Universitat Politècnica de Catalunya

Master Thesis

An approach to quantifying hardware diversity against common cause failures

---

Author:
Sergi Alcaide i Portet

Supervisor:
Dr. Jaume Abella Ferrer
Dr. Carles Hernandez Luz

A thesis submitted in fulfillment of the requirements
for the degree of Master in Innovation and Research in Informatics

in the

Departament d'Arquitectura de Computadors (AC)
Facultat d'Informàtica de Barcelona (FIB)
at Computer Architecture - Operating Systems (CAOS) department -
Barcelona Supercomputing Center (BSC)

January 14, 2018

Declaration of Authorship

I, Sergi Alcaide i Portet, declare that this thesis titled, "An approach to quantifying hardware diversity against common cause failures" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master degree at the Universitat Politècnica de Catalunya.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Universitat Politècnica de Catalunya or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

Universitat Politècnica de Catalunya

Abstract

Reliability is an important concern in computing systems. In some cases, full redundancy is used for fault tolerant purposes so that, if one of the units experiences a fault, the other(s) still operate correctly, thus preventing the fault from becoming a system failure. However, redundancy may not be enough if system replicas are identical since identical faults in all replicas may lead to the same (erroneous) output. These identical faults can occur due to common cause faults (using automotive jargon). For instance, an electrical interference may produce exactly the same fault in all replicas, which provide the same erroneous output, thus defeating the purpose of using redundancy.

Diversity has been regarded as a desirable property of redundant systems, since it allows them to behave differently in front of a given fault. Diversity relates to the realization of the same function with different means. Such diversity can be applied at any abstraction level, spanning from the system design to the physical implementation. However, although, diversity is a well understood concept, there is no current metric which is able to quantify it. In this thesis, we cover this gap by introducing DIMP, a low-cost diversity metric based on analyzing the paths of the circuits. Relating it to the particular case of automotive microcontrollers that implement lockstep cores, we show that DIMP can provide relevant information for addressing the common cause faults. In particular, we have validated DIMP in the context of timing faults with SPICE simulations. Additionally, we have also extended DIMP metric to cover other types of faults like the ones caused by systematic process variations.

Acknowledgements

First, I want to deeply thank my advisors Carles and Jaume for their guidance and mentoring through the development of this thesis.

I also want to thank the rest of the people of the CAOS group at BSC who always offer help when I needed.

Furthermore, I would like to acknowledge BSC for financially supporting my master studies.

Last but not least, I would like to thank my parents and my sister for their unconditional support both in my studies and in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Safety-related systems must undergo an exhaustive validation and verification (V&V) process before being deployed to prove that their safety goals are met. This process needs to collect enough evidence to prove that either the risk of death or serious injury to people, the loss or severe damage to equipment/property and/or the environmental harm due to malfunctioning behavior of the system can be regarded as negligible. Therefore, a hazard analysis and risk assessment is needed for safety-related systems to determine the degree of prevention or mitigation needed in order to avoid unreasonable risk.

These safety goals are defined at the scope of the whole system, but also they are propagated to all the components so their composition preserves the safety goals. This translates into attaching to each component a Safety Integrity Level, which determines the type and amount of evidence needed in the V&V process to prove that the component under analysis will meet its specifications. For instance, in the case of the automotive domain, the ISO26262 functional safety standard is used for the V&V process of electrical and electronic systems, and defines four Automotive Safety Integrity Levels (ASIL), from A to D, being D the highest integrity level and A the lowest [4]. Other similar examples can be found in other domains, such as avionics (e.g., DO178B/C [5]), railway (e.g., EN50126/8 [6]) and general electronic safety-related systems (e.g., IEC61508 [7]) among others.

When mapping safety integrity levels to hardware components such as microcontrollers, meeting the requirements for high integrity levels requires providing hardware

components with specific safety mechanisms. For instance, in the context of the automotive domain, microcontrollers providing ASIL C and D functionalities often require some form of hardware redundancy [8, 9]. Although redundancy is very useful against most faults, it is vulnerable to faults that can produce the same failure on each of the instances of the system (e.g., due to voltage droops). This type of failures are often produced by *common cause faults* and need, not only redundancy but also *diversity*, so that the manifestations of the fault are different in each diverse instance and hence, the fault can be detected before becoming a failure.

Diversity can be obtained by using different implementations of the same hardware functionality and can be applied at different abstraction levels (e.g., gate level, transistor level). However, although diversity is well-understood at a qualitative level, it is hard to quantify it since it relates to the specific target that creates the common cause fault and the abstraction level at which it is considered. Some efforts have been done to quantify diversity systematically [10, 11], but specific realizations have only been proven successful for random faults. Unfortunately, the existing diversity quantification methods were not designed targeting common cause faults, whose nature and effects are different. Therefore, industry faces the complex issue of delivering diverse implementations without practical means to quantify their effectiveness. These issues exacerbate with the increasing complexity of microcontroller designs needed to keep pace with the performance requirements of the safety-related functionalities in different domains such as automotive [12, 13] and avionics [14] among others.

1.2 Contribution

In this thesis we cover the gap of quantifying diversity by proposing DIMP, a new metric that systematically compares circuits and measures the diversity based on path analysis of these circuits. The proposed metric has a relatively low cost in terms of computational effort and complexity compared to the existing diversity quantification metrics. In this thesis, we suit DIMP to the particular case of timing faults as a representative example of common cause faults, and apply it at the gate-level granularity. The results are compared against existing diversity metrics and we show that

in controlled scenarios, where specific outputs are expected, our metric is the only metric delivering a meaningful answer.

We have also specialized the proposed metric to cover other type of common cause faults. In particular, we consider the impact of systematic process variation to define our fault model. In this regard, we have extended the evaluation method and compare the original DIMP implementation with the results of the specialized metric. Unfortunately, we have found that there is no strong correlation between the specialized metric and the error manifestations for the considered fault model. However, although we cannot correlate the results of the new metric with the evaluation method, we propose some ideas that can be explored in the future to overcome some of the problems we have faced.

1.3 Structure of the Thesis

The rest of this thesis is organized as follows:

Chapter 2 presents some background about safety related systems and the most widely used safety mechanisms to help the reader understanding the scope of the thesis and the main problem we want to solve.

Chapter 3 presents DIMP, our solution to quantify diversity in a systematic manner, and its evaluation and validation. In this chapter we also compare DIMP against other existing diversity quantification approaches.

Chapter 4 extends DIMP to cover systematic faults originated by systematic process variations.

Chapter 5 presents the conclusions and gives some ideas about future work to be done in this area.

Chapter 6 contains the bibliography used for this thesis.

Appendix A: Shows the complete results for one of the experiments of chapter 3.

The work done in this thesis has been published in the Design Automation conference (DAC), which is the premier conference in the area of circuits and systems, and also presented in the ACACES Summer School. The exact references of the publications can be found at Appendix B: Published Work.

Chapter 2

Background

2.1 Faults, Failures and Errors

In this thesis, we use the common terminology in fault-tolerant systems:

Fault is a defect within the system, it can be due to multiple reasons: Erroneous design, software bug, random hardware fault, due to cosmic rays, defect of some component, etc.

When faults are not correctly fixed they can lead to **errors**. An **error** is a deviation from the required operation of the system, or subsystem. Errors are the manifestations of the faults, although a fault may not lead to an error.

For example, consider a 2 (inputs) AND gate at some circuit design. The expected output of this gate is a '1' since both inputs are '1'. If in one of the inputs there is a fault that causes one of the inputs to switch to a '0', we will have an error because the output of this gate will not be '1'. However, if the fault causes the input to be stuck at '1', since the correct output value is a '1', the output is correct anyway. Despite that in both cases we have the same fault, only in one of them the fault becomes an error.

A system **failure** occurs when a system *fails* to perform its required operation. The presence of an error might cause the whole system to deviate from its required operation or not. One of the main goals of the safety-critical systems is minimizing the likelihood that errors lead to system failures.

2.2 Safety Related Systems

A safety related system or safety-critical system, is a system whose failure or malfunctioning may result in a death or an injury to a person, loss or severe damage to equipment/property or an environmental harm. These systems must be protected to ensure safety is not compromised and failures occur with negligible probability. For example, in the aircraft industry a figure of 10^{-9} accidents per hour is often quoted as an acceptable target in the standards [15].

Failures can be produced by different kinds of faults: bugs in the software, faults in the hardware, spontaneous malfunctioning of the system, etc. Some of them can be corrected during their testing before the deployment but others, such as transient faults, require specific mechanisms to ensure that those faults will not cause a system failure, or that at least failures will only occur within a given target failure rate.

Transient faults, also called single event upsets (SEU's) or soft errors, are faults that can appear in some parts of our system and can alter its behavior momentaneously. They are random in nature, temporary and unpredictable. Because of their characteristics, one solution to recover from their effects is to power off the system and restart it. However, in general it is difficult to detect when a transient fault is affecting our system, and at the same time, powering off or reseting a safety-critical system is not always possible.

One of the sources of transient faults are the voltage variations caused by the power supply. Voltage variations affect some of the transistors and can potentially modify their behavior. In 2000, the likelihood of having transient faults was low, less than one fault per year per thousand computers [16], making fault-tolerant computers attractive only for mission-critical applications, such as online transaction processing, space programs, transportation systems (e.g., railway, avionics, automotive), nuclear engineering, medicine, in other words, to the systems that are regarded as safety-critical. Since then, industry has evolved following Moore's Law, which states that the number of transistors in the same area is almost doubled every 2 years. As a consequence of this, the probability that one of the transistors suffers a transient fault has increased, not just because we doubled the number of transistors frequently, but also because other sources of transient faults, like capacitive crosstalk or inductive

crosstalk, have also become more likely with the shrinking of transistors size. For instance, new physical features of the transistors, which are smaller and usually operate at a lower operative voltage than the previous generation, make them more vulnerable to these and other transient faults.

Process variation must also be taken into account. Process variation is inherent to the manufacturing process and affects the attributes of transistors (lengths, widths, oxide thickness) [17]. The effect of the manufacturing variability becomes specially important as the size of transistors shrinks since relative variations increase and lead to increased fault rates. This kind of faults, once appear, remain during all the life of the system and for this reason they are called permanent faults.

Regardless of the type of fault, transient or permanent, processors need to have some mechanisms, that despite the costs (at least for the moment), protect safety-critical systems. Systems with the ability to operate correctly in spite of the presence of faults are referred to as fault-tolerant.

Fault tolerance requires at least two basic mechanisms: fault detection and recovery. Fault detection allows the construction of fail-stop components, which are components that, in case of fault, stop the propagation of the fault to avoid its propagation to the outputs and to other components. Fault detection also alerts the system that an error has been detected so the system can activate the appropriate recovery mechanism. Recovery mechanisms restore the execution from a point in which we ensure the system was error free or simply correct system state so that error-free operation can continue. These two mechanisms are the basis of highly reliable systems.

2.3 Safety Mechanisms for Safety Related Systems

2.3.1 Fault Detection

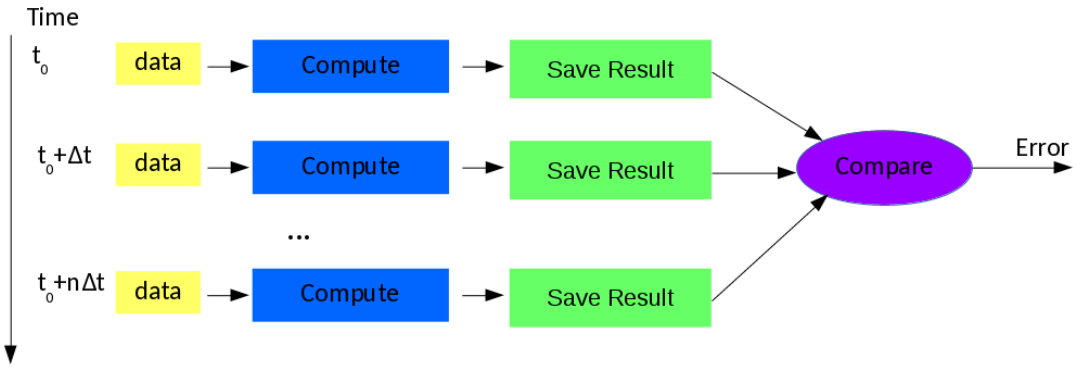
For fault detection mechanisms we have two main strategies: Hardware fault detection and Software fault detection. In general, hardware fault detection performs better than software fault detection by two main reasons. The first one is that hardware fault detection has a lower latency catching faults than software, using software the detection is delayed more and thus, it can occur that the erroneous outputs have already been propagated. The second is that hardware fault detection usually incurs

lower performance overhead compared to software, producing a lower performance degradation due to the fault detection mechanism.

