Frederico Ferlini

METHODOLOGY TO ACCELERATE DIAGNOSTIC COVERAGE ASSESSMENT: MADC

Tese submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina para a obtenção do Grau de Doutor em Engenharia Elétrica
Orientador: Prof. Dr. Eduardo Augusto Bezerra
Coorientador: Prof. Dr. Djones Vinicius Lettnin

Florianópolis
2016

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Ferlini, Frederico
METHODOLOGY TO ACCELERATE DIAGNOSTIC COVERAGE ASSESSMENT: MADC / Frederico Ferlini ; orientador, Eduardo Augusto Bezerra ; coorientador, Djones Vinicius Lettnin. - Florianópolis, SC, 2016.
135 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia Elétrica.

Inclui referências

1. Engenharia Elétrica. 2. Fault Injection. 3. Fault Emulation. 4. Functional Safety . 5. ISO 26262. I. Bezerra, Eduardo Augusto. II. Lettnin, Djones Vinicius. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia Elétrica. IV. Título.

Frederico Ferlini

METHODOLOGY TO ACCELERATE DIAGNOSTIC COVERAGE ASSESSMENT: MADC

Esta Tese foi julgada adequada para obtenção do Título de Doutor em Engenharia Elétrica, e aprovada em sua forma final pelo Programa em Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.

Florianópolis, 23 de Setembro de 2016

Prof. Marcelo Lobo Heldwein, Dr.
Universidade Federal de Santa Catarina
(*Coordenador do Curso*)

Banca Examinadora:

Prof. Alberto Bosio, Dr.
Université de Montpellier
(Videoconferência)

Prof. Luigi Dillilo, Dr.
Université de Montpellier
(Videoconferência)

Prof. Rui Policarpo Duarte, Dr.
Instituto Superior Técnico
(Videoconferência)

Prof. Fernando Rangel de Sousa, Dr.
Universidade Federal de Santa Catarina

Prof. Hector Pettenghi Roldán, Dr.
Universidade Federal de Santa Catarina

Prof. Eduardo Augusto Bezerra, Dr.
Universidade Federal de Santa Catarina
(*Orientador*)

I dedicate this work
to my beloved wife.

ACKNOWLEDGMENTS

I would like to acknowledge the tremendous support of my beloved wife. I also would like to express my gratitude to my family and friends that always believed in me. Last but not least, I would like to thank my advisors which I am highly indebted for their help thus enabling this.

It always seems
impossible until
it’s done.
(Nelson Mandela)

RESUMO

Os veículos da atualidade vêm integrando um número crescente de eletrônica embarcada, com o objetivo de permitir uma experiência mais segura aos motoristas. Logo, a garantia da segurança física é um requisito que precisa ser observada por completo durante o processo de desenvolvimento. O padrão ISO 26262 provê medidas para garantir que esses requisitos não sejam negligenciados. Injeção de falhas é fortemente recomendada quando da verificação do funcionamento dos mecanismos de segurança implementados, assim como sua capacidade de cobertura associada ao diagnóstico de falhas existentes. A análise exaustiva não é obrigatória, mas evidências de que o máximo esforço foi feito para acurar a cobertura de diagnóstico precisam ser apresentadas, principalmente durante a avalição dos níveis de segurança associados a arquitetura implementada em hardware. Estes níveis dão suporte às alegações de que o projeto obedece às métricas de segurança da integridade física exigida em aplicações automotivas. Os níveis de integridade variam de A à D, sendo este último o mais rigoroso. Essa Tese explora o estado-da-arte em soluções de verificação, e tem por objetivo construir uma metodologia que permita acelerar a verificação da cobertura de diagnóstico alcançado. Diferentemente de outras técnicas voltadas à aceleração de injeção de falhas, a metodologia proposta utiliza uma plataforma de hardware dedicada à verificação, com o intuito de maximizar o desempenho relativo a simulação de falhas. Muitos aspectos relativos a ISO 26262 são observados de forma que a presente contribuição possa ser apreciada no segmento automotivo. Por fim, uma arquitetura OpenRISC é utilizada para confirmar os resultados alcançados com essa solução proposta pertencente ao estado-da-arte.

Palavras-chave: Injeção de Falhas. ISO 26262. Integridade Funcional. Cobertura de Diagnóstico. Plataforma de Hardware dedicada à Verificação. Mecanismos de Segurança.

ABSTRACT

Modern vehicles are integrating a growing number of electronics to provide a safer experience for the driver. Therefore, safety is a non-negotiable requirement that must be considered through the vehicle development process. The ISO 26262 standard provides guidance to ensure that such requirements are implemented. Fault injection is highly recommended for the functional verification of safety mechanisms or to evaluate their diagnostic coverage capability. An exhaustive analysis is not required, but evidence of best effort through the diagnostic coverage assessment needs to be provided when performing quantitative evaluation of hardware architectural metrics. These metrics support that the automotive safety integrity level – ranging from A (lowest) to D (strictest) levels – was obeyed. This thesis explores the most advanced verification solutions in order to build a methodology to accelerate the diagnostic coverage assessment. Different from similar techniques for fault injection acceleration, the proposed methodology does not require any modification of the design model to enable acceleration. Many functional safety requisites in the ISO 26262 are considered thus allowing the contribution presented to be a suitable solution for the automotive segment. An OpenRISC architecture is used to confirm the results achieved by this state-of-the-art solution.

Keywords: Fault Injection. ISO 26262. Functional Safety. Diagnostic Coverage. Hardware-assisted Verification Platform. Safety Mechanism.

RESUMO EXPANDIDO

INTRODUÇÃO

Os veículos da atualidade vêm integrando um número crescente de eletrônica embarcada, com o objetivo de permitir uma experiência mais segura aos motoristas. Logo, a garantia da segurança física é um requisito que não pode ser negligenciado e o padrão ISO 26262 provê medidas para garantir que este seja observado durante o processo de desenvolvimento de sistemas embarcados para automóveis de produção em série.

Entre essas medidas, a injeção de falhas é fortemente recomendada pelo ISO 26262 para verificar o correto funcionamento de mecanismos de segurança implementados no circuito e de sua capacidade de cobertura referente ao diagnóstico de falhas existentes. A análise exaustiva não é obrigatória, mas evidências de que o máximo esforço foi praticado para acurar a cobertura de diagnóstico precisam ser fornecidas. Isso se aplica, especialmente, durante a avalição dos níveis de segurança associados a arquitetura implementada em hardware. Estes níveis, denominados ASIL (do inglês, *Automotive Safety Integrity Level*), dão suporte às alegações de que o projeto obedece as métricas de segurança da integridade física especificada de acordo com a aplicação automotiva alvo. Os ASILs são definidos no ISO 26262 e vão de A à D, ou seja, do mais brando até o mais rigoroso, respectivamente.

Essa pesquisa explora o estado-da-arte em soluções de verificação visando construir uma metodologia baseada na injeção de falhas que permita acelerar a verificação da cobertura de diagnóstico alcançada. Diferentemente de outras técnicas voltadas à aceleração de injeção de falhas, a metodologia proposta MADC (*Methodology To Accelerate Diagnostic Coverage Assessment*) utiliza uma plataforma de hardware dedicada à verificação para maximizar o desempenho da inserção de falhas. Muitos aspectos do ISO 26262 são observados de forma que a original contribuição deste trabalho possa ser apreciada no segmento automotivo. Por fim, resultados obtidos da aplicação da MADC sobre uma arquitetura OpenRISC são apresentada.

OBJETIVOS

As evidências utilizadas para dar suporte ao se aclamar que um produto é *functional safety* precisam ser coletadas sobre o efetivo projeto do circuito integrado em desenvolvimento. Quando isso não é feito, então um conjunto de argumentos robustos que são necessários para justificar a utilização de um modelo abstrato como representante do real projeto do circuito em desenvolvimento. A geração desse tipo de argumentação pode exigir significante esforço para comprovar que há suficiente correlação entre o modelo efetivo do circuito e o modelo abstrato usado para análise. Visando reduzir esse oneroso esforço adicional, esse trabalho propõe MADC para acelerar a campanha de injeção de falhas sem que seja necessário qualquer modificação no modelo do circuito ou no nível de abstração utilizado. Antes de entrar em mais detalhes referentes a MADC proposta, o escopo dessa pesquisa precisa ser definido. Portanto, injeção de falhas nesse trabalho significa a imitação do efeito de falhas em circuitos integrados descritos a nível de portas lógicas.

O principal objetivo dessa pesquisa é ***propor uma metodologia que explora ao máximo as vantagens encontradas nas mais avançadas soluções de verificação funcional para acelerar a injeção de falhas visando verificar os mecanismos de segurança implementados assim como sua capacidade de diagnosticar falhas respeitando os preceitos do ISO 26262***. Para alcançar esses objetivos, os seguintes caminhos foram identificados:

- Investigar soluções de verificação funcional modernas que possam ser empregadas na aceleração de injeção de falhas;
- Familiarizar-se com os padrões mais atuais sobre *functional safety* de modo que suas recomendações fossem observadas pela metodologia de injeção de falhas desenvolvida;
- Construir um modelo prático visando conferir o desempenho da metodologia implementada diante de diferentes soluções;
- Explorar vantagens das mais avançadas ferramentas de verificação para minimizar a intrusão da solução proposta.

O uso de uma plataforma dedicada à verificação funcional para acelerar a injeção de falhas acarretou nas seguintes questões:

- Será viável a utilização de emuladores dedicados a verificação funcional para aceleração de injeção de falhas?
- A metodologia proposta apresenta alguma vantagem cuja não seria viável se fosse baseada em outra forma de aceleração?
- Há limitações na metodologia proposta e como mitigá-las?

Essa pesquisa exibe resultados referentes à injeção de falhas permanentes e o caso de teste é um microprocessador descrito ao nível de portas lógicas. Dado que o grau de detalhamento do modelo do circuito utilizado influencia no desempenho, então se acredita que o caso de teste escolhido consegue destacar as vantagens da solução proposta. Com isso, a contribuição original dessa pesquisa pode ser identificada pelo ganho em desempenho com a metodologia proposta baseada na aceleração de injeção de falhas permanentes sem a necessidade de alterar o modelo do circuito sendo verificado. É importante ressaltar que MADC, por não ser intrusiva, reduz o esforço referente a geração de justificativas exigidas pelo ISO 26262 quando modelos abstratos do circuito são utilizados na análise de segurança funcional. Até este momento, o autor desconhece trabalho semelhante considerando o escopo definido acima.

METODOLOGIA

Parte do esforço intrínseco em garantir níveis adequados de segurança está na necessidade de se gerar evidências que corroborem a capacidade dos eletrônicos utilizados em automóveis de reagir previsivelmente quando da ocorrência de falhas. *Safety mechanisms* (SM) são adicionados ao sistema embarcado visando maximizar a cobertura de falhas passíveis de serem diagnosticadas (do inglês, *diagnostic coverage* – DC) e minimizar possiblidade delas resultarem acidentes. Em outras palavras, DC representa a porcentagem da probabilidade falhas que são anteparada pelos SMs implementados. Dessa forma, os SMs precisam ter sua funcionalidade verificada e se sua participação no nível DC do sistema permite atender os requisito do ASIL especificado para aplicação.

Há diversas técnicas de verificação e validação que vem sendo investigadas visando atender desafios inerentes à *functional safety*. Em meio a essas técnicas está simulação de falhas cuja é tradicionalmente empregada na geração de testes de manufatura de circuitos integrados. Essa pesquisa explora soluções de verificação funcional avançados para acelerar injeção de falhas quando comparado com técnicas baseadas em simulação, além de sempre observar os preceitos descritos no ISO 26262. Logo, MADC permite reduzir o tempo gasto com a análise de DC dos SMs implementados, especialmente quando o circuito está descrito ao nível de portas lógicas e o desempenho do simulador acaba sendo impactada devido à complexidade do modelo executado. A investigação do DC não se limita ao nível de portas lógicas uma vez que os valores precisam ser constantemente estimados de modo que medidas necessárias possam ser tomadas oportunamente. Com isso, soluções integráveis ao fluxo de verificação do circuito sendo desenvolvido é essencial.

Simuladores de falhas avançados foram recentemente anunciadas por fornecedores de ferramentas de projeto de circuito integrado atendendo requisitos do ISO 26262. Por outro lado, injeção de falhas baseada em simulação pode ser proibitivamente longa. Dessa forma, o emprego de plataformas de emulação baseadas em hardware dedicados para acelerar a campanha de injeção de falhas foi investigado. A MADC foi construída sobre uma dessas plataformas comerciais disponíveis cuja combina aceleração de simulação e emulação para otimizar a verificação funcional. Como injeção de falhas não é nativamente habilitada nessa plataforma, então algumas adaptações foram feitas para permitir que esse meio fosse empregado nesse estudo. Essa plataforma pode ser empregada independentemente do nível de abstração da descrição do circuito, logo, a MADC pode ser estendida para além do nível de portas lógicas.

