quad \text{with } t = \lceil \frac{d}{2} \rceil \quad (3.30)$$

Let us concentrate on the special case $d = 3$ and $t = 2$, assuming that $p_1 = p_2 = p$. The previous formula becomes:

$$\begin{aligned} p_{fail} &\leq \binom{3}{2} p^2 + 2 \binom{3}{1} \binom{2}{1} p^2 + \binom{3}{2} p^2 + (4 - 2) \binom{3}{2} p^2 \\ &\leq c_3 p^2 \quad \text{with } c_3 = 24 \end{aligned} \quad (3.31)$$

This result shows that a NAND gate encoded through the triple repetition code and embedded into a noisy circuit of similar gates still fails with a probability $O(p^2)$. Nonetheless, because of the fluctuations in the inputs, the bound (3.31) we obtained is higher than the bound (3.29) with perfect inputs. This new bound corresponds to a threshold:

$$p_{th} = \frac{1}{c_3} = \frac{1}{24} \simeq 4\% \quad (3.32)$$

Since we aim at designing gates integrated into large circuits of encoded logic, we adopt from now on this value for the threshold.

3.3.3 Recursive multiplexing

Seeking the best architecture for the embedding of faulty components into reliable circuits, we are faced with an apparent dilemma:

- On the one hand, we would like to preserve the principle of successive hierarchical voting which leads to a double exponential decrease in the failure probability of the global circuit.
- On the other hand, the construction of arbitrarily reliable systems with components of fixed deficiency require that none of them should be so crucial as to compromise the computation by its failure. In the simple triple modular redundant design, this condition was obviously not fulfilled, since a single error in the last majority voting would generate a wrong result with high probability. This remark has led us to equalize the impact of each voting device in the circuit and adopt the multiplexing as a design principle.

There is actually no incompatibility between those design guidelines: we may build circuits computing on a code with a recursive structure provided by the concatenation, but in which the computation is distributed over many devices of similar importance. We now combine recursive majority voting and multiplexing to get the best of both worlds.

Let us start by considering a multiplexed circuit of recursion level 2. From this example, we will generalize to constructions of higher recursion levels. This scheme, presented on figure 3-8, is obtained from the multiplexed circuit on figure 3-5 by the following procedure:

- Replace each NAND gate in the circuit of figure 3-5 by its multiplexed version. The former NAND are now encoded gates taking two bundles in input and delivering one bundle in output.
- Operate majority voting on the three output bundles in the following way. Let us note $b_{1,1}, b_{1,2}, b_{1,3}$ the outputs of the upper multiplexed NAND, $b_{2,1}, b_{2,2}, b_{2,3}$ the outputs of the second multiplexed NAND, and $b_{3,1}, b_{3,2}, b_{3,3}$ that of the bottom multiplexed NAND. Perform then three 3-majority voting in parallel on the wires $b_{1,1}, b_{2,1}, b_{3,1}$, on the wires $b_{1,2}, b_{2,2}, b_{3,2}$ and on $b_{1,3}, b_{2,3}, b_{3,3}$. In order for each majority voting to be fault-tolerant, we do it three times in parallel, triplicating each output wire of the multiplexed NAND gates. The output bundle is the reunion of all 3-majority gates outputs.

We call $N^{(k)}$ and $M^{(k)}$ respectively a multiplexed NAND and a distributed majority voting system of level k . The procedure of recursive multiplexing may appear slightly complicated because the multiplexed processor $N^{(k)}$ involves both $N^{(k-1)}$ and $M^{(k)}$, the latter being defined through a distinct recursion. As in the simple recursive triple modular redundant design, each increment of the recursion level creates a new stage of majority voting. The multiplexing at an arbitrary recursion level is described through the definitions (3.3.1), (3.3.2) and (3.3.3) which are gathered on a single page for convenience. These definitions are also illustrated on figure 3-9.

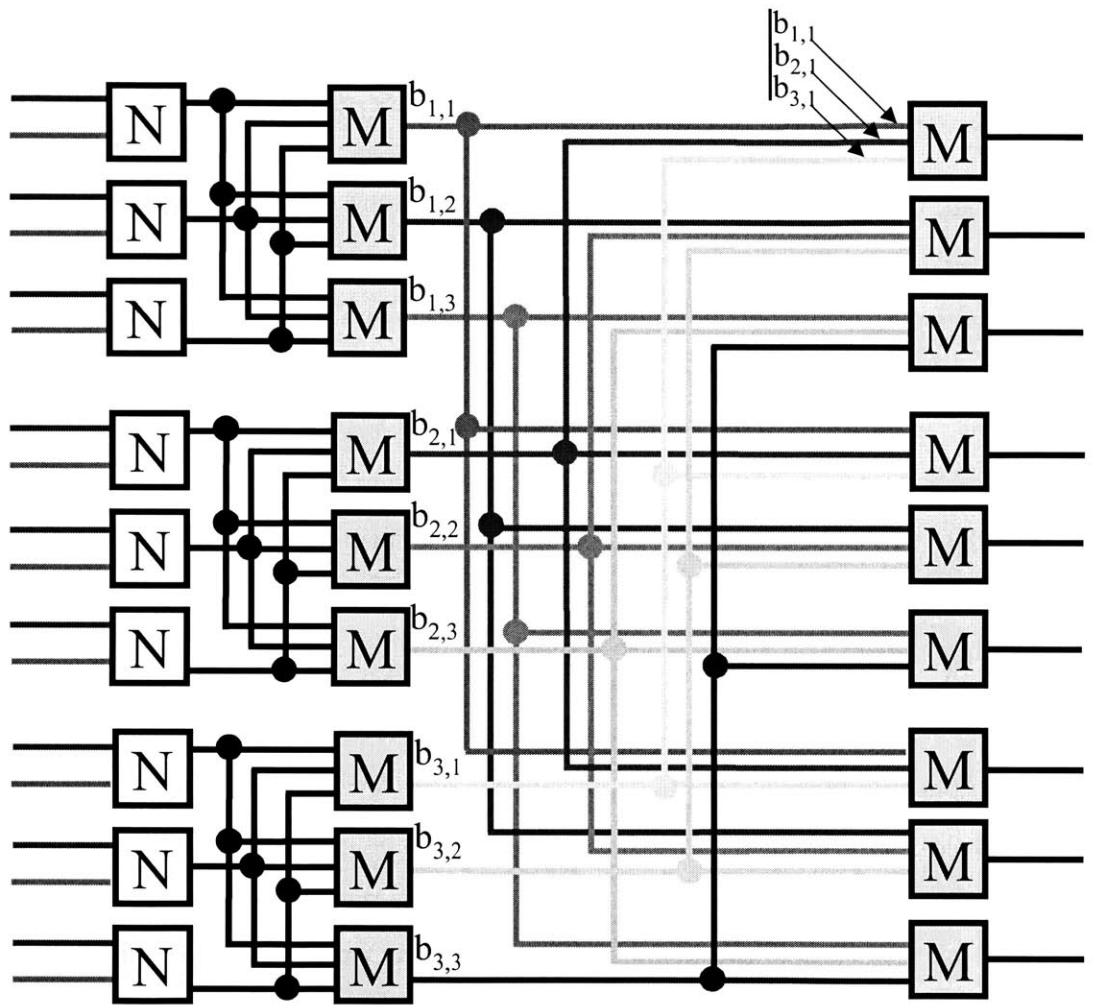


Figure 3-8: Multiplexed NAND of recursion level 2.

Definition 3.3.1 : We call $b^{(k)}$ -bundle a bundle of 3^k wires. A $b^{(k)}$ -bundle contains three $b^{(k-1)}$ -bundles which we note $b_1^{(k-1)}, b_2^{(k-1)}, \dots, b_{3^{k-1}}^{(k-1)}$.

Definition 3.3.2 (Recursive Distributed 3-Majority Voting):

A recursive distributed 3-majority voting system of level k , noted $M^{(k)}$, is a gate taking in input the data carried by a $b^{(k)}$ -bundle of wires and whose output is defined recursively as being:

- for $k=1$ the majority vote of the three single wires $b_1^{(0)}, b_2^{(0)}, b_3^{(0)}$.

- for $k \geq 2$ it is obtained by the following procedure.

The input $b^{(k)}$ consists in 3 bundles $b_1^{(k-1)}, b_2^{(k-1)}, b_3^{(k-1)}$

or in 9 bundles $b_{1,1}^{(k-2)}, b_{1,2}^{(k-2)}, b_{1,3}^{(k-2)}, \dots, b_{3,3}^{(k-2)}$.

We reorganize the bundles $b_{1,j}^{(k-2)}, b_{2,j}^{(k-2)}, b_{3,j}^{(k-2)}$ into a bundle $c_j^{(k-1)}$.

We send each of the $c_j^{(k-1)}$ into a majority voting $M^{(k-1)}$.

The reunion of their outputs noted $d_j^{(k-2)}$ forms a bundle $d^{(k-1)}$ which defines the output of $M^{(k)}$.

Definition 3.3.3 (Recursive Multiplexed Triple Modular Implementation):

A multiplexed triple modular redundant (MTMR) implementation of level k of a NAND gate, noted $N^{(k)}$, is a gate which takes in input two bundles $a^{(k)}, b^{(k)}$ (each corresponding to the encoding of a logical bit) and whose output is recursively defined as being:

- for $k = 0$ the output of a NAND
- for $k \geq 1$ the scheme obtained by the following procedure.

The two input bundles $a^{(k)}, b^{(k)}$ may be decomposed into three pairs of bundles $a_j^{(k-1)}, b_j^{(k-1)}$ which are sent into an encoded NAND $N^{(k-1)}$ for each $j \in \{1, 2, 3\}$.

The outputs of the three NAND $N^{(k-1)}$ form a bundle $c^{(k)}$, which is sent into three identical distributed 3-majority voting systems $M^{(k)}$. The reunion of their outputs $d_j^{(k-1)}$ forms a bundle $d^{(k)}$ defining the output of $N^{(k)}$.

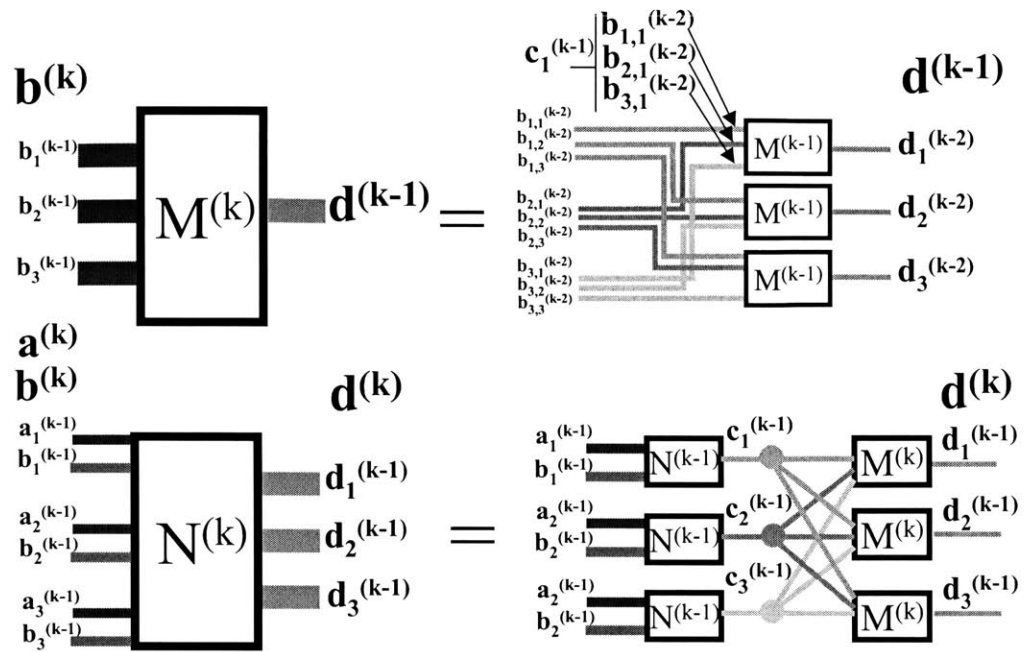


Figure 3-9: Recursive multiplexing of a NAND gate.

Recursively multiplexed circuits of level k perform indeed an encoded computation on the triple repetition code concatenated k times with itself. The hierarchical structure of recursive concatenation gives to the code the shape of a balanced tree (figure 3-10). Because we are using a single code, the triple repetition, each node has exactly three branches. Elementary 3-majority gates operate on the leaves, while the subsystems $N^{(1)} \dots N^{(k-1)}$ contained in $N^{(k)}$ act on subtrees.

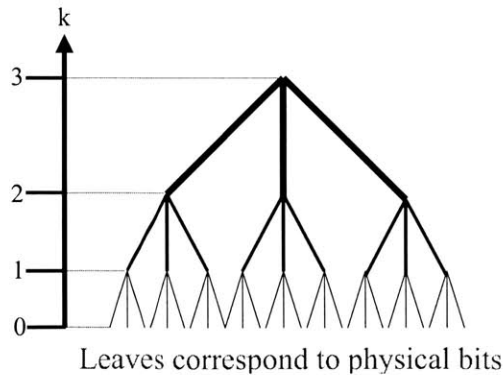


Figure 3-10: Structure of the code : ternary tree of depth k .

3.3.4 Reliability and resource consumption in recursively multiplexed circuits

We now express the reliability and resource consumption of multiplexed circuits as a function of their recursion level. Fortunately, those circuits turn out to be almost as efficient as the schemes of section 3.2 with perfect majority voting. In other words, the additional cost implied by the faults in the 3-majority gates does not prevent the obtention of reliable circuits with an acceptable resource overhead.

Resource Scaling

An immediate recursion shows that a 3-majority distributed voting system of level k involves 3^k majority gates. We note $N_1(k)$ and $N_2(k)$ the respective numbers of NAND and 3-majority gates in an encoded NAND of level k . The definition of recursive multiplexing then implies the relations:

$$N_1(k) = 3N_1(k-1) \quad \text{and} \quad N_2(k) = 3N_2(k-1) + 3^k \quad (3.33)$$

Using the fact that $N_1(0) = 1, N_2(0) = 0$ this recursion is easily solved in:

$$N_1(k) = 3^k \quad \text{and} \quad N_2(k) = 3^k k \quad (3.34)$$

Thanks to our assumption that 3-majority and NAND gates have the same technology, we may simply consider that the circuit involves: $N(k) = 3^k(k+1)$ devices. If we suppose that all the devices are granted the same layout area A_0 and the same power P_0 , neglecting also the additional space occupied by the wiring, we obtain the area and power consumption associated with a MTMR scheme of level k :

$$A(k) = 3^k(k+1) A_0 \quad \text{and} \quad P(k) = 3^k(k+1) P_0 \quad (3.35)$$

It is also of interest to estimate the scaling of the duration of the encoded computation with the recursion level. This time is proportional to the depth of the circuit,

that is the maximum number of components separating an input from the output. Each increment of the recursion level adds a stage of majority voting. Noting T_0 the computing time associated with an elementary NAND or with a 3-majority gate (both times are taken as identical), we obtain a processing time for the gate $N^{(k)}$:

$$T(k) = (k + 1) T_0 \quad (3.36)$$

The resource “time” scales thus linearly with the recursion level k .

Reliability of an encoded NAND taking perfectly encoded inputs

We can attempt to compute the exact failure probability of a multiplexed system with reliably unreliable components, that is devices strictly p_1 -computing NAND gates and strictly p_2 -computing 3-majority gates. We note $p_1^{(k)}$ and $p_2^{(k)}$ the respective failure probabilities of the systems $N^{(k)}$ and $M^{(k)}$. The distributed voting system $M^{(k)}$ fails only if at least two of the subsystems $M^{(k-1)}$ fail. The multiplexed system $N^{(k)}$ fails if at least two of the processors $N^{(k-1)}$ fail while the distributed voting systems $M^{(k)}$ succeeds or if at least two of the $N^{(k-1)}$ succeed while $M^{(k)}$ fails. These events are disjoint so their probabilities add. Taking again the function f associated with a perfect majority voting:

$$f(x) = 3x^2(1 - x) + x^3 \quad (3.37)$$

we obtain the following recursive relations for $p_1^{(k)}$ and $p_2^{(k)}$:

$$p_1^{(k)} = f\left(p_1^{(k-1)}\right)\left(1 - p_2^{(k)}\right) + f\left(1 - p_1^{(k-1)}\right)p_2^{(k)} \quad \text{and} \quad p_2^{(k)} = f\left(p_2^{(k-1)}\right) \quad (3.38)$$

These equations maybe rewritten in a more symmetric way:

$$\begin{aligned} p_1^{(k)} &= f_{3,p_2^{(k)}}\left(p_1^{(k-1)}\right) = f\left(p_1^{(k-1)}\right)f\left(1 - p_2^{(k-1)}\right) + f\left(1 - p_1^{(k-1)}\right)f\left(p_2^{(k-1)}\right) \\ \text{and} \quad p_2^{(k)} &= f^{(k)}(p_2) \end{aligned} \quad (3.39)$$

$p_1^{(k)}$ is expressed through the function $f_{3,\epsilon}(p)$, deviation probability of the output of a 3-majority gate failing with probability ϵ and whose inputs deviate with probability p . The expression of $p_1^{(k)}$ at an arbitrary recursion level k is a complicated function of p_1, p_2 . It will reveal useful, in the next chapters, to have an expression for the reliability of an encoded NAND through a simpler invertible bound. Besides, the latter computation assumed “reliably unreliable” components, requirement which is too stringent in many design situations.

In order to get a more general and useful result, we now suppose that the components *weakly* p -compute the NAND and the 3-majority gates, and we seek a simple bound for the failure probability of the encoded NAND $N^{(k)}$. By noticing that $f(p) \leq 3p^2 \leq 1$, provided that $p \leq \frac{1}{3}$, we obtain that $p_1^{(k)} \leq 6 p_1^{(k-1)2}$. The number 6 corresponds indeed to the possible error locations with *perfect* inputs. As in the bound (3.10), $p_1^{(k)}$ decreases doubly exponentially with the recursion level k :

$$p_1^{(k)} \leq \frac{1}{c} (c p)^{2^k} \quad \text{with } c = 6 \quad (3.40)$$

Reliability of an encoded NAND taking corrupted inputs

As we did for simple distributed schemes in section 3.3.2, we want to take into account the input noise of an encoded NAND inserted in a circuit of similar gates.