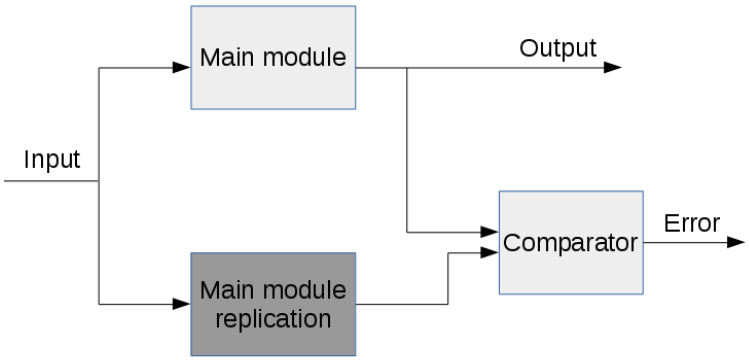
2.3.2 Redundancy, Sphere of Replication and Lockstep Execution

One of the most common hardware fault detection mechanisms is the use of redundancy. Redundancy means duplicating part of the resources of the system that will perform the same task in order to increase reliability. We have two different kinds of redundancy:

- Time redundancy
- Space redundancy



(A) Time redundancy



(B) Space redundancy

FIGURE 2.1: Examples of different redundancy

Time redundancy (2.1a) requires to execute the same operation (an instruction, a whole program) more than once at the same functional unit. **Space redundancy**(2.1b) on the other hand, is to have multiple functional units that execute the

same software/operation. After all executions are performed, either in time redundancy or space redundancy, a comparison is done with the outputs from each execution in order to find for potential errors. If results mismatch, it means that at least one execution has gone wrong, because in regular systems there is only one correct result. Note that redundancy can be complete or partial. For instance, for space redundancy one may replicate full pieces of data or codify properties of such data (e.g. parity). For time redundancy, one may execute an operation and sample the result twice from the output circuit instead of reexecuting the operation completely.

When a mismatch is detected, recovery actions need to be activated to restore the system from a safety point. Then, the system will re-execute, and repeat the process until no mismatch in the outputs is found. In that respect, redundancy highly relays on the probability that all the executions produce the same erroneous outputs is almost zero. For most of the cases this premise it is valid but, as we will show later, there are some cases in which this cannot be taken for granted.

Time redundancy is used mainly to protect against transient faults. Because transient faults can appear and disappear in time, the transient fault will unlikely be present during all the executions. In this context, the execution affected by a transient fault and the fault-free execution, or the execution with a different transient fault, will produce different outputs allowing the comparator to effectively detect a mismatch between executions. In the low probable case that the transient fault remains during all the executions, then time redundancy will fail only if, the transient faults produces exactly the same erroneous outputs in every execution. The protection against permanent faults is completely ineffective with time redundancy, because permanent faults will produce the same erroneous outputs for all the executions and the comparator will not detect the errors.

Space redundancy on the other hand, is able to detect the presence of errors caused by permanent faults, since it is very likely the permanent fault will only affect one of the executions, and thus, the fault will be effectively detected by the comparator. Transient faults can also appear, and like in the time redundancy, as long as they not affect all the executions (instances) in the same way, different outputs will be generated and the comparator will detect the mismatch.

Space redundancy increases area costs because we have to add more functional

units that do not add new functionalities, but since such functional units can execute in parallel the multiple required executions, the impact in performance is lower than for time redundancy, that needs to execute them sequentially. In most systems, replicating the full system is not needed.

The granularity at which redundancy is applied is known as the *Sphere of Replication* [18] or SoR for short. In the context of microprocessors for example, SoR can be defined at the instructions granularity so that the output of each instruction is compared before it is committed. However, this approach has a huge impact on performance since we will need to perform a comparison at every instruction, so very often. On the other hand, if the SoR is set at this granularity the recovery mechanism is much simpler and faster. In fact, the SoR presents a trade-off between the performance and the overhead that is determined by the granularity at which the replication is performed (see the Figure 2.2 extracted from [1]). Fine granularities will also reduce the amount of work that the recovery system will do since we will detect the fault earlier.

In a particular application of redundancy, some authors also mentioned the idea of using threads [18] instead of replicating the whole core and take advantage of some properties like locality in caches. With this redundancy consideration, replication is provided by the actual inherent redundancy of the system and so minimizes the required hardware overhead. However, because they use the same hardware, they are vulnerable to permanent faults. Moreover, the increase in the performance overhead must be considered with this approach.

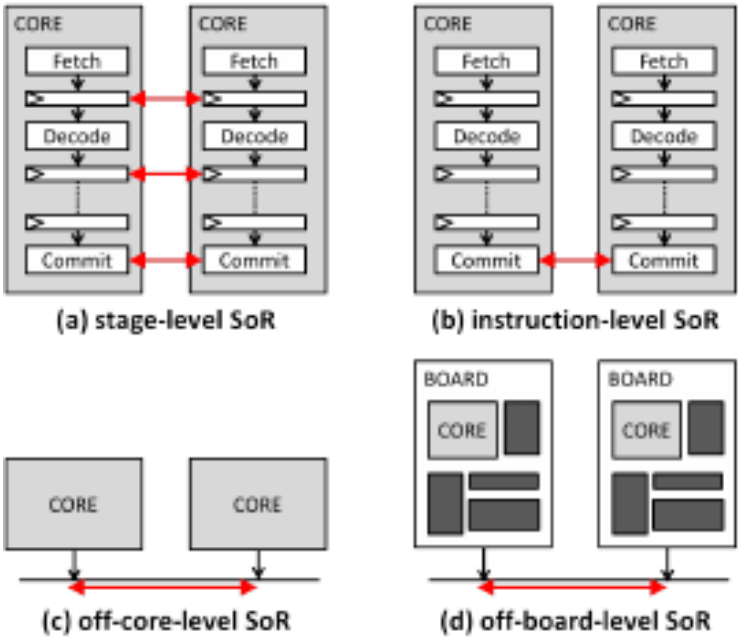


FIGURE 2.2: Schematics for different SoR granularities, [1]

2.3.3 Examples from Fault-Tolerant Platforms for Automotive Safety-Critical Applications

Lockstep execution is a space redundant fault-tolerant system used on processors, that is widely extended in safety critical systems and has been used in that industry for many years. For instance, the Infineon AURIX microcontroller [19], implements a dual-core lockstep processor to satisfy the reliability requirements of ASIL-D applications for the automotive domain. Following, we analyze different lockstep approaches.

Lockstep Dual Processor Architecture

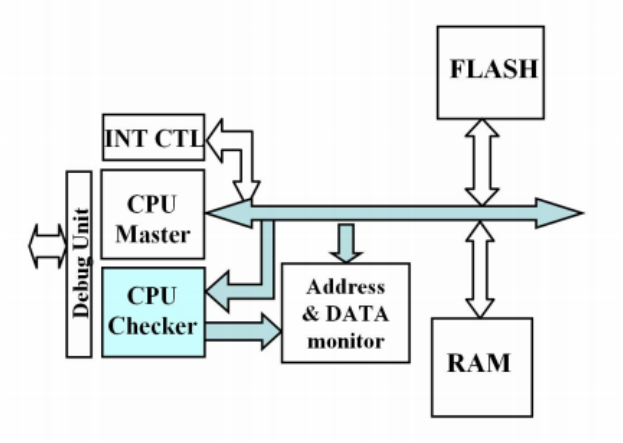


FIGURE 2.3: Lockstep dual processor architecture, [2]

In a dual lockstep architecture (figure 2.3) two processors usually referred as the *master* and the *checker*, execute the same code being strictly synchronized. The master has to access to the system memory and drives all the systems outputs while the checker continuously executes the instructions moving on the bus. The outputs of the checker, both addresses and data, feed the monitor, which is a comparator that checks the consistency of all the data and addresses, the outputs from the master are observed directly from the bus. When a mismatch appears on the comparator this indicates that at least one of the CPU's had a fault (fault detection). However, it is not possible to know which one executed correctly and which one had the fault, so the system has to re-execute again the same program piece. In this architecture the SoR is only the CPU which means the monitor is not capable of detecting bus and memory errors. Memory and bus must be protected using some form of error correction and detection codes. Lockstep architectures can be employed in safety critical system providing the capability of detecting any single errors, permanent or transient, occurring indifferently in one of the CPU's, the memory system or the communication channel.

Loosely-Synchronized Dual Processor Architecture

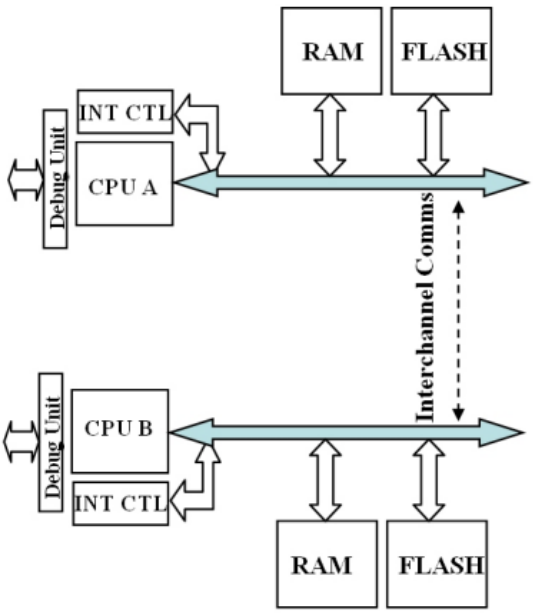


FIGURE 2.4: Loosely-Synchronized Dual Processor Architecture, [2]

In a loosely-synchronized architecture (Figure 2.4) two CPU's run independently having access to distinct memory system. A real-time operating system running on both CPU's handles the interprocess communication that is the responsible for checking and synchronizing the error detection. In this architecture the SoR is larger that in the previous architecture (subsection 2.3.3), which directly implies an increase in the cost of the system since we have to replicate more functional units and more area needed for the same functionalities.

The subset (or set) of tasks designed as critical are executed on both CPU's. The image is copied in both memories and the outputs are compared after each run on a time triggered basis. Each processor is responsible for keeping their memory consistency. The errors are detected with the comparison of values and prevent them to be sent them out of the system. We can also use what is called a *sanity-check*, which is to execute a predefined computation in order to find and identify a faulty component.

Triple Modular Redundancy System

Apart from the architectures we have seen, we can apply the same techniques: lockstep dual and loosely synchronized, but using 3 instances instead of 2. In this case we have

a majority voter component instead of a simple comparator. This component has to select the option that has more votes. Thus, if one of the instances has a fault, can be corrected in flight since the others are regarded as fault free. This reduces the overhead of the recovery system. In other words TMR increases performance. The fault coverage is also increased, as now we are able to detect two different faults on two different instances, that produce different outputs.

Using 3 instances instead of 2 has increased costs. However, we have also increased reliability since now the system can detect up to 2 errors. Furthermore, in the case of a single error we gain performance since the error can be corrected in flight avoiding costly re-executions. An example of this system is shown in Figure 2.5.