Diferentes abordagens de aceleração de injeção de falhas foram proposta nas últimas décadas. Emuladores de falhas baseados em dispositivos reconfiguráveis como *Field-Programable Gate Arrays* são tradicionalmente usados para superar o desempenho limitado de simuladores de falhas. Metodologias baseadas em emulação apresentam desvantagens considerando os requisito do ISO 26262. Uma delas é o fato da análise ser realizada sobre um modelo do circuito que não é aquele efetivamente enviado para fabricação. Isso porque métodos baseados em intrusão ou em emuladores cuja tecnologia fim difere da aplicação final são usados. A MADC observa esses aspectos de modo que a injeção de falhas ocorre no mesmo modelo utilizado no fluxo de fabricação do circuito integrado.

Os trabalhos relacionados foram divididos entre os que visam a aceleração da injeção de falhas daqueles que mostram o uso da injeção de falhas no contexto de validação da integridade funcional. O conhecimento adquirido com o segundo grupo ajudou a esclarecer os requisitos no ISO 26262 que precisam ser observados quanto ao uso de injeção de falhas. Desse modo, foi possível identificar quais os pontos das técnicas de aceleração apresentados no primeiro grupo que poderiam ser explorados no contexto em que esse trabalho foi desenvolvido.

RESULTADOS

Os primeiros experimentos realizados com a MADC ajudaram a validar os resultados obtidos se comparado com soluções comerciais. Experimentos mais complexos mostram o potencial da solução proposta nessa pesquisa, em especial no que se refere a desempenho. Vale lembrar que a MADC oferece um melhor desempenho na análise de cobertura de falhas observando os requisitos do ISO 26262. Dessa forma, MADC pode servir ao segmento automobilístico com uma solução no estado da arte não intrusiva para acelerar injeção de falhas. Enfim, diversos avanços possíveis para a MADC desenvolvida são discutidos ao mesmo tempo em que trabalhos futuros são apresentados.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABREVIATIONS

µC	Microcontroller
ALU	Arithmetic Logic Unit
AoU	Assumption of Usage
ASIL	Automotive Safety Integrity Level
ATPG	Automatic Test Pattern Generation
BBW	Brake-By-Wire
CCF	Common Cause Failure
CCU	Central Control Unit
DBW	Driver-By-Wire
DC	Diagnostic Coverage
DfT	Design-for-Testability
DIS	Draft International Standard
DUT	Design Under Test
E/E/PE	Electric/Electronic and Programmable Electronic
ECU	Electronic Control Unit
EDA	Electronic Design Automation
FIT	Failure In Time
FMC	Failure Mode Coverage
FMEA	Failure Mode Effects Analysis
FMEDA	Failure Mode Effects, and Diagnosis Analysis
FPGA	Field-Programmable Gate Arrays
FSR	Functional Safety Requirement
FTA	Fault Tree Analysis
GL	Gate-Level
HARA	Hazard Analysis and Risk Assessment
HDL	Hardware Description Language
HWR	Hardware Safety Requirement
IC	Integrated Circuit
IFSS	Incisive Functional Safety Simulator®
ISO	International Organization for Standardization
LF	Latent Fault

MADC	Methodology to Accelerate DC Assessment
MPF	Multi-Point Fault
MUX	Multiplexer
NDA	Non-Disclosure Agreement
OEM	Original Equipment Manufacturer
PF	Perceived Fault
PVSG	Potential to Violate the Safety Goal
QM	Quality Management
RF	Residual Fault
RTL	Register Transfer Level
SA0	Stuck-At-0
SA1	Stuck-At-1
SAT	Stuck-At Faults
SBB	Steer-By-Brake
SBST	Software Based Self-Test
SBW	Steer-By-Wire
SEE	Single Event Effect
SEL	Single Event Latchup
SET	Single Event Transient
SEU	Single Event Upset
SF	Safe Fault
SG	Safety Goal
SM	Safety Mechanism
SoC	System-on-a-Chip
SOP	Start-Of-Production
SPF	Single Point Fault
SPFM	Single Point Fault Metric
S-R	Safety-Related
SWR	Software Safety Requirement
TB	Testbench
TB-Q	Testbench Qualification
TBW	Throttle-By-Wire
TSR	Technical Safety Requirement

TABLE OF CONTENTS

1. INTRODUCTION

Car manufacturers are integrating gradually more electronics in the vehicles in order to provide a safer experience for their customers – the drivers. The existing integrated circuit (IC) technology provides the computation power required to develop applications that can process the complex data provided by multiple sensors, transforming physical readings into electronic signals [1]. Capturing the surrounding information allows creating systems capable of deciding whether there are traffic hazards around and which actions should be taken to prevent or at least to minimize the risk of harming those involved. The data processing power available in the car permits to implement safety applications among many others like infotainment and car-to-car communication.

The automotive OEMs (Original Equipment Manufacturers) and their suppliers - Tier 1, Tier 2, and so forth – are investing on innovative car utilities in order to explore the new market opportunities made possible by the current technology. Nowadays, there are many valued features to consider when selecting a new car, which go far beyond the traditional characteristics, such as engine, internal space, design or fuel consumption. Ninety percent of the car novelties are based on electronics [2] and some are already available in most modern cars, such as: autopilot system, self- parking, no blind spots camera system, collision avoidance by auto emergency braking assist, gesture control of the infotainment system, cloud-based dashboard access, car keys with parking remote control among many other interesting features [3]. With so many new car features being launched on each automotive tradeshow, it seems that this is a lucrative market for the OEMs and suppliers that will continue to expand – the money spent on the vehicle electronics is expected to grow about ten percent by 2022 [2].

1.1. MOTIVATION AND CHALLENGES

With various sensors in the car – some directly exposed to the outside world to proper read the physical measures – there is a need for specific packaging, circuit board protection and extra wiring to connect the sensors to the Electronic Control Units (ECU) [1]. Approximately, up to 100 Kg of cabling weight can be found in a car nowadays [4]. The additional weight cannot be neglected since it can be linked to the fuel consumption increase: 50 kg weight ~ 0.1 liter/100 Km [5]. Moreover, OEMs need to meet new environment regulation hence reducing the CO_2 emission of the next vehicle generations. As a result, innovative automotive communication has become a research topic of broad and current interest that aims to find solutions for the cabling weight while still considering security and the required bandwidth challenges [2]. Independently of taking an innovative cabled networking connection or even a wireless solution for the communication, safety must always be taken into account.

At the end of the day, safety is a non-negotiable requirement that must be considered throughout the vehicle development process. The ISO 26262 standard provides guidance to ensure that such requirements are implemented [6]. Part of the additional effort required to increase safety is to provide evidence that the electronics integrated into the car are not just functionally correct but also they can handle random faults that may occur in the system due to aging or radiation sources, for example. To cope with possible fault scenarios, mechanisms are integrated into the design in order to add diagnose capability and to make the driver aware of any existing problem. In the case of a critical real-time feature, which correction may not be possible, then the system must guarantee minimal controllability to the driver that should then be able to avoid an accident of bigger proportions.

The safety mechanisms (SM) added to the design are used to improve the diagnostic coverage (DC) – i.e., the percentage of existing failure probability that is prevented by the SM – for different failure modes. The SMs must be functionally verified as well as the DCs achieved with them must be assessed. Fault injection is highly recommended for the functional verification of SMs and to evaluate their DC contribution. Although an exhaustive analysis is not required, evidence of best effort through the DC assessment must be provided when performing a quantitative evaluation of hardware architectural metrics. These metrics are used to identify whether the required automotive safety integrity level (ASIL) – ranging from A (lowest) to D (strictest) levels – was obeyed. Best effort, in this case, is pushed by the governmental regulations, and it can only be justified by using state-of-the-art techniques together while performing an assessment as thoroughly as possible.

There are many advanced verification technologies that are being explored or tailored to cope with different functional safety challenges. The selection of which SMs to be implemented starts at the conceptual phase of the design, much sooner than any implementation is made available. The DCs at this stage are estimated based on previous data, expert judgment, or guided by reference information from standards like the ISO 26262. Along the project development, the requirements initially defined must be continuously tracked while more detailed analysis is executed in order to provide evidence that these requirements are achieved. With progressive design refinement and implementation, a rationale approach is used to map the requirements to the corresponding parts of the design, thus allowing an analysis considering only safety-related (S-R) parts. Even after this “divide and conquer” approach is performed, the number of faults that are covered by an SM can still be unfeasible to verify through fault injection due to the design complexity. Therefore, sampling and other statistics methods are used to obtain confident assessment results.

This research explores the most advanced functional verification solutions in order to enable fault injection acceleration for SM verification and DC assessment. The research work developed aims to enable a more thorough safety analyses hence permitting to achieve higher accuracy of the metrics used to provide evidence that the design reached the required ASIL level. For instance, software based SMs – e.g., software based self-test (SBST) – typically requires long runs until the diagnostic routine completes, and the fault detection can be confirmed. For such kind of SM, concurrent fault simulators – commonly used in the Design-for-Testability (DfT) flow – are not suitable since they do not support complex testbenches (TB), which can reproduce the interaction with the software. Concurrent simulators normally only support designs at gate-level (GL), thus benefiting from the structured and modular description to optimize the fault simulation. On the other hand, it requires extracting static test vectors during normal simulation due to the lack of support of advanced TB, which is a time-consuming task. As a result, this kind of tool is not suitable for verifying the SMs during the development process. This forces the safety assessment to be executed once and only when the final GL netlist is available, what is risky since the cost of a design change at this stage is unacceptable. Therefore, the design is overprotected with extra SMs hence consuming unnecessary area and power to guarantee it will pass the safety certification process. Another consideration is that transient – fault models required by the ISO 26262 – are not typically supported by concurrent fault simulators.

Fault injection solutions to be used during the development phase are being announced by Electronic Design Automation (EDA) vendors, such as Incisive Functional Safety Simulator® (IFSS) [7], Certitude® [8], Z01X® [9], among others. However, fault injection campaigns using simulation can be prohibitively long, especially if the assessment is performed at the system level or if the SM being verified corrects the fault hence requiring most of the runs to be executed completely in order to classify the fault.

Targeting the fault injection campaign speed-up, the utilization of hardware emulator platforms, such as Veloce® [10], Zebu® [11], and Palladium® [12], provided by the biggest EDA vendors has been investigated. The thesis research used the former one that is provided by Cadence Design Systems, Inc., which combines simulation acceleration and emulation capabilities to boost verification throughput and productivity. Since this kind of platform is intended for typical functional verification, some investigation was necessary to permit leveraging Palladium for fault injection acceleration. The most important advantages of Palladium are the seamless acceleration despite the design description level: register-transfer-level (RTL) or GL. This allowed creating a non-intrusive acceleration approach for DC assessment of SMs. The proposed solution is based on technology edge tools, and the methodology has been conceived aiming to become a recommended approach to be used by future functional safety standard releases.

Different fault injection approaches have been proposed through the last decades. Many of them aiming to optimize the assessment of dependability attributes – e.g., availability, reliability, and safety – of fault tolerant designs [13] – especially against soft errors. The techniques can be grouped according to the technology/method that underlies the fault injection, which can be: hardware or physical injection, simulation and emulation. Software-based fault injection can be another group when distinguishing hardware design that process software [14]. Each approach has specific characteristics that define its application suitability. For example, simulation-based approaches are likely to provide more controllability and observability, while there is a compromise between accuracy and performance that must be considered. Field-programmable gate arrays (FPGA) are the most common technology used to emulate fault injection by providing the performance that lacks in the simulation based solutions. Minimal design modifications – also called instrumentation – are required to enable similar controllability and observability tough. Despite the facilities costs and the setup complexity, hardware or physical fault injection requires a prototype only available too late in the design flow and hence not considered here.

The FPGA-based approach is the most similar to the solution explored in this research. However, no design model alteration neither the FPGA bitstream generation are required by using Palladium as the underlying technology to accelerate fault injection. Only the static analysis algorithm implemented must be executed to collect all faults that can be accelerated hence keeping the same design model for simulation, emulation, and even sign-off. Since Palladium is a powerful machine specially created to accelerate the simulation, it has most features of a simulation tool thus providing similar controllability and observability, which are leveraged in the implemented solution. When considering functional safety, then the latest representation of the design must be used for the analysis. Therefore, the acceleration platform used has another significant characteristic, and that is: RTL or GL of design description has minimal influence on the possible performance gain. Consequently, the DC assessment can be performed over the design model most close to the sign-off thus satisfying an ISO 26262 certification auditor. Therefore, the static analysis implemented together with a platform like Palladium enabled the invention of the methodology presented, which permits accelerating the DC assessment.