We can indeed directly transpose the argumentation and counting developed in section 3.3.2, replacing the term “wire” by “bundle”: in an encoded NAND $N^{(k)}$, either two or more bundles $c_j^{(k-1)}$ feeding its 3-majority voting systems $M^{(k)}$ deviate and the result of the encoded NAND $N^{(k)}$ is wrong with high probability, or the corruption in the output wires originates from the systems $M^{(k)}$. Therefore the results obtained for the distributed majority voting system taking imperfect inputs apply. The failure probability of $N^{(k)}$ decreases thus doubly exponentially with the recursion level k , but we must take the new threshold value $c_3 = 24$ valid for noisy inputs:

$$p_1^{(k)} \leq \frac{1}{c_3} (c_3 p)^{2^k} \quad \text{with } c_3 = 24 \quad (3.41)$$

3.4 Conclusion: Threshold Theorem with imperfect majority voting

This chapter exposed how the duplication of a computation, combined with an appropriate recursive structure, could provide an efficient protection against device failures. The discussion began with the assumption that reliable majority gates were available. In this frame a recursive construction was derived, which enabled the reliable computation of formula through networks of faulty components at the cost of a logarithmic overhead, provided that their reliability was above a threshold. The concept of distributed majority voting was then introduced, circumventing the requirement of perfect majority voting, and enabling the construction of satisfactory restoring organs out of unreliable 3-majority components. This gain in generality came at the following price:

- The requirement on the maximum deficiency is more stringent with imperfect majority voting: the threshold probability is $\frac{1}{24}$ instead of $\frac{1}{3}$ with reliable 3-majority gates.
- The number of gates with imperfect majority voting involved is $N(k) = (k+1)3^k$ instead of $N(k) = \frac{3}{2} 3^k$ with reliable 3-majority gates.

Asymptotically the latter change is minor since the number of gates still scales exponentially. Therefore *the threshold theorem (3.2.1) prevails even with imperfect majority voting*, which is an important result. The effect of imperfect majority voting is mostly to decrease the threshold, that is the maximum failure probability of the components compatible with a reliability improvement through a redundant design.

So far this theorem has been applied with the purpose of proving the existence of efficient constructions simulating reliable circuits with faulty components. In those constructions, the failure probability of the elementary components is fixed and does not increase with the recursion level. This is not the case in real design situations, where a certain amount of resources is shared among the different constituents of the circuit. In the following chapters, we analyze the possibilities offered by the threshold theorem, taking this time into account the drop in resource availability for each device.

Chapter 4

Fined Grained Fault-Tolerance: Qualitative Approach

We now introduce a new idea, which will underlie all the further developments in this thesis: the failure probability of a logic component is a continuous function of different technological resources, typically its layout area, power consumption and time processing. This concept is indeed extremely general and relevant for any real computing system. In this line of thought, reliability becomes itself a fully *fungible* asset in the following sense: for any device consuming a certain amount of technological resources and with a finite failure probability, there exists an other component consuming less and whose reliability is arbitrarily close. In other words, any arbitrarily small amount of reliability may be given up to save some area, power or time. There is thus no fundamental reason to distinguish between the former and the latter while optimizing the performance of a circuit. We investigate the application of the fault-tolerant constructions reviewed in Chapter 3 into this new context, procedure which is original to our knowledge.

This chapter analyzes qualitatively the potentialities offered by this design method along the following outline. Section 4.1 describes the noise model in CMOS transistors proposed and tested by Sarpeshkar, Delbruck and Mead [RDM93], which shows explicitly how physical resources condition the failure probability of this component. This example will serve in Chapter 5 as a case study of our design methods. Sec-

tion 4.2 formulates the problem of optimization through a fine-grained fault-tolerant design. Section 4.3 brings preliminary answers to this question. In particular, this section gives a general mathematical condition on the dependence of reliability towards the resources, which states whether a fault-tolerant design is efficient. Section 4.4 explores the full conversion possibilities of the fungible resource “reliability”. Section 4.5 concludes on the results exposed in this chapter.

4.1 Resource-dependent reliability:

Example of the CMOS transistor

4.1.1 Notion of noise

Because of their crucial role in the electronics industry, the behavior of CMOS transistors has been carefully studied by the electrical engineering community. Accurate noise models for this component have been established and verified experimentally [RDM93] which we describe here, following closely the approach of [Sar98].

The notion of noise is concomitant to that of randomness and irrelevancy. A source of noise is a random variable whose result has been declared irrelevant. In the case of additive noise, this random variable is added to an other one considered as relevant, which is the source of information. The behavior of an electronic device is described through the voltages between different points of the component. Any voltage v is the sum of two terms:

- v_S , random voltage called signal, which is a function of random variables attached to a source of information.
- v_N , random voltage called noise, whose origin is the multitude of uncontrolled physical processes happening at the microscopic scale.

In digital circuits, the voltage $v = v_S + v_N$ is later converted into a boolean through a map $B : \Re \rightarrow \{0, 1\}$. The signal voltage, attached to a discrete and finite set of boolean variables, should take discrete values.

Let us focus on the voltage difference v between the output and the source of a CMOS transistor. The “source of information” is here the single bit carried by the output. The signal voltage should take one of the two possible values: $v_S \in \{v_0, v_1\}$. A typical choice for B is:

$$B(v) = \begin{cases} 0 & \text{for } v < \frac{v_0+v_1}{2} \\ 1 & \text{for } v > \frac{v_0+v_1}{2} \end{cases} \quad (4.1)$$

If we define v_{th} by:

$$v_{th} = \frac{v_1 - v_0}{2} \quad (4.2)$$

we obtain the probability that the output be wrongly interpreted for each value of the signal:

$$\begin{aligned} p(B(v) \neq B(v_S)|v_S = v_0) &= \int_{v_{th}}^{+\infty} dv p_V(v_N = v) \\ \text{and } p(B(v) \neq B(v_S)|v_S = v_1) &= \int_{-\infty}^{-v_{th}} dv p_V(v_N = v) \end{aligned} \quad (4.3)$$

It is reasonable to assume that the distribution of v_N is symmetric around the origin, since physical processes contributing to the noise voltage are generally invariant under parity. Consequently:

$$p(B(v) \neq B(v_S)|v_S = v_0) = p(B(v) \neq B(v_S)|v_S = v_1) \quad (4.4)$$

and thus the choice (4.1) for B leads to a symmetric error profile.

While microscopic physical events within the CMOS cannot be addressed individually, their statistical behavior and their impact on the noise voltage may be controlled: the distribution of v_N is a well-defined function of the resource allocation. Usually such functions may be simply bounded, but here it can be determined precisely: this is a typical case in which the processors can be taken as “reliably unreliable,” essentially because the different noise sources have been identified and measured. In less favorable situations such as radiative failures, where the noise is

inherent to environmental factors and may vary on time, this property is not valid anymore.

The noise voltage may be characterized by its power spectrum. We note v the voltage difference between gate oxide and the source of the transistor and A , I , P respectively the total area of the transistor, the DC current and the power going through this device. Power and current are related to each other by:

$$P = v_0 I \tag{4.5}$$

where v_0 is a fixed power voltage supply imposed by the technology of the CMOS transistor. Furthermore, we note f_h and f_l respectively the highest and lowest frequencies of operation. We will rather think in terms of output current than in terms of voltage between the gate oxide and the source, but both descriptions are equivalent thanks to Ohm's law. In a CMOS transistor, the noise receives two important contributions.

Shot Noise

The shot noise comes from the wave-particle duality of the electrons involved in the electric current. Because of their incoherent relative motions, their arrival times on any cross section of the wire are Poisson-distributed. The Fourier transform of the autocorrelation function associated with a poissonian process (which is the power spectrum by Wiener-Khintchine's theorem) is the sum of a Dirac distribution and a constant term. The latter corresponds to the "white" part of the noise process. According to the law of large numbers, relative fluctuations of the current should decrease significantly with the number of particles going through a cross section of the wire per time unit. This prediction is verified to the extent that voltage fluctuations are decreasing functions of the current, but two regimes need to be distinguished,

depending on whether the transistor is functioning below or above threshold:

$$\Delta v_{SN}^2 = \frac{K_w(\gamma)}{I^\gamma} (f_h - f_l) \quad \text{with} \quad \gamma = \begin{cases} 1.0 & \text{for } P < P_{th} \quad (\text{subthreshold regime}) \\ 0.5 & \text{for } P > P_{th} \quad (\text{above threshold regime}) \end{cases} \quad (4.6)$$

$K_w(\gamma)$ and the threshold power P_{th} are fixed parameters inherent to the CMOS technology [Sar98]. Since v_{SN} is the sum of a great number of independent voltage fluctuations associated with each single charge, the central limit theorem shows that the corresponding voltage distribution is a Gaussian of standard deviation Δv_{SN} .

1/f Noise

This noise has a different physical origin. Because of the impurities in the gate oxide of the transistor, there is a leakage of the charges present on this surface, leading to fluctuations of the number of charges situated on the oxide. The current delivered by the transistor output is a function of the difference between the gate oxide voltage and the threshold (the common voltage reference being the source of the transistor). Therefore everything happens as if the threshold of the transistor was jittering, following the gate oxide voltage fluctuations. Let us see how these fluctuations scale with the resources. The conversion of charge variations into voltage variations depends on the capacitance C of the transistor, proportional to its area A :

$$\frac{dV}{dQ} = \frac{1}{C} \sim \frac{1}{A} \quad (4.7)$$

Thus the voltage power spectrum (given by $|\tilde{v}(f)|^2$ where $\tilde{v}(f)$ is the Fourier transform of the voltage evaluated at the frequency f) associated to the fluctuations of a *single* charge scales as A^{-2} . But the number of fluctuating charges is proportional to the number of “in and out” states for the charges (consequence of Fermi golden rule in quantum mechanics), which is in turn proportional to number of defects. Since the density of impurities is approximately constant, the number of fluctuating charges scales like the area A . Because the fluctuations associated with each impurity are independent, *their contribution to the power spectrum add*. Taking both effects into

account - the simultaneous increase of the capacitance and of the number of fluctuating charges -, we obtain a scaling of the voltage fluctuations as $A^{-2} \times A = A^{-1}$. This is reflected in the relation:

$$\Delta v_{1/f}^2 = \frac{K_f}{A} \int_{f_l}^{f_h} \frac{df}{f} = \frac{K_f}{A} \log \left(\frac{f_h}{f_l} \right) \quad \text{with} \quad K_f = \frac{B}{C_{ox}} \quad (4.8)$$

In this formula C_{ox} is the capacitance of the transistor per unit area, while B stands for the density of impurities in the oxide, constants typical of the material used in the transistor. The great number of tiny fluctuations associated with the $1/f$ noise results again in a gaussian voltage distribution of standard deviation $\Delta v_{1/f}$.

4.1.2 Resulting error probability

The output noise voltage $v_n = v_{SN} + v_{1/f}$ is the sum of two independent gaussian noise processes. Consequently, the corresponding distribution is a Gaussian of standard deviation:

$$\Delta v_n^2 = \Delta v_{SN}^2 + \Delta v_{1/f}^2 = \sqrt{\frac{K_w(\gamma)}{I\gamma} (f_h - f_l) + \frac{K_f}{A} \log \left(\frac{f_h}{f_l} \right)} \quad (4.9)$$

The bandwidth Δf and the time processing T of the device are related by the relation:

$$\Delta f = f_h - f_l = \frac{2\pi}{T} \quad (4.10)$$

The power consumption P is proportional to the intensity I of the current going through the transistor (equation (4.5)). Combining the equation (4.5) with (4.9) and (4.10), the standard deviation of the noise voltage distribution maybe rewritten:

$$\Delta v_n^2 = \left(\frac{K'_w(\gamma)}{P\gamma} + \frac{K'_f}{A} \right) \frac{1}{T} \quad \text{with} \quad \gamma = \begin{cases} 1.0 & \text{for } P < P_{th} \\ 0.5 & \text{for } P > P_{th} \end{cases} \quad (4.11)$$

where $K'_w(\gamma)$ and K'_f are fixed parameters derived from the expressions of K_w , K_f , and from the relations (4.5), (4.10). We note p_V the noise voltage distribution:

$$p_V(v) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{v^2}{2v_n^2}\right) \quad (4.12)$$

Inserting this distribution in equation (4.3) leads us to the failure probability of a CMOS transistor:

$$p(A, P, T) = \int_{v_{th}}^{+\infty} dv p_V(v) = \frac{1}{2} \operatorname{erfc}\left(1 / \sqrt{\left(\frac{K'_w(\gamma)}{P^\gamma} + \frac{K'_f}{A}\right) \frac{1}{T}}\right)$$

with $\gamma = 1.0$ for $P < P_{th}$
and $\gamma = 0.5$ for $P > P_{th}$ (4.13)

This relation captures the trade-off between the reliability of this device and the allowance of a certain amount of physical resources such as area, power and time. From now on, we call such relations reliability-resource laws (or sometimes simply reliability laws).

Although we have expanded it in a specific technological context, the idea that the reliability of a computational system is conditioned by the consumption of physical resources is extremely general. Indeed, the behavior of natural computers is essentially statistical and related to energy exchange or entropy creation. The operating mode of neurons is perhaps the most relevant illustration of this concept. Neurons receive through their synapses messengers of a chemical nature, whose interpretation requires in turn the performance of chemical reactions. The efficiency of their information processing relies simultaneously on sufficient energy supply (the reactions needed are sometimes endothermic or require some energy for their rate to be acceptable) and on the availability of messengers in sufficient quantity around the synapses. In those natural computing systems, there seems to be also a deep connection between reliability, processing time and power.

4.2 Problem Statement

In this section, we lay out the problem associated with the fault-tolerant design of resource-efficient circuits. We begin by raising the essential question and hypothesis which we will guide us in the subsequent developments of this thesis. We then treat two simple examples enlightening the problematic of fine-grained fault-tolerance. Afterwards, we define a framework by introducing assumptions and hypothesis regarding the error model, the reliability-resource law of the device and the circuit partitioning. We then present two optimization problems corresponding to different design situations, in which either the resource availability, or a requirement in the device reliability is fixed. Both problems will provide us with a different perspective from which to appreciate the benefits of a fine-grained fault-tolerant design.

4.2.1 Fundamental question and hypothesis

Redundancy and efficiency are usually considered to be at odds. In the common language, redundancy is synonymous for waste. At best, in the context of fault-tolerant computing, modular redundancy appeared as the simplest and least efficient way to achieve some reliability improvement. Let us adopt the opposite attitude and ask what could seem an absurd question:

Question: *Can redundancy help save resources ?*

This point deserves indeed to be seriously addressed. If reliability is fungible, it comes only at a certain price in terms of resource consumption. And this price becomes more and more significant as one moves towards high levels of reliability. This consideration enlightens a new role for redundancy: instead of struggling to reach a very low failure probability with a single component, it might be more efficient to duplicate the computation into several components and adopt a more permissive attitude towards the behavior of each element.

This alternative between multiple faulty identical processors and a single perfectly reliable one is similar to the choice between analog and digital computation. In an

analog computation, a continuous signal is carried by a single wire and a very low noise level is required to preserve the information encrypted in the signal. In a digital computation, the information is distributed over many wires, each of those carrying a signal taking its values in a small set (containing typically 10 elements if the computation is decimal) instead of in a quasi-continuum. Because of this discretization, the signal within each wire becomes robust to a significantly higher level of noise. In a context where noise can only be limited through the mobilization of resources, Sarpeshkar showed that digital computation could be more efficient than analog in many instances [Sar98]. This fact is also confirmed by the actual prominence of digital systems. Advocating for redundant systems based on the distribution of computational task between several elements is thus no more nonsensical than opting for the digital over the analog!

Let us investigate how the use of modular redundancy could lead to resource-efficient design. The last section showed how the physical reality of a device could connect its reliability to energy, space and time consumption. Nonetheless, this connection is manyfold in the following sense: granted a certain amount of resources, one can either allocate it at once to a single component, or organize a multiplexed circuit in which the resources would be divided among several components. The relation between resource consumption and reliability is different in this latter gate. For sake of concreteness, we consider the case of a device whose reliability depends in a continuous manner of its layout area only. Assuming that we have at our disposal a certain area A_T , we can choose to either allocate it at once to the component, or to organize a multiplexed circuit of the same size. These two design alternatives are depicted on figure 4-1.

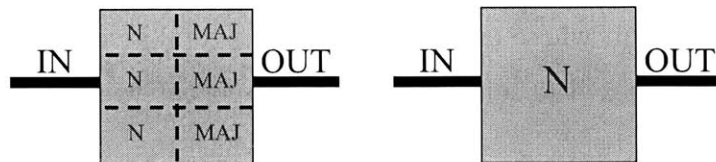


Figure 4-1: Two alternative resource allocations for a NAND gate of layout area A_T .

We could also imagine to use multiplexed constructions of higher recursion levels on the same layout. For each level, assuming the same area is granted to each component, we obtain a different trade-off between the total area consumed and the reliability of the circuit. We can now formulate the main hypothesis that will drive the analysis in the remainder of this thesis:

Hypothesis: *There exists a fine-grained multiplexed triple modular redundant design leading to a better trade-off than that associated with the direct allocation of resources.*

Given a reliability-resource law, it is not *a priori* obvious to see whether this hypothesis is valid or not. Indeed, when dividing the layout of a chip in order to organize a multiplexed circuit, two opposite effects compete:

- Each component is less reliable because the same pool of resources is divided by a greater number of devices.
- The distributed majority voting tempers the gravity of each failure occurring within the circuit.

The structure of the reliability-resource law determines whichever of these effects is dominant. Our main task will be to find out the laws and reliability range for which the hypothesis is verified.

4.2.2 Two first examples

We now explore on two examples how the relation between reliability and resources determines the efficiency of a fault-tolerant design. We aim at optimizing the reliability of a circuit whose layout is spread on a total area A_T . If we choose to divide it into a multiplexed circuit, each of the component will be granted the area $\frac{A_T}{6}$. Let us investigate two opposite situations, in which the reliability-resource law is either very flat, or very steep. The corresponding curves are presented in figure 4-2.

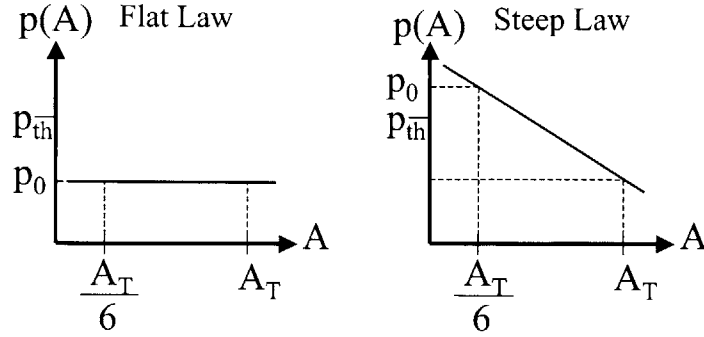


Figure 4-2: Fault-tolerant design in steep and flat reliability laws.

Flat Reliability-Resource law: In this case, the drop in resource allocation affecting the components of a multiplexed circuit does not change significantly their reliability. If the failure probability of a device of layout area A_T is below the threshold of equation (3.32), then:

$$p\left(\frac{A_T}{6}\right) = p_0 < p_{th} = \frac{1}{c_3} \quad \text{with } c_3 = 24 \quad (4.14)$$

This is the situation presented on figure 4-2. Noting p_{fail} the failure probability of the multiplexed component:

$$p_{fail} \leq \frac{1}{c_3} \left(c_3 p\left(\frac{A_T}{6}\right) \right)^2 \simeq \frac{1}{c_3} (c_3 p(A_T))^2 < p(A_T) \quad (4.15)$$

which shows that multiplexing improves the reliability of this device. A fault-tolerant design is advantageous for this reliability-resource law, so the hypothesis would be valid in this case.