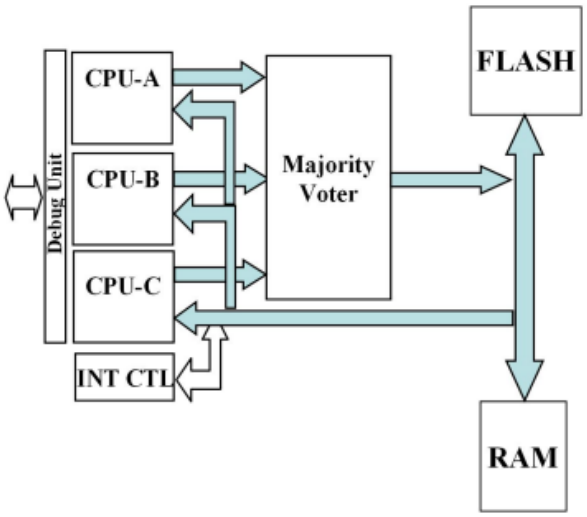


FIGURE 2.5: Triple modular redundancy, TMR [2]

We see that we can obtain new architectures by merging the ones we have seen so far and adding more instances. However, although the new architectures may add more protection, using more than three instances is not often employed because the costs of the final system are too high, compared to the obtained benefit.

2.4 Robustness Verification

Robustness verification has become one of the fundamental stages in the verification process for any new design. Verification for complex microcontrollers challenges product design cycles, what can lead to financial loss and severe delays if it is done at the

final production stages. Hence, designers have been striding to move this procedure towards the early stages of the design. Simulation-based verification has been shown to reduce costs associated with the robustness verification process as any misbehavior or defect can be corrected at earlier stages. Simulation-based verification can be carried out at different abstractions levels. Simulation at lower levels such as physical or gate level can be extremely time-consuming. With higher levels of abstraction like RTL, the burden is reduced but it is still overwhelming. Simulation-based verification using Instruction-Set Simulator (ISS) arises as one of the most promising approaches to reduce the timing constraint. However, modeling processors at higher levels reduces information required for more accuracy simulation. Fault injection is carried out at the level of instructions and smaller granularity faults, such as transistor or physical faults, cannot be modeled. Therefore, fault injections modify directly the instruction and become an error, since a modification in the instruction level cannot be masked. However, J. Espinosa et al. [20] show that correlation between fault injection at ISS and the RTL level exists for different permanent fault models.

The Architectural Vulnerability Factor (AVF), is a metric calculated in the robustness verification process to quantify the robustness of a processor architecture. Although the term is widely used in different applications, the methodology of how is calculated depends on the particularities of the employed architecture. Different methodologies have been proposed to calculate this value, the most used is assuming the AVF can be approached by the percentage of faults injected that lead to an error w.r.t. the total of faults injected.

2.5 Diversity

Design diversity has been proposed in the past to protect redundant systems against common cause faults [21]. Diversity is defined as achieving *different* implementations of the same hardware functionality. Those different implementations can be obtained at different abstraction levels. Different ways of implementing diversity are shown on the Figure 2.6.

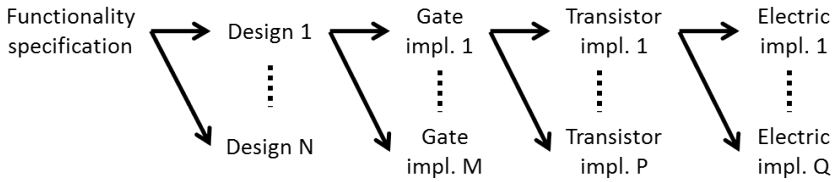


FIGURE 2.6: Different ways of achieving diversity for a given functionality, extracted from [3]

The first option on the figure will be the use of two circuit designs with completely different architectures. Given a multiplier we could implement it using Booth's multiplication algorithm or using multiply-accumulate algorithm. This option will have an enormous cost of design, because both designs must be developed independently and in different teams to ensure that the designs are completely diverse. However, the resulting designs may be very secure against common cause faults like voltage droops or crosstalk because the probability that one fault leads to exactly the same error is very low.

A second option could be the use of different gate implementations, for example by using two different libraries when performing the synthesis of the designs. Thus, we may avoid failures produced by process variation. We could try to use different transistor geometries or different electrical characteristics for each instance, which could avoid faults produced by voltage droops, or at least avoid producing the same errors.

Diversity, as explained, can be obtained by different methods, some will have a higher increase of the costs than others and each method will be specialized in offering more protection against specific fault types than the others. Despite the abstraction level, the objective is the same for all, use diversity in a lockstep system so when a transient or permanent common fault appears and affects all the instances, produces different error manifestations. With this premise the system will be able to detect a fault and perform the pertinent actions, achieving then a common fault tolerance, and not leading to a common cause failure. However, for other types of faults that only affect one of the instances, transient or permanent, diversity is not needed since the fault will only affect one of the outputs and a mismatch will be discovered at the comparison of the outputs. In those cases redundancy is enough to protect the system.

2.5.1 Other efforts to quantify diversity

To our knowledge, only one methodology has been proposed to quantify diversity [10, 11, 22] systematically. This methodology proposes an abstract diversity metric D that can be suited for any fault type. Mitra et al. define D as:

$$D = \sum_{f_i, f_j} P(f_i, f_j) d_{i,j}$$

where $d_{i,j}$ is the probability that both designs do not produce the same erroneous output when a fault is injected in each of them. $P(f_i, f_j)$ is the probability of having the fault f_i and f_j at each circuit instance. However, deriving fault probabilities for a given fault pair is not practical in the general case, if otherwise we consider that all faults are equally probable, then D can be computed systematically using the equation given in [10, 11, 22]:

$$D_{INJ} = \frac{1}{m} \sum_{i,j} d_{i,j}$$

m stands for the number of fault pairs injected in both instances and $d_{i,j}$ has the same value like in the first expression. D_{INJ} bases in injecting the same number of faults randomly in two non-identical instances. Then D_{INJ} represents how reliable are the two instances in front of random injected faults. But as we explained before, common cause failures are due to one fault. Thus, in our case we cannot apply this fault model. Also, if we consider two almost identical implementations which have both a 10% error probability when a fault occurs, D_{INJ} will obtain around 1% of the time errors in both instances. And will likely lead to different erroneous outputs, which will traduce in a D_{INJ} close to 0. In our model instead, we expect that when we apply the same fault, the 10% of the times will produce an error in both instances, and in those cases, very likely will lead to same erroneous outputs (because are almost identical). But following D_{INJ} one could conclude that those circuits are not diverse at all because D_{INJ} is close to 0. Here we must take into account the fact that 90% of the faults do not produce an error in both instances and thus, should not be taken into account when calculating diversity since it relates to the robustness of the circuits instead.

2.5.2 Main problem of diversity for the Industry

One may think the most important problem of *diversity* for the Industry, is the increase in terms of cost of deploying two diverse implementations. However, the main problem is that, despite the cost of deploying two diverse implementations, once we have both implementations side to side, there is no mechanism or method to measure how (quantitatively) diverse those implementations are.

We have explained some mechanisms that can lead to diverse designs, like creating two different teams that work independently in each implementation, subcontract two different companies to build the same components, but the result of these methods cannot be quantified. Therefore the formulation of the problem is that, despite the metric explained on Section 2.5.1, which we have seen that do not measure exactly diversity, there is not any other mechanism to *quantify* diversity *quantitatively*, thus Industry cannot validate the diversity of their systems.

The target of this thesis is trying to mitigate the impact that common cause failures have in the system. Common cause failures are caused by a single fault, also called common-mode faults, that affects all the instances of a lockstep execution. This faults can be for example voltage droops, crosstalk, systematic process variation etc. The errors produced by these faults are difficult to detect because in most of the cases they produce the same error for all the instances, specially if the instances are just replicas. Thus, causing that the comparator/monitor cannot observe any difference because in fact there is not, although all results can be potentially wrong. Also most of the faults are transient, so even with the *sanity-checks* we are not able to detect them.

Chapter 3

DIMP: A DIversity Metric based on circuit Path analysis

In this chapter we present DIMP a metric proposed to quantify diversity in a pair of designs. We start by explaining the rationale behind DIMP algorithm. Then, we detail our implementation of DIMP and how we evaluate it. For the evaluation we use 3 different implementations of ISCAS'89 circuits and applied our DIMP implementation. For comparison purposes we also compute D_{INJ} to these circuits. Finally, we validate DIMP using diverse implementations of an 8-bit full adder and other representative circuits and using SPICE simulations.

3.1 Motivation of DIMP

Using space redundancy, as shown before, protects systems against transient and permanent faults. However, protection against the particular case of common cause faults is not provided. Common cause faults can either be permanent or transient, but the characteristic that defines them is that affect multiple instances inside our redundant system. Therefore, if instances inside the SoR are affected simultaneously and identically, redundant elements will produce the same erroneous outputs and the output comparator will not detect the errors, leading to a system failure.

In order to protect systems against common cause faults, diversity must be used across the different instances inside the SoR. However, there is no mechanism able to calculate diversity across circuits systematically. In this master thesis we propose

DIMP to cover this gap and offer a systematic mechanism to accurately quantify diversity and thus, improving system robustness against common cause faults.

3.2 Rationale behind DIMP

DIMP is built on the idea that lack of diversity occurs when starting a signal from an input pin (I_i) and traversing through the path until we reach an output pin (O_j), the devices that the signal goes through are *similar*. Therefore, in order to quantify diversity we must consider the devices that are traversed from a given input to a given output, also called a path. Comparison must be done on paths that have the same input and output since they are in fact carrying out the same functionality.

Once all irrelevant parts of the circuit have been filtered out and the devices traversed to get from I_i to O_j kept, we are able to perform the comparison. Before that, we must consider to put a weight for each path to reflect the *importance* of each path in the manifestation of the given fault. For example if we consider the timing faults, longer paths can be more susceptible than shorter paths. Then one may consider that the shorter paths are less likely to induce erroneous outputs and thus, be less relevant than longer paths. In that case we can put a small weight in paths that are shorter than a given threshold for instance.

3.3 A realization of DIMP

```
1  DIMP = 0
2  MaxDIMP = 0
3  For i = 1 to N
4    For j = 1 to M
5      Paths^1_{i,j} = ⋃ ( Paths ( I^1_i , O^1_j ) )
6      Paths^2_{i,j} = ⋃ ( Paths ( I^2_i , O^2_j ) )
7      While Paths^1_{i,j} ≠ ∅ and Paths^2_{i,j} ≠ ∅ do
8        Take p^1_k ∈ Paths^1_{i,j} and p^2_l ∈ Paths^2_{i,j}
                 with highest overlap(p^1_k, p^2_l)
9        DIMP = DIMP + weight(p^1_k, p^2_l) · (1 − overlap(p^1_k, p^2_l))
10       MaxDIMP = MaxDIMP + weight(p^1_k, p^2_l)
11       Remove p^1_k from Paths^1_{i,j}
12       Remove p^2_l from Paths^2_{i,j}
13     Endwhile
14     RemainingPaths_{i,j} = Paths^1_{i,j} ∪ Paths^2_{i,j}
15     While RemainingPaths_{i,j} ≠ ∅ do
16       Take any p_q ∈ RemainingPaths_{i,j}
17       DIMP = DIMP + weight(p_q, ∅)
18       MaxDIMP = MaxDIMP + weight(p_q, ∅)
19       Remove p_q from RemainingPaths_{i,j}
20     Endwhile
21   Endfor
22 Endfor
23 Return DIMP / MaxDIMP
```

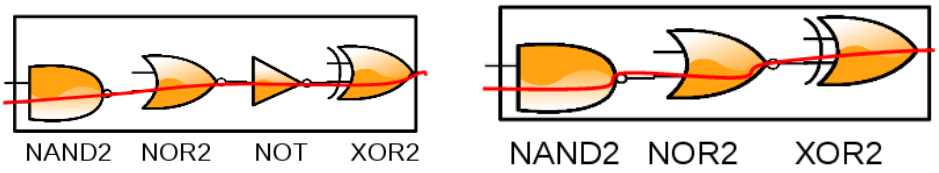
FIGURE 3.1: Pseudocode of a realization of DIMP.