1.2. OBJECTIVES AND ORIGINAL CONTRIBUTION

In the functional safety context, the evidence utilized to support the claim of safety compliance must be collected based on the design model being developed. Otherwise, a set of strong arguments needs to be prepared in order to justify the utilization of an abstract model used for the safety assessment. The development of such argumentation can require an overwhelming effort dedicated to the generation of evidence that the design model used accurately represent the actual design model. The work developed in the context of this thesis seeks the provision of a fault injection acceleration solution that does not rely on changing the design representation, in order to enable the acceleration. Given the distinct possibilities of employing fault injection, it is important to define the research scope clearly. From this moment on, fault injection means the imitation of fault effects at the semiconductor level, using the design model prior production – i.e. HDL description, either RTL or GL.

The main objective of this thesis is to **propose a methodology to leverage the most advanced functional verification solutions in order to accelerate fault injection for SM verification and DC assessment**. To meet this thesis objective, a few means were identified:

- To investigate the available technology edge solutions for function verification, which can be explored to accelerate the fault injection campaigns;

- To become functional safety literate in order to tailor the conception of a methodology taking into account the state-of-the-art recommendations presented in the standards;
- To build a proof-of-concept flow based on the developed scripts that enable seamless utilization of different tools to compare the results and validate the correctness of the proposed methodology;
- To implement a solution leveraging the tools available to analyze the design in order to minimize the intrusiveness of the proposed acceleration approach.

As already mentioned, the utilization of hardware-assisted verification platforms towards the acceleration of fault injection is investigated. Therefore, the following research questions are considered:

- Is the fault injection using a hardware-assisted platform feasible?
- Does the proposed methodology provide any advantage towards functional safety which would not be achieved with other acceleration solutions?
- Are there limitations and what can be done do minimize their impact?

This research focuses on the injection of permanent faults. A microprocessor architecture, described at gate-level, has been chosen as a case study. The detail level of the design model used can significantly impact the simulation performance. Hence it is believed that a test case described at gate-level can better highlight the advantages of the proposed solution. Within this scope, the Thesis original contribution can be defined as the performance gain achieved with the proposed methodology via the fault injection acceleration of permanent faults without requiring modification or a different design model while reusing the existing verification environment – e.g., not requiring TB transformation to allow using traditional DfT fault simulators. To the best of this author's knowledge, there is no published work considering this specific scope.

The proposed methodology utilization is not limited to the gate-level netlist and the automotive scope. In fact, it is a generic fault injection acceleration solution that can be leveraged in other industry segments like avionics or applied at different abstraction levels. For that reason, fault injection techniques relevant to aerospace applications are discussed as well as methods pertinent to the automotive area. However, it is important to emphasize that this Thesis focuses exclusively on addressing the challenges derived from the ISO 26262, which is the functional safety standard tailored for the automotive applications. The results achieved with the developed proof-of-concept highlights the potential benefit offered by the proposed methodology.

1.3. ORGANIZATION

The thesis is divided in the following manner: Chapter 0 provides an ISO 26262 overview, and the functional safety context covered in this Thesis. Chapter 3 analyzes related work in the area of fault injection but within the safety context and its different applications. Chapter 0 details the many fault injection concepts, and it also discusses the state-of-the-art acceleration approaches targeting the fault injection optimization. A literature review covering different fault injection acceleration techniques is presented in Chapter 5. The characteristics of each reviewed acceleration approaches are discussed in order to identify their suitability to the functional safety domain. Therefore, Chapter 5 allows highlighting how the Thesis' original contribution distinguishes from the rest of the solutions found in the literature. Chapter 6 goes through the developed non-intrusive approach that enables fault injection acceleration by leveraging the hardware-assisted platform. Additionally, Chapter 6 describes the proposed methodology to connect the fault injection campaign to the ISO 26262 world through an FMEDA. Chapter 7 reports the experiments performed using an OpenRISC architecture, and it discusses the results achieved. At the end of Chapter 7, a high-level comparison between the proposed methodology and the most similar related work is made, which permits emphasizing the potential benefit of the proposed methodology. In conclusion, Chapter 0 summarizes the thesis research achievements and comments future work.

2. FUNCTIONAL SAFETY AND ISO 26262 INTRODUCTION

2.1. ISO 26262 TERMINOLOGY AND SCOPE

In a broad sense, safety is a dependability attribute that represents the probability of a system that cannot operate correctly any longer to interrupt its functions in a way that nothing catastrophic happens to the users and the environment [13]. The "absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems" is the definition of functional safety according to the ISO 26262 vocabulary [15]. A graphical interpretation of this definition is illustrated in Figure 1. In a vehicle, hazards can be due to mechanical, electrical or even hydraulic problems. ISO 26262 limits its concern to the malfunction behavior of Electric/Electronic and Programmable Electronic (E/E/PE) systems thus not including electrical shocks or any other source of risk.

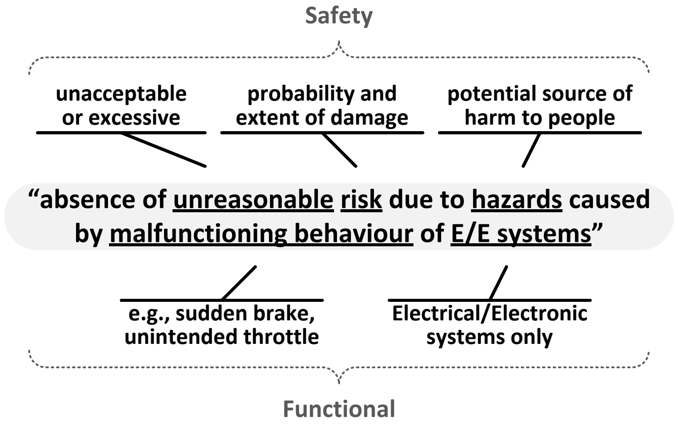

Safety
unacceptable or excessive
probability and extent of damage
potential source of harm to people
"absence of unreasonable risk due to hazards caused by malfunctioning behaviour of E/E systems"
e.g., sudden brake, unintended throttle
Electrical/Electronic systems only
Functional

Figure 1 – Functional safety definition according to the ISO 26262.

The ISO 26262 has its first nine parts – chapters of the standard – published late 2011 followed by the part ten in 2012. The standard on its first edition has series production passenger cars with maximum gross vehicle mass no greater than 3,500 Kg as intended scope of application [15]. The second edition of the ISO 26262 is planned to be released in 2018 with wider scope by removing the vehicle mass limitation and also including a specific chapter for motorcycles [16]. This new edition is going to contain the part eleven that will cover the application of the standard concepts on semiconductors, which is vaguely considered in the current release [17]. Until the official publication, the standard committee has periodically meetings to agree on the progressing draft of the ISO document, which is accessible for members – including OEMs, EDA vendors, safety consultants, and others.

2.2. BRIEF HISTORY OVERVIEW AND LEGAL ASPECTS

Functional safety has not been introduced by the automotive standard that drives the topic of this research. Aviation segment has the important DO-178 "Software Considerations in Airborne Systems and Equipment Certification" published in 1982, which matured to the DO-178A – released in 1985 – and ten years later became the well-known DO-178B. At the same time, it has been developed the "Aerospace Recommended Practice" guidelines ARP4754A and ARP4761, which were published later in 1996 putting the system and safety assessment into place [18]. The hardware counterpart is the DO-254 that provides guidance to ensure safe operation of airborne electronic designs that was announced in 2000 [19]. In the same year, the remaining four parts of the IEC 61508 standard were released after the first three parts became available in 1998 [20]. Figure 2 shows some of the most well-known safety standards.

The "functional safety of E/E/PE safety-related system" standard, or IEC 61508, is a "generic" standard, which served as the basis for drafting the functional safety guidelines tailored to different industry segments [21] – some indicated in Figure 2. Therefore, IEC 61508 supported the creation of safety standards applied medical device software (e.g., IEC 62304), nuclear power plant systems (e.g., IEC 61513), machinery control systems (e.g., IEC 62061), industrial processes (e.g., IEC 61511), railway application (e.g., EN 50126), and many others. This long list also includes ISO 26262 covering the automobile electronics. The draft international standard (DIS) of the ISO 26262, which is prepared by the committee before the official release, was published in 2009. The same year the "unintended acceleration" case with the Toyota Lexus ES 350 killed all four occupants and triggered an escalation of investigations, back to 2002, of driver's complaints reporting similar problem [22]. The investigation found 89 deaths suspected to be caused by defects on the Electronic Throttle Control System (ETCS), but the National Aeronautics and Space Administration (NASA) – assigned to inspect the design – could not point to a specific design flaw. However, NASA reported many technical questionable procedures and the jury, of the civil lawsuit against Toyota, was convinced that ETCS defects caused the deaths. Additional to millions of recalls and millionaire civil penalties, the carmaker was fined in more than a billion of dollars in 2014 [23]. To prevent such cases that the ISO 26262 was created, similar to the IEC 61508, which was influenced by the lessons learned from tragedies like Bhopal in 1984, Chernobyl in 1986 and Piper Alpha in 1988 [24].

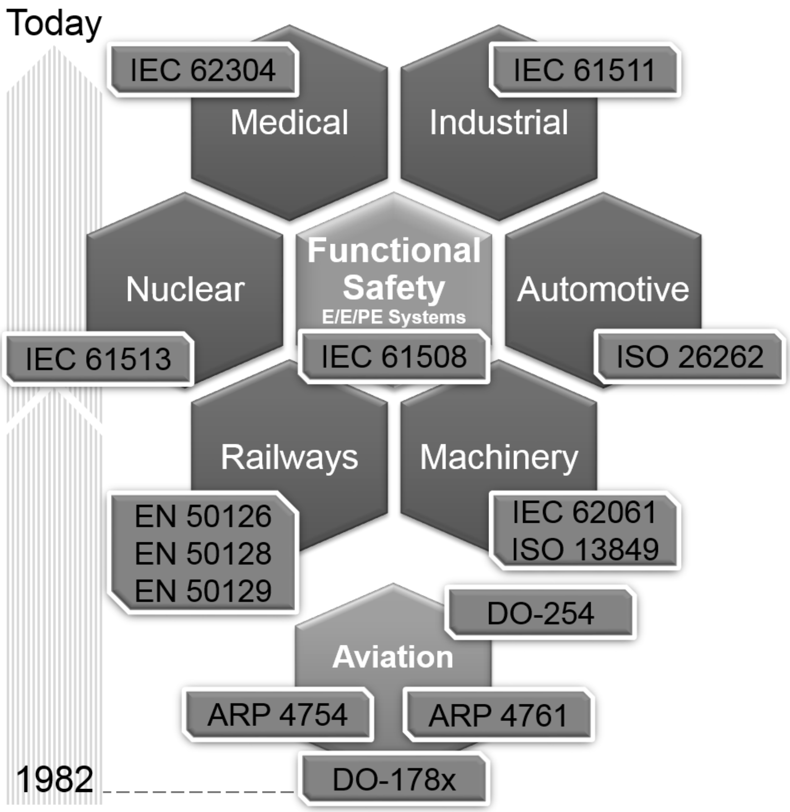


Figure 2 - Functional safety standards historic connection.

Regulation bodies from nine countries – e.g., USA, Germany, Japan – engaged for more than eight years until publishing the first version of the ISO 26262 containing the state-of-the-art framework for achieving functional safety [25] [26]. Despite the technical focus of the recommendations in the standard, there are circumstantial norms or legal aspects that cannot be ignored. For instance, according to the European Regulation 661/2009, the current state of science and technology must be used to design vehicle safety. This makes the ISO 26262 more than just a recommendation due to its contemporary functional safety guidelines. The Toyota “unintended acceleration” example and the liability risk shared across the OEMs and its suppliers together with the growing number of electronics integrated into the cars, made the automotive industry align and contribute to the ISO 26262. While the standard imposes additional effort of complying with a bundle of new requirements, it also serves as legal protection from unreasonable liability. Such protection can only be achieved if the designer can convince it has done its “best-effort” to assure safety [27]. This is supported by the Safety Case, which compiles all relevant functional safety information derived from the other work products – i.e., documentation – required by the ISO 26262. The Safety Case must clear state what is being claimed – i.e., scope, context and requirements – about the system, the evidence – e.g., work products, test reports – and the safety arguments that communicate the relationship between the evidence and what is claimed [28]. Additionally, best-effort evidence must be provided including the adoption of reasonable economic and technical measures to guarantee maximum safety. In other words, the “liability risk” translated into the “hazard potential” of the application is what establishes the “best effort” extent a supplier has to commit in order to determine the appropriate state-of-the-art to be used [29].

2.3. AUTOMOTIVE SAFETY LIFECYCLE

The automotive safety lifecycle – see Figure 3 – is outlined in the ISO 26262. Part 2 of the standard describes requirements – e.g., competences, traceability, plans – that must be managed from development to decommissioning of the product, as well as pre-requisites for safety activities applicable to specific phases [30].