Steep Reliability-Resource law: On the contrary, if the reliability-resource law is steep, the division of the total resource A_T among each component of the multiplexed circuit seriously deteriorates their reliability. On figure 4-2, we see that the failure probability of each component in the multiplexed circuit is such that:

$$p\left(\frac{A_T}{6}\right) = p_0 > p_{th} = \frac{1}{c_3} \quad \text{with } c_3 = 24 \quad (4.16)$$

Therefore we obtain the bound:

$$\frac{1}{c_3} \left(c_3 p \left(\frac{A_T}{6} \right) \right)^2 \geq p \left(\frac{A_T}{6} \right) \geq p(A_T) \quad (4.17)$$

which does not guarantee that a multiplexed design brings any improvement in the component reliability.

4.2.3 Framework

Error model

We mention here again the important assumptions, already evoked in Chapter 2, concerning the error model in noisy circuits:

- *Any device in the circuit is subject to errors.*
- *The devices in the circuit are subject to soft errors only.*
- *Errors happen independently in the different components.*
- *Processors fail independently from their inputs.*

Reliability-Resource law

The reliability of the device is described by an equation such as:

$$p = f_{\gamma_1.. \gamma_n}(R_1, \dots, R_n) \quad (4.18)$$

where $R_1...R_n$ are the resources used by the gate and $\gamma_1.. \gamma_n$ parameters depending on the technology employed. The figure 4-3 shows a typical reliability-resource law.

We make the following assumptions regarding the dependence of the reliability of the components in the circuit towards their resource allocation:

- *Reliability is a fungible resource.* In other words, the function $f_{\gamma_1.. \gamma_n}$ giving the failure probability is continuous with respect to each of its resource variables.

- *All components in the circuit are ruled by the same reliability law.*

The latter assumption would be illegitimate if we were to apply it to d -majority gates taking a big number of inputs, because the complexity of these gates is greater than that of a NAND. That is why we focus our analysis on the possibilities offered by recursive multiplexed triple modular redundant schemes, which involve exclusively 3-majority and NAND gates.

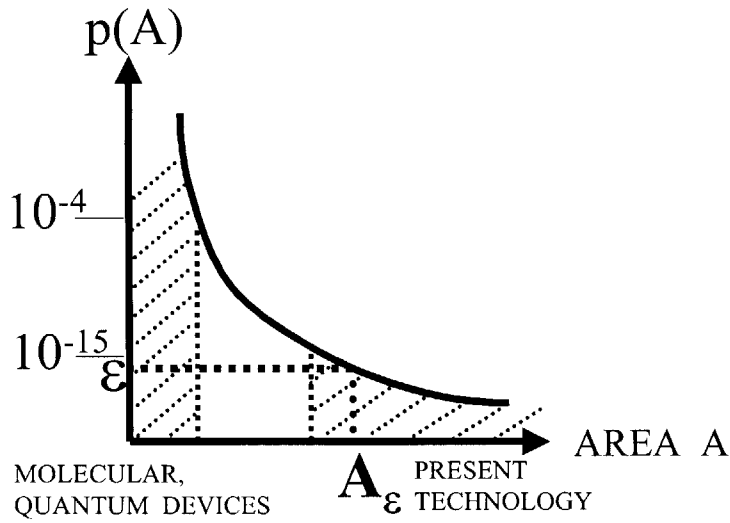


Figure 4-3: General reliability-resource law.

Furthermore, we will make two hypothesis regarding the reliability-resource law:

- *The component can be made arbitrarily reliable pending sufficient resource availability:*

$$\lim_{R_1 \rightarrow +\infty, R_2 \rightarrow +\infty, \dots, R_n \rightarrow +\infty} f_{\gamma_1 \dots \gamma_n}(R_1, \dots, R_n) = 0$$

- *Allocating more resources always lowers the failure probability of the device, in other words $f_{\gamma_1 \dots \gamma_n}$ is a strictly decreasing function in each of the variables R_1, \dots, R_n .*

From now on, we denote $p(R_1, \dots, R_n)$ the reliability-resource laws.

Circuit Partitioning

Regarding the partitioning of a layout into recursively multiplexed circuit, we suppose that:

- *The partitioning is done at no cost.* In particular, we neglect the cost of wires and communication between subcomponents.
- *Each component is granted the same amount of resources.* This assumption merely reflects our policy of resource allocation. Nonetheless, if some components were granted much more area than others, pitch-matching problems could arise in the layout, leading to a breakdown of the first assumption.

Let us translate these assumptions in terms of failure probabilities. We have seen in the equations (3.35) and (3.36) the scalability of the resources area, power and time for multiplexed schemes. Assuming that we allocate the resources R_1^0, \dots, R_n^0 to each basic unit and denoting R_1^k, \dots, R_n^k the consumption associated with a MTMR implementation of level k (definition (3.3.3)), we may define the scaling factors:

$$r_i^k = \frac{R_i^k}{R_i^0} \quad (4.19)$$

We will often specialize to the simpler problem of single parameter reliability-resource laws. They will be denoted $p(A)$, where A is an *extensive* resource which we take as the layout area, but which may represent the power as well. Assuming a global allocation R_1, \dots, R_n , the failure probability of a multiplexed scheme of level k is bounded by:

$$p_{fail} \leq p_{ft} \left(k, \frac{R_1}{r_1^k}, \dots, \frac{R_n}{r_n^k} \right) = \frac{1}{c_3} \left(c_3 p \left(\frac{R_1}{r_1^k}, \dots, \frac{R_n}{r_n^k} \right) \right)^{2^k} \quad \text{with } c_3 = 24 \quad (4.20)$$

Again we have taken the value $c_3 = 24$ for the threshold to account for the embedding of the encoded gate into a noisy circuit of similar components.

4.2.4 Two different points of view

We define here two different optimization problems allowing one to state whether the trade-off associated with a fault-tolerant design can be more advantageous.

Optimization With Limited Resource Availability

Given a *fixed* pool of resources R_1, \dots, R_n , we now aim at determining the optimal recursion level $k_{opt}(R_1, \dots, R_n)$, minimizing the failure probability $p_{ft}(k, R_1, \dots, R_n)$ of a MTMR construction consuming the whole pool of resources. We will say that we have found a successful fault-tolerant design with the resources R_1, \dots, R_n when $k_{opt}(R_1, \dots, R_n)$ is non-zero.

It is easy to see that, for each set of resources R_1, \dots, R_n , the optimal level of redundancy is to be taken within a finite set. Indeed, the only possible valid values for k are upper-bounded k_m , lowest integer such that:

$$p_{ft}(k_m, R_1, \dots, R_n) < \frac{1}{c_3} \quad (4.21)$$

This optimization problem is thus always solvable by inspection of a finite set of values. Its interest may appear as limited, since the objective of chip designers would rather be to achieve a certain reliability with the least possible amount of resources. But this point of view clearly shows how a fault-tolerant design can boost the reliability of a chip. We shall adopt the acronym OWLRA as short-hand notation for optimization with limited resource availability.

Optimization With Reliability Requirement

This is now the reliability of the encoded gate which is fixed. In this optimization problem, we look for the resource allocation to the elementary components and for the recursion level of the multiplexed construction minimizing the resource consumption of the global circuit and compatible with the reliability requirement.

In mathematical terms, we wish to find a set of parameters k, R_1^0, \dots, R_n^0 minimizing

a weight function $w(R_1^0 r_1^k, \dots, R_n^0 r_n^k)$ under the constraint:

$$p_{ft}(k, R_1^0, \dots, R_n^0) \leq \epsilon \quad (4.22)$$

where the function p_{ft} is defined in the equation (4.20). The weight function reflects the cost attributed to each resource and may be fixed by the designer, but we will naturally assume that:

$$\forall i \in \{1, \dots, n\} \quad \frac{\partial w(R_1, \dots, R_n)}{\partial R_i} > 0 \quad (4.23)$$

This problem is indeed much harder to solve in the case of reliability laws depending on several different resources. Its solution is not expected to be a unique set of parameters, but rather a collection of continuous surfaces of \mathfrak{R}^n (of dimension $n - 2$) corresponding to resource parameters satisfying the constraint $p_{ft}(k, R_1, \dots, R_n) = \epsilon$ and minimizing $w(R_1, \dots, R_n)$. The ensemble of possible values should be chosen within a reciprocal image of p_{ft} defined by the relations:

$$\frac{1}{c_3} \leq p_{ft}(0, R_1^0, \dots, R_n^0) \leq \epsilon \quad (4.24)$$

We simply state here that resources allocated to each unit should be chosen such that its failure probability is below the threshold - otherwise a modular redundant encoding worsens the reliability - and above the reliability requirement - otherwise resources are wasted -.

Definition 4.2.1 : *We shall say that there is a successful fault-tolerant design in the optimization with reliability requirement when there are two sets of parameters:*

- *A set of "fault-tolerant" parameters k, R_1^0, \dots, R_n^0 such that $k > 0$ and $p_{ft}(k, R_1^0, \dots, R_n^0) \leq \epsilon$.*
- *A set of direct parameters R'_1, \dots, R'_n such that $p_{ft}(0, R'_1, \dots, R'_n) = \epsilon$*

satisfying:

$$\max_{i=1 \dots n} \frac{R_i^0 r_i^k}{R'_i} < 1 \quad (4.25)$$

From now on we adopt the acronym OWRR as short-hand notation for optimization with reliability requirement.

4.3 Initial Solutions

We derive here some initial answers to the problem laid out in the previous section. We begin by showing a simple yet essential result: for reliability-resource laws with a residual error probability, a fault-tolerant design is always advisable for high reliability requirements. Afterwards, we derive a mathematical condition enabling to state the existence of an efficient fault-tolerant design for a reliability-resource law and a certain reliability requirement. Finally, we show that the two optimization problems formulated in section 4.2 lead to equivalent answers in terms of confirming or invalidating the hypothesis defined in the same section.

4.3.1 Efficiency of a fault-tolerant design for reliability laws with residual error

We release here one of the hypothesis of section 4.2.3 regarding the reliability-resource laws: we consider a device ruled by a law for which there is a residual error probability ϵ_r independent of the resource allocation. This law is represented in figure 4-4. We also assume that the residual error is below threshold ($\epsilon_r < \frac{1}{24}$), which is largely satisfied by actual components. The following claim shows the advantage of a fault-tolerant design.

Claim 4.3.1 : *For laws with a residual error ϵ_r below threshold, a multiplexed design is always advisable for reliability requirements $\epsilon < \epsilon_r$.*

Proof: This reliability cannot be reached by direct allocation. However, if we consider a building block (elementary device) of area A_0 such that:

$$\epsilon_r < p_0 = p(A_0) < p_{th} = \frac{1}{c_3} \quad (4.26)$$

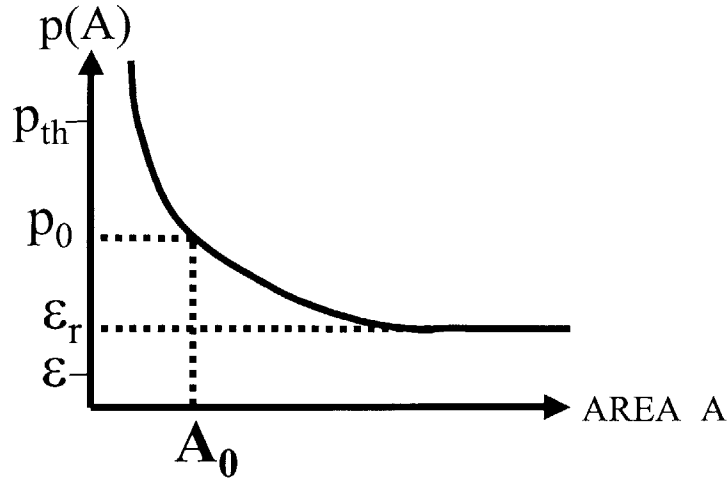


Figure 4-4: Reliability-resource law with residual error.

then a multiplexed construction of level k with those elementary components has a failure probability bounded by:

$$p_{fail} \leq \frac{1}{c_3} (c_3 p_0)^{2^k} \quad (4.27)$$

When k becomes large, thanks to (4.26) the failure probability p_{fail} tends toward 0. Therefore there is a fault-tolerant construction of finite area which allows one to achieve the reliability requirement ϵ .

The latter claim legitimates the hypothesis of section 4.2.3 that a device be perfectly reliable pending sufficient resource allocation: if this is not the case, one can obtain through appropriate fault-tolerant constructions a new reliability law in which the residual error disappears (provided that the residual failure probability is below threshold).

4.3.2 Efficiency condition of a multiplexed triple modular redundant design

The purpose of this paragraph is to derive a mathematical condition addressing the efficiency of a MTMR design of a NAND gate (definition (3.3.3)).

We focus in the first place on a reliability law which involves only a single resource identified as area. We consider the problem of optimization with a reliability requirement $p_{fail} \leq \epsilon$. The direct design requires a layout area of $A_\epsilon = p^{(-1)}(\epsilon)$. We wish to achieve the same level of reliability with elementary components of smaller size A_0 and of failure probability $p(A_0)$, embedded in a multiplexed voting system. From the previous section, we know that A_0 is constrained to be in the interval $I_\epsilon =]p^{-1}(1/c_3), p^{-1}(\epsilon)[=]A_{1/c_3}, A_\epsilon[$. For any recursion level k such that:

$$\frac{1}{c_3} (c_3 p(A_0))^{2^k} \leq \epsilon \quad (4.28)$$

the corresponding multiplexed circuit reaches the reliability objective. We shall note $k(\epsilon, A_0)$ the smallest of these integers. It is given by the formula:

$$k(\epsilon, A_0) = \left\lceil \log_2 \left(\frac{\log(c_3 \epsilon)}{\log(c_3 p(A_0))} \right) \right\rceil \quad (4.29)$$

Under the assumption of perfect partitioning, the scaling of multiplexed constructions established in (3.35) gives the layout area required for a scheme of level k :

$$A(k) = A_0 (k + 1) 3^k \quad (4.30)$$

The trade-off associated with the recursive construction is efficient if:

$$A_{ft} = A(k_0) \leq A_\epsilon \quad (4.31)$$

Because the failure probability is a strictly *decreasing* function of the area, we can reformulate this condition as follows:

Claim 4.3.2 (Condition of fault-tolerant efficiency):

There exists an efficient fault-tolerant construction if:

$$\begin{aligned} \exists A_0 \in]p^{(-1)}(c), p^{(-1)}(\epsilon)[\quad \text{s.t.} \quad p[A_0 \quad (k(\epsilon, A_0) + 1) \quad 3^{k(\epsilon, A_0)}] \geq \epsilon \\ \text{where} \quad k(\epsilon, A_0) = \left\lceil \log_2 \left(\frac{\log(c_s \epsilon)}{\log(c_s p(A_0))} \right) \right\rceil \end{aligned} \quad (4.32)$$

If this condition is verified, then the hypothesis of section 4.2 is valid.

We can without major difficulty generalize this approach to reliability laws involving several resources by using a weight function w . Keeping the notation of the previous section, the condition (4.32) becomes:

$$\begin{aligned} \exists (R_1^0, \dots, R_n^0) \quad \exists (R_1, \dots, R_n) \quad \text{s.t.} \quad \begin{cases} p(R_1, \dots, R_n) = \epsilon \quad \text{and} \\ w(R_1^0 r_1^{3,k}, \dots, R_n^0 r_n^{3,k}) < w(R_1, \dots, R_n) \end{cases} \\ \text{where} \quad k = \left\lceil \log_2 \left(\frac{\log(c_s \epsilon)}{\log(c_s p(R_1^0, \dots, R_n^0))} \right) \right\rceil \end{aligned} \quad (4.33)$$

These equations reflect the fine balance between the overhead incurring through the triplication and the improvement brought by distributed majority voting. Contrary to what intuition suggests, these relations show that introducing redundancy in the design of a processor can sometimes enable to save resources. The success of a fault-tolerant design depends both on the structure of the reliability-resource law $p(A)$ and on the reliability requirement ϵ .

4.3.3 Equivalence of the optimization with limited resource availability and with reliability requirement

In the rest of this thesis, we will devote a greater attention to the problem of optimization with reliability requirement. Nonetheless, when comparing the suitability for a fault-tolerant implementation in two different technologies, it will be useful to switch from an optimization problem to the other.

Indeed both point of views are closely related, and both optimizations may be

equivalently used to state the existence of an efficient fault-tolerant design. We verify this quickly in the following claim:

Claim 4.3.3 : *In terms of stating the successfulness of the recursive fault-tolerant design over the classical design, the problems of optimization with limited resource availability (OWLRA) and the optimization with reliability requirement (OWRR) are equivalent.*

Proof : Let us assume that there is a successful fault-tolerant set of parameters in the OWLRA problem. That is to say:

$$\exists k \in N \quad s.t. \quad p_{ft} \left(k, \frac{R_1}{r_1^k}, \dots, \frac{R_n}{r_n^k} \right) < p_{ft}(0, R_1, \dots, R_n) \quad (4.34)$$

We set $\epsilon = p_{ft}(0, R_1, \dots, R_n)$. Because the function p_{ft} is a continuous function of the resources, there exists $\rho < 1$ such that:

$$p_{ft} \left(k, \rho \frac{R_1}{r_1^k}, \dots, \rho \frac{R_n}{r_n^k} \right) < p_{ft}(0, R_1, \dots, R_n) \quad (4.35)$$

Then we have found by definition a successful fault-tolerant design in the OWRR problem.

Conversely, let us assume that we have a successful fault-tolerant design in the OWRR problem, that is two sets of parameters k, R_1^0, \dots, R_n^0 and R'_1, \dots, R'_n verifying the properties of definition (4.2.1). We then have:

$$\forall i \in \{1..n\} \quad \frac{R'_i}{r_i^k} > R_i^0 \quad (4.36)$$

which ensures that:

$$p_{ft} \left(k, \frac{R'_i}{r_i^k}, \dots, \frac{R'_i}{r_i^k} \right) < p_{ft}(k, R_1^0, \dots, R_n^0) \leq \epsilon = p_{ft}(0, R'_1, \dots, R'_n) \quad (4.37)$$

This proves that the optimal recursion level $k_{opt}(R'_1, \dots, R'_n)$ associated with the OWLRA optimization is not zero, and that we have a successful fault-tolerant design for this problem also.

4.4 Reliability is a fungible resource

As we saw on a real technological example, the reliability of a physical computing device may be exchanged in a continuous fashion for other physical resources, attribute which we summed up under the term of fungibility. In order to take a step further in the analysis of this property, it is revealing to think of a phase space in which each device would be mapped to a point $(R_1, \dots, R_n, \epsilon)$ of \mathfrak{R}^{n+1} describing the resources mobilized and its reliability, and to investigate what points of the phase space are accessible.