Now we describe a particular realization of DIMP that we have implemented. This realization is intended for fault types like voltage droops, residual faults escaping diagnosis coverage and permanent faults in the root of the power network of the circuits instances. Pseudocode of the realization can be seen on figure 3.1.

First, we initialize the two main values, $DIMP$ and $MaxDIMP$ to 0. As we will iterate over the designs, $DIMP$ will be the diversity value, while $MaxDIMP$ will be the maximum value of diversity possible. The loops at lines 3 and 4, stand for the number of inputs (N) and the number of outputs, (M). First we will group all the paths for each pair $< I_i, O_j >$ separately, lines 5 and 6. Since we work at gate level, the devices correspond to all gates that traverse from the input to the output. Then $Paths^1_{i,j}$, stands for all the paths from circuit 1 that traverse from input i to output j, and each path consists in an ordered list of gates.

Once all the paths are grouped in sets, we will iterate over all the sets, line 7. Given a pair $< I_i, O_j >$ we will have paths from circuit 1 and paths from circuit 2, we will select the pair of paths, one for each circuit, that have the highest overlapping. We define $overlap(p_k^1, p_l^2)$ as the number of gates that repeat across paths in the same order, even if some other gates are interleaved. This is a problem known as LCS [23] (*Longest Common Subsequence problem*). We divide the value by the total number of gates in both paths, otherwise the longer paths will easily have high values instead of shorter paths that are similar.

For instance if $p_k^1 = \{NAND2, NOR2, NOT, XOR2\}$ and $p_l^2 = \{NAND2, NOR2, XOR2\}$, see figure 3.2 then $\{NAND2, NOR2, XOR2\}$ will be the longest common subsequence and the $overlap(p_k^1, p_l^2) = 6/7$, since 6 gates repeat in both paths out of 7 total gates. Other definitions of $overlap(p_k^1, p_l^2)$ are possible.



(A) Example p_k^1 (B) Example p_l^2

FIGURE 3.2: Small example: $\{NAND2,\ NOR2,\ XOR2\}$ is the solution of LCS

Because $DIMP$ variable is the diversity value, we increase $DIMP$ with the non-overlap (see line 9), which is subtracting from 1 the value of the overlapping, and then we multiply it with relative weight ($weight(p_k^1, p_l^2)$) of those paths. Since we target timing we define $weight(p_k^1, p_l^2)$ as the maximum gate count of those paths. Other considerations of weights can be used according to the fault model being used. $MaxDIMP$ is also increased with the relative weight (line 10), but as we explained $MaxDIMP$ is the maximum potential value of diversity, so accordingly, we have 0 overlapping in all the cases. Then we remove the paths considered from the sets in lines 11 and 12.

We continue evaluating and subtracting paths from the sets until one of the sets is empty. If the other set is still not empty, we consider the rest of the paths of the non-empty set as 0 overlapping (fully diverse) paths because we cannot compare against any other path from the other circuit. Once we iterate over all the sets, we

will have the $DIMP$ and $MaxDIMP$ value computed. At line 23 we finally return the diversity value as the division of the diversity of the circuits ($DIMP$) divided by the maximum potential diversity ($MaxDIMP$).

Notice that in the case of two identical circuits the overlapping will be 1 in all the cases, thus leading to a value of $DIMP = 0$ and leading to diversity 0. On the other hand having two completely different circuits (e.g., one implemented with NOR gates and the other with NAND gates) will have a 0 overlapping in all the cases which will lead to $DIMP = MaxDIMP$ and resulting in a diversity of 1.

3.4 DIMP Implementation

Our diversity algorithm takes as inputs all the paths for a given circuit. Thus, we need first to create a tool that creates the input for diversity evaluation.

3.4.1 Inputs tool

First, we create a small Python scripting tool that analyzes a circuit descriptor file in VHDL and lists all the paths of the circuit. Note that we require the VHDL implementation of the circuit to define a gate-level implementation of the circuit or a netlist. The program reads from an input file in VHDL and identifies for all gates their inputs and outputs. Then, for each input net of the circuit we create a N-ary tree, in which their nodes are the gates. Next, we traverse the tree at the same time we are building it. When we arrive to an output pin, which is a leaf, we write the path and continue traversing the tree. The format in which the paths are listed is the following:

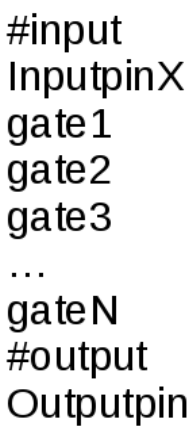


FIGURE 3.3: Format of the file listing all paths

3.4.2 Diversity implementation

We have implemented the algorithm show in section 3.3 in C++. Once we have read the input file that contains all the paths that is computed by the developed Python scripts, we iterate over all the input-output pairs. For each set of paths of an input-output pair, we create a matrix in which the columns represent the paths from circuit 1 and rows represent the paths from circuit 2. Each cell contains the "overlap" of both paths. Since each path contains a list of the gates, we have used Hunt and Szymanski algorithm [24] to obtain the Longest Common Sequence (LCS) value. We have modified the code from [25], to change the granularity of the solution since we consider as the minimum element the gate names and the code considers the letters.

Therefore, cells of the matrix contain the overlapping value of two paths. Then, we search for the highest overlapping and perform the increase of DIMP and MaxDIMP values accordingly. Since we already have computed all the overlapping values, we simply deactivate the rows and columns used, and search for the highest overlapping again until we either use all columns, all rows or both. In the case that one of the designs has more paths in a particular set (non-square matrix), once the other design runs out of paths, the remaining paths will be considered as fully diverse.

We repeat this process for all input-output pairs and, finally, we perform the division of DIMP and MaxDIMP values to obtain the diversity value.

3.5 DIMP evaluation

To test our metric, we have two different sets of circuits, the first ones are obtained from the ISCAS'89 bench suite [26]. We have used a synthesizing tool to obtain different implementations, we used three synthesis optimizations for different delay targets, being $0.3ns$ the most demanding and $1.0ns$ the most relaxed constraint. We have also used different designs implementations for an 8-bit full adder which we will use for illustration purposes.

For all the circuits considered we have computed the diversity using the metric proposed in [10] and our DIMP metric. To compute D_{INJ} we have followed the methodology described in [20] since its computation requires injecting faults. For fault-injection we have used FALLES Fault Injector tool that we describe in the next section. For each experiment we have injected randomly stuck-at faults at gate-level and explored all the input combinations. For each input combination we have injected 10,000 different stuck-at faults, and we compare the outputs against the golden run. Due to the size of the circuits, we cannot perform this experiment in all the ISCAS'89 circuits. This is in fact a limitation of the D_{INJ} metric that is not suffered by DIMP.

We have also used the SPICE [27] tool, to compute the diversity w.r.t. voltage droops applying it to the different implementations of the 8-bit full adder. We inspected the outputs of the different implementations while we modify the voltage, reproducing voltage droops, and we analyzed if they produce the same error manifestations or not.

3.5.1 FALLES Fault Injector

In order to inject faults in our circuit implementations, we have used the FALLES Fault Injector jointly developed by BSC and the UPV [20]. FALLES Fault Injector comprises a set of scripts that enable injecting faults into a VHDL model and simulate it to observe the consequences using the Modelsim or Questasim simulators. Faults are injected by means of simmulation commands. The injection and the posterior analysis phase are automated using scripts, that can be highly personalized. FALLES supports different fault models – both permanent and transient faults are supported. It is built around the ModelSim Simulator [28] environment.

3.5.2 ISCAS evaluation

To test DIMP, we have implemented ISCAS circuit using a synthesis tool with three different constraints: $0.3ns$, $0.6ns$ and $1.0ns$ to obtain three diverse implementations of the same design circuit. On Table 3.1 we can see the results of applying D_{INJ} to the different ISCAS'89 circuits implementations. Each column shows the results of applying the mentioned metric between two of the three implementations using FALLES Fault Injector.

	0.3 vs 0.6	0.3 vs 1.0	0.6 vs 1.0
s1196	0.999945	0.999998	0.999986
s1238	0.999999	0.999999	0.999999
s1488	0.998937	0.999332	0.998904
s1494	0.999998	1	0.999962
s27	1	1	1
s298	0.996409	0.999571	0.996084
s382	1	0.998712	1
s386	0.999878	1	0.99947
s400	0.99855	0.998784	0.9988
s444	1	0.999234	1
s953	1	1	1

TABLE 3.1: Diversity results computed injecting random faults (D_{INJ}) using FALLES Fault Injector.

All results are very close to 1, this would mean that all the implementations are very diverse among them no matter what synthesizing constraint we have used. However, after we analyzed in detail the different implementations, we can observe that the paths that are not time critical are quite similar, and we have small synthesizing optimizations in those paths. Therefore we should not expect values so close to 1 for all the cases.

This occurs because D_{INJ} is suitable for random faults and the probability that two random faults affecting to each of the circuit instances produce the same erroneous outputs is almost zero. In fact, when this metric is computed systematically for random faults it measures how robust a circuit is w.r.t. random faults rather than how diverse two given circuits are. Thus, if we apply this metric to 2 almost identical circuits, 2 random faults will rarely produce the same erroneous outputs in both circuit

instances, so $D_{INJ} \approx 1$. However, the actual diversity of those circuits would be close to 0.

Results of applying DIMP to the same subset of ISCAS'89 than D_{INJ} are shown on table 3.2. Results of all ISCAS'89 circuits are on appendix A. In this case we observe diversity values have ranges from 0.6 to 1, which better represents the differences across the different implementations.

	0.3 vs 0.6	0.3 vs 1.0	0.6 vs 1.0
s1196	0.87885	0.990901	0.989536
s1238	0.881473	0.980223	0.977516
s1488	0.898724	0.951677	0.815758
s1494	0.884235	0.98916	0.973529
s27	1	0.949367	1
s298	0.77582	0.991486	0.981879
s382	0.678801	0.873572	0.901327
s386	0.933984	0.960784	0.791227
s400	0.60297	0.880554	0.880322
s444	0.601725	0.879051	0.827288
s953	0.625419	0.892768	0.885153

TABLE 3.2: DIMP's results on ISCAS'89 circuits.

We observe a particular behavior in almost all the cases, the diversity values have the higher values on the pair $0.3ns$ vs $1.0ns$. This is an expected behavior since in those circuits the constraints applied are the ones more different, one is the more demanding ($0.3ns$) and the other the more relaxed ($1.0ns$) optimization. However, there are two circuits which do not follow this pattern: s382 and s27. We have inspected the gate implementations of these two circuits and have realized that the utilization of some gates in some paths produces some cascade effects on the automatic choice of other gates in the same path. Those other gates, therefore, may repeat across implementations targeting $0.3ns$ and $1.0ns$, but not for the implementations targeting $0.6ns$. Analogously, lower diversity values are in almost all the cases, in the pair $0.3ns$ and $0.6ns$. For most of the circuits, the delay without optimizations is close to $0.7ns$. This results in applying similar timing optimizations in both $0.3ns$ and $0.6ns$, which end up using the same gates relatively often. For the $1.0ns$ target,

instead, the optimizations focus on reducing power since most of them already pass the timing constraint, thus leading to more diverse implementations w.r.t. the other targets.