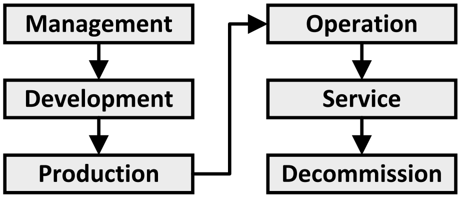


Figure 3 – Automotive safety lifecycle according to the ISO 26262.

The next five parts of the standard specify distinct stages of the ISO 26262. In Figure 4, the clauses that describe the subphases of each stage are indicated in the following manner: "part number"-"clause". Part 3 start requiring the definition of the item being considered, which correspond to features at the vehicle level like:

- Electronic Throttle Control Systems (ETCS);
- Electrical Hydraulic Power Steering (EHPS);
- Advanced Driver Assisted System (ADAS);
- Tire Pressure Monitoring System (TPMS);
- Electronic Steering Column Lock (ESCL);
- Electronic Stability Program (ESP);
- Emergency Brake Assistant (EBA);
- Antilock Braking System (ABS);
- Adaptive Cruise Control (ACC);
- Traffic Sign Recognition (TSR);
- Electronic Parking Brake (EPB);
- Electric Power Steering (EPS);
- Steer-By-Brake (SBB);
- **X**-By-Wire – i.e. **X** = **D**rive, **T**hrottle, **B**rake, **S**teer, etc.

After the item definition with regards to its functions, interfaces, use cases, the safety cycle is then initiated by identifying if the project corresponds to a new design, modification or a reuse of existing product in an automotive application. The hazard analysis and risk assessment (HARA) subphase comes next and the objectives are:

- identify and categorize the hazards that malfunctions in the item can trigger;
- determine the ASIL considering the estimate of severity, exposure and controllability probability factors of each hazardous event based on the provided item's functional behavior;

- formulate the safety goals to prevent or mitigate the unreasonable risk of each hazardous events with an ASIL assigned.

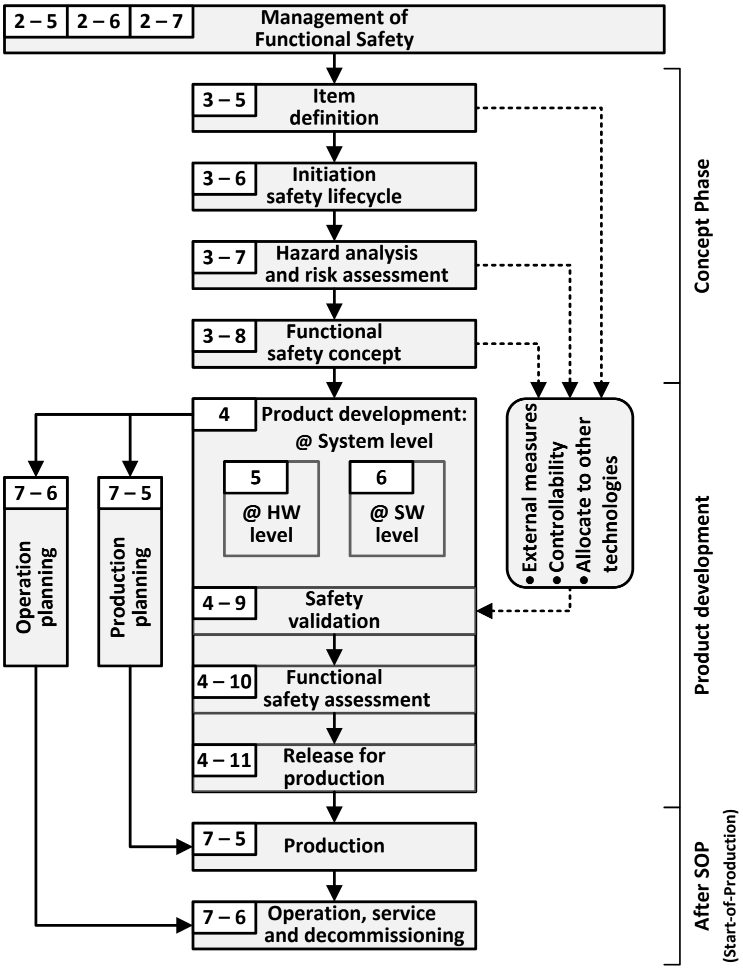


Figure 4 - Safety lifecycle subphases. (Adapted from ISO 26262)

2.4. ISO 26262 – CONCEPT PHASE EXERCISE

A fault tolerant Drive-By-Wire (DBW) example is used here to demonstrate a HARA [31]. The example consists of a Steer-By-Wire (SBW), a Brake-By-Wire (BBW), and a Throttle-By-Wire (TBW) sub-systems. The elements of the DBW system are: central control unit (CCU); brake, steer and throttle ECUs and actuators; communication bus; and the steer wheel, the pedals, and their sensors. The user interface shall permit the driver to control the speed and the steering angle of the vehicle through the interaction with the pedals and steering wheel, respectively. Therefore, the steer and speed control need to be able to turn and to regulate acceleration of the vehicle. Figure 5 diagram shows how the elements composing the DBW system interact.

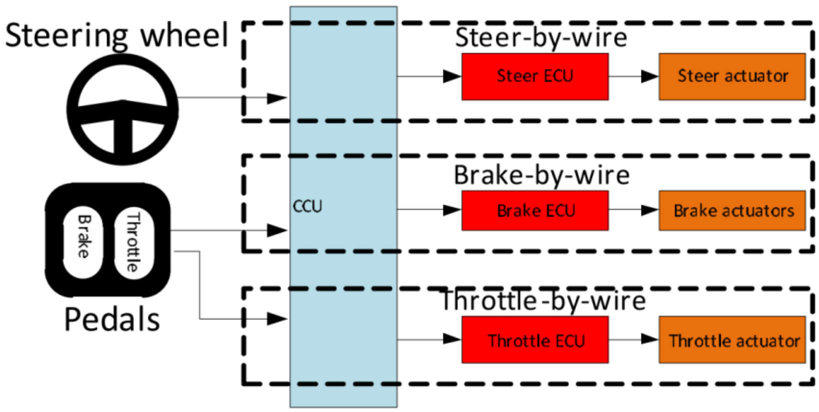


Figure 5 – The block diagram of the DBW example. (Source: [31])

2.4.1. Item Definition

An example of item definition for the DBW system is shown in Table 1. The DBW definition contains the description of the intended functionality of the item, which is composed of the SBW, BBW, and TBW subsystems.

Table 1 – Item definition of the DBW example with the subsystems functionality description. (Adapted from [31])

System	Function	Description of intended function
SBW	Steer control	Control the steering angle of the vehicle

TBW	Speed control	Increase the speed of the vehicle
BBW	Speed control	Decrease the speed of the vehicle

The item definition shall also include operation modes, environment conditions, legal requirements (e.g., the drive legislation of the interesting regions), already known failure modes and hazards (e.g., from known safety-related incidents). Sufficient information must be provided in order to permit conducting the subsequent activities, which are the HARA and later the "Functional Safety Concept".

2.4.2. Hazard Analysis and Risk Assessment

The HARA starts by analyzing the operating situations – i.e., functional modes and environment constraints – on which an item's malfunction will result in a hazardous event. Brainstorming, field studies, "Failure Mode Effects Analysis" (FMEA), among other techniques are used to systematically identify the as many as possible hazardous events and their consequences. Table 2 is a reduced version of the hazard identification done for the DBW system, but with a similar format used in the ISO 26262 guidelines.

Table 2 – Hazardous event identification for the DBW example. (Adapted from [31]).

ID	**Failure mode**	**Hazardous Event**		**Possible consequences**
		Hazard	**Situation**	
H01	Throttle omission	Sudden lack of throttle	Low speed at crosslevel	Rear end or train collision
H02	Throttle omission	Sudden lack of throttle	High-speed at motorway	Rear end or side collision
H03	Brake commission	Unexpected full brake	High-speed at highway	Traffic Accident
H04	Brake stuck at value	Constant partial brake	Avg. speed at main roads	Loss of car stability
H05	Steer angle commission	Unexpected strong veer	Taking a bend at high-speed	Loss of car control

All hazardous events identified in the exercise, which results in the Table 2 information, must be classified with respect to the severity, probability of exposure and controllability. All three classification parameters are estimated using a defined rationale.

The failure mode corresponds to the manner in which an item or one of its elements fails. If an item's failure mode occurs during a specified operational condition, then the worst-case scenario is considered as the possible consequence. Table 3 can be used to classify the severity of such consequences. The severity parameter has four classes that are associated with the different potential harm levels of the persons at risk, such as, drivers, passengers, cyclists, and pedestrians.

Table 3 – Classes of severity according to the ISO 26262.

Class	Description
S0	No injuries
S1	Light and moderate injuries
S2	Severe and life-threatening injuries (survival probable)
S3	Life-threatening injuries (survival uncertain), fatal injuries

Also, split into four classes, the controllability parameter relates to the probability of the driver or another person potentially at risk to gain control of the hazardous situation and avoid the harm. Therefore, the operating condition is analyzed, and the controllability classification is decided using the Table 4.

Table 4 – Classes of controllability according to the ISO 26262.

Class	Description
C0	Controllable in general
C1	Simply controllable
C2	Normally controllable
C3	Difficult to control or uncontrollable

The probability of exposure to the operational situation of the hazardous event is what defines the remaining parameter needed for the hazard classification. Table 5 has the five classes that distinguish the different probability levels of exposure.

Table 5 – Classes of exposure probability regarding operational situations according to the ISO 26262.

Class	Description
E0	Incredible
E1	Very low probability
E2	Low probability
E3	Medium probability
E4	High probability

2.4.3. Automotive Safety Integrity Level Determination

Using Table 6 is possible then to determine the ASIL for each hazardous event identified.

Table 6 – ASIL determination based on the hazardous classification parameters according to the ISO 26262.

ASIL Classification		**Controllability**		
Severity	**Exposure**	**C1**	**C2**	**C3**
S1	**E1**	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	**E1**	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	**E1**	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

The ASIL has four different levels corresponding to the item’s or element’s requirements to be fulfilled in order to avoid an unreasonable risk. The levels are represented by the first four letters of the Latin alphabet where A and D represent the least and the most stringent levels, respectively. Some hazardous event not even get the ASIL A assigned, and therefore, no additional requirement must be observed hence the Quality Management (QM) process already in place is sufficient.

The hazardous events of the DBW example in Table 7 are derived from the operating situation and the hazard identified in Table 2. For each hazardous event, the classification parameters are defined thus permitting to assign correspondent ASIL according to Table 6. Notice that no further requirements must be observed for the hazardous events H01 and H02 of the DBW example, however, the ASILs D, B, and D levels were assigned to the Table 7 entries H03, H04, and H05, respectively.

Table 7 – ASIL determination for the hazardous event identified DBW example. (Adapted from [31])

ID	Hazardous Event	Parameters	ASIL
H01	Sudden lack of throttle at low speed passing over a rail cross	S3,C2,E1	QM
H02	Sudden lack of throttle at high speed driving in a motorway	S1,C1,E4	QM
H03	Unintended full brake applied while driving at highway speed limit	S3,C3,E4	D
H04	Constant partial brake pressure suddenly applied when driving with main road speed	S3,C2,E4	B
H05	Unintended strong veer off the direction at high speed in the highway	S3,C3,E4	D

2.4.4. Safety Goal Specification

At least one top-level safety requirement, known as Safety Goal (SG), must be defined for those hazardous events that had an ASIL assigned – i.e., no less than A level. Therefore, SGs had to be defined only for the hazardous events H03, H04, and H05. The SG from the DBW example is shown in Table 8. Statements like the one highlight in SG01 presumes highest possible reliability will be targeted [31]. SG01 is also used as an example of combining similar SG by keeping the highest ASIL level assigned. Additionally, an operating mode without the unreasonable level of risk can be specified by the SG, for example, the “safe state” underlined in SG03 that needs to be achieved and maintained to prevent the possible harm.

Table 8 – Safety goals defined for the hazardous events

ID	**ASIL**	**Safety Goals**	
H03	D	SG01	The brake *functionality shall not fail*
H04	B	SG01	The brake *functionality shall not fail*
		SG02	Warn the driver when at degraded function
H05	D	SG03	The steer system needs to be stuck at value in case of failure, <u>enabling SBB functionality</u>

2.4.5. Derivation of the Safety Requirements

At the “functional safety concept” subphase, it is defined the Functional Safety Requirements (FSR) for each SG resulted from the HARA activity. An FSR specifies behaviors or measures that shall be implemented. The FSR specification can include the transitioning to the safe state to achieve, the fault tolerance time interval, the warning mechanisms (e.g., ABS warning lamp), the functional redundancies (e.g. SMs), and so forth. The FSRs must also be mapped to the corresponding system elements they apply. Table 9 contain some FSRs and the allocated elements from the DBW example.