We consider a family of processors corresponding to devices with the same technology, parameterized by a resource vector of \mathfrak{R}^n , two components with distinct resource allocation being different processors. After some investigation on their mechanism, we may establish a reliability-resource law for these devices. When (R_1, \dots, R_n) describes \mathfrak{R}^n , the phase point describes a manifold of \mathfrak{R}^{n+1} characterized by the function $x_{n+1} = p(x_1, \dots, x_n)$, which we shall call “physical” manifold. We may also define a minimal surface by picking up, for each resource allocation (R_1, \dots, R_n) , the reliability associated with the optimal recursion level. We already referred to this problem as optimization with limited resource availability. This surface is continuous, but it is generally not be a manifold because a curve traced on this surface might encounter singularities associated with changes of the optimal recursion level k . The discussion of the previous section sheds an interesting light on the accessible phase trajectories, but it also raises an apparent contradiction:

How can several reliability-resource laws be appropriate to describe a family of systems sharing the same technology, when such laws are established on the physical origin of the noise ?

Indeed, the microscopic physical processes involved in a processor and its recursively multiplexed implementation are the same. But the paradox disappears if one remarks that these different reliability laws are associated with devices managing differently the same amount of physical disorder. In other words, while the physical behavior is dictated by the amount of resources available, its interpretation is manifold. Two

processors with different multiplexing levels acquire the information through a different sampling realized on a similar set of physical events. In this line of thought, there is no special reason to distinguish the “physical” reliability-resource law from other laws derived by multiplexing.

Because of the many possible trades-off, from each point of the accessible phase space, one might move on a countable class of manifolds associated with multiplexed implementations of different recursion levels k (see figure 4-5).

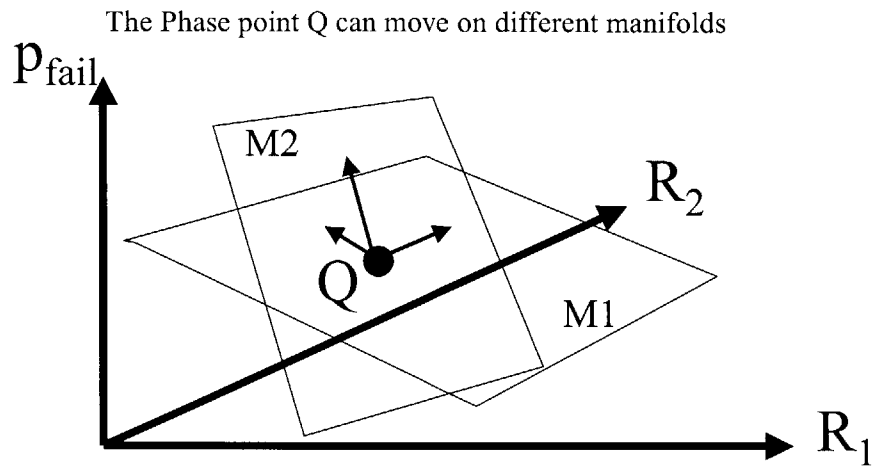


Figure 4-5: Possible trajectories of the phase point.

Let us investigate for a moment what happens with a single parameter reliability law. The failure probability of the device is again a function of its layout area. If we assume that the wires are reliable, it is possible to build devices of similar reliability with increased resource allocation: one just need to increase the empty space between the components of the layout. On the graph presented on figure 4-6, the phase point can move horizontally towards component of similar reliability but of increased resource consumption. The figure also shows the accessible phase space.

Our ability to leave some resources unused legitimates the hypothesis that the failure probability of a component be a decreasing function of the resources: if the failure probability of a component were not a decreasing function of the area, for a total layout of size A it would still be possible to grant the area $A' < A$ to the device,

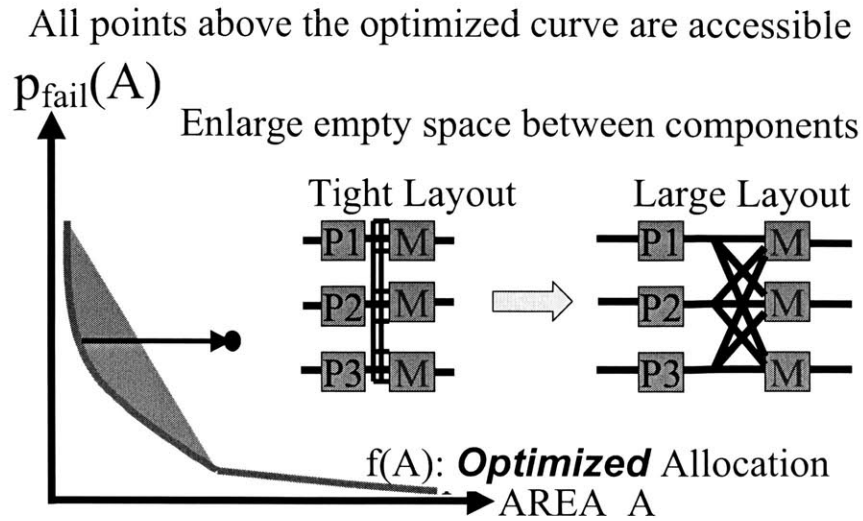


Figure 4-6: Accessible phase space for a single resource reliability law.

optimizing its reliability by leaving some empty space on the layout.

A reliability law with several resources exhibits the same property: from each point of the minimal surface, it is possible to increase resource allocation and move towards less efficient devices. Let us invoke again the generic example of a CMOS transistor whose failure probability is conditioned by area, power and time. We wish to move along the vector $(\lambda_1 A, \lambda_2 P, \lambda_3 T, \lambda_4)$ with $\forall i \lambda_i > 0$. We may increase the resources consumption along this vector by putting empty space on the layout, additional resistors in the wires or waiting after the computation is performed. We may also decrease the reliability of the transistor by decreasing the resource allocated to the component, while increasing the general resource consumption. The set of accessible points is $\{(\mathfrak{R}^+)^n \times]0, \frac{1}{2}[\}$ minus the volume S enclosed below the minimal surface. The accessible space is represented on the figure 4-7. For sake of clarity, we have not plotted the resource R_3 which would require a figure in four dimensions.

4.5 Conclusion

We have introduced in this chapter the essential notion that reliability is a fungible resource, and illustrated this concept on an the CMOS transistor. Starting from

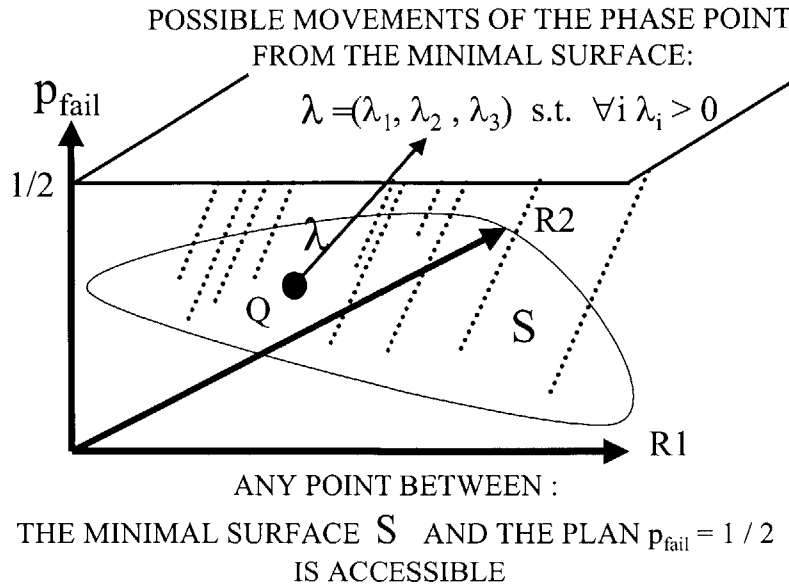


Figure 4-7: Accessible phase space for a reliability law with several resources.

the hypothesis that fault-tolerant schemes may outperform the traditional design in terms of resource allocation, we have introduced two different points of view from which to perform the analysis: the optimization with limited resource availability and the optimization with reliability requirement. We have shown that a fault-tolerant design is advantageous for reliability-resource laws with a residual error probability. Furthermore, we have translated the efficiency of a multiplexed design into a mathematical condition. We have also proven that both optimization problems lead to equivalent answers. Last, by introducing the notion of phase space, we have given a more comprehensive view on the fungibility of reliability as a resource.

So far we have approached qualitatively the possibilities offered by recursive multiplexing at the fine-grain level. It is now time to take a step further in the analysis and bring quantitative evaluations of the gain in resources and reliability provided by this design method. This is the purpose of the next chapter.

Chapter 5

Fine-grained Fault-Tolerance: Quantitative Analysis

In this chapter we treat quantitatively the optimization through fine-grained fault tolerance by introducing new measures of efficiency for this design method and by evaluating them on specific instances.

Section 5.1 describes a systematic procedure allowing one to estimate, through the reliability-resource law, the efficiency of a multiplexed triple modular redundant design. Section 5.2 establishes criteria of comparison enabling to infer, from a comparison of their reliability-resource laws, the relative performance of a fault-tolerant design in different devices. In section 5.3 these techniques are applied to identify, within the family of power and exponential functions, a class of laws and reliability requirements for which such a design is efficient. Section 5.4 brings an answer to the relevance of a multiplexed design in a CMOS transistor. Section 5.5 concludes by mapping the classes of reliability laws which satisfy the hypothesis of section 4.2.

5.1 Systematic approach of fault-tolerant optimization

This section covers more specifically the optimization of multiplexed constructions in devices ruled by a single parameter reliability-resource law $p(A)$, and verifying the assumptions laid out in section 4.2.3.

5.1.1 Identification of the optimal areas

Let us begin by recalling some important results of Chapter 4. $k(\epsilon, A_0)$ denotes the smallest integer k such that:

$$\frac{1}{c_3} (c_3 p(A_0))^{2^k} \leq \epsilon \quad (5.1)$$

According to the bound (3.41) for the failure probability of recursively multiplexed circuits, in order to achieve a reliability $p_{fail} \leq \epsilon$ in a circuit whose elementary devices have a layout area A_0 , it is sufficient to adopt a recursion level $k(\epsilon, A_0)$. Inverting the equation (5.1), we obtained the expression:

$$k(\epsilon, A_0) = \left\lceil \log_2 \left(\frac{\log(c_3 \epsilon)}{\log(c_3 p(A_0))} \right) \right\rceil \quad (5.2)$$

Although the reliability ϵ could be in principle achieved with a lower recursion level, $k(\epsilon, A_0)$ is the optimum choice that can be obtained from the bound (5.1). Nonetheless, this bound has two advantages: it exhibits clearly the double exponential decrease of the failure probability with the recursion level, and it is also a simple monomial of the elementary gate failure probability $p(A_0)$. This scaling of the failure probability with the recursion level captures the efficiency of the fault-tolerant constructions. Besides, it will reveal useful to have an invertible expression relating the reliability of the elementary processors to that of the encoded circuit. That is why we choose to base on the bound (5.1) our evaluation of the failure probability in multiplexed circuits.

Consistently, given an elementary gate area A_0 and a reliability requirement ϵ , we shall always adopt the recursion level $k(\epsilon, A_0)$ in a recursive MTMR scheme. The couple of reals (ϵ, A_0) then fully determines the MTMR construction, and we call it a building set. As mentioned earlier, we consider for the multiplexed construction only values of A_0 such that:

$$A_0 \in I_\epsilon =]p^{(-1)}(\frac{1}{c_3}), p^{(-1)}(\epsilon)[=]A_{1/c_3}, A_\epsilon[\quad (5.3)$$

On the one hand, if A_0 falls below this range, the elementary gates have a failure probability over threshold and majority voting might actually deteriorate the reliability of the circuit. On the other hand, if A_0 is situated above this interval no recursion is obviously needed to obtain a failure probability $p_{fail} \leq \epsilon$ and some resources are indeed wasted. We denote S_A the reunion of those “acceptable” building sets:

$$S_A = \{(\epsilon, A_0) \in (\mathfrak{R}^{++})^2 \mid A_0 \in I_\epsilon =]A_{1/c_3}, A_\epsilon[\} \quad (5.4)$$

Given an acceptable building set (ϵ, A_0) the MTMR design is efficient if:

$$p(A_0, (k(\epsilon, A_0) + 1) 3^{k(\epsilon, A_0)}) \geq \epsilon \quad (5.5)$$

The acceptable building sets (ϵ, A_0) satisfying the condition (5.5) will be qualified of MTMR efficient. Whether there exists such building sets or not depends in general on the structure of the reliability-resource law. It is natural to introduce the following partial order on the set S_A :

$$(\epsilon_1, A_1) \preceq (\epsilon_2, A_2) \Leftrightarrow \begin{cases} \epsilon_1 \leq \epsilon_2 & \text{and} \\ A_1 (k(\epsilon, A_1) + 1) 3^{k(\epsilon, A_1)} \leq A_2 (k(\epsilon, A_2) + 1) 3^{k(\epsilon, A_2)} \end{cases} \quad (5.6)$$

We also adopt the convention that $(\epsilon_1, A_1) \prec (\epsilon_2, A_2)$ when one of the two inequalities in (5.6) is strict. As we shall see, for each $\epsilon < \frac{1}{c_3}$ there exists a corresponding minimal element $(\epsilon, A_{min}(\epsilon)) \in S_A$. Our task reduces to finding those elements. It is indeed

possible to focus our search for $A_{min}(\epsilon)$ on a finite set of values:

Claim 5.1.1 (Optimal Areas):

An acceptable building set $(\epsilon, A_0) \in S_A$ may be a minimal element of S_A only if:

$$A_0 \in S_\epsilon = \{A(k, \epsilon) \mid k \in \mathbb{N} \text{ s.t. } \frac{A_\epsilon}{(k+1)3^k} \geq A_{1/c_3}\}$$

$$\text{where } A(k, \epsilon) = p^{(-1)}\left(c_3^{-1+1/2^k} \epsilon^{1/2^k}\right) \Leftrightarrow \frac{1}{c_3} [c_3 p(A(k, \epsilon))]^{2^k} = \epsilon \quad (5.7)$$

Proof: By mean of contradiction, we assume that $(\epsilon, A_0) \in S_A$ is minimal and such that $A_0 \notin S_\epsilon$:

either $\forall k \in \mathbb{N} \quad A_0 \neq A(k, \epsilon)$,

or $\exists k_0 \text{ s.t. } A_0 = A(k_0, \epsilon) \text{ and } A_\epsilon < A_{1/c_3}((k_0 + 1)3^{k_0})$.

To obtain a contradiction, we only need to exhibit in each case an element $(\epsilon', A') \in S_A$ such that $(\epsilon', A') \prec (\epsilon, A_0)$.

Let us assume that we are in the first case: $\forall k \in \mathbb{N} \quad A_0 \neq A(k, \epsilon)$. Then, the choice $\epsilon' = \epsilon$ and $A' = A(k(\epsilon, A_0), \epsilon)$ works. Indeed, by definition of $k(\epsilon, A_0)$ and $A(k, \epsilon)$:

$$\frac{1}{c_3} [c_3 p(A_0)]^{2^{k(\epsilon, A_0)}} \leq \epsilon \quad (5.8)$$

$$\frac{1}{c_3} [c_3 p(A')]^{2^{k(\epsilon, A_0)}} = \epsilon \quad (5.9)$$

The comparison between the equations (5.8) and (5.9) tells us that $p(A_0) \leq p(A')$. Since p is a decreasing function, this implies $A' \leq A_0$. Because of the hypothesis, $A' = A(k(\epsilon, A_0), \epsilon) = A_0$ is excluded. Consequently $A' < A_0$.

Furthermore, the equation (5.9) tells us that $k(\epsilon, A_0)$ is a sufficient recursion level for a multiplexed scheme of building set (ϵ, A') . Therefore $k(\epsilon, A') \leq k(\epsilon, A_0)$.

Combining $A' < A_0$ and $k(\epsilon, A') \leq k(\epsilon, A_0)$ yields $(\epsilon', A') \prec (\epsilon, A_0)$. Contradiction.

If we are in the second case $A_0 = A(k_0, \epsilon)$ for $A_\epsilon < A_{1/c_3}(k_0 + 1)3^{k_0}$.

$A_0 = A(k_0, \epsilon)$ implies that $k(\epsilon, A_0) = k_0$. Taking this time $(\epsilon', A') = (\epsilon, A_\epsilon)$, for which

$k(\epsilon, A') = 0$, we obtain:

$$A' (k(\epsilon, A') + 1)3^{k(\epsilon, A')} = A_\epsilon < A_{1/c_3} (k_0 + 1)3^{k_0} < A_0 (k_0 + 1)3^{k_0} \quad (5.10)$$

Again $(\epsilon', A') \prec (\epsilon, A_0)$. Contradiction.

The advertised existence of a minimal element $(\epsilon, A_{min}(\epsilon))$ for each $\epsilon < 1/c_3$ comes as a corollary of this claim: the optimal choice for the building set falls within a finite set of optimal areas S_ϵ . In order to have simple notations, we keep the convention that $A_\epsilon = A(0, \epsilon) = p^{(-1)}(\epsilon)$. We shall say that the reliability-resource law $p(A)$ can be efficiently fault-tolerantly implemented (or MTMR implemented) on the range of reliability ϵ for which $A_{min}(\epsilon) < \epsilon$. This is equivalent to saying that the hypothesis of section 4.2 is valid. Also, with a slight language abuse, we will sometimes speak of the gain provided by a fault-tolerant design in a reliability law, even if strictly speaking this design applies to a device.

5.1.2 Simplification of the Triple Modular Redundancy Efficiency Condition

The former discussion simplifies the efficiency condition (4.32). When A_0 is an optimal area, the integer parts in the expression of $k(\epsilon, A_0)$ are in fact no longer necessary. Since one can without loss of generality restrict the choice of building sets to these areas, the efficiency condition becomes:

$$\begin{aligned} \exists A_0 \in]p^{(-1)}(c), p^{(-1)}(\epsilon)[\quad \text{s.t.} \quad p[A_0 (k(\epsilon, A_0) + 1) 3^{k(\epsilon, A_0)}] \geq \epsilon \\ \text{where} \quad k(\epsilon, A_0) = \log_2 \left(\frac{\log(c_s \epsilon)}{\log(c_s p(A_0))} \right) \end{aligned} \quad (5.11)$$

This condition is now analytical, which is an advantage for further mathematical treatments.

5.1.3 Quantitative measures of efficiency:

Definition and evaluation

The identification of a discrete set of optimal areas for the building blocks of the circuit allows one to obtain easily quantitative measures of efficiency of the multiplexed triple modular redundant design:

Definition 5.1.1 : *We shall call resource gain:*

$$r(\epsilon) = \max_{k=0\dots+\infty} \frac{A_\epsilon}{A(k, \epsilon) (k+1)3^k}$$

Definition 5.1.2 : *We shall call reliability gain:*

$$s(A) = \max_{k=0\dots+\infty} \frac{c_3 p(A)}{\left(c_3 p \left(\frac{A}{(k+1)3^k} \right) \right)^{2^k}}$$

Definition 5.1.3 : *We denote $p_{MTMR}(A)$ the function giving the failure probability of the optimal MTMR circuit with a total resource allocation A .*

For single parameter reliability-resource laws, when this notation is not ambiguous, we shall also refer to $p_{MTMR}(\epsilon)$ as the failure probability of the optimal MTMR circuit whose resource allocation would lead to ϵ with a direct design.