3.5.3 8-bit full adder designs

We have used 4 different implementations of an 8-bit full adder design. First, we have implemented an 8-bit full-adder made of 8 1-bit full adders implemented with NAND gates (FA-NAND). In order to have another implementation fully diverse we have replicated the FA-NAND but replacing all NAND gates with an AND gate followed by a NOT gate, we called this design as FA-AND. Then we have built a full adder based on half adders (which use OR, AND and XOR gates) that we called FA-2. Finally, we mixed the FA-NAND and the FA-AND in the sense that we use 4 bits with the components of FA-NAND and 4 bits with the components of FA-AND, we called this design as FA-MIX.

On the next figures we can see the designs for 1 bit of the different full adder designs. As explained, FA-MIX uses both AND (figure 3.4) and NAND (figure 3.5).

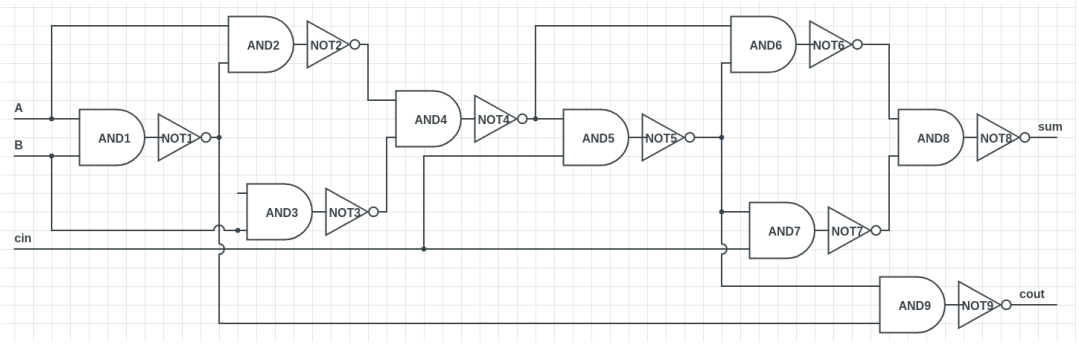


FIGURE 3.4: 1-bit full adder design for FA-AND and FA-MIX

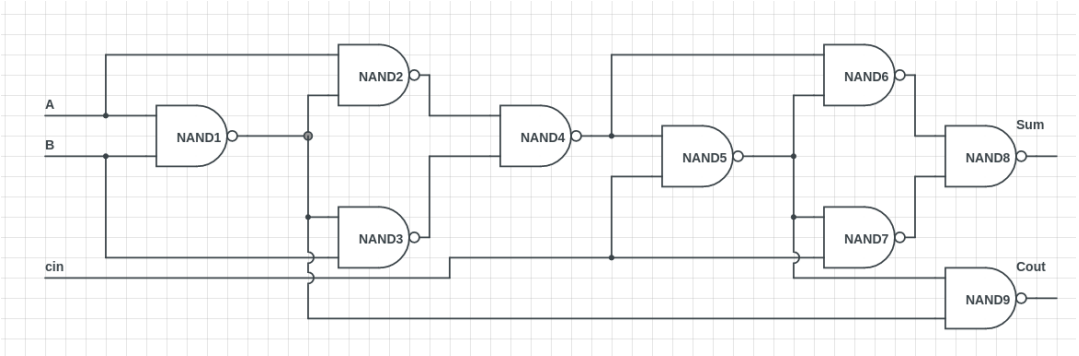


FIGURE 3.5: 1-bit full adder design, using only NAND gates, for FA-NAND and FA-MIX

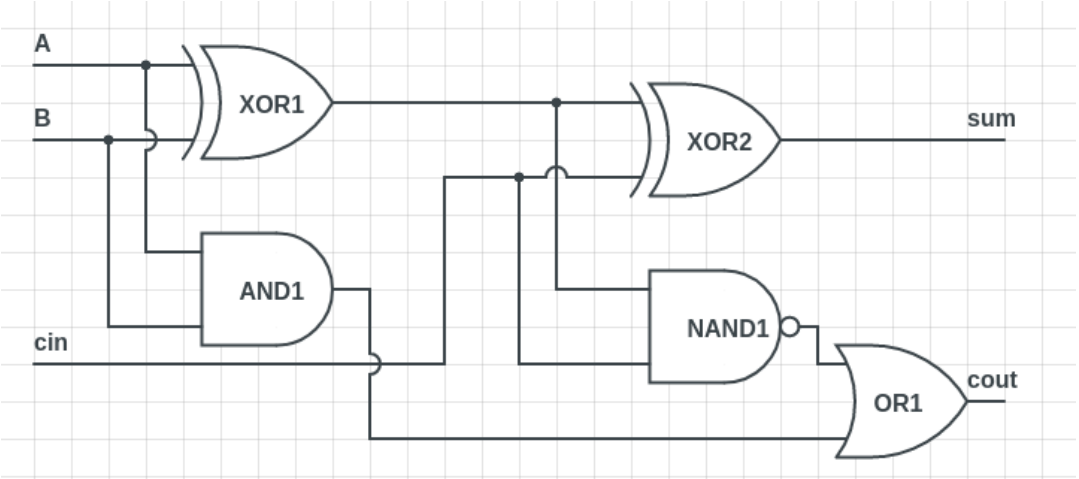


FIGURE 3.6: 1-bit full adder design, based on half adders, used only in FA-2

3.5.4 SPICE validation

SPICE

SPICE (*Simulation Program with Integrated Circuit Emphasis*) is a general-purpose analog electronic circuit simulator. It is used in order to check the integrity of circuits designs in integrated circuits and/or board-level design and to predict circuit behavior. There have been three main stable releases, SPICE1 SPICE2 and SPICE3. The first and the second are written in Fortran while the last one is written in C. In our experiments we have used the third version, in particular the HSPICE Synopsys tool. We have used SPICE to observe the behavior of the circuits when we decrease the operating voltage and specially how the outputs are affected.

Validation

Results of applying DIMP to the full adder circuits are shown in the next table.

	FA-NAND vs FA-AND	FA-NAND vs FA-2	FA-NAND vs FA-MIX	FA-AND vs FA-2
DIMP	1	1	0.41	0.96

TABLE 3.3: DIMP's results applied to full adders circuits.

Pairs FA-NAND / FA-2 and FA-NAND / FA-AND have both maximum diversity according to DIMP. This is naturally correct based on how these circuits are designed, since both pairs do not have any gate in common.

Instead, the pair FA-AND / FA-2 has diversity less than 1 according to DIMP. Again, if we look at the 1-bit designs, in this case the AND gate appears in both designs. Thus, we should expect a value below 1. However, the designs are very different and therefore, the DIMP value is very close to 1. The last pair is the most similar according to DIMP. FA-NAND and FA-MIX, have the same circuits for 4 of their 8 bits adders. Thus, makes sense that DIMP diversity is the lowest of all pairs.

We used SPICE to simulate the full-adders designs and try to reproduce voltage droops. The experiments consisted in downgrading the nominal voltage, which is $1V$, to $0.6V$ where a large fraction of errors was expected and observe the effects produced on the outputs values. With SPICE simulations, we compute the percentage of outputs providing different results for different circuit pair combinations. Note that this metric is different from the one provided by DIMP. A value 1 represents the maximum diversity, 0 represents either that the circuits are identical or that the outputs provided by the circuits are all correct. However, the results will help us understanding if DIMP is able to capture the diversity of the error manifestations caused by voltage droops.

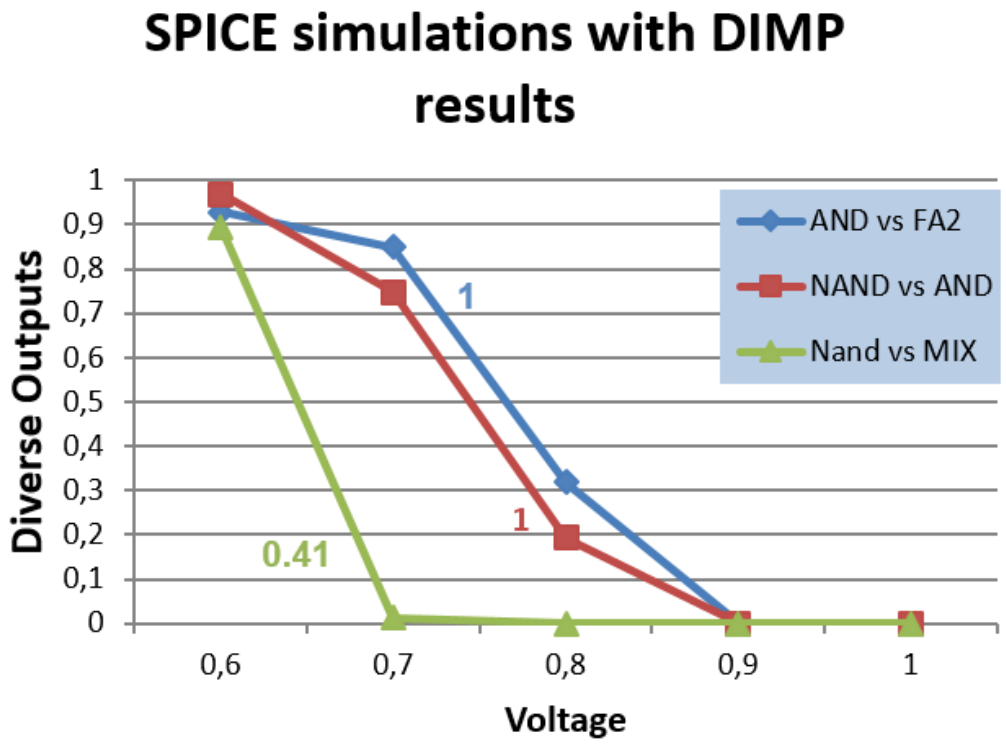


FIGURE 3.7: Format of the input circuits descriptions

In figure 3.7 we can see the results of the SPICE simulation for the different pairs we have considered for DIMP with the DIMP values next to them. As expected all pairs have a value of 0 on the nominal voltage ($1V$), because at nominal voltage all the circuits produce correct results. When decreasing the nominal voltage, we see that the pairs FA-AND vs FA-2 and FA-NAND vs FA-AND start to produce diverse erroneous output at $0.8V$. Those are the pairs that, according to DIMP, have more diversity and in this figure we confirm that the topological diversity translates in diverse error manifestation. For the case of the pair FA-NAND vs FA-MIX, until $0.7V$ either they produce correct results or they produce the same erroneous outputs. It is the behavior that DIMP predicted for this pair.

3.5.5 Computation complexity

An important property of our metric is that it can be computed systematically and that its computational costs are low. In order to test the computational power required to compute DIMP, we use different synthetic circuits with different sizes and observe the amount of time it takes to compute their diversity. We have used synthetic circuits to understand the scalability of our metric and validate that the circuits assessed with SPICE require a complexity quite close to the one computed for the synthetic circuits when they are tailored to the specific size.

Since design size and complexity are not simple to measure, we modify different parameters to observe how our algorithm behaves. The parameters that we considered are: the number of paths, the sizes of the paths and the number of inputs and outputs. We created a tool that creates synthetic circuits based on these parameters. Figure 3.8 shows the amount of time our implementation took for computing DIMP for the different synthetic circuits. The complexity is higher when number of paths and their sizes are increased and the number of inputs and outputs are low. However, computational costs are in general low, and only for some circuit configurations computation take up few hours. We have confirmed also this with the ISCAS circuits where only 2 circuits from this suite required more than one hour. Instead, the computation of D_{INJ} may require several days of computation for some of the circuits. The reason for this huge computational cost is explained by the fact that D_{INJ} requires performing exhaustive fault-injection campaigns. Although a computational cost of few hours is

not negligible, note that diversity needs to be quantified only once for each pair of diverse implementations during product development and thus, such cost seems to be affordable. For larger circuits, in order to reduce timing, we could simply do the quantification at different granularity: instead of quantifying the whole designs, we could use it on the different functional unit blocks (e.g., ALU, Register files). Therefore, using this methodology designs could be evaluated considering their components and also the multiple fine-grained quantifications could be done in parallel.