Table 9 – FSRs from the DBW example. (Adapted from [31])

ASIL	**SG**	**Functional Safety Requirement**		
D	SG01	FSR01	Distributed brakes	BBW-ECU
		FSR02	CCU redundancy: HW/SW [a]	CCU
B	SG02	FSR03	Interaction with the driver	CCU
D	SG03	FSR04	SSB technology	BBW-ECU
		FSR05	Steer stuck at forward position	SBW-ECU

[a] Hardware and Software redundancy.

The “product development at the system level” phase (see Figure 4) also has the initialization exercise when it is specified functional safety activities for the further subphases. One of the activities is to derive the Technical Safety Requirements (TSR) from the FSRs. The TSR shall specify the SM details, such as the detection, alarming and control of faults in the system, ways to achieve and maintain, measures to avoid or warn degradation. Subsystem elements can have TSRs derived from the TSRs allocated at the system level, and this can be iteratively done through the design hierarchy. Optionally, TSRs can be further refined into hardware (HWR) or software (SWR) safety requirements.

2.4.6. Requirements Allocation through the Supply Chain

Different from the IEC 61508, the ISO 26262 considers the typical automotive scenario where a system can be built with the elements provided by multiple companies in the supply chain of the OEM. Therefore, TSRs must be communicated through the companies. This communication is supported by the Development Interface Agreement (DIA) where the data to be exchanged between the customer, and the supplier is defined. The supply chain illustrated in Figure 6 highlight the importance of the DIA also protects the companies involved from unreasonable liability.

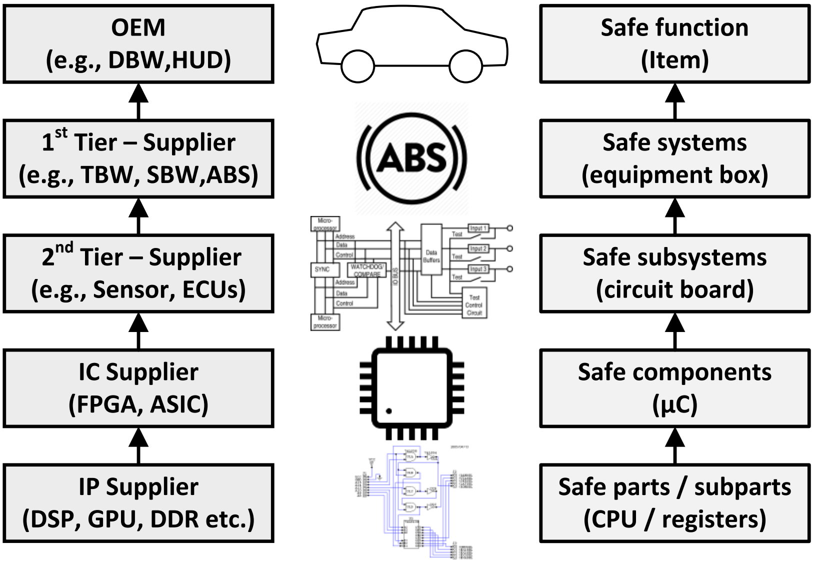


Figure 6 – Automotive supply chain illustration.

The SGs with the determined ASIL, the FSRs and the derived TSRs at the vehicle level are typically defined by the OEM as indicated in Figure 7. If this data is not available, then they can be defined based on assumptions of usage (AoU) whenever a supplier is going to commercialize a generic subsystem as a safety element out of context (SEooC).

An item as defined in the ISO 26262 can be composed of a hierarchy of systems, which the subsystem leafs are built by multiple components. These components are split into hardware parts (or software-units) that can be further divided into sub-parts. Higher level TSRs are further refined across the system hierarchy throughout the supply chain by deriving new detailed TSRs. The leaf requirements are allocated to the components, for example, a software application or a microcontroller (µC). Such components may have specific requirements – i.e., HWR and SWR, respectively – that are derived from the leaf TSRs as illustrated in Figure 7.

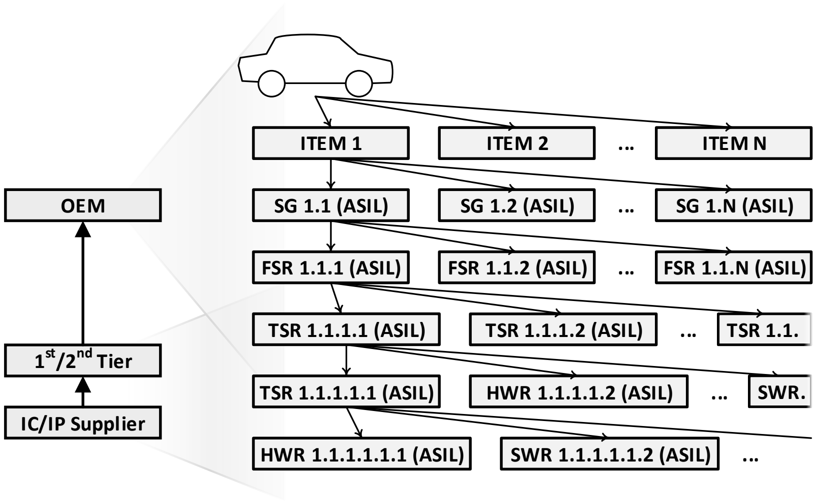


Figure 7 - Requirements communication throughout the supply chain.

This complex multilevel requirements definition must be consistent and traceable in order to guarantee that no SG is overlooked. Additionally, the strictest ASIL level assigned to the SG must traverse together with the requirements, which are allocated to the components. Therefore, all FSR, TSR, HWR and SWR illustrated in Figure 7 must have the same ASIL assigned to their parent SG (SG 1.1). This traversal assignment ensures that the correct safety measures are implemented in the required components. A microcontroller is an example of a complex *component*, which is subdivided into its major *parts*, for instance, a CPU, memory controller, and other peripherals. A *part* itself can be further split into *subparts* – e.g., arithmetic logic unit (ALU), debug interface, timer, and interruption controller. The *component* dismemberment allows achieving a stricter ASIL by evidencing the *parts* or the *subparts* – e.g., debug module interface – that are not S-R and hence can not contribute to the SG violation probability [28]. Moreover, this detailed analysis increases the confidence of the safety assessment.

2.5. CHAPTER REMARKS

This Chapter briefly discussed the liability impact on the “best-effort” practiced by the OEMs and suppliers. The legal aspect pushes the companies in the supply chain always to consider state-of-the-art solutions like the one proposed in this research. The effort level commit by the companies is also based on the ASIL determined according to the identified hazardous event of the target application. Therefore, it is shown how the ASIL, which is defined for SG at the vehicle level, traverse with the safety requirements that are communicated by DIAs throughout the supply chain until getting allocated into hardware and software components. This standard overview is essential to understand the ASIL influence on the next subphases. The V-Model defined in the ISO 26262 is illustrated in Figure 8, which shows how the activities are distributed in the V-Model flow. After starting the development of the hardware and software parts, verification is the next subphase, which is discussed in the following chapter.

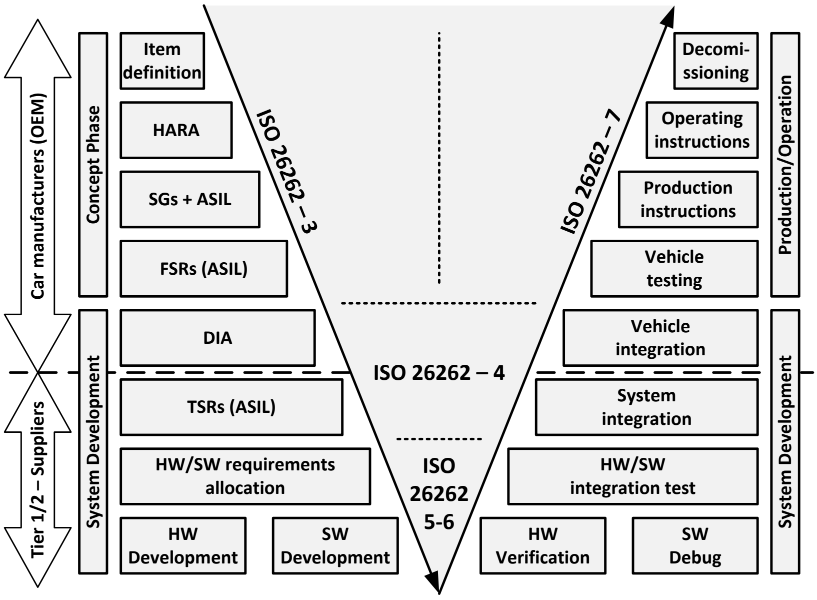


Figure 8 – ISO 26262 V-Model illustration. (Adapted from ISO 26262)

3. FAULT INJECTION IN THE CONTEXT OF ISO 26262

3.1. SAFETY ASSESSMENT BASIC CONCEPTS

Failure probability imposes a great challenge with the advance of the process technology required to develop a complex systems-on-a-chip (SoC) with higher performance [14]. Fault avoidance through design inspection and testing processes are not enough to prevent failures due to aging or radiation effects during the system lifetime. Fault tolerance and fault diagnosis mechanisms are used to increase the reliability, thus allowing to develop robust circuits that fulfill the needs of critical applications. Such mechanisms are commonly implemented by using some kind of redundancy like replicated hardware, specific software routines, data parity, and others.

3.1.1. Hardware Architectural Metrics

The term “safety mechanism” is used in the ISO 26262 that embraces different fault tolerant mechanisms which can be employed in the safety device. The collection of SMs selected to integrate the design and to increase its reliability must be checked regarding their capability to prevent faults from propagating and violating SGs. The standard requires a rigorous analysis of the incidence probability of random faults in the hardware. The safety analysis uses a set of objective metrics in order to enable the audition by an external authority. The hardware architectural metrics allows to assessing if the selection of SMs implemented is sufficient to detect and or to control the failure rate proportion defined according to the ASIL assigned. Therefore, the single point faults metric (SPFM) and latent faults metric (LFM) percentages must be assessed.

3.1.2. Fault Classification

A single point fault (SPF) has the potential to directly violate an SG unlike the multi-point fault (MPF). However, an MPF that remains latent (MPF_L), in combination to another independent fault, has the indirect potential to violate an SG. Naturally, SMs are first considered to cover as much SPFs as possible. Whenever an SM is used like this, the remaining uncovered SPFs are called residual faults (RF) hence with the same SPF characteristics – potential to directly violate an SG. Then, in the presence of an SM, the residual SPFs become RFs, while the covered SPFs are called MPFs. Therefore, it is beneficial that SMs added to cope with SPFs can also diagnose as much MPF_L as possible thus improving the LFM by increasing the number of latent faults that are detected (MPF_D). There are faults which cannot directly violate an SG, but independently of the detection of an SM, their presence can be perceived by the driver that notices some performance degradation or any other problem indication like black smoke leaving the car exhaust. This kind of perceived multipoint fault (MPF_P) is not considered at microcontroller level since it is not possible to judge the detection ability of the driver at this level [28].

Notice that the violation of an SG by an MPF is typically associated with the occurrence of a second MPF, but no more than two faults – sometimes MPFs are even referred to as dual-point faults [28]. In general, a fault with the characteristics of an MPF, yet requiring two or more extra faults to be able to cause a violation, can be considered a safe fault (SF) – unless evidence during the safety concept shows otherwise. Faults considered SF shall not contribute to the violation of an SG.

3.1.3. Failure Modes and Fault Models

To overcome the SPF associated with a short circuit of a resistor, for example, three resistors in series can be used instead, thus allowing the short circuit of each resistor to be considered SFs [28]. The short circuit in this example represents one possible failure mode of a component like a resistor. Permanent and transient are common digital semiconductors failure modes hence they are considered in this research. The permanent failure mode is triggered by the occurrence of stuck-at-faults (SAT) – i.e., a signal gets stuck-at-0 (SA0) or stuck-at-1 (SA1). On the other hand, single-event-transient (SET) and single-event-upset (SEU) faults, temporarily disrupt the normal functionality of combinatorial and sequential parts of a circuit, respectively. For that reason, the faults SET and SEU compose the transient failure mode. Notice that only random faults – i.e. during operational lifetime – are being considered since the systematic faults are covered by implementing and properly managing the safety activates throughout the safety lifecycle.

3.1.4. Failure Rate

The probability of a component failure mode occurs is called the base failure rate (λ), and the unit is FIT (failure in time). One FIT corresponds to no more than a single failure within one billion operating hours – i.e. 10^9 h. The FIT budget is defined for the item SGs at the vehicle level according to the ASIL determined [32]. FIT rate targets are derived from the budget and allocated to the elements through the system hierarchy – similar to the derivation of safety requirements from the SGs. For example, Figure 9 illustrates the FIT budget for an ASIL D item that is communicated across the supply chain until defining the target FIT to be observed at the component level. In other words, the accumulative failure rate of those elements – i.e., parts and subparts of components – that contribute to the violation of an SG shall not surpass the FIT budget. This evaluation must be done in order to guarantee the probability to violate each SG is not exceeded – i.e., each SG's FIT budget is observed. The assessment of the hardware architecture metrics and the SG violation probability due to random hardware failures is required in order to claim ISO 26262 compliance.