In fact, for each reliability requirement ϵ , both maximums are taken on finite sets. For the reliability gain $r(A)$, the relevant recursion levels are bounded by $k_{max}(A)$, smallest integer such that:

$$(k+1)3^k > \frac{A}{A_{1/c_3}} \quad (5.12)$$

As for the resource gain $s(\epsilon)$, it is sufficient to investigate the recursion levels up to the smallest integer $k_{max}(A_\epsilon)$ such that:

$$(k+1)3^k > \frac{A_\epsilon}{A_{1/c_3}} \quad (5.13)$$

Since components of failure probability $p_{fail} \leq 10^{-16}$ can be considered as perfectly reliable for all practical applications, even for slowly decreasing reliability laws such as the family of power laws, only a “reasonable” number of recursion levels needs to be explored. As an example, for the power-like decreasing laws $p(A) = \frac{1}{A^\gamma}$, the maximum relevant recursion level is:

$$k_{max}(\epsilon), k_{max}(A) \sim \frac{\log(A_\epsilon)}{\log(3)\gamma} \sim \frac{|\log(\epsilon)|}{\log(3)\gamma} \leq 16 \frac{\log(10)}{\log(3)} \frac{1}{\gamma} \simeq \frac{32}{\gamma} \quad (5.14)$$

In fact, in order for k_{max} to become macroscopic, the component should be ruled by a reliability law or by a power-like decreasing with a very small exponent. This will not be the case for the devices we investigate in the present thesis, and it will be largely sufficient in practice to explore the first 100 recursion levels. Although the evaluation of the resource gain is slightly more complicated, since it requires the inversion of a reliability-resource law, the estimation of both measures of efficiency remains essentially a straightforward numerical task.

5.2 Comparison of reliability-resource laws

In this section, we construct a toolbox allowing one to compare the relative efficiency of a fine-grained multiplexed design in distinct reliability-resource laws. We begin by proving that two reliability laws related to each other by a dilatation of their resource variables yield similar gains with a multiplexed design. This remark will help us establish classes of reliability laws equivalent from the point of view of fault-tolerance. We then derive two criteria of comparison applicable in different contexts. The first property is relative to different single parameter reliability laws whose resource scale similarly. An important corollary is the notion of critical exponent, which will be very useful in the mapping of the efficiently fault-tolerant implementable reliability laws. The second property involves two reliability laws without any necessary similarity of their resources: both laws can actually involve a different number of resources, or resources which scale differently.

5.2.1 Invariance of the efficiency towards dilatation of the reliability-resource law

In physical terms, we state here that the resource gain provided by a multiplexed construction does not depend on the system of unit chosen for the parameter entering the reliability-resource law.

Claim 5.2.1 : *Let p and $p_\lambda = p(\lambda A)$ be two reliability-resource laws for $\lambda > 0$.*

The resource gains associated with p and p_λ are identical.

Proof: Let $0 < \epsilon < 1/c_3$ be a reliability requirement (below threshold), $A(k, \epsilon)$ and $A_\lambda(k, \epsilon)$ the optimal areas of level $k \in N$ associated with the respective laws p and p_λ . By definition of these two areas:

$$p(A(k, \epsilon)) = c_3^{-1+1/2^k} \epsilon^{1/2^k} \quad \text{and} \quad p_\lambda(A_\lambda(k, \epsilon)) = p(\lambda A_\lambda(k, \epsilon)) = c_3^{-1+1/2^k} \epsilon^{1/2^k} \quad (5.15)$$

Since p is an injection as a *strictly* decreasing function of the resources, the previous equation implies:

$$A_\lambda(k, \epsilon) = \frac{A(k, \epsilon)}{\lambda} \quad (5.16)$$

All the optimal areas are thus equally contracted. In particular, the resource gains are equal:

$$\frac{A_\lambda(k, \epsilon)}{A_\lambda(0, \epsilon)} = \frac{A(k, \epsilon)}{A(0, \epsilon)} \quad (5.17)$$

As an important corollary, we obtain that the reliability range on which p and p_λ can be efficiently fault-tolerantly implemented is the same. This statement will help us identify the relevant parameters in the classification of the resource laws with respect to their fault-tolerant implementability. This result is not only valid for single parameter resource laws, but can be extended immediately to reliability laws with several resources: again the optimal allocations for the fault-tolerant and for the direct implementation are dilated by the same factor.

5.2.2 Comparison of single parameter reliability-resource laws

We know from the previous chapter that incrementing the recursion level, in a multiplexed construction with fixed resources, simultaneously lowers the resource availability for each component and increases the number of fault-path tolerated by the circuit. The gain provided by multiplexing results from a balance between those two effects. Whichever is dominant depends on the structure of the reliability law, as we saw on the two examples of section 4.2.2: if the failure probability decreases in a very steep way with the resources then the reliability deterioration of the elementary gates is prominent, while if it decreases only slowly the gain obtained by majority voting prevails. This suggests a comparison of the performance of a fault-tolerant design in distinct reliability laws by inspection of their slopes. In particular, we may extend the eligibility of a law to a fault-tolerant design to a class of other laws decreasing more slowly with the resource allocation. We now transform these heuristic considerations into rigorous properties.

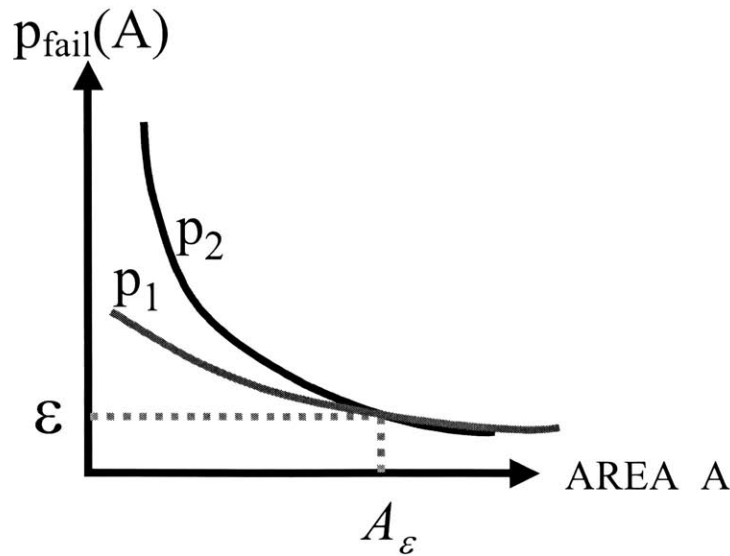


Figure 5-1: Comparison of two reliability-resource laws for a single reliability requirement ϵ .

Let us start with a very specific situation in which a direct inspection of the

reliability laws allows one to compare their fault-tolerant implementability. In the context presented on the figure 5-1, for the specific reliability requirement ϵ , the law p_2 is obviously more suitable than p_1 for a fault-tolerant design of the component. We verify this fact quickly in the following lemma:

Lemma 5.2.1 : *Let p_1, p_2 be two single parameter reliability-resource laws such that:*

$$p_1(A_\epsilon) = p_2(A_\epsilon) = \epsilon \quad \text{and} \quad \forall A \leq A_\epsilon \quad p_1(A) \leq p_2(A) \quad (5.18)$$

If the law p_2 admits an efficient fault-tolerant implementation for the reliability requirement ϵ , then the law p_1 also admits such an implementation. Besides, p_1 yields a greater resource gain than p_2 for this reliability requirement.

Proof: Considering a potential building block A_0 , we compare the fault-tolerant areas A_1^{ft} and A_2^{ft} required for both devices. These areas are given by the equation:

$$A_i^{ft} = A_0 r^{k_i(\epsilon, A_0)} \quad \text{where} \quad k_i(\epsilon, A_0) = \left\lceil \log_2 \left(\frac{\log(c_3 \epsilon)}{\log(c_3 p_i(A_0))} \right) \right\rceil \quad (5.19)$$

Since for any possible value A_0 , we have $p_1(A_0) < p_2(A_0)$ obviously $k_1(\epsilon, A_0) < k_2(\epsilon, A_0)$ and hence $A_1^{ft} < A_2^{ft}$, which yields the desired result.

Naturally this lemma allows us also to prove negative results on the efficiency of a MTMR design: if we know that it is inefficient for the law p_1 and for the reliability requirement ϵ , it will be *a fortiori* inefficient for the law p_2 at the same reliability requirement.

We may underline that the precedent lemma applies to a restrictive context in which a common resource allocation gives the same reliability with both laws p_1 and p_2 . Strictly speaking, their comparison through the lemma (5.2.1) is thus restrained to the reliability requirement ϵ . Fortunately this drawback can be circumvented by renormalizing one of the reliability laws. This technique enables a comparison of their fault-tolerant implementability on a finite range of reliability:

Claim 5.2.2 :

Let p_1 and p_2 be two single-parameter reliability-resource laws such that:

$$\forall \epsilon \in [\epsilon_1, \epsilon_2] \quad \forall A \in]A_{1/c_3}^1, A_\epsilon^1[\quad p_1(A) \leq p_2(\lambda_\epsilon A)$$

$$\text{where } \lambda_\epsilon = \frac{A_\epsilon^2}{A_\epsilon^1}, \quad p_1(A_\epsilon^1) = p_2(A_\epsilon^2) = \epsilon \quad (5.20)$$

Then the law p_1 yields better resource gains than p_2 on the range of reliability $[\epsilon_1, \epsilon_2]$.

Proof: The lemma (5.2.1) proves that, for each $\epsilon \in [\epsilon_1, \epsilon_2]$, better resource gains can be obtained with the law p_1 than with the law $p_2 \circ \lambda_\epsilon$ through a fault-tolerant design. The invariance of the resource gain in a dilatation of the resource variables of a reliability law tells us that the gains of $p_2 \circ \lambda_\epsilon$ and p_2 are equal:

$$r_{p_2 \circ \lambda_\epsilon}(\epsilon) = r_{p_2}(\epsilon) \quad (5.21)$$

This completes the proof.

Notion of critical exponent

This notion comes as a corollary of the previous result, and it will play an important role in the discussion of the fault-tolerant implementability on specific instances. Without loss of generality, because of the invariance through dilatation of the resource variables, we consider laws such that $A_{1/c_3} \geq 1$. The purpose of this requirement is to ensure that, for the reliability range of interest, the higher the exponent γ , the steeper the decrease of the failure probability as a function of the resource allocation. For reliability laws with high exponents γ , the division of the resources through multiplexing worsens significantly the reliability of elementary components. It seems then natural that the resource gain at a certain reliability ϵ should be a decreasing function of the exponents γ . Let us use the claim (5.2.2) to establish this fact.

Claim 5.2.3 :

Let $p(A)$ be a single parameter reliability law such that $A_{1/c_3} \geq 1$, and $0 < \gamma_1 < \gamma_2$ be two exponents. The law $p(A^{\gamma_1})$ yields greater resource gains than the law $p(A^{\gamma_2})$.

Proof: Let us consider a reliability requirement ϵ . In order to use the property (5.2.2), we need to compare the laws $p(A^{\gamma_1})$ and $p(\lambda_\epsilon^{\gamma_2} A^{\gamma_2})$. Denoting $A_\epsilon, A_\epsilon^1, A_\epsilon^2$ the areas such that:

$$p(A_\epsilon) = p(A_\epsilon^1) = p(A_\epsilon^2) = \epsilon \quad (5.22)$$

one obtains:

$$\lambda_\epsilon = \frac{A_\epsilon^2}{A_\epsilon^1} = A_\epsilon^{1/\gamma_2 - 1/\gamma_1} < 1 \quad (5.23)$$

Our goal is to show:

$$\forall A \in]A_{1/c_3}^1, A_\epsilon^1[\quad p_1(A) \leq p_2(\lambda_\epsilon A) \Leftrightarrow A^{\gamma_1} \geq A_\epsilon^{1-\gamma_2/\gamma_1} A^{\gamma_2} \quad (5.24)$$

The latter inequality is true if $A_\epsilon > 1$, which is a consequence of the hypothesis that $A_{1/c_3} > 1$. Thus the claim is proven.

In particular the latter result implies that for any reliability resource law $p(A)$ such that $A_{1/c_3} > 1$, and for any reliability requirement ϵ , the set of exponents γ :

$$S_{p,\epsilon,r} = \{ \gamma \in \mathbb{R}^{+*} \mid p(A^\gamma) \text{ MTMR implementable with resource gain } r \text{ at reliability } \epsilon \}$$

is an interval $]0, \gamma(r, \epsilon)[$. The maximum is included in the interval because the function: $\gamma \mapsto r_{p(A^\gamma)}(\epsilon)$ (resource gain associated with the law $p(A^\gamma)$ at ϵ) is a maximum of a finite number of continuous functions of γ and is consequently also a continuous function of γ . From now on, we note $\gamma(r, \epsilon)$ the maximum exponent γ such that the law $p(A^\gamma)$ is MTMR implementable with resource gain r at the reliability ϵ . The claim (5.2.3) tells us that $\gamma(r, \epsilon)$ is a decreasing function of the resource gain r .

Definition 5.2.1 : *We call critical exponent of a law p at the reliability ϵ the maximum exponent $\gamma_c(\epsilon)$ such that $p(A^\gamma)$ is efficiently MTMR implementable at the reliability requirement ϵ .*

For each reliability requirement ϵ , the critical exponent $\gamma_c(\epsilon)$ partitions the family of laws in two groups:

- The laws with exponent $\gamma \leq \gamma_c(\epsilon)$ for which the hypothesis is valid.
- The laws with exponent $\gamma > \gamma_c(\epsilon)$ for which the hypothesis of section 4.2 may not be verified, on the basis of the failure probability evaluation through the bound (5.1).

5.2.3 Comparison of reliability laws with different resources

We proved in Chapter 4 that, in terms of stating the efficiency of a fault-tolerant design, the problems of optimization with reliability requirement and with limited resource availability are equivalent. The latter offers, however, a better insight into the relative fault-tolerant implementability of two reliability-resource laws involving distinct resources. Indeed, in the OWLRA approach, the relative performance of a multiplexed design in two laws appears directly in the evolution of the elementary gate failure probability with the recursion level:

Claim 5.2.4 *Let $p_1(R_1, \dots, R_{n_1}), p_2(S_1, \dots, S_{n_2})$ be two reliability laws such that:*

$$\begin{aligned}
 & p_1(R_1, \dots, R_{n_1}) = p_2(S_1, \dots, S_{n_2}) = \epsilon \\
 \text{and } \forall k \geq 1 \quad & p_1\left(\frac{R_1}{r_1^k}, \dots, \frac{R_{n_1}}{r_{n_1}^k}\right) \leq p_2\left(\frac{S_1}{s_1^k}, \dots, \frac{S_{n_2}}{s_{n_2}^k}\right)
 \end{aligned} \tag{5.25}$$

where r_i^k and s_i^k are the scaling factors (defined in section 4.2) associated with a MTMR design of level k . The reliability gain achievable through a multiplexed design in the device of law p_1 with a circuit of total resources R_1, \dots, R_{n_1} is better than that obtained in the device of law p_2 with a circuit of resources S_1, \dots, S_{n_2} .

Proof: Under the condition (5.25), a MTMR scheme of level k yields a circuit more reliable for the law p_1 than for the law p_2 with the considered resources allocations:

$$\begin{aligned}
 \frac{1}{c_3} \left(c_3 p_1\left(\frac{R_1}{r_1^k}, \dots, \frac{R_{n_1}}{r_{n_1}^k}\right) \right)^{2^k} & < \frac{1}{c_3} \left(c_3 p_2\left(\frac{S_1}{s_1^k}, \dots, \frac{S_{n_2}}{s_{n_2}^k}\right) \right)^{2^k} \\
 \Leftrightarrow p_{1_{ft}}(k, R_1, \dots, R_{n_1}) & < p_{2_{ft}}(k, S_1, \dots, S_{n_2})
 \end{aligned} \tag{5.26}$$

Since the direct construction gives to both devices the same reliability, the previous equation directly reflects the reliability gains:

$$r_1 = \max_{k=0..+\infty} \frac{p_1(R_1, \dots, R_{n_1})}{p_{1ft}(k, R_1, \dots, R_{n_1})} > r_2 = \max_{k=0..+\infty} \frac{p_2(S_1, \dots, S_{n_2})}{p_{2ft}(k, S_1, \dots, S_{n_2})} \quad (5.27)$$

This result turns out to be very useful in the extension of properties established for single-parameter reliability laws to more complex and realistic laws involving several resources. We will apply it in section 5.4.

5.3 Treatment of specific reliability-resource laws

We solve here the problem laid out in section 4.2 for two classes of reliability laws associated with a power and an exponential decrease of the failure probability with the amount of resources allocated to the component. One might want to consider the following families of functions:

$$p(A) = \frac{1}{\lambda A^\gamma} \quad \text{and} \quad p(A) = \frac{1}{2} \exp(-\lambda A^\gamma) \quad \text{where} \quad \lambda, \gamma > 0 \quad (5.28)$$

The power law may only describe a certain range of resource allocation, because the failure probability should not diverge when few resources are granted to the device. We shall assume that it describes the component on a range including the interval of failures probabilities $]0, \frac{1}{c_3}[$, which is the range of interest for our constructions. In the interval of resource allocation $[0, A_{1/c_3}[$ the device is ruled by an other law such that $\lim_{A \rightarrow 0} p(A) = \frac{1}{2}$.

The exponential law is taken as valid for any resource allocation, the factor $\frac{1}{2}$ in the expression ensuring that the corresponding device becomes totally unreliable when no resource is granted to it.

The property (5.2.1) tells us that the parameter λ is irrelevant for the evaluation of the fault-tolerant implementability of reliability-resource laws. That is why we

concentrate our attention on the influence of the exponent γ .

This section begins by introducing a possible representation of the efficiency of a fault-tolerant design in a family of laws. Afterwards the results associated with power and exponential laws are presented and commented. We will see that a fault-tolerant design can lead to significant reliability improvements when the exponent γ is below the critical value (definition (5.2.1)).

5.3.1 Representation of the efficiency of a Multiplexed Triple Modular Redundant design

In order to represent the efficiency of a MTMR design for a family of reliability laws indexed by their exponent, we plot the following quantities:

- The critical exponent $\gamma_c(\epsilon)$ as a function of ϵ .
- The exponent $\gamma(r, \epsilon)$, maximum exponent for which the resource gain r can be achieved through a MTMR design at the reliability ϵ , as a function of r for different reliability requirements ϵ .
- The reliability gain through the function $p_{MTMR}(\epsilon)$, optimized failure probability, as a function of the reliability ϵ obtained by a direct design. The exponent γ is fixed in this graph.