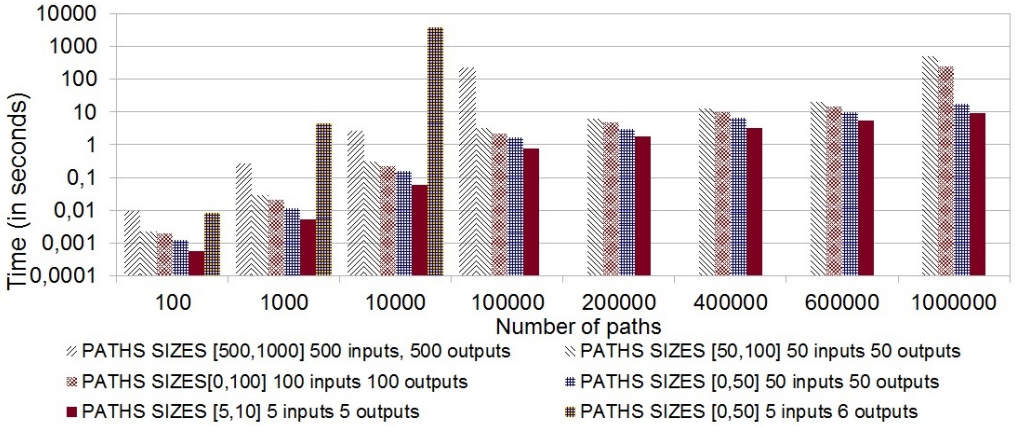


FIGURE 3.8: Computational complexity. Square brackets show the range of the depth of the paths

3.6 Conclusions

Lockstep execution is often used in safety-related microcontrollers to meet the safety requirements for the highest integrity levels. However, although lockstep has been proven efficient to detect independent faults across the redundant cores, it is unable to detect those faults that lead to a common cause failures unless diverse designs are used. Therefore, industry requires a mechanism to quantify systematically diversity across designs.

This thesis tackles this challenge by providing DIMP, a low-cost metric to quantify diversity based on circuit path analysis. DIMP can be suited to different abstraction levels (e.g., gate level, transistor level) and different fault types. We provide a specific realization of DIMP suited to gate level and timing faults. Our evaluation shows the effectiveness and low cost of DIMP to quantify diversity and the validation using

SPICE simulator shows that the metric results correlate with common cause failures of the 8-bit full adder simulations.

Chapter 4

Specialization of DIMP

In this chapter we tailor DIMP to a specific common cause fault. In particular, we have tailored it to faults caused by systematic process variation. We explain all the modifications that are required to tailor the DIMP metric and the evaluation of this specification. However, we will show that despite our efforts, we could not correlate the results of the new metric with the ones obtained using fault injection. We analyze the reason of this mismatch and show how it can be improved.

4.1 Considered Fault Model

Process variation is an inherent consequence of the manufacturing process and the inability of such process in producing exact copies of all circuit elements due to physical limitations and other imperfections. Process variation affects the attributes of transistors (lengths, widths, oxide thickness) and wires (length, width) when integrated circuits are fabricated. These manufacturing deviations can slightly or significantly modify the behavior of the circuit by the alteration of transistor and wire electrical properties under some stress circumstances.

To understand how DIMP behaves when we tailor this metric to the faults caused by manufacturing deviations, we have to define an appropriate fault model. For this model, we consider some gates are more vulnerable to certain conditions – such as magnetic fields or voltage droops – than others due to the impact of systematic process variations. Therefore, during the simulations we assume that only some gates are affected by faults while all other parts of the design are fault free. To model this, we will only inject faults to the selected gates.

As system model we again assume a lockstep design in which both instances are within the SoR. Therefore, system failure will occur when both instances generate the same erroneous outputs.

4.1.1 Tailoring DIMP to the specific fault model

We can easily adapt DIMP to the new specific fault model. In fact, we make DIMP remain almost the same as shown in section 3, but imposing a restriction when we create the sets of paths. Since we know that the probability of some gates to be faulty is zero for the specific fault model, we now consider only the paths that contain at least one gate of the gates regarded as more likely to fail due to process variations. Thus, the specialization will ignore parts of the circuit and will focus only on the paths that contain the selected gate or set of gates. We can see the new restriction in the algorithm at line 5 and 6 of Figure 4.1.

1 $DIMP = 0$
2 $MaxDIMP = 0$
3 **For** $i = 1$ **to** N
4 **For** $j = 1$ **to** M
5 $Paths^1_{i,j} = \bigcup \left(Paths \left(I^1_i, O^1_j \right) \mid Filtered\_Gate \in Paths \left(I^1_i, O^1_j \right) \right)$
6 $Paths^2_{i,j} = \bigcup \left(Paths \left(I^2_i, O^2_j \right) \mid Filtered\_Gate \in Paths \left(I^2_i, O^2_j \right) \right)$
7 **While** $Paths^1_{i,j} \neq \emptyset$ **and** $Paths^2_{i,j} \neq \emptyset$ **do**
8 Take $p^1_k \in Paths^1_{i,j}$ and $p^2_l \in Paths^2_{i,j}$
with highest $overlap(p^1_k, p^2_l)$
9 $DIMP = DIMP + weight(p^1_k, p^2_l) \cdot (1 - overlap(p^1_k, p^2_l))$
10 $MaxDIMP = MaxDIMP + weight(p^1_k, p^2_l)$
11 Remove p^1_k from $Paths^1_{i,j}$
12 Remove p^2_l from $Paths^2_{i,j}$
13 **Endwhile**
14 $RemainingPaths_{i,j} = Paths^1_{i,j} \cup Paths^2_{i,j}$
15 **While** $RemainingPaths_{i,j} \neq \emptyset$ **do**
16 Take any $p_q \in RemainingPaths_{i,j}$
17 $DIMP = DIMP + weight(p_q, \emptyset)$
18 $MaxDIMP = MaxDIMP + weight(p_q, \emptyset)$
19 Remove p_q from $RemainingPaths_{i,j}$
20 **Endwhile**
21 **Endfor**
22 **Endfor**
23 Return $\frac{DIMP}{MaxDIMP}$

FIGURE 4.1: Pseudocode of the modification for specialized DIMP.

4.1.2 Gate Selection

The new model requires a gate selection of the designs to decide which gates we select. The rationale behind this is that, due to process variability, some gates may be more vulnerable than others since they have different sizes and electrical properties, and this will also translate in a systematically higher fault probability. Note that a specific type of gate is not only determined by the logic function implemented, but also by its strength. The strength of a gate is associated to its ability to switch a given capacitance and relates to the size of the transistors used to build it. To model this, we create a tool that lists the gates of a design and their number of occurrences. Using this tool, we noticed that most of the gates appear only once in the designs. Since we want to have a representative number of occurrences but still want to select just one gate type, we simply decided to choose the gate with more instances for each design. We also considered using the two most occurring gates for two circuits in order to observe how the metric will behave with more gate representatives. For these two circuits, we performed 3 different experiments, one using both gates and two using one of the gates. Table 4.1 summarizes the gates selected for each circuit. Our working set is a subset of the ISCAS'89 circuits.

Circuit		**0.3**		**0.6**		**1.0**	
	Gate Filtered	Num Gates	% of total gates	Num Gates	% of total gates	Num Gates	% of total gates
s1196	**ND2M2W**	28	4.90%	44	9.57%	38	12.62%
	INVM2W	28	4.90%	48	10.43%	22	7.31%
	both	56	9.81%	92	20.00%	60	19.93%
s1238	**ND2M2W**	39	6.29%	59	12.04%	34	10.97%
	INVM2W	30	4.84%	26	5.31%	34	10.97%
	both	69	11.13%	85	17.35%	68	21.94%
s1488	**ND2M2W**	56	10.67%	47	9.29%	30	8.98%
s1494	**ND2M2W**	51	8.89%	23	4.47%	28	7.98%
s386	**NR2M2W**	8	7.14%	7	6.25%	11	15.49%

TABLE 4.1: Selected gate for each circuit and percentage respect to the total

4.2 Evaluation

Our fault model differs from the one in Section 3, since now we consider another particular case of common mode fault, systematic defects caused by process variations.

Therefore, we need to use a different evaluation that can be less generic and more precise to the particular fault model.

4.2.1 FALLES Fault Injector

We have used the FALLES Fault Injector for the validation. To do so, we have modified some of the parameters. In particular, we have modified the fault injection to make it being able to choose between injecting always a stuck-at 1 fault or a stuck-at 0, and forcing the faults to be injected at the beginning of the simulation to have a more predictable behavior that we can use to validate DIMP.

4.2.2 Matching mechanism

FALLES Fault Injector is the tool in charge of performing the fault injection, simulating the consequences and finally analyzing the results. The analysis phase consists in comparing the results of fault-injection with the Golden Run[1]. Then, we will have a huge file reporting all the erroneous outputs of the design.

According to our system model, we must match the cases where a common cause failure appears. This is when both designs created the same erroneous outputs with the same conditions (same input values). Therefore, we will extend the FALLES Fault Injector tool with a new phase which is the comparison between two experiments reports.

Due to the size of both inputs files, there is heavy work to process them. However, since one of the restrictions of the matching is that errors must appear under the same input values, for a given error, we only need to consider those errors that were produced with the same input values. Thus, one easy way to parallelize the work consists of splitting the errors based on the input values producing them.

In order to perform this task, we created a C++ tool that runs on a parallel cluster. The program receives 2 analysis reports from the analysis phase of FALLES Fault Injector, and replicates the files for each worker thread. Then, using MPI (Message Passing Interface) as a parallel programming model, the master thread informs the workers about the errors they must match (by giving them the input values). Matching

[1]Golden Run is an execution without errors. In this context it is used to be compared with other executions to find errors.

is then done in parallel by all the workers: they read their input files and select only the errors assigned. Then, they perform the matching and once they are done, they send back a message to the master with the information of how many matches they found and the total number of errors considered. The master collects information from all the workers and calculates the final value. An overview of the software pipeline is shown in Figure 4.2.

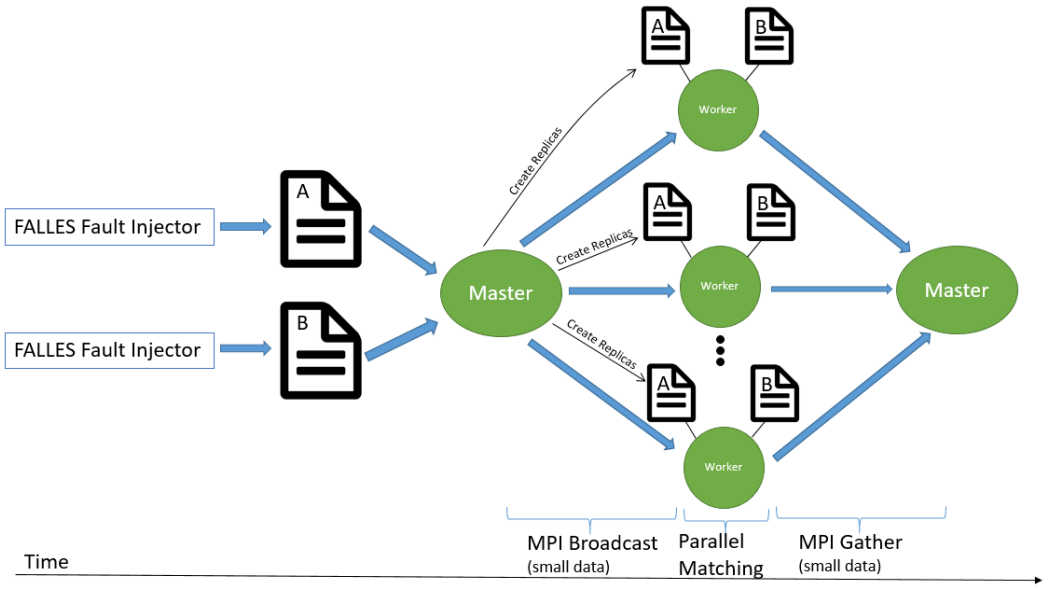


FIGURE 4.2: Software pipeline of the matching mechanism.