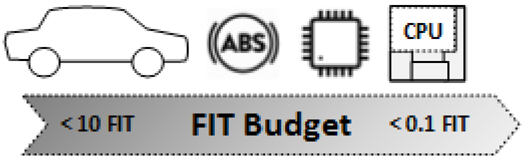


Figure 9 – Allocating the target FIT budget to be observed in order to guarantee the probability to violate an SG is not exceeded at the vehicle level.

3.1.4.1. Failure Rate Estimation

The estimated base failure rate for semiconductors can be extracted from recognized industry sources like SN29500 and IEC/TR 62380 with respect to permanent faults, or from JEDEC standards like the JESD89 for faults of the transient failure mode. Standards may provide too conservative numbers that do not represent the process and technology in place. Therefore, expert judgment, field experience, tests, and other inputs can also be used provided adequate confidence level on the statistics and approach employed to perform the estimations [32]. For complex components like microcontrollers, there may be parts which are not safe relate or SMs that cover only specific subparts of the design. In this case, it makes senses to divide the design further into parts and subparts thus distributing the component base failure rate according to the area proportion occupied by microcontroller's modules for example. The area figures are conservatively estimated at earlier stages, then, they are updated during the development process – e.g., whenever a GL netlist more mature becomes available. After allocating the base failure rate to each part and subpart of the component according to the failure modes, the hardware architectural metrics – i.e., SPFM and LFM – need to be calculated.

3.1.4.2. Base Failure Rate Composition

Figure 10 illustrate how it is composed the component base failure rate.

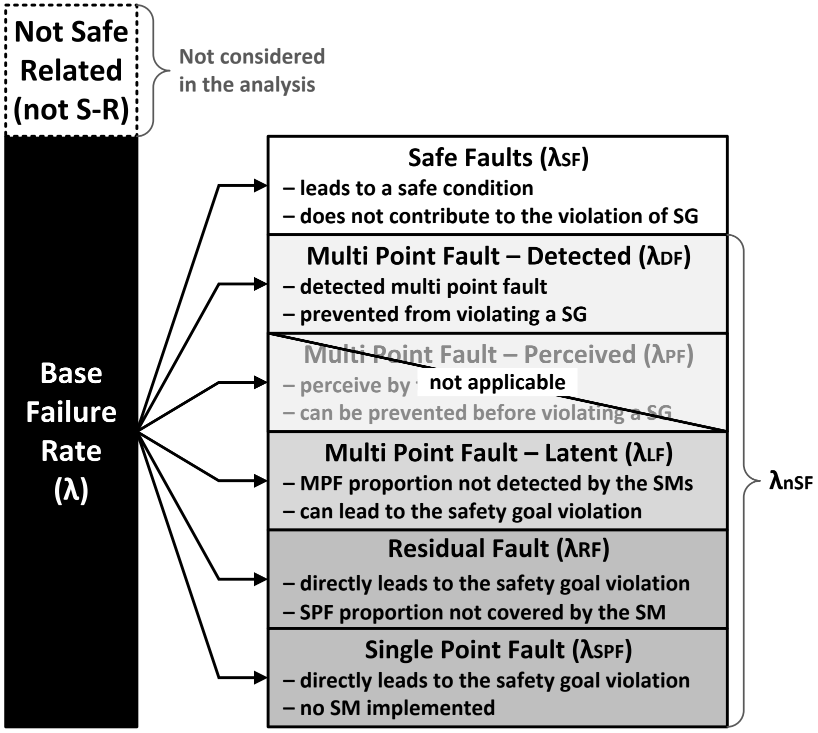

Not Safe Related (not S-R)
Not considered in the analysis
Base Failure Rate (λ)
Safe Faults (λSF)
– leads to a safe condition
– does not contribute to the violation of SG
Multi Point Fault – Detected (λDF)
– detected multi point fault
– prevented from violating a SG
~~Multi Point Fault – Perceived (λPF)~~
~~– perceive by~~
~~– can be prevented before violating a SG~~
not applicable
Multi Point Fault – Latent (λLF)
– MPF proportion not detected by the SMs
– can lead to the safety goal violation
Residual Fault (λRF)
– directly leads to the safety goal violation
– SPF proportion not covered by the SM
Single Point Fault (λSPF)
– directly leads to the safety goal violation
– no SM implemented
λnSF

Figure 10 – Component base failure rate split according to the classification of the faults of a certain failure mode.

The possible faults of a failure mode, within the area of each part or subpart, must be individually classified into SF, SPF, RF, and MPF thus corresponding to proportions of the base failure rate – i.e., λ_{SF}, λ_{SPF}, λ_{RF}, and λ_{MPF}, respectively. The λ_{MPF} is split further according to the different MPF subcategories symbolized by λ_{DF}, λ_{PF}, and λ_{LF} as shown in Figure 10 – the acronyms DF, PF, and LF denote MPF_D, MPF_P, and MPF_L and they are one-to-one interchangeable. The faults that can occur in an element, which is not S-R, are also considered SFs but are not taken into account for the safety analysis. Therefore, the base failure rate for each failure mode is calculated using (1).

$$\lambda = \lambda_{SF} + \lambda_{SPF} + \lambda_{RF} + \lambda_{MPF} \tag{1}$$

Where the MPF failure rate calculated in (2) is composed of the proportions corresponding to the amount of faults classified as perceived, latent, and detected. The perceived failure rate is not considered in the following equations since it is not applicable to the focus of this research – i.e., semiconductor level.

$$\lambda_{MPF} = \lambda_{PF} + \lambda_{LF} + \lambda_{DF} \tag{2}$$

3.2. HARDWARE ARCHITECTURAL METRICS CALCULATION

3.2.1. Failure Rate Proportions Matching the Fault Classification

Each fault despite the failure mode – e.g., permanent or transient – can be classified using the flow diagram adapted from the ISO 26262 in Figure 11. The faults on an element that is not S-R, even if classified as safe faults, are not considered in the calculation. Notice that faults can only be tagged as SPF if no SM prevents any other fault from the same failure mode from directly violating an SG. In this case, when there is no SM to prevent SPFs, yet there can be faults that do not directly violate an SG, which shall be automatically tagged as latent faults unless there is an SM that only covers MPF faults. The arrow in Figure 11 indicates when a fault that does not directly violate an SG becomes an MPF and therefore can be either detected or classified as a latent fault.

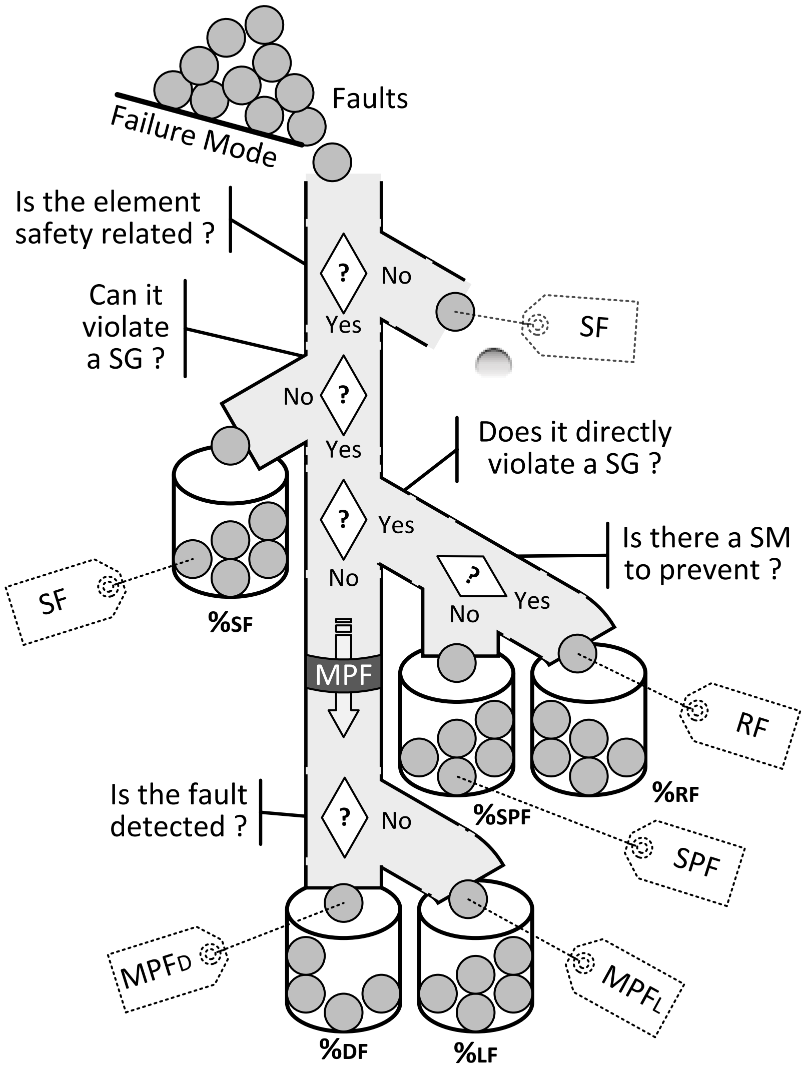


Figure 11 – Fault classification illustration using dropping balls as faults that are distributed according to the answer to the question through the path.

The fault classification analogy shown in Figure 11 utilizes balls symbolizing faults that are tagged and split into bins denoting the possible classifications. Given that all faults of a specific failure mode shall be tagged, then the percentage of all balls found in each bin corresponds to the proportion of the failure rate associated with each classification. Figure 12 illustrates base failure rate shares related to the percentage of faults in each classification bin.

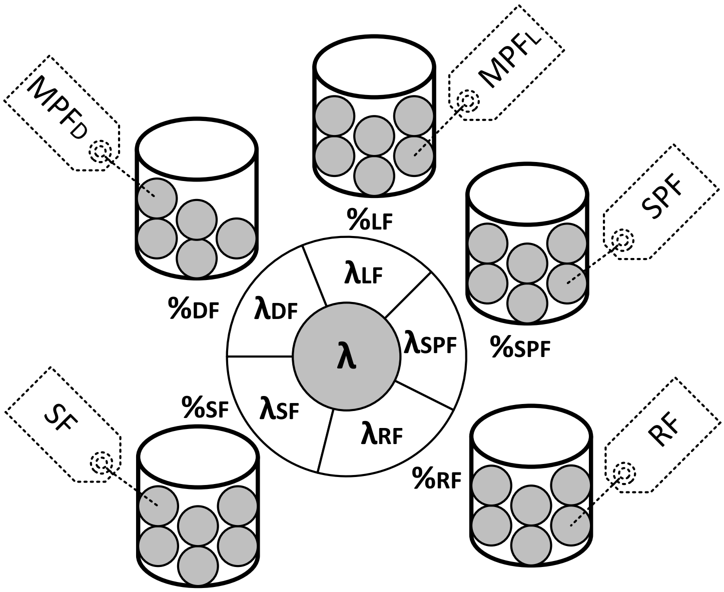


Figure 12 – Failure rate shares associated with the proportion of faults classified in each group.

The interface to the debug module of a microcontroller that is not used during the safety-related operation can have its failure rate marked as safe and its contribution to the hardware architecture metrics ignored, as already mentioned. However, there are faults in the inner logic of the debug unit that may have the potential to violate an SG hence need to be further classified into SPF, RF or MPF. Therefore, different from the interface, the debug inner logic cannot be considered not S-R. At the same time, faults in the inner logic that do not drive signals to the rest of the microcontroller shall be marked as safe, but the proportion of SF cannot be neglected because the element is S-R.

3.2.2. Diagnostic Coverage and the Failure Rate Calculation

After finding the component's parts and subparts not S-R, the safety analysis continues with the identification of SFs within S-R elements. For each S-R element, the further classification of the remaining faults is carried out differently given the presence or not of at least one SM. Without any SM, the faults which the effect propagates out the element are classified as SPFs and the rest as LF. If there is an SM that prevents at least one SPF, then all remaining SPFs are tagged as RFs. The percentage of faults prevented by an SM from directly violating an SG is called either failure mode coverage (FMC) or diagnostic coverage – i.e., DC. In this case, the capability of the SM to cover RFs is symbolized by DC_{RF} percentage. Likewise, DC_{LF} is the percentage of MPFs that are detected by the SM hence not allowing these faults to become LFs.

3.2.2.1. Calculation Steps with Residual and Latent Faults Coverage

Figure 13 illustrate the stepwise calculation of the base failure rate.