A mere inspection of the first graph reveals directly the reliability laws with each family leading to a successful MTMR design at a certain reliability requirement ϵ . The second graph allows to determine, for any resource gain objective r , what exponents γ are acceptable. The last plot shows the jump in reliability offered by a MTMR design over the direct design for a fixed value of the exponent γ .

5.3.2 Power laws: $p(A) = 1/A^\gamma$

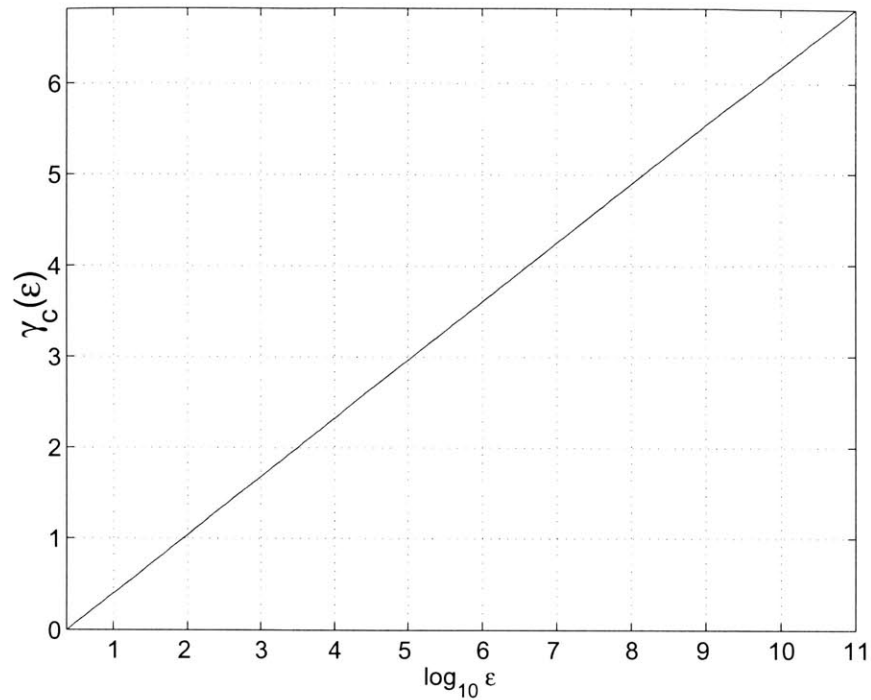


Figure 5-2: Critical exponent for the family of power laws $p(A) = A^{-\gamma}$ as a function of the reliability.

For reliability-resource laws decreasing as a power of the area, figure 5-2 shows that the critical exponent depends strongly on the reliability requirement. For power laws, at high reliability requirements, the fault-tolerant design can be efficient with fairly high exponents: as an example, for the reliability requirement $\epsilon = 10^{-9}$, the hypothesis of section 4.2 is still valid with exponents up to $\gamma = 5.5$.

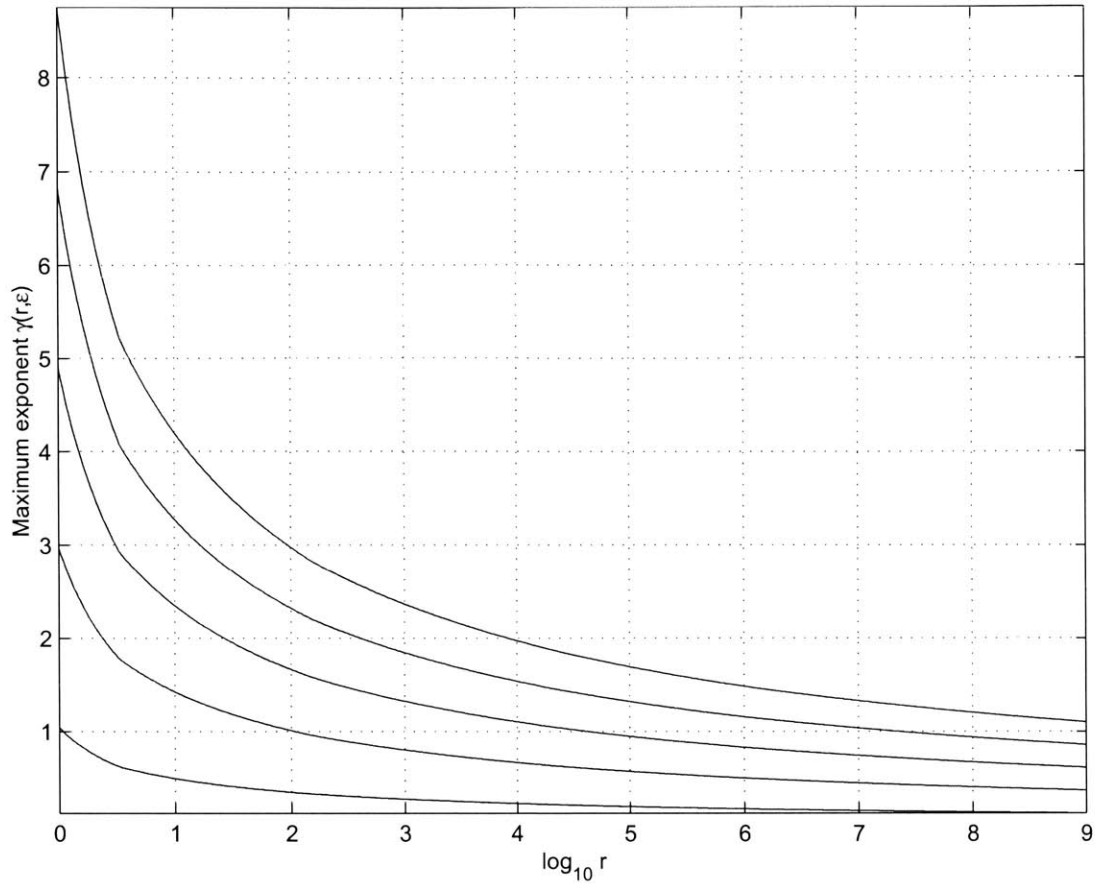


Figure 5-3: Maximum exponent $\gamma(r, \epsilon)$ for the family of power laws $p(A) = A^{-\gamma}$ as a function of the resource gain r for different reliability requirements $\epsilon = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}, 10^{-15}$. Upper curves correspond to higher reliability requirements.

Figure 5-3 shows that significant resource gain can be achieved for reliability laws with fairly high exponents. For instance, a law decreasing as the area $p(A) = 1/A$ yields a resource gain of $r = 100$ for the reliability requirement $\epsilon = 10^{-6}$.

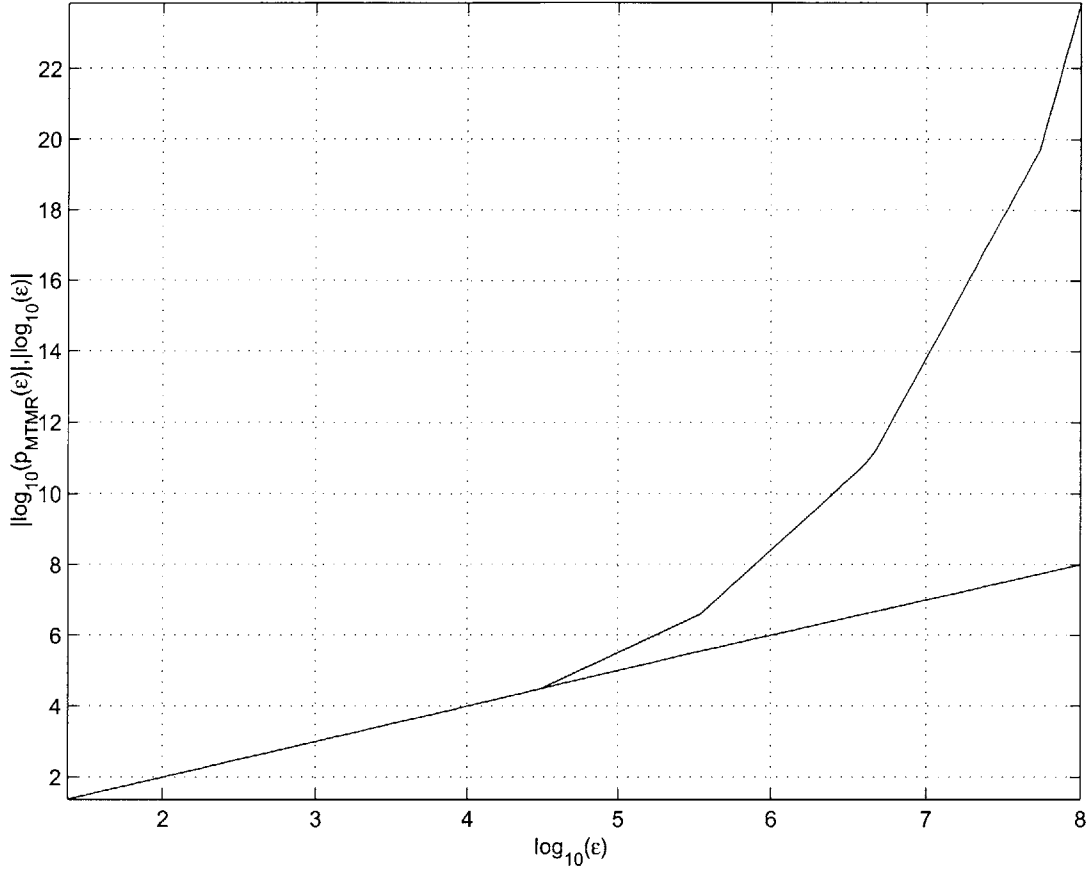


Figure 5-4: Reliability obtained through optimized allocation for the power law $p(A) = 1/A^2$ as a function of the reliability resulting from the direct allocation of the same amount of resources. The latter reliability is also plotted on the bottom curve in order to highlight the gain provided by the optimization.

In the case of the power law $p(A) = 1/A^2$, figure 5-4 reveals that, without changing the global resource allocation, a multiplexed design can bring devices whose reliability is several orders of magnitude higher. As an example, the figure 5-4 shows that, with resources leading to a failure probability of $\epsilon = 10^{-7}$ through a traditional design, one obtains a failure probability close to $p = 10^{-14}$ for the appropriate recursively multiplexed circuit. Angular points correspond to changes in the optimum recursion level. The reliability range for which the curves associated with $p_{MTMR}(\epsilon)$ and ϵ differentiate is the range of fault-tolerant implementability of the law $p(A) = 1/A^2$.

5.3.3 Exponential laws: $p(A) = \frac{1}{2} \exp(-A^\gamma)$

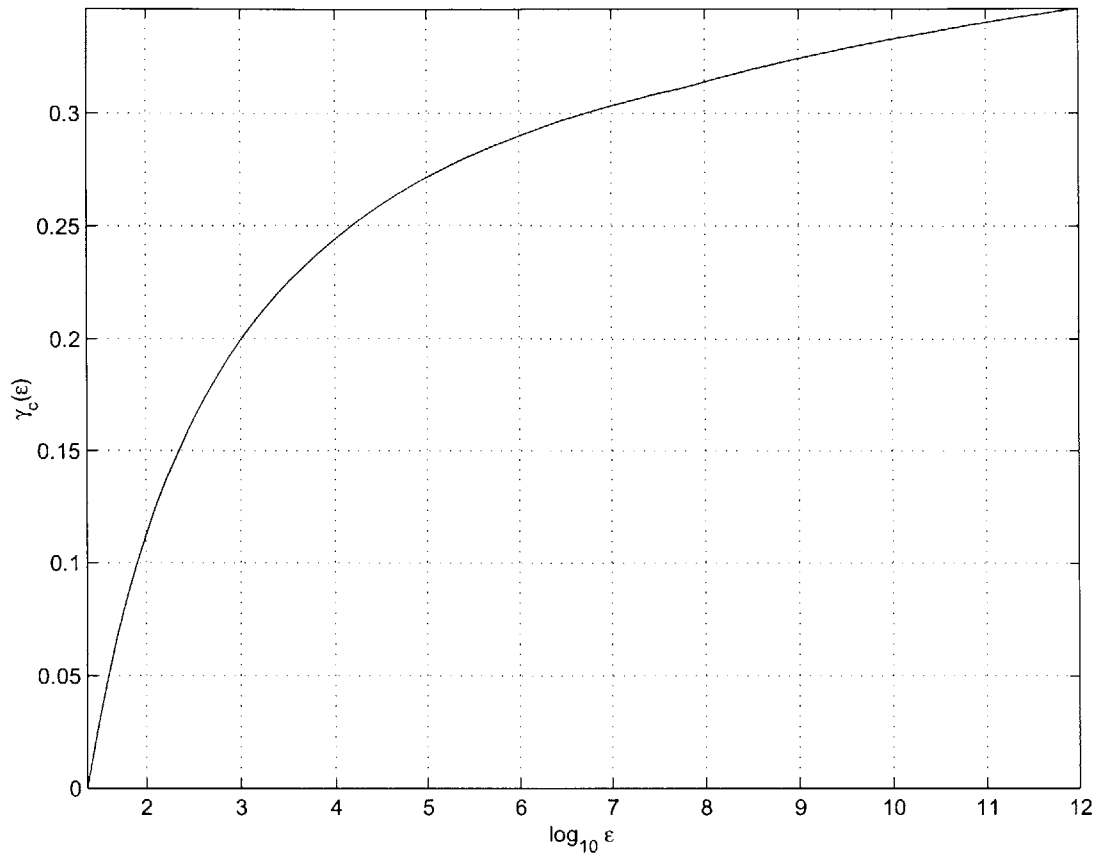


Figure 5-5: Critical exponent for the family of exponential laws $p(A) = \frac{1}{2} \exp(-A^\gamma)$ as a function of the reliability requirement.

For exponential laws, figure 5-5 reveals that the exponents compatible with a fault-tolerant design are much smaller. This fact is not surprising because the exponents γ have a different role in an exponential and in a power-like decreasing function. At the same reliability requirement $\epsilon = 10^{-9}$, the critical exponent is now $\gamma_c = 0.33$.

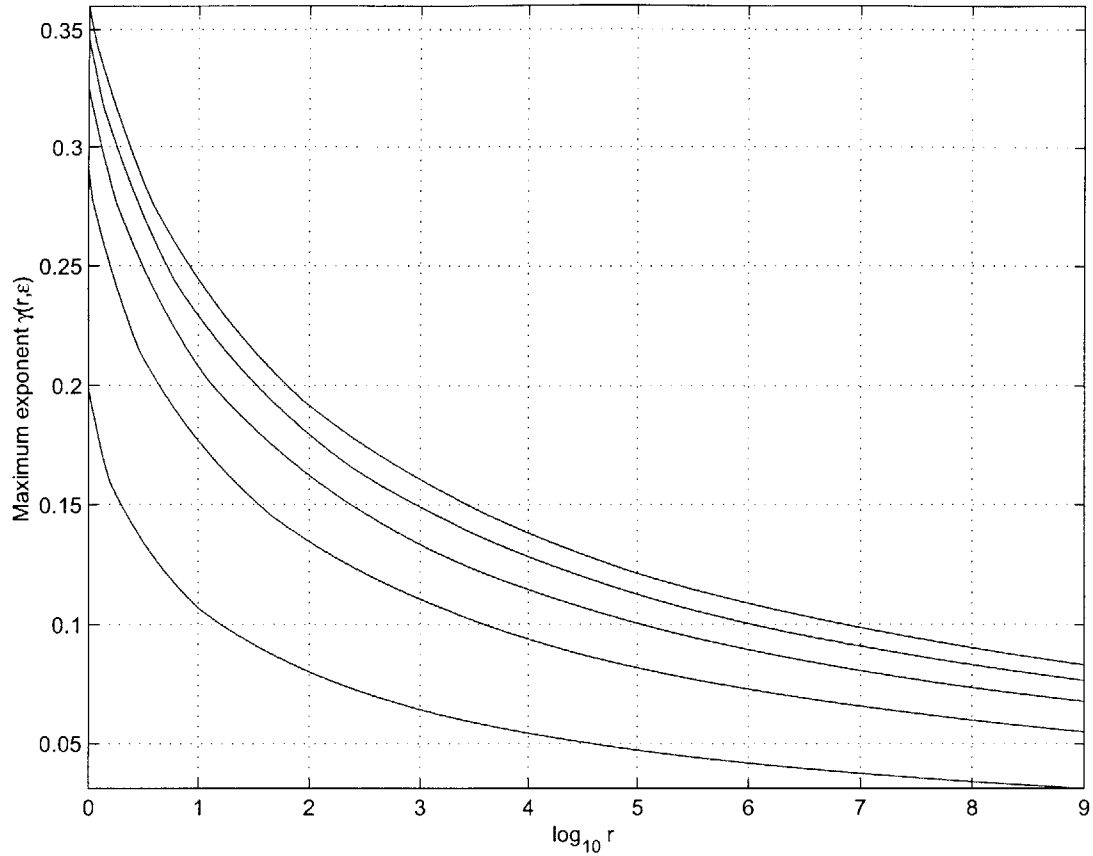


Figure 5-6: Maximum exponent $\gamma(r, \epsilon)$ for the family of exponential laws $p(A) = \frac{1}{2} \exp(-A^\gamma)$ as a function of the resource gain r for different reliability requirements $\epsilon = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}, 10^{-15}$. Upper curves correspond to higher reliability requirements.

Consistently with the last graph, we see on figure 5-6 that significant gains are not compatible with high exponents. As an example, in order to achieve a resource gain of $r = 100$ at the reliability requirement $\epsilon = 10^{-6}$, it is necessary to have an exponent below $\gamma(100, 10^{-6}) \simeq 0.13$.