4.2.3 ISCAS evaluation

For this experiments we cannot use all ISCAS'89 circuits because we are evaluating all states, instead of doing it sampling a subset of them in a statistical manner. This makes the amount of experiments required to perform increase a lot with the number of inputs. Therefore, we reduce the working set to a subset of the ISCAS'89 circuits. Table 4.2 shows the ISCAS circuits we have employed for this fault injection campaign.

ISCAS'89 Circuit	Inputs	Outputs
s1196	16	14
s1238	16	14
s1488	10	19
s1494	10	19
s386	9	7

TABLE 4.2: ISCAS'89 working set

Because we model a lockstep design, the evaluation consists in quantify the number of cases that faults appearing in both designs could lead to a system failure. Following the fault model, the faults will be injected on both instances of a specified gate. Since we simulate each design separately, after the injection phase, we perform a matching that pairs the cases that having the same input values lead to the same erroneous output. Finally, we calculate the fraction between the errors paired against all the errors produced.

Simulation consists in performing an injection at the beginning of the simulation. The injection we have used is a stuck-at 1 fault. Then, we vary the input values until we cover all the possible inputs (2^n cases, n number of inputs). This is the reason why we cannot use all the circuits from ISCAS'89. In order to let signals propagate through the circuit, we wait for a small amount of time between each input change ($20ns$). We repeat all the process for each input value and for all the possible injection points of the design.

Once the simulation is finished, the analyzer component of the FALLES Fault Injector reports the errors found compared to the Golden Run. Since the file contains the injections done on all the possible injection points, we must filter all the injections that were not done in the instances of the selected gates. With the filter process we eliminate some of the unnecessary data reported by the FALLES Fault Injector to reduce the size of the files. Note that due to the high amount of required injections, reducing the amount of tracked information is crucial to keep the problem tractable. In fact we only save for each error: the timestamp of the error, the input values, and the erroneous output value.

After the previous process, we run our matching tool on a parallel cluster to perform the matching. Because our model is a lockstep design, we will match those errors that have the same input and output values, which are the ones that can lead to a common cause failure. In order to parallelize the process, we divide the amount of inputs between the number of workers, so each worker has to analyze the errors of $\frac{2^n}{w}$ inputs, where w is the number of workers and n the number of inputs of the design. Notice that there is no work balancing since we do not know a priori the number of errors reported for each input. Even with the parallelization, this final step can take up 2 days, mostly due to the large file sizes.

After the matching phase, we are ready to compute the final value. The value consists in a rate between the number of errors that we matched divided by the total number of errors produced. This can be seen as an approximation of the AVF on our model considering we divide the number of errors that can lead to the system failure by all the errors produced by the injections.

CIRCUIT + FILTER		Generic DIMP	Specialized DIMP	Matching percentage
s1196				
	0.3∼0.6	0.879962	0.699909	63.25%
Gate: ND2M2W	0.3∼1.0	0.989662	0.987574	60.88%
	0.6∼1.0	0.987646	0.972088	62.26%
	0.3∼0.6	0.879962	0.596522	71.30%
Gate: INVM2W	0.3∼1.0	0.989662	0.964201	60.74%
	0.6∼1.0	0.987646	0.935969	63.31%
	0.3∼0.6	0.879962	0.667136	70.13%
Gate: INVM2W + ND2M2W	0.3∼1.0	0.989662	0.983005	68.19%
	0.6∼1.0	0.987646	0.967317	68.60%
s1238				
	0.3∼0.6	0.881975	0.529878	39.42%
Gate: ND2M2W	0.3∼1.0	0.979199	0.956967	34.43%
	0.6∼1.0	0.976734	0.936694	62.73%
	0.3∼0.6	0.881975	0.628794	64.98%
Gate: INVM2W	0.3∼1.0	0.979199	0.982548	52.54%
	0.6∼1.0	0.976734	0.977272	55.70%
	0.3∼0.6	0.881975	0.573767	61.13%
Gate: INVM2W + ND2M2W	0.3∼1.0	0.979199	0.968309	54.56%
	0.6∼1.0	0.976734	0.957104	65.82%
s1488				
	0.3∼0.6	0.76609	0.673546	43.95%
Gate: ND2M2W	0.3∼1.0	0.988728	0.901524	34.48%
	0.6∼1.0	0.976887	0.794531	39.41%
s1494				
	0.3∼0.6	0.838476	0.880282	32.33%
Gate: ND2M2W	0.3∼1.0	0.952615	0.923454	35.39%
	0.6∼1.0	0.936673	0.932814	39.81%
s386				
	0.3∼0.6	0.821197	0.71831	58.31%
Gate: NR2M2W	0.3∼1.0	0.968137	0.879121	55.43%
	0.6∼1.0	0.935917	0.820628	62.78%

TABLE 4.3: Results from Generic DIMP, Specialized DIMP and the Matching percentage from the fault injection

Results

A summary of the results obtained with the experiments can be seen in Table 4.3. We have divided the results for each circuit and gate filtered. For each comparison we have the values for the generic DIMP, which are the same shown on Section 3, plus the values for the specialized DIMP and the matching percentage from the fault injection. From the results we expect having the highest percentage of matching for the circuits that are less diverse. Additionally, we expect the specialized DIMP results to be more precise than the regular DIMP metric and follow the mentioned trend more clearly.

In general, the results of Specialized DIMP still follow the pattern seen in the generic DIMP metric, which is that the lowest values are on the 0.3 *vs* 0.6 pairs and the highest on 0.3 *vs* 1.0 pairs. We believe that the reason of why both metrics have similar results is due to the selection of gates: since we selected the most representative gates of the circuit, results are similar. Notice that the values of the 0.3 *vs* 0.6 pairs are lower than on the generic DIMP, because now we consider paths that at least have one gate in common, the selected one. Therefore, overlapping will never be 0 on those paths, except when we run out of paths from one of the circuits and the other one still has unmatched paths.

As mentioned before, one should expect matching percentages to be higher when the DIMP values are close to 0, since we have lower DIMP values when gates across paths are "similar" in both designs. More precisely, in the case of specialized DIMP, paths that contain the potentially erroneous gates are the ones analyzed and the rest of the design is ignored. Considering we know in advance the potentially erroneous gates, we know also, that the paths analyzed with specialized DIMP are the ones that will traverse the erroneous signal. Therefore, lower values of specialized DIMP will indicate that the mentioned paths are very similar and thus, will potentially have similar erroneous outputs.

It is important to remark that matching percentages measured are very high for all circuits (lowest value is above 30%). Thus, that means that at least 1 out of every 3 injections can be matched and potentially lead to a system failure. This is a very interesting result and can give us an estimation of how important is having diverse implementations to face common case faults since the percentage of matching

in identical designs will be even higher.

In summary, we cannot generally see a clear correlation between DIMP, neither the generic or the specialized, and the matching percentage. Only in three cases, (s1196-INVM2W, s1488-ND2M2W and s1238-INVM2W) the behavior is the one expected. In that respect, in the next section we try to understand the reason for this and propose different approaches to improve the results.

4.3 Removing the aliasing

As shown before, our metric does not fit the needs of the specified fault model. Therefore, we need to reconsider all the steps taken and try to identify the reason why we are obtaining such results. In general, it can be either that specialized DIMP is not well suited for the particular fault model or that experiments are not representative enough, or both.

If we start considering that the specialized DIMP algorithm is wrong, this still cannot explain per se why the generic DIMP, which we have already shown to behave correctly, cannot also correlate with the results. Although the regular DIMP implementation is more generic, we expect at least some degree of correlation, but this is not the case. Furthermore, the results of the matching seem to have different patterns for each circuit. Thus, we consider that in our results we are having some form of noise or aliasing that makes results not to be consistent across circuits.

Surprisingly, matching percentages are huge, being 32% the lowest value. This means that almost 1 out of 3 errors can potentially cause a system failure. We believe that this is again because of the aliasing since we are comparing faults that are injected at gates belonging to different paths and, therefore, DIMP quantification is not considering that effect. Note that DIMP metric is computed on a per-path basis and therefore, matching errors originated at different paths can only contribute to introducing noise to the obtained results.

4.3.1 Modifications on the matching

In order to remove this aliasing, we introduce a new filter or constraint in the matching process. In fact, we impose to the matching of errors, that injections belong to gates

that are in the same path. Now, when we apply the filter after the FALLES Fault Injector tool, we must save the path in which the injection was made to be able to filter out this result. With this new restriction we expect the noise within the matching percentages to be reduced and having better correlation for the specialized DIMP.

CIRCUIT + FILTER		Generic DIMP	Specialized DIMP	Matching percentage
s1196				
	0.3∼0.6	0.879962	0.699909	1.36%
Gate: ND2M2W	0.3∼1.0	0.989662	0.987574	0.23%
	0.6∼1.0	0.987646	0.972088	1.61%
	0.3∼0.6	0.879962	0.596522	1.04%
Gate: INVM2W	0.3∼1.0	0.989662	0.964201	0.06%
	0.6∼1.0	0.987646	0.935969	1.03%
	0.3∼0.6	0.879962	0.667136	0.97%
Gate: INVM2W + ND2M2W	0.3∼1.0	0.989662	0.983005	0.23%
	0.6∼1.0	0.987646	0.967317	2.61%
s1238				
	0.3∼0.6	0.881975	0.529878	0.66%
Gate: ND2M2W	0.3∼1.0	0.979199	0.956967	7.26%
	0.6∼1.0	0.976734	0.936694	3.67%
	0.3∼0.6	0.881975	0.628794	0.15%
Gate: INVM2W	0.3∼1.0	0.979199	0.982548	0.27%
	0.6∼1.0	0.976734	0.977272	0.17%
	0.3∼0.6	0.881975	0.573767	0.26%
Gate: INVM2W + ND2M2W	0.3∼1.0	0.979199	0.968309	0.11%
	0.6∼1.0	0.976734	0.957104	0.18%
s1488				
	0.3∼0.6	0.76609	0.673546	3.37%
Gate: ND2M2W	0.3∼1.0	0.988728	0.901524	3.08%
	0.6∼1.0	0.976887	0.794531	5.43%
s1494				
	0.3∼0.6	0.838476	0.880282	0.13%
Gate: ND2M2W	0.3∼1.0	0.952615	0.923454	2.90%
	0.6∼1.0	0.936673	0.932814	1.61%
s386				
	0.3∼0.6	0.821197	0.71831	12.56%
Gate: NR2M2W	0.3∼1.0	0.968137	0.879121	0.22%
	0.6∼1.0	0.935917	0.820628	0.74%

TABLE 4.4: Results after the matching modifications

Results

Looking at the results shown in Table 4.4, the first thing we see is that, as expected, the new restriction has reduced the matching percentages significantly, being now 12.56% the highest value among all the circuits. However, in spite of that, results are still inconsistent across different circuits. The expected behavior can only be seen in 3 out of 9 experiments (s1196-INVM2W, s1238-INVM2W ND2M2W and s386-NR2M2W), which means that, despite we have removed the inter-paths aliasing, we cannot match the specialized DIMP with the our expectations yet.

We have also evaluated some new experiments where we used two gates at the same time. Unfortunately, these results are also inconsistent between them, since only one of them matches with the expected behavior.