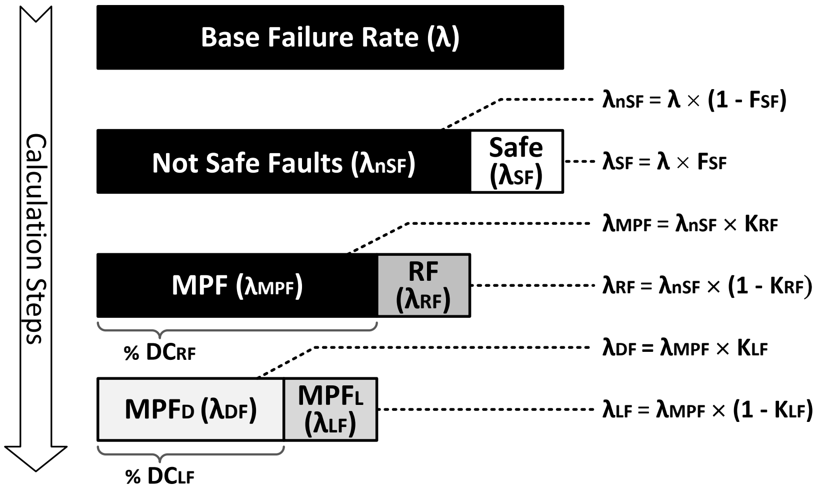

Calculation Steps
Base Failure Rate (λ)
Not Safe Faults (λ_{nSF})
Safe (λ_{SF})
$\lambda_{nSF} = \lambda \times (1 - F_{SF})$
$\lambda_{SF} = \lambda \times F_{SF}$
MPF (λ_{MPF})
RF (λ_{RF})
$\lambda_{MPF} = \lambda_{nSF} \times K_{RF}$
$\lambda_{RF} = \lambda_{nSF} \times (1 - K_{RF})$
% DC_{RF}
MPFD (λ_{DF})
MPFL (λ_{LF})
$\lambda_{DF} = \lambda_{MPF} \times K_{LF}$
$\lambda_{LF} = \lambda_{MPF} \times (1 - K_{LF})$
% DC_{LF}

Figure 13 – Calculation steps of the failure rate proportions when there is an SM that covers RFs as well as LFs.

The DC_{RF} and the DC_{LF} are percentages without a unit, so they can be used to either calculate the two corresponding failure rates fraction or to find the number of faults associated with each of these two classifications. The K_{RF} and the K_{LF} fractions correspond to the DC_{RF} and DC_{LF} percentages, respectively. Additionally, there is the failure rate fraction associated with the safe faults that are represented by F_{SF}. Figure 13 illustrates the steps for calculating the failure rate proportions of a given failure mode when there is an SM that can cover RFs and detect LFs.

In Figure 13 the SFs failure rate fraction (λ_{SF}) has its counterpart indicated by λ_{nSF}. The λ_{nSF} given by (3) and the K_{RF} are used to calculate the failure rates λ_{MPF} and λ_{RF} in (4) and (5), respectively.

$$\lambda_{nSF} = \lambda \times (1 - F_{SF}) \qquad (3)$$

$$\lambda_{MPF} = \lambda_{nSF} \times K_{RF} \qquad (4)$$

$$\lambda_{RF} = \lambda_{nSF} \times (1 - K_{RF}) \qquad (5)$$

Next, in the calculation steps of Figure 13, the MPF failure rate (λ_{MPF}) is split into λ_{DF} and λ_{LF} using the LF DC fraction in the equations (6) and (7), respectively.

$$\lambda_{DF} = \lambda_{MPF} \times K_{LF} \qquad (6)$$

$$\lambda_{LF} = \lambda_{MPF} \times (1 - K_{LF}) \qquad (7)$$

3.2.2.2. Calculation Steps with Residual Fault Coverage

Without an SM to prevent a fault from directly violating an SG, the λ_{SPF} needs to be calculated instead of the λ_{RF}. However, the λ_{SPF} and λ_{MPF} derivation from the λ_{nSF} can not be done using the K_{RF} since there is no SM. Therefore, the standard represents as F_{PVSG}, the failure rate fraction associated with the standalone faults with potential to violate the safety goal (PVSG) as shown in Figure 14.

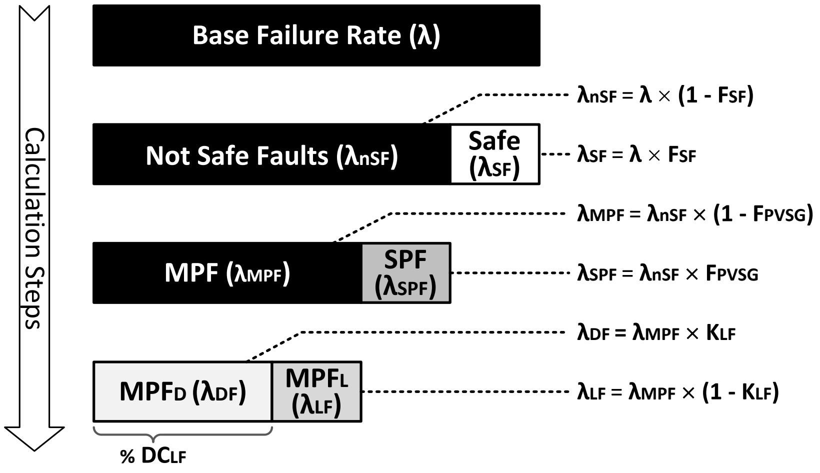


Figure 14 - Calculation steps of the failure rate proportions associated with SPF and MPF when the SM only detects LF.

In the situation illustrated in Figure 14, the λ_{SPF} and λ_{MPF} are given by (8) and (9), respectively. The λ_{DF} and λ_{LF} failure rates in Figure 14 are calculated in the same way as in the situation from Figure 13 since there is an SM responsible for the detection of the MPFs.

$$\lambda_{MPF} = \lambda_{nSF} \times (1 - F_{PVSG}) \tag{8}$$

$$\lambda_{SPF} = \lambda_{nSF} \times F_{PVSG} \tag{9}$$

3.2.2.3. Calculation Steps without Residual or Latent Faults Coverage

For the sake of completeness, Figure 15 illustrates the situation where there is no SM whatsoever thus making λ_{LF} equal to λ_{MPF}. It is important to notice that in each situation, the failure rates λ_{SPF} and λ_{RF} are not used together. Since the base failure rate given by (1) considers λ_{SPF} and λ_{RF}, then whenever one is applicable the other one must be zero on each situation.

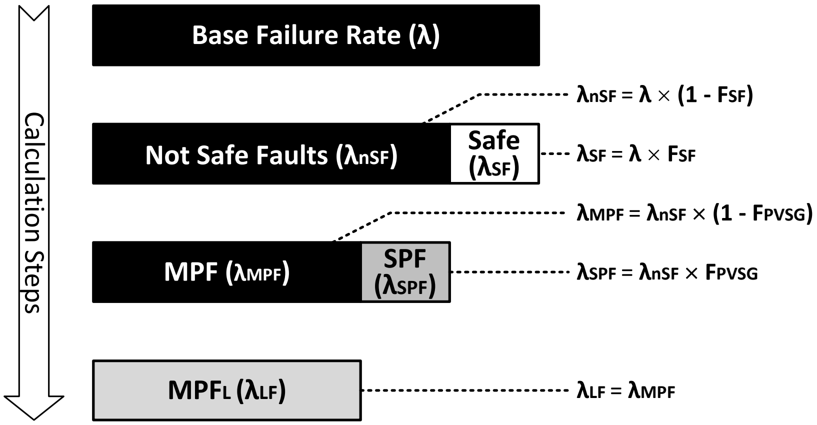


Figure 15 – Calculation steps of the failure rate proportions associated with SPF and MPF for the situation without any SM in place.

3.2.3. Hardware Architectural Metrics Computation

The calculations steps shall be done for each part or subpart of the component being analyzed. After that, the component's hardware architectural metrics SPFM and LFM can be calculated. Therefore, to enable the calculation of SPFM and LFM, the following tasks must be performed:

- Split the component into (sub)parts for the safety analysis;
- Identify the applicable failure modes – e.g., permanent;
- Distribute the component's base failure rate to each element corresponding to its area occupation proportion;
- Determine whether each element is S-R or not;
- Verify the percentage of safe faults if any;
- Indicate whether there are SMs in place;
- Collect the DC_{RF} and DC_{LF} of the SMs in place;
- Derive the λ_{LF} and λ_{SPF} or λ_{RF} from the base failure rate;
- Calculate the component's total $\Sigma(\lambda_{LF})$ and $\Sigma(\lambda_{SPF})$ or $\Sigma(\lambda_{RF})$;
- Compute the SPFM and LFM given by (10) and (11).

$$SPFM = 1 - \frac{\sum(\lambda_{SPF} + \lambda_{RF})}{\sum \lambda} \tag{10}$$

Notice that the λ_{SPF} and λ_{RF} are subtracted from the total failure rate in (11) because the faults that directly violate an SG are not latent anymore thus cannot contribute to LFM. Additionally, it is important to highlight that the greater the F_{SF}, the better provided the base failure rate given by (1) hence justifying a more detailed safety analysis for some cases.

$$LFM = 1 - \frac{\sum(\lambda_{LF})}{\sum(\lambda - \lambda_{SPF} - \lambda_{RF})} \tag{11}$$

The SPFM and LFM target values to achieve are established by the ASIL determined to the SG, which the probability of violation is being analyzed. Table 10 shows the target reference values defined in the ISO 26262 standard. This qualitative detailed safety analysis is only applicable for SG with ASIL higher than A.

Table 10 – SPFM and LFM target values. (Source ISO 26262).

Metrics	**ASIL**	**B**	**C**	**D**
$SPFM = 1 - \frac{\sum(\lambda_{SPF} + \lambda_{RF})}{\sum \lambda}$		≥ 90 %	≥ 97 %	≥ 99 %
$LFM = 1 - \frac{\sum(\lambda_{LF})}{\sum(\lambda - \lambda_{SPF} - \lambda_{RF})}$		≥ 60 %	≥ 80 %	≥ 90 %

3.3. FAILURE MODE EFFECTS, AND DIAGNOSTIC ANALYSIS

At initial phases of the safety lifecycle, an FMEA can be performed in order to describe the item’s function, identify and classify the hazardous events to determine the ASIL, which becomes assigned to the SGs defined to avoid unreasonable risk. An FMEA tool is normally based on complex spreadsheet or database where all the aforementioned data is entered through the FMEA worksheets [33]. This kind of qualitative analysis does not consider the failure rate or DC metrics [34]. Therefore, the quantitative evaluation of the safety metrics is commonly supported by an extended FMEA, named “Failure Mode Effects, and Diagnostic Analysis” (FMEDA).

Table 11 shows a worksheet based on the microcontroller FMEDA example given in the ISO 26262 part [28]. The Table 11 FMEDA uses informative entry values for the sake of the calculation demonstration.

The part and subpart of the microcontroller analyzed are the CPU and its ALU instance. The permanent and transient failure modes are considered for the ALU subpart. The microcontroller has two SMs in place. The mechanism number one (M1) is a monitor of the CPU that can detect faults which cause the software to run out of sequence. The M2 is SBST executed at key-on – i.e., vehicle startup – to detect latent faults. Any fault detected by M1 or M2 activates an output signal of the microcontroller and a system level requirement shall be specified to make proper use of this signal – e.g., go to a safe state or alarm the driver. M1 can also detect transient faults hence the CPU can be reset in order to resume the fault-free state. If the error persists after reset, then the fault is considered permanent. Thus the FMEDA has no LF entry for the transient failure mode. No SPF entry appears in the example because there are SMs for all failure modes. If a subpart without any SM is added to the FMEDA, then its row would have 0% as DC^P_{RF}, and the failure rate associated with the SPF could be placed in the λ^P_{RF} column. This is possible because λ^P_{SPF} and λ^P_{RF} are complementary with respect to the contribution to the SPFM and LFM calculations given by (10) and (11), respectively.

Table 11 – Microcontroller FMEDA examples adapted from the ISO 26262.

Design (μC)				**Permanent Failures**								**Transient Failures**				
						SPF/RF			**MPF**					**SPF/RF**		
Part	**Subpart**	**S-R** (Y / N)	**FM**[a] (P / T)	$\boldsymbol{\lambda^P}$ (FIT)	$\mathbf{F^P_{SF}}$ (%)	**SM**[b] (Ref.)	$\mathbf{DC^P_{RF}}$ (%)	$\boldsymbol{\lambda^P_{RF}}$ (FIT)	**SM** (Ref.)	$\mathbf{DC^P_{LF}}$ (%)	$\boldsymbol{\lambda^P_{LF}}$ (FIT)	$\boldsymbol{\lambda^T}$ (FIT)	$\mathbf{F^T_{SF}}$ (%)	**SM** (Ref.)	$\mathbf{DC^T_{RF}}$ (%)	$\boldsymbol{\lambda^T_{RF}}$ (FIT)
CPU	ALU	Y	P	25	20	M1	90	2	M2	60	7.2					
		Y	T									15	20	M1	90	1.2
			Totals	$\Sigma(\lambda^P)$				$\Sigma(\lambda^P_{RF})$			$\Sigma(\lambda^P_{LF})$	$\Sigma(\lambda^T)$				$\Sigma(\lambda^T_{RF})$

[a] The Failure Modes (FM) considered are Permanent (P) and Transient (T). The initial letters are used to mark the entries associated with each FM.