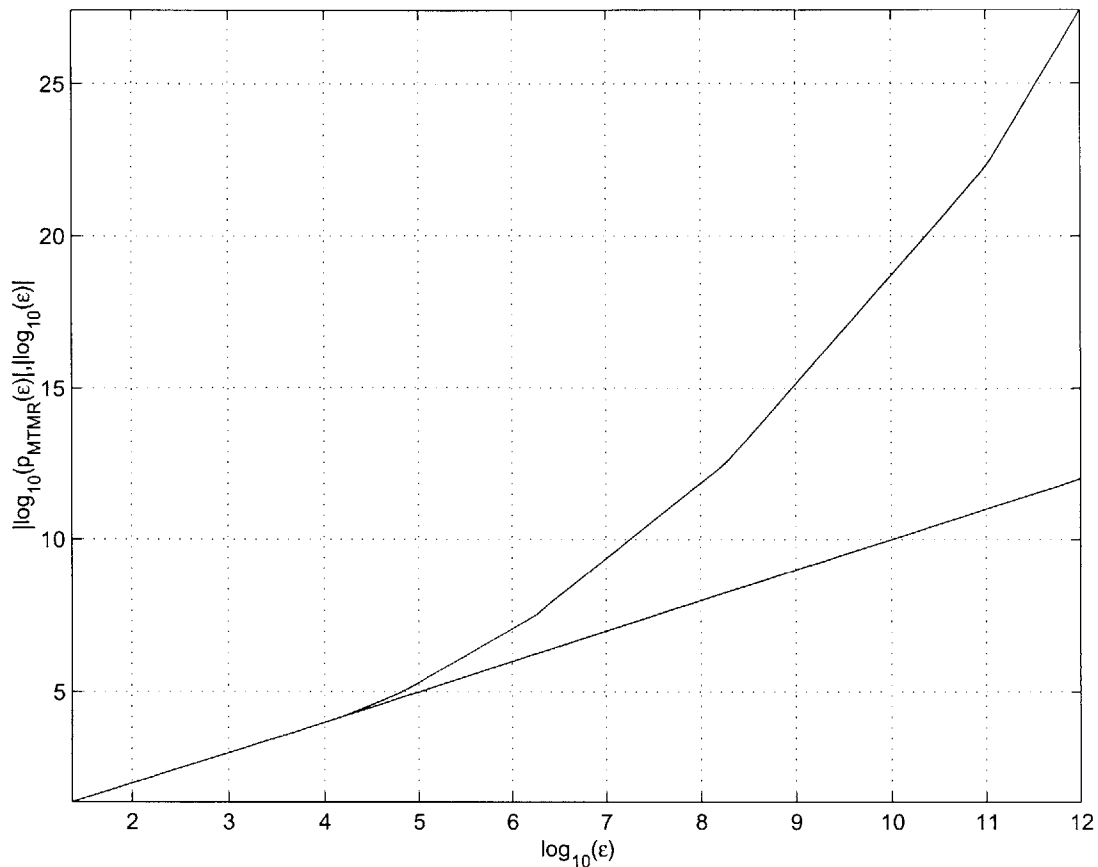


Figure 5-7: Reliability obtained through optimized allocation for the exponential law $p(A) = \frac{1}{2} \exp(-A^{1/4})$ as a function of the reliability resulting from the direct allocation of the same amount of resources. The latter reliability is also plotted on the bottom curve in order to highlight the gain provided by the optimization.

The plot on figure 5-7, associated with the law $p(A) = \frac{1}{2} \exp(-A^{0.25})$, shows that significant reliability gains may be achieved with laws whose exponent is below and close to the critical value. Indeed, for a resource allocation leading normally to a failure probability of $\epsilon = 10^{-9}$, the optimized multiplexed scheme yields a failure probability of $p_{\text{MTMR}}(10^{-9}) = 10^{-15}$, which corresponds to six orders of magnitude of reliability improvement. The fact that the critical exponent at this reliability requirement $\epsilon = 10^{-9}$ be $\gamma_c(10^{-9}) = 0.33$, which is only slightly above the exponent considered here ($\gamma = 0.25$), does not prevent from the obtention of important reliability gains. Angular points still correspond to changes in the optimal recursion level.

5.4 CMOS Transistor

The treatment of the power and exponential reliability laws was interesting from a theoretic point of view. However, to our knowledge, no physical computational system exhibits this dependence of reliability towards the resources. In order to apply the previous results on a device whose reliability law has been effectively measured, we turn back to the example of the CMOS transistor laid out in Chapter 4.

5.4.1 Reliability laws $p(A) = \frac{1}{2} \operatorname{erfc}(A^\gamma)$

To state the relevance of a fault-tolerant design for the CMOS transistor, we start again from its reliability-resource law:

$$p(A, P, T) = \frac{1}{2} \operatorname{erfc} \left(1 / \sqrt{\left(\frac{K'_w(\gamma)}{P^\gamma} + \frac{K'_f}{A} \right) \frac{1}{T}} \right)$$

with $\gamma = 1.0$ for $P < P_{th}$ and $\gamma = 0.5$ for $P > P_{th}$

(5.29)

This law involves a new family of functions derived from the complementary error function through a variation of the exponent. This function results from the integration of a gaussian distribution above a definite threshold. In the great majority of physical systems, the noise is essentially the sum of a big number of small contributions associated to independents microscopic events. The law of large numbers implies that this sum is a random variable with a gaussian distribution. The study of reliability laws associated with the function erfc is thus not only appropriate for the CMOS, but also potentially for any other device.

The corresponding family of laws is very closely related to that of exponential laws:

$$\frac{1}{2} \operatorname{erfc}(x) \simeq \frac{1}{2} \frac{\exp(-x^2)}{\sqrt{x}} \quad \text{when } x \rightarrow +\infty \quad (5.30)$$

Since the scaling is essentially dictated by the exponential, we can expect a high degree of similarity between the plots for both families of laws. Nonetheless it is necessary to

address specifically the fault-tolerant implementability of this new family of functions, in order to treat rigorously the case of the CMOS transistor and to study other devices with gaussian noise. We now follow step by step the systematic procedure exposed in section 5.1.3 and plot the graphs defined in section 5.1.3.

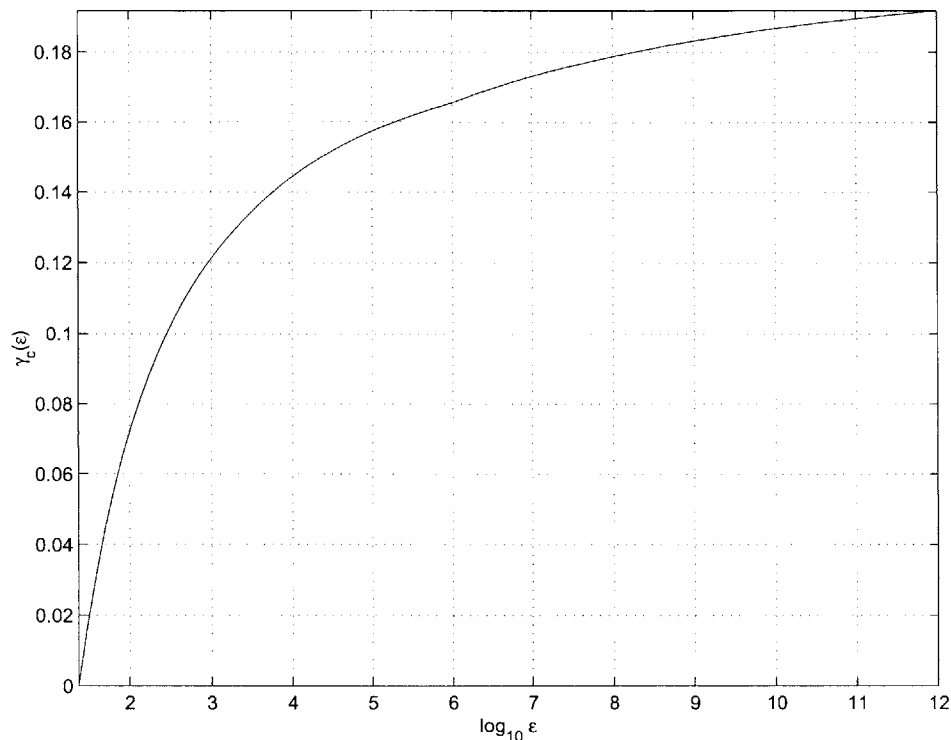


Figure 5-8: Critical exponent for the family laws $p(A) = \frac{1}{2}\text{erfc}(A^\gamma)$ as a function of the reliability requirement.

According to figure 5-8, a fault-tolerant implementation is only likely to bring some improvements with complementary erfc laws only if $\gamma \leq 0.19$, within the range of reliability requirements up to $\epsilon = 10^{-12}$. In the example of a reliability requirement $\epsilon = 10^{-9}$, the law can only bring an improvement if $\gamma_c \leq 0.185$. While this plot is very similar to that obtained with an exponential law on figure 5-5, the approximation $\frac{1}{2}\text{erfc}(x) \simeq \frac{1}{2}\exp(-x^2)$ would yield a slightly lower value $\gamma_c^{(app)} \leq \frac{0.33}{2} \simeq 0.165$. It is worth mentioning also that the critical exponent is almost constant on a wide range of reliability requirements: for $\epsilon = 10^{-6} \dots 10^{-12}$, $\gamma_c = 0.161 \dots 0.183$

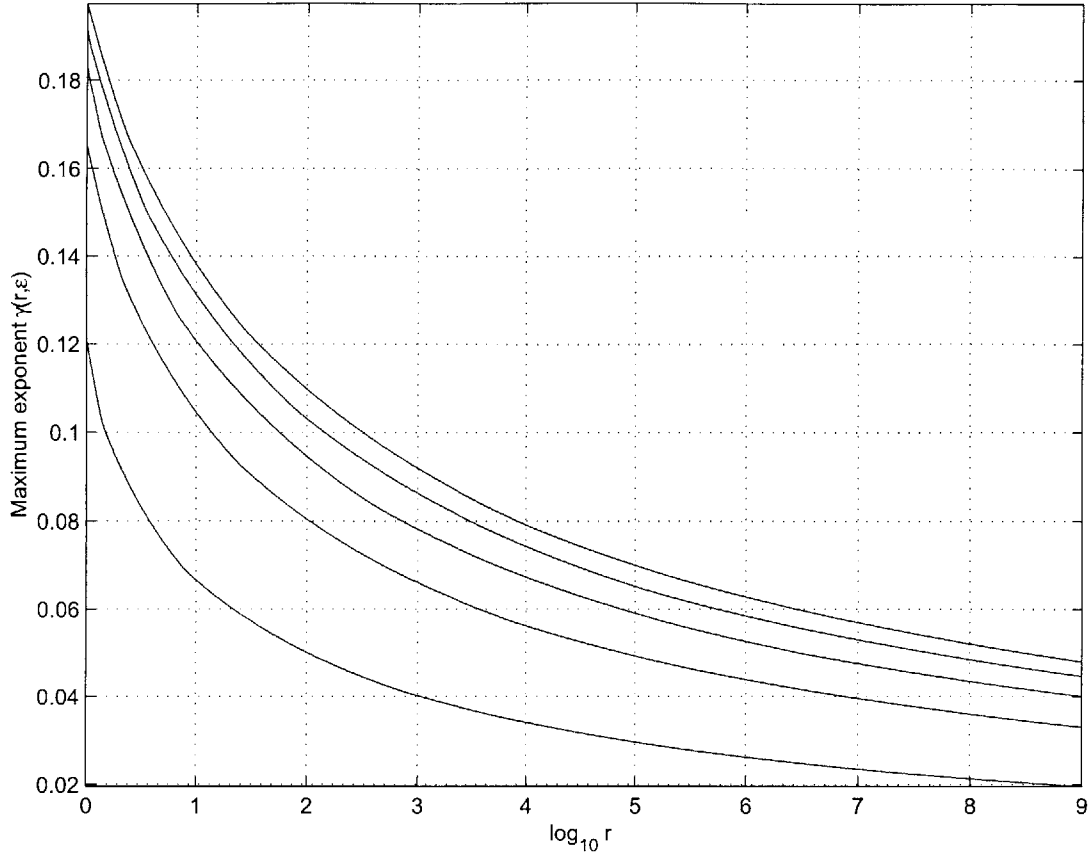


Figure 5-9: Maximum exponent $\gamma(r, \epsilon)$ for the family of laws $p(A) = \frac{1}{2}\text{erfc}(A^\gamma)$ as a function of the resource gain r for different reliability requirements $\epsilon = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}, 10^{-15}$. Upper curves correspond to higher reliability requirements.

Again, figure 5-9 is very similar to the plot obtained for exponential laws on figure 5-6. It also reveals that a fault-tolerant design may only be efficient in laws with low exponents. In order to get a resource gain of $r = 100$ for the reliability requirement $\epsilon = 10^{-9}$, it is this time necessary to have an exponent γ below $\gamma(100, 10^{-6}) = 0.08$. This value is more optimistic than that yielded by the approximation $\frac{1}{2}\text{erfc}(x) \simeq \frac{1}{2}\exp(-x^2)$, which is $\gamma^{(app)}(100, 10^{-6}) = 0.065$.

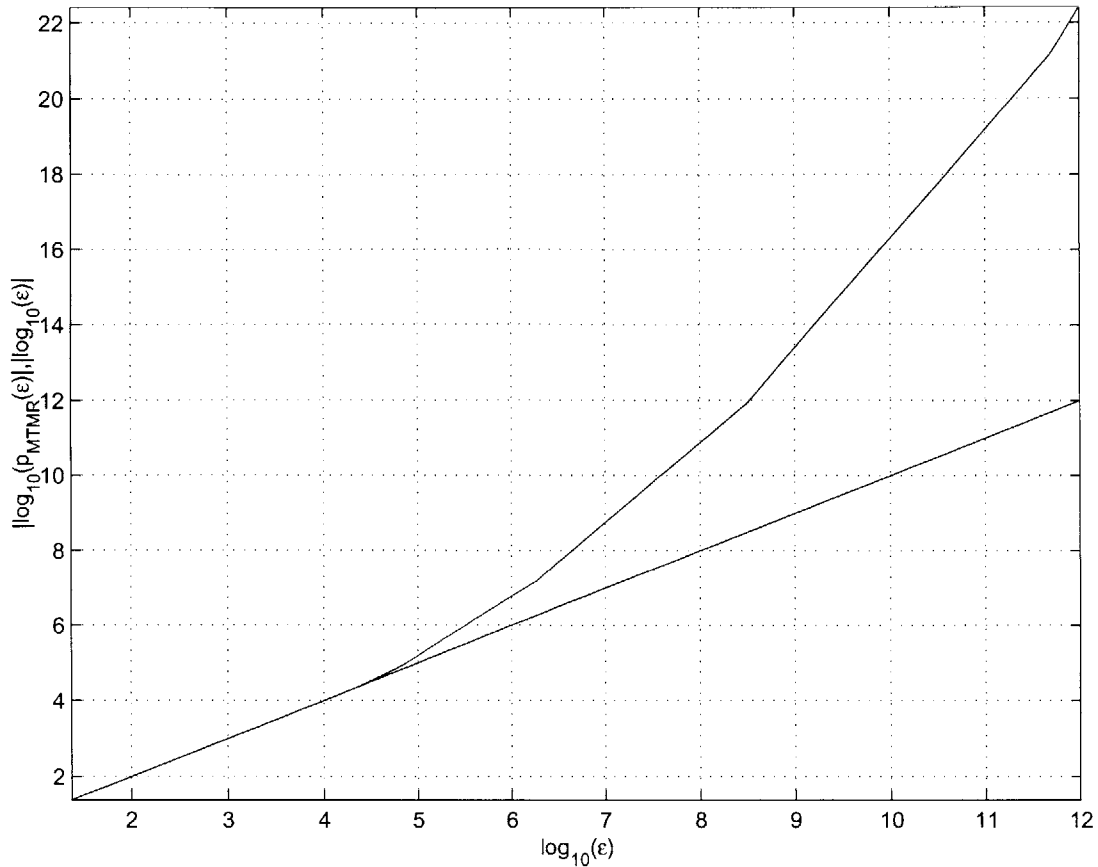


Figure 5-10: Reliability obtained through optimized allocation for the law $p(A) = \frac{1}{2}\text{erfc}(A^{0.15})$ as a function of the reliability resulting from the direct allocation of the same amount of resource. The latter reliability is also plotted on the bottom curve in order to highlight the gain provided by the optimization.

For exponents lower and close to the critical value, figure 5-10 shows that significant reliability gain can still be expected. With $\gamma = 0.15$, it is still possible to gain several order of magnitude in reliability: with an amount of resources which would lead to a failure probability of $\epsilon = 10^{-9}$ with a direct allocation, one obtains a failure probability of $p_{MTMR}(10^{-9}) \leq 10^{-13}$ with the optimized multiplexed design.

5.4.2 Relation to the fault-tolerant implementability of a CMOS transistor

We still assume that, for the reliability regime below threshold, the area A and power P obtained by direct allocation are such that $A, P > 1$. Taking into account the square root present in the reliability law, its lowest exponent $\gamma'_1 = 0.25$ is associated with the power in the above threshold regime, while the exponent associated with the area is always $\gamma'_2 = 0.5$. The graph on figure 5-8 reveals that, unfortunately, none of those exponents are compatible with an efficient fault-tolerant implementation in the case of a single-parameter erfc reliability laws. We can extend this result to the multi-parameter law associated with a CMOS transistor and show that the fault-tolerant design is not suitable for this component:

Claim 5.4.1 :

The CMOS transistor governed by the reliability-resource law (4.13) cannot be efficiently fault-tolerantly implemented.

Proof:

Let (A, P, T) be the total area, power and processing time allocated to the design of a CMOS transistor. Let us define the parameter single resource law:

$$p_2(A) = \frac{1}{2} \operatorname{erfc}(\lambda A^{1/4}) \quad (5.31)$$

where $\lambda > 0$ is chosen such that $p_2(A) = p(A, P, T)$. In order to apply the claim (5.2.4) we need to show:

$$\forall k \geq 1 \quad p\left(\frac{A}{(k+1)3^k}, \frac{P}{(k+1)3^k}, \frac{T}{(k+1)}\right) \geq p_2\left(\frac{A}{(k+1)3^k}\right) \quad (5.32)$$

Indeed:

$$\begin{aligned}
p\left(\frac{A}{(k+1)3^k}, \frac{P}{(k+1)3^k}, \frac{T}{(k+1)}\right) &\geq \frac{1}{2} \operatorname{erfc}\left(\frac{((k+1)3^k)^{1/4}}{\sqrt{\left(\frac{K'_w(\gamma)}{P^\gamma} + \frac{K'_f}{A[(k+1)3^k]^{3/4}}\right) \frac{(k+1)}{T}}}\right) \\
&\geq \frac{1}{2} \operatorname{erfc}\left(\frac{((k+1)3^k)^{1/4}}{\sqrt{\left(\frac{K'_w(\gamma)}{P^\gamma} + \frac{K'_f}{A}\right) \frac{1}{T}}}\right) \\
&\geq p_2\left(\frac{A}{(k+1)3^k}\right) \tag{5.33}
\end{aligned}$$

Consequently, the claim (5.2.4) tells us that the multi-parameter law $p(A, P, T)$ yields a lower resource gain than the law $p_2(A)$. Because of the invariance of resource gain towards dilatation of reliability law (claim (5.2.1)), p_2 yields the same gains as the law $p(A) = \frac{1}{2} \operatorname{erfc}(A^{1/4})$. But the figure 5-8 shows that this law cannot be efficiently fault-tolerantly implemented at any reliability requirement. This is true *a fortiori* for the law $p(A, P, T)$.

5.5 Conclusion

In this chapter we were able to classify a range of reliability-resource laws according to their faculty to be efficiently fault-tolerantly implemented or not. Some results are summed up in a chart on figure 5-11.

RELIABILITY LAW	RANGE OF TMR EFFICIENCY FOR $\epsilon=10^{-9}$
$p(A) = \frac{1}{A^\gamma}$	$\gamma \leq 5.5$
$p(A) = \frac{1}{2} \exp(-A^\gamma)$	$\gamma \leq 0.33$
$p(A) = \frac{1}{2} \operatorname{erfc}(A^\gamma)$	$\gamma \leq 0.185$

Figure 5-11: Range of efficiency for multiplexed triple modular redundancy

Within each family, the laws whose exponents are below these critical values verify the hypothesis of section 4.2 for the reliability requirement $\epsilon = 10^{-9}$. Furthermore, we have seen that in this case the reliability gain achievable through a fault-tolerant design can be of several orders of magnitude. Using the criterions of comparison of section 5.2, we have shown that the recursive multiplexing is not suitable for CMOS transistors. Nonetheless, it could still be a powerful design technique for computing systems exhibiting a gaussian noise with low exponents.