4.4 Conclusions

Lockstep is widely used in safety-related microcontrollers to meet the standard requirements of highest integrity levels. In this chapter, after validating the generic DIMP for the case of timing faults, we tried to specialize it in order to focus to a certain systematic fault type. In particular, we have targeted the impact of systematic process variation since these faults are becoming more relevant as transistors become smaller. Therefore, system failures due to this kind of faults will continue increasing while we continue reducing transistor's geometry and increasing their count.

We have developed a set of mechanisms in order to evaluate our metric, that consisted in a selection gate tool, a filter for the outputs of the FALLES Fault Injector and a parallel matching program. Despite our efforts in finding a correlation between the specialized DIMP and the percentage of matching reported by our evaluation method, we have failed to show the expected correlation. We have deeply investigated the reasons for that, but we think that they are related to the following causes. For the generic DIMP, diversity is computed on a per path basis and the particular fault-model we have employed in this chapter is not suitable for this restriction. In that respect, we have specialized DIMP to be able to filter on a per-path basis and also failed on showing the expected behavior. Our intuition here is that after filtering out so many gates – the ones not belonging to the path and the ones not having a specified

gate – and also having in mind that the circuits that we were able to evaluate are the smallest ones, the remaining parts of the circuits are too small to provide any meaningful result. We plan to perform modifications to our evaluation method to solve that limitation. However, we leave this as future work.

Chapter 5

Conclusions and Future Work

Reliability is one of the most important design constraints in safety-critical systems. In this thesis we have focused in the utilization of lockstep execution as one of the most relevant existing safety mechanisms to protect designs. Lockstep execution is a particular implementation of space redundancy that is very suitable to protect processors and microcontrollers. Lockstep designs provide protection against permanent faults and soft errors (transient faults) that appear in one of the instances. Unfortunately, lockstep processors cannot protect against common cause faults without the use of diversity since they will provide identical error manifestations in both core replicas.

Diversity can be intuitively achieved by different methods but there is still not a method able to quantifying it on a systematic manner. Even in the automotive standards (ISO26262: 1.28) that require processor designers to employ diversity, there is a lack of concretion on how to quantify it and its requirements stick to: *different solutions satisfying the same requirement with the aim of independence (1.61)* [29]. The need for diversity in the safety critical systems industry calls for a metric that helps clarifying to what extent the diversity requirements have been fulfilled for a particular design implementation.

In this thesis we have made an effort in this direction and presented DIMP, a systematic method to quantify diversity, which is based on a circuit path analysis. DIMP can be suited to different abstraction levels and can also be used at different granularities (e.g., full adders, full designs). Still, there is a long path until there is an automated quantification of diversity in the safety critical systems, but we think that DIMP is a first step towards it.

We have seen that DIMP matches the expectations when used on smaller and

controlled design implementations (full adders). More generically, we have also show that DIMP is well suited to the case of timing-failures validating this by means of voltage droops simulations using SPICE. There are steps that need to be done together with the industry such as carrying out experiments with industrial-relevant examples or integrating diversity quantification as a part of the safety standards such as ISO26262 in the automotive domain to further understand the benefits of DIMP, but our preliminary findings are promising, as acknowledged with our publication in the automotive track of a top conference (Design Automotion Conference).

Furthermore, we have tried to extend DIMP to cover other fault models with the purpose of understanding to what extent DIMP is able to quantify diversity w.r.t. systematic fault patterns. In particular, we selected the systematic process variation as the root of the faults to model and considered that process variations cause some specific gates to be more prone to errors than others.

However, although we carried out a huge and exhaustive number of experiments for which we had to develop different tools, we have observed that DIMP is not yet well suited to specific fault models like systematic faults caused by process variations. We have deeply analyzed the reasons why the DIMP metric fails to provide meaningful diversity quantification results for that particular fault model and tried to solve this by proposing a specialization of DIMP tailored to this particular case. Unfortunately, we were not able to prove the expected correlation between the diversity values computed and the percentage of errors that can potentially produce the same erroneous outputs. There is still work that can be done to try to improve results such as using fewer specific gates, increasing the size of the designs to increase the representativeness of the results, and/or use DIMP at different granularities, but all this work will be done as future work and beyond the scope of this thesis.

An important observation we can make after the thorough evaluation performed in this thesis, is that DIMP works well for fault models affecting specific paths of a circuit. This is for example the case of timing-faults. Timing faults are only relevant for the path or set of paths that are close to the timing properties of the critical path. The reason for this is that DIMP quantifies diversity on a per-path basis and thus, searches for similarities across paths. However, some other type of faults, like the ones affecting some gates systematically, are not necessarily captured properly with DIMP

since several faulty gates not belonging to the same path can contribute to a given erroneous output. We let as future work the corroboration of this hypothesis.

Bibliography

[1] C. Hernandez and J. Abella, "Live: Timely error detection in light-lockstep safety critical systems," in *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, (New York, NY, USA), pp. 25:1–25:6, ACM, 2014.

[2] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, and S. Pezzini, "Fault-tolerant platforms for automotive safety-critical applications," in *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '03, (New York, NY, USA), pp. 170–177, ACM, 2003.

[3] S. Alcaide, C. Hernandez, A. Roca, and J. Abella, "DIMP: A low-Cost Diversity Metric based on circuit Path analysis," in *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, (New York, NY, USA), pp. 45:1–45:6, ACM, 2017.

[4] *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[5] RTCA and EUROCAE, *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[6] CENELEC, *EN50126. Railway Applications: The Specification and Demonstration of Dependability, Reliability, Availability, Maintainability and Safety (RAMS)*, 2012.

[7] International Electrotechnical Commission, *IEC61508. Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2010.

[8] Freescale Semiconductor, "Qorivva MPC5643L microcontroller data sheet. rev. 9," 2013.

[9] Infineon, *Tricore 1. 32-bit Unified Processor Core v1.3*, October 2005.

[10] S. Mitra, N. Saxena, and E. McCluskey, "A design diversity metric and analysis of redundant systems," *IEEE Transactions on Computers*, vol. 51, no. 5, 2002.

[11] S. Mitra, N. Saxena, and E. McCluskey, "Techniques for estimation of design diversity for combinational logic circuits," in *DSN*, 2001.

[12] R. Charette, "This car runs on code," in *IEEE Spectrum online*, 2009.

[13] P. Clarke, "Automotive chip content growing fast, says gartner," 2010. http://www.eetimes.com/electronics-news/4207377/Automotive-chip-content-growing-fast.

[14] G. Edelin, "Embedded systems at thales: the artemis challenges for an industrial group," in *Presentation at the ARTIST Summer School in Europe 2009*, 2009.

[15] J. Bowen, "The ethics of safety-critical systems," *Commun. ACM*, vol. 43, pp. 91–97, Apr. 2000.

[16] A. Wood, "Data integrity concepts, features, and technology," *White paper, Tandem Division, Compaq Computer Corporation*, 1999.

[17] S. Mittal, "A survey of architectural techniques for managing process variation," vol. 48, 02 2016.

[18] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, (New York, NY, USA), pp. 25–36, ACM, 2000.

[19] Infineon, "AURIX Multicore 32-bit Microcontroller Family to Meet Safety and Powertrain Requirements of Upcoming Vehicle Generations." http://www.infineon.com/cms/en/about-infineon/press/press-releases/2012/INFATV201205-040.html.

[20] J. Espinosa, C. Hernandez, J. Abella, D. de Andres, and J. C. Ruiz, "Analysis and rtl correlation of instruction set simulators for automotive microcontroller robustness verification," in *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, (New York, NY, USA), pp. 40:1–40:6, ACM, 2015.

[21] R. A. Ashraf, O. Mouri, R. Jadaa, and R. F. Demara, "Design-for-diversity for improved fault-tolerance of tmr systems on fpgas," in *2011 International Conference on Reconfigurable Computing and FPGAs*, pp. 99–104, Nov 2011.

[22] S. Mitra, N. R. Saxena, and E. J. McCluskey, "Efficient design diversity estimation for combinational circuits," *IEEE Trans. Comput.*, vol. 53, pp. 1483–1492, Nov. 2004.

[23] Wikipedia, "Longest common subsequence problem — wikipedia, the free encyclopedia," 2017. [Online; accessed 20-October-2017].

[24] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Commun. ACM*, vol. 20, pp. 350–353, May 1977.

[25] "Longest common subsequence," 2017. https://rosettacode.org/wiki/Longest_common_subsequence#C.2B.2B.

[26] F. Brglez, D. Bryan, and K. Kozminski, *Combinational profiles of sequential benchmark circuits*, 1989.

[27] L. W. Nagel and D. Pederson, "Spice (simulation program with integrated circuit emphasis)," Tech. Rep. UCB/ERL M382, EECS Department, University of California, Berkeley, Apr 1973.

[28] I. Corporation, "Modelsim *- Intel FPGA," 2017. https://www.altera.com/products/design-software/model---simulation/modelsim-altera-software.html.

[29] ISO, "ISO26262 - Road vehicles — Functional safety — Part 1: Vocabulary," 2011. [Online; accessed 1-January-2018].

Acronyms

V&V Validation and Verification. 1

ASIL Automotive Safety Integrity Level. 1, 2

SEU Single Event Upset. 4

SoR Sphere of Replication. 8–11, 18, 33

TMR Triple Modular Redundancy. 12

AVF Architectural Vulnerability Factor. 14, 37

RTL Register-Transfer Level. 14

ISS Instruction-Set Simulator. 14

DIMP Diversity Metric based on circuit Path analysis. 18–20, 23, 25, 26, 28–32

SPICE Simulation Program with Integrated Circuits Emphasis. 18, 24, 28, 29, 31

LCS Longest common subsequence problem. 21, 23

MPI Message Passage Interface. 35

Appendix A

DIMP on ISCAS'89

	0.3 vs 0.6	0.3 vs 1.0	0.6 vs 1.0
s1196	0.87885	0.990901	0.989536
s1238	0.881473	0.980223	0.977516
s13207	0.712577	0.878367	0.903798
s1423	0.781312	0.933486	0.929485
s1488	0.898724	0.951677	0.815758
s1494	0.884235	0.98916	0.973529
s15850	0.902511	0.949534	0.698047
s208_1	0.739288	0.928969	0.851585
s27	1	0.949367	1
s298	0.77582	0.991486	0.981879
s344	0.597537	0.912125	0.929464
s349	0.664762	0.934632	0.917834
s35932	0.728343	0.897606	0.881384
s382	0.678801	0.873572	0.901327
s38417	0.793578	0.96827	0.869858
s38584	0.724989	0.845901	0.813881
s386	0.933984	0.960784	0.791227
s400	0.60297	0.880554	0.880322
s420_1	0.996923	0.999903	0.991359
s444	0.601725	0.879051	0.827288
s510	0.596014	0.917991	0.943923
s526	0.556871	0.888151	0.880187
s526n	0.571667	0.898126	0.910305
s5378	0.885166	0.963412	0.893173
s641	0.789491	0.878476	0.892225
s713	0.872436	0.955024	0.901885
s820	0.895561	0.976798	0.945946
s832	0.965325	0.966387	0.927405
s838_1	0.965755	0.999818	0.980561
s9234_1	0.568182	0.728395	0.582278
s953	0.625419	0.892768	0.885153

TABLE A.1: DIMP results for ISCAS'89 circuits

Appendix B

Published Work

DIMP: A Low-Cost Diversity Metric based on Circuit Path Analysis

Sergi Alcaide, Carles Hernandez, Antoni Roca, and Jaume Abella, in Proceedings of the 54^{th} Annual Design Automation Conference 2017, DAC '17, (New York, NY, USA), pp. 45:1–45:6, ACM, 2017.

Abstract presented in ACACES Summer School 2017. Fiuggi (Italy), July 10-14.

POSTER: DIMP: A Low-Cost Diversity Metric based on Circuit Path Analysis

Sergi Alcaide, Carles Hernandez, Antoni Roca, and Jaume Abella

ACACES Summer School Poster Session 2017,

Fiuggi (Italy), July 12th