[b] The example has two SMs with the reference names: M1, which can cover permanent and transient residual faults; and M2 that detects latent faults.

After the Table 11 is complete with all relevant parts and subparts of the microcontroller, then the summations $\Sigma(\lambda^P)$ and $\Sigma(\lambda^P_{RF})$ can be used in (10) to calculate the microcontroller's SPFM associated with the permanent failure mode. The same can be done with $\Sigma(\lambda^T)$ and $\Sigma(\lambda^T{}_{RF})$ to calculate the SPFM for transient faults. To find the LFM, the summation $\Sigma(\lambda^P{}_{LF})$ together with $\Sigma(\lambda^P)$ and $\Sigma(\lambda^P{}_{RF})$ can be used in (11). Notice that such detailed analysis provided by FMEDA tools allows identifying which parts or subparts that most contribute to the total failure rate or that need a higher DC thus allowing to specify new requirements to achieve the SPFM and LFM requisites.

The Appendix D in the ISO 26262 – Part 5 [32] shows the generic hardware of an embedded system – e.g., power supply, clock, digital and analog I/O, and specific to semiconductors there are: ALU, register bank, interrupt handling, etc. For each generic element, the standard provides the common SMs and their typical achievable DC for different failure modes of the element. This start point information can be used as guidance for the selection of the SM and the target DC to achieve in order to comply with the SPFM and LFM derived from the ASIL. However, the techniques mentioned in Appendix D are not exhaustive, and other SM might be used. Additionally, there are constraints that the appendix cannot entirely consider when evaluating the DC of a generic SMs – e.g., the quality of the self-test executed by SBST, the periodicity that an SM shall be triggered to diagnose LF. Therefore, any kind of SM can be used provided evidence that corroborates the DC claimed.

3.4. RELATED WORK: FAULT INJECTION APPLICATION

Fault injection is broadly accepted to evaluate the response of a circuit in the presence of faults [35]. Thus, it plays a key role as a method to verify fault tolerance techniques integrated into the design of resilient systems. ISO 26262 strongly recommends fault injection to evaluate the completeness and correctness of the SMs for the higher ASILs – i.e., C and D. The same level of recommendation, independent of the ASIL, applies to the functional verification of the SMs. At SoC level, some SMs may only be triggered by the injection of a fault in the system since an external TB can not directly control SoC's internal subparts' inputs and outputs. Additionally, constraints like “Fault Reaction Time” and “Fault Tolerant Time Interval” must be verified to assure compliance with the requirements. Figure 16 illustrates the mentioned SMs' time constraints.

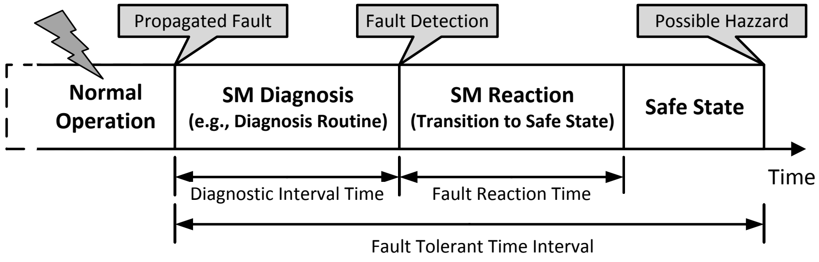


Figure 16 – SM’s time constraints. (Adapted from ISO 26262).

Yogitech S.p.A is a member of the technical committee for the ISO 26262, and it has announced to be the lead of the ISO 26262 Part 11 [36]. This company provides state-of-the-art solutions for semiconductor safety design including fault injection campaign to verify DC. The academic contributions [37] [38] [39] from the Yogitech staff give a detailed introduction with regard the usage of fault injection to assess safety at semiconductor level. On the other hand, DC verification based on fault injection is not well promoted in the ISO 26262 standard. Yogitech is a safety consultant company experienced on guiding the customers throughout the safety certification process, therefore, its published research permits to better understand how fault injection is used within the safety context. Meanwhile, it serves as a reference material until ISO 26262 – Part 11 is not officially released, which will bring more details on fault injection and other particularities of safety application at semiconductors level.

Yogitech indicates how the DC verification is performed using fault injection in [37], but the hardware architectural metrics calculated based on the IEC 61508 – not equal to the ISO 26262 metrics. The certification body TÜV SÜD has approved the so-called “fRMethodology” presented. A “sensitive zone” is an identified site of the design in which the faults within its input cone converge, henceforth, leading to the activation of failure mode – see the sensitive zone illustration [38] in Figure 17. The design is partitioned into sensitive zones, and their failure rates are calculated and stored in an FMEDA database. The base failure rates of the gates within sensitive zone cones are defined according to the failure mode – e.g., permanent in glue logic during power-up/down or normal operation, transient in registers or logic. The number of gates extracted from the IC description and their elementary failure rates among other parameters – e.g., frequency of use (only for transient) and DC – are stored in the database too. This information is used to calculate the hardware architectural metrics of the IEC 61508. Next, fault injection is used to validate the FMEDA database. Evidence that the workload – i.e., TB and tests – used is appropriate needs to be provided in order to guarantee the sensitive zones are properly stimulated while injecting the faults. Coverage collection, the number of faults injected, test's run-time, and other profiling information are used to check the confidence of the fault campaign results.

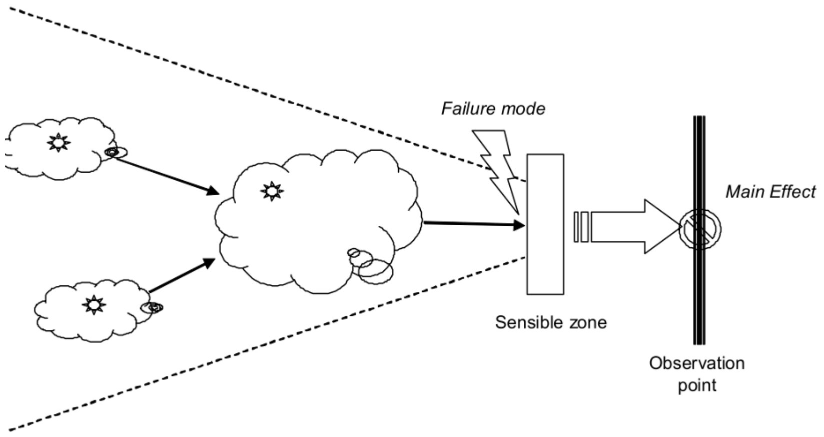


Figure 17 - Sensitive zone illustration (Source: [38]).

Prior the ISO 26262 release, Yogitech was already working on DC assessment based on fault injection at semiconductor level [38]. In collaboration with the *Politecnico di Torino*, a new tool was proposed considering the requirements specified by IEC 61508 for safety-related systems [39]. The tool is intended to work with any functional verification environment, regardless the EDA tools available. The goal of the presented solution is to enable the verification of SMs at different abstraction levels by leveraging the functional simulators used in the verification environment. For that reason, the fault models had to be implemented according to the existing tools on each environment setup due to the lack of fault injection support by the standard simulators available at that time [40]. In the tool proposal, an example of the SEU model is provided using the Functional Verification Language *e* (IEEE 1647) for verification flows containing the Incisive® Specman® Elite. Figure 18 shows the Fault injection environment proposed in [39].

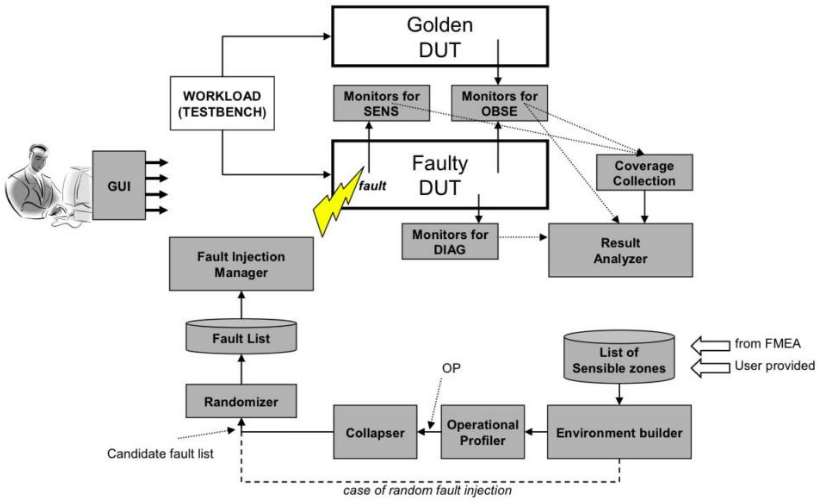


Figure 18 – Fault injection environment proposed in [39].

After the FMEA is available, the SMs are implemented, and the sensitive zones are extracted, two other fault campaign elements are defined: the observation and the diagnostic points. There are monitors for these elements and also for the sensitive zones – in Figure 18, they are referred as OBSE, DIAG, and SENS monitors, respectively. The faults within the sensitive zone's cone that do not trigger the SENS monitor are irrelevant for the analysis since they do not perturb the functionality. On the other hand, the number of faults, which propagate through the SENS until the OBSE monitors, contribute to the fraction of dangerous faults. Therefore, special registers or primary outputs of the design under test (DUT) are commonly selected as observation points. The DIAG monitors are typically located at the output of the SMs thus providing the information of which faults were covered by the diagnostic functions implemented in the DUT.

The golden – i.e., fault free – instance of the DUT is executed in parallel to the faulty instance, and both share the same workload stimulus. The workload selected is either device or mission oriented – i.e., for generic usage of the device or a specific application, respectively. When the former kind of workload is selected, then the number of faults to inject can be reduced by not considering parts of the circuit that are not used by the application. With the workload defined, the fault-free run is analyzed in order to identify the most suitable moments to inject the faults – e.g., assure that the transient faults injected in a memory element precede a read access hence the effect can propagate before being override by a write command. The workload completeness is measured by the amount of SENS and OBSE monitors that are excited at least once This is done by enabling toggle coverage where the monitors are located [38].

For each fault, the simulation stops at the injection time, then the fault model is executed, and the simulation run continues until the test finishes. At the same time, the OBSE and DIAG monitors keep collecting data that will be analyzed after all faults are injected in order to calculate the DC. However, an exhaustive analysis is typically not feasible, hence only a set of all faults can be injected. Therefore, it is important to select the optimal fault candidates thus avoiding the injection of faults that have no effect. The experiments show that the fault list generation based on the workload profile reduced the number of faults to inject by 50%. The test case used is a simple router with one input and three possible output channels to where the data packets can be routed. Each channel has a buffer working as a queue with 16 words of 8 bits. Table 12 shows that the router design has 9,648 fault candidates [39]. After analyzing the workload, the set of faults to be injected was reduced to only 4,886 faults.

Table 12 – Average fault injection time of the experiments on [39].

Design description	**Number of faults**	**Average fault injection time**
Router (without optimization)	9,648	4.1s
Router (50% fault set reduction)	4,886	3.7s
32-bit RISC Processor	25,000	56.6s

Eleven hours of simulation time were necessary to inject all faults while the optimized set of faults took about five hours to complete [39]. This overall time corresponds to 4.1 and 3.7 seconds, respectively, to the average simulation time for each fault injection – see Table 12. Although the overall time is drastically reduced, the optimization does not provide a significant improvement on the average fault injection time. Additional experiments with the tool presented have been done during the validation of real safety critical design based on a 32-bit RISC processor. Table 12 shows that 25 thousand faults were selected to be injected – the original number of fault candidates was not disclosed. In this example, the average simulation time required for each fault injection was almost one minute. Thus showing that the fault simulation time is independent to the optimization presented. Another relevant consideration is that the tool reuses the available functional verification tools in order to minimize the flow setup time.

The Industrial Technology Research Institute (ITRI) in collaboration with the *National Tsing Hua University* has also published their experience on DC assessment based on fault injection [41]. While discussing the related works, the paper reaches a common understanding "that the description of the probabilistic hardware metrics in ISO 26262 is intrinsically not easy to follow". Additionally, it highlights that other researches in the literature do not disclose how the safety evaluation is performed, which support the proof for claimed ASIL achieve. For instance, the "safety analysis report" – with the component's FMEDA – of the ISO 26262 certified microprocessors from Texas Instruments Inc. (TI) cannot be accessed without a non-disclosure agreement (NDA) [42]. More details about the TI's safety evaluation are only available to customers who require tailored safety analysis according to the target application.

Given the lack of references, the ITRI limits its paper contribution on reporting their experience on the functional safety assessment of a MIPS-like microprocessor. Therefore, no kind of optimization or novel methodology is proposed. However, its detailed description provides deep insight on how the DC evaluation can be performed when considering the