Chapter 6

Conclusion

This thesis dealt with an original procedure of resource optimization in computing systems through the use of fault-tolerant techniques. Inspired by the evolution of near-term technologies toward components subject to high rates of soft errors, we introduced an original point of view on reliability: reliability as a resource of the computational system, both physical and fungible. This new perspective allowed us to apply usual fault-tolerant schemes to a new task: the design of more efficient devices.

6.1 Summary and contributions

After introducing the concepts of fault-tolerance, we defined a framework suitable for probabilistic encoded logic. Afterwards, we reviewed the modular-redundancy based architectures and we evaluated their performance. We then introduced the main assumption of this thesis: the failure probability of a computing system depends continuously on the presence of resources. Reliability is an asset which can be converted for other physical resources. This conversion is manifold and can be realized along a countable class of trades-off obtained thanks to fault-tolerant constructions. In this context, we formulated the problem of optimization through a fine-grained fault-tolerant design. We determined whenever such a design is appropriate, and we evaluated the associated gains in resources and reliability. We applied this analysis to the CMOS transistor, but our treatment encompassed more generally

the family of devices subject to a gaussian noise. Surprisingly, an adapted use of redundancy may allow these components to rely on fewer resources. This is a new and important concept.

6.2 Open Problems

It could be fruitful to look for other fault-tolerant schemes than recursively multiplexed triple modular redundant circuits. Indeed, it is in principle possible to define logic gates encoded in a different code than the recursive concatenation of the triple repetition code. For instance, one could imagine schemes based on d -modular multiplexing in which the degree of modularity would vary with recursion level. Such schemes would still be recursively defined but not self similar anymore. Nonetheless, in order for the analysis to be valid, one should assume that the NAND and the d -majority gates are ruled by different reliability laws, which would complicate significantly the optimization problem.

Furthermore, a non-recursive logic gate encoding can also be investigated. We have seen in section 2.4 the successful protection of a specific computation through a non-recursive family of codes, example due to Beckmann [Bec92]. There might exist such a class of codes applicable to general boolean computation and compatible with a gate encoding in which error propagation is contained, along the guidelines of section 2.2. It would be interesting to derive the corresponding fault-tolerant circuits and explore the associated trades-off, which might outperform those obtained with the concatenation of repetition codes.

An other important improvement would be to account for the cost of the increased wiring imposed by multiplexing. One could estimate the resources devoted to wires, or investigate the trades-off in presence unreliable communications. This model of circuit would have an interesting counterpart in quantum computing systems: the movement of quantum bits in those architectures is indeed not reliable [COI⁺04].

6.3 Relation to emerging technologies

If fault-tolerant architectures can be profitable to current computing systems, they will be without any doubt the cornerstone of emerging technologies such as quantum computing or molecular electronics in which dynamic errors overwhelm static ones. Indeed, scalable fault-tolerant quantum computing architectures are currently investigated [COI⁺04]. There is also a growing interest in the introduction of multiplexed modular redundancy in nano-devices [HJ02], [NSF02]. In both quantum computing and molecular electronics, new trades-off between reliability and resources will have to be considered. We hope that the procedure of optimization through fault-tolerant design derived in this thesis will help engineers and physicists face the technological challenges in those fields.

Appendix:

Definitions and Acronyms

Acceptable building set:

Element of the set $S_A = \{(\epsilon, A_0) \in (\mathfrak{R}^{+*})^2 \mid A_0 \in I_\epsilon =]A_{1/c_3}, A_\epsilon[\}$.

$b^{(k)}$ -**bundle:**

Definition 3.3.1 : We call $b^{(k)}$ -bundle a bundle of 3^k wires. A $b^{(k)}$ -bundle contains three $b^{(k-1)}$ -bundles which we note $b_1^{(k-1)}, b_2^{(k-1)}, \dots, b_{3^{k-1}}^{(k-1)}$.

Combinational Circuit:

Definition 2.3.5 : A combinational C circuit made out of processors from S is a map from $\mathbf{R}(\{0, 1\}^M)$ to $\mathbf{R}(\{0, 1\}^N)$ obtained by composition of processors from S .

Computation:

Definition 2.3.4 : Let $\epsilon \in]0, 1/2[$ and P be a processor computing a function $F : \{0, 1\}^m \rightarrow \{0, 1\}$ in the sense of definition (2.3.2) .

- We shall say that a processor P *weakly* ϵ -computes F if and only if :

$$\forall \mathbf{x} \in \{0, 1\}^m \quad p(\delta = 1 \mid \mathbf{X} = \mathbf{x}) \leq \epsilon \quad (1)$$

- We shall say that a processor P *strictly* ϵ -computes F if and only if :

$$\forall \mathbf{x} \in \{0, 1\}^m \quad p(\delta = 1 \mid \mathbf{X} = \mathbf{x}) = \epsilon \quad (2)$$

- A processor P is said to *reliably* compute F if it 0-computes f .

Correct computation:

Definition 2.3.10 : Let $T : \{0, 1\} \rightarrow \{0, 1\}^l$ and $D : \{0, 1\}^l \rightarrow \{0, 1\}$ be two functions such that $\mathbf{D} \circ \mathbf{T} = Id$ on $\{0, 1\}$. Let C be a combinational circuit implementing a T -encoded computation of $F : \{0, 1\}^m \rightarrow \{0, 1\}^n$, taking the inputs $\mathbf{X} = X_1, \dots, X_{m \times l}$ and delivering the outputs $\mathbf{Y} = Y_1, \dots, Y_{n \times l}$.

We shall say that a computation yielding $\mathbf{X} = (x_1, \dots, x_{m \times l})$ and $\mathbf{Y} = (y_1, \dots, y_{n \times l})$ is a *correct* (\mathbf{T}, \mathbf{D}) -computation of F if and only if:

$$(D[(y_1, \dots, y_l)], \dots, D[(y_{(n-1)l+1}, \dots, y_{nl})]) = F[D(x_1, \dots, x_l), \dots, D(x_{(m-1)l+1}, \dots, x_{ml})] \quad (3)$$

Critical exponent:

Definition 5.2.1 : We call critical exponent of a law p at the reliability ϵ the maximum exponent $\gamma_c(\epsilon)$ such that $p(A^\gamma)$ is efficiently MTMR implementable at the reliability requirement ϵ .

Distributed 3-Majority Voting of level k :

Definition 3.3.2 : A distributed 3-majority voting system of level k , noted $M^{(k)}$, is a gate taking in input the data carried by a $b^{(k)}$ -bundle of wires and whose output is defined recursively as being:

- for $k=1$ the majority vote of the three single wires $b_1^{(0)}, b_2^{(0)}, b_3^{(0)}$.
- for $k \geq 2$ it is obtained by the following procedure.

The input $b^{(k)}$ consists in 3 bundles $b_1^{(k-1)}, b_2^{(k-1)}, b_3^{(k-1)}$

or in 9 bundles $b_{1,1}^{(k-2)}, b_{1,2}^{(k-2)}, b_{1,3}^{(k-2)}, \dots, b_{3,3}^{(k-2)}$.

We reorganize the bundles $b_{1,j}^{(k-2)}, b_{2,j}^{(k-2)}, b_{3,j}^{(k-2)}$ into a bundle $c_j^{(k-1)}$.

We send each of the $c_j^{(k-1)}$ into a majority voting $M^{(k-1)}$.

The reunion of their outputs noted $d_j^{(k-2)}$ forms a bundle $d^{(k-1)}$ which defines the output of $M^{(k)}$.

Deviation:

Definition 2.3.8 : Let C be a boolean circuit and W be the random variable associated with a wire of the circuit. We shall say that the wire W deviates during a computation when it takes a different value from that yielded by a perfect computation in C .

Encoded computation:

Definition 2.3.9 : Let $T : \{0, 1\} \rightarrow \{0, 1\}^l$ be a function. Let $F : \{0, 1\}^m \rightarrow \{0, 1\}$ be a boolean function, and C be a combinational circuit with the inputs $\mathbf{X} = X_1, \dots, X_{m \times l}$ and outputs $\mathbf{Y} = Y_1, \dots, Y_{n \times l}$, built out of the processors P_1, \dots, P_N . We shall say that C represents the T -encoded implementation of F if and only if:

$$\forall (x_1, \dots, x_m) \in \{0, 1\}^m \quad r(T(\mathbf{x})) = T[F(\mathbf{x})] \quad (4)$$

Fault-Path:

Definition 2.3.2 : Let C be a combinational circuit consisting in the processors P_1, \dots, P_N . We shall call fault-path the random vector $\mathbf{E} = (E_1, \dots, E_n)$ where the coordinates E_i are binary random variables such that:

$$E_i = 1 \quad \Leftrightarrow \quad P_i \text{ fails}$$

Fault-Tolerance:

Definition 2.1.1 : We shall say that a system is *fault-tolerant* when it preserves a *desirable behavior* under the occurrence of a certain rate of *internal faults*.

Fault-Tolerant Implementability:

A reliability-resource law is efficiently fault-tolerantly implementable on the reliability range on which MTMR constructions yield resource gains greater than one.

MTMR implementation of level k : Multiplexed triple modular implementation of level k .

Multiplexed Triple Modular Implementation of level k :

Definition 3.3.3 : A multiplexed triple modular redundant (MTMR) implementation of level k of a NAND gate, noted $N^{(k)}$, is a gate which takes in input two bundles $a^{(k)}, b^{(k)}$ (each corresponding to the encoding of a logical bit) and whose output is recursively defined as being:

- for $k = 0$ the output of a NAND
- for $k \geq 1$ the scheme obtained by the following procedure.

The two input bundles $a^{(k)}, b^{(k)}$ may be decomposed into three pairs of bundles $a_j^{(k-1)}, b_j^{(k-1)}$ which are sent into an encoded NAND $N^{(k-1)}$ for each $j \in \{1, 2, 3\}$. The outputs of the three NAND $N^{(k-1)}$ form a bundle $c^{(k)}$, which is sent into three identical distributed 3-majority voting systems $M^{(k)}$. The reunion of their outputs $d_j^{(k-1)}$ forms a bundle $d^{(k)}$ defining the output of $N^{(k)}$.

Optimal Area: Element $A(k, \epsilon)$ of the set S_ϵ defined in claim (5.1.1).

OWLRA: Optimization With Limited Resource Availability, problem is defined in section 4.2.3.

OWRR: Optimization With Reliability Requirement, problem is defined in section 4.2.3.

Perfect Computation:

Definition 2.3.7 : Let C be a combinational circuit consisting in the processors P_1, \dots, P_N , of inputs $\mathbf{X} = X_1, \dots, X_m$ and outputs $\mathbf{Y} = Y_1, \dots, Y_n$.

- A *perfect* (or *fault-free*) computation of x_1, \dots, x_m is a computation yielding $\mathbf{X} = (x_1, \dots, x_m)$ and $\mathbf{E} = (E_1, \dots, E_n) = \mathbf{0}$
- The *perfect result* $r(x_1, \dots, x_m)$ the value taken by the vector $\mathbf{Y} = (y_1, \dots, y_n)$ in a perfect computation. The function $r(x_1, \dots, x_m)$ is the function *represented* by the circuit.

PMTMR(A): Function giving the failure probability of the optimal MTMR circuit with a total resource allocation A .

Processor:

Definition 2.3.2 : A processor P computing $F : \{0, 1\}^m \rightarrow \{0, 1\}$ and delivering a single output is a map:

$$G : \mathbf{R}(\{0, 1\}^m) \longrightarrow \mathbf{R}(\{0, 1\}) \quad (5)$$

$$(X_1, \dots, X_m) \longmapsto Y = F(X_1, \dots, X_m) + \delta \quad (6)$$

where δ is a random variable reflecting the errors occurring in the processor. δ is independent from the inputs (X_1, \dots, X_m) , and two variables δ associated with distinct processors are independent.

Definition 2.3.3 : Two processors implement the same gate if they compute the same function F up to a composition with a permutation of their inputs. Two processors implementing the same gate are identical if the distribution of their respective random variables δ are equal.

Reliability-resource law or Reliability law:

Relation between the failure probability of a component and its consumption of resources satisfying the assumptions of section 4.2.3.

Resource gain:

Definition 5.1.1 : We shall call resource gain:

$$r(\epsilon) = \max_{k=0\dots+\infty} \frac{A_\epsilon}{A(k, \epsilon) (k+1)3^k}$$

Reliability gain:

Definition 5.1.2: We shall call reliability gain:

$$s(A) = \max_{k=0\dots+\infty} \frac{c_3 p(A)}{\left(c_3 p \left(\frac{A}{(k+1)3^k} \right) \right)^{2^k}}$$

Separate, Systematic Codes:

Definition 2.1.2 : A *systematic* error correcting code is a code whose codewords are composed of two distinct parts: the original word, and parity check symbols derived from it. A systematic error correcting code is *separate* if the encoded operations are performed separately on the original data and on the parity check symbols.

TMR implementation of level k : Triple Modular Implementation of level k

Triple Modular Implementation of level k :

Definition 3.2.1:

Let P be a deficient processor computing a boolean function F with a single output. A triple modular redundant (TMR) implementation of the processor P of level k (noted $P^{(k)}$) is:

- for $k = 0$ the processor P .
- for $k \geq 1$ the scheme obtained by embedding the outputs of three TMR implementation of P of level $k - 1$ into a 3-majority gate.

Bibliography

- [Bec92] P.E. Beckmann. Fault-tolerant computation using algebraic homomorphisms. *PhD Thesis, Massachusetts Institute of Technology*, 1992.
- [CAC⁺96] B. Culbertson, R. Amerson, R. Carter, P. Kuekes, and G. Snider. Tera-mac custom computer: Extending the limits with defect tolerance. *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 2–10, November 1996.
- [COI⁺04] D. Copsy, M. Oskin, F. Impens, T. Metodiev, A.W. Cross, F.T. Chong, I.L. Chuang, and J. Kubiawicz. Toward a scalable, silicon-based quantum computing architecture. *Journal of Selected Topics in Quantum Electronics*, to appear in 2004.
- [DO77a] R.L. Dobrushin and S.I. Ortyukov. Lower bound for the redundancy of self correcting arrangements of unreliable functional elements. *Prob. Inform. Trans.*, **13**, 59–65, 1977.
- [DO77b] R.L. Dobrushin and S.I. Ortyukov. Upper bound on the redundancy of self correcting arrangements of unreliable functional elements. *Prob. Inform. Trans.*, **13**, 203–218, 1977.
- [Eli58] P. Elias. Computation in the presence of noise. *IBM Journal of Research and Development*, **2**(10), 1958.

- [EP98] W. Evans and N. Pippenger. On the maximum tolerable noise for reliable computation by formulas. *IEEE Trans. Inf. Theory*, **44**(3), 1299–1305, May 1998.
- [G85] A. Gál. Lower bounds for the complexity of reliable boolean circuits with noisy gates. *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pp. 594–601, 1985.
- [G86] P. Gács. Reliable computation with cellular automata. *Journal of Computer and System Sciences*, **32**(1), 15–78, 1986.
- [Had02] C.N. Hadjicostis. *Coding approaches to fault tolerance in combinational and dynamic systems*. Boston : Kluwer Academic Publishers, 2002.
- [HJ02] J. Han and P. Jonker. A system architecture solution for unreliable nano-electronic devices. *IEEE Trans. on Nanotechnology*, **1**(4), 201–208, 2002.
- [HKS98] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, **280**, 1716–1721, 1998.
- [HW91] B. Hajek and T. Weller. On the maximum tolerable noise for reliable computation by formulas. *IEEE Trans. Inf. Theory*, **37**, 388, 1991.
- [Lik99] K. K. Likharev. Single-electron devices and their applications. *Proc. IEEE*, **87**, 606–632, 1999.
- [McE76] R.J. McEliece. *Theory of Information and Coding*. Cambridge University Press, 1976.
- [MP43] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, **5**, 115–133, 1943.
- [MSST00] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Toward robust integrated circuits: The embryonics approach. *Proc. IEEE*, **88**, 516–541, 2000.

- [Neu56] J. Von Neumann. Probabilistic logic and the synthesis of reliable organisms from unreliable components. *Automata Studies, Ann. of Math. Studies*, **34**, 43–98, 1956.
- [NSF02] K. Nikolic, A. Sadek, and M. Forshaw. Fault-tolerant techniques for nanocomputers. *Nanotechnology*, **13**(3), 357, June 2002.
- [Pip85] N. Pippenger. On networks of noisy gates. *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, pp. 30–36, 1985.
- [Pip89] N. Pippenger. Invariance of complexity measures for networks of unreliable gates. *J. ACM*, **36**, 531–9, 1989.
- [PST91] N. Pippenger, G.D. Stamoulis, and J.N. Tsitsiklis. On a lower bound for the redundancy of reliable networks with noisy gates. *IEEE Trans. Inf. Theory*, **37**(3), 639–643, 1991.
- [RCM⁺97] R.A. Reed, M.A. Carts, P.W. Marshall, C.J. Marshall, O. Musseau, P.J. McNulty, D.R. Roth, S. Buchner, J. Melinger, and T. Corbière. Heavy ion and proton-induced single event multiple upset. *IEEE Trans. Nuclear Science*, **44**(6), 2224–2229, July 1997.
- [RDM93] R. Sarpeshkar, T. Delbrück, and C.A. Mead. White noise in mos transistors and resistors. *IEEE Circuits and Devices Magazine*, **9**(6), 23–29, November 1993.
- [Sar98] R. Sarpeshkar. Analog versus digital: Extrapolating from electronics to neurobiology. *Neural Computation*, **10**(7), 1601–1638, October 1998.
- [Sha48a] C.E Shannon. A mathematical theory of communications (part i). *Bell System Technical Journal*, **27**(7), 379–423, 1948.
- [Sha48b] C.E Shannon. A mathematical theory of communications (part ii). *Bell System Technical Journal*, **27**(10), 623–656, 1948.

- [Sho96] P.W. Shor. Fault-tolerant quantum computation. *37th Symposium on Foundations of Computer Science*, pp. 56–65, 1996.
- [Ste96] A.M. Steane. Multiple particle interference and quantum error correction. *Proc. R.Soc. London A*, **452**, 2551–2576, 1996.
- [Tay90] M. G. Taylor. Reliable information storage in memories designed from unreliable components. *The Bell System Journal*, **47**(10), 2299–2337, 1990.
- [WC63] S. Winograd and J.D. Cowan. *Reliable Computation in the Presence of Noise*. MIT Press, Cambridge, Massachusetts, 1963.
- [Win80] S. Winograd. On multiplication of polynomials modulo a polynomial. *SIAM Journal of Computing*, **9**(2), 225–229, 1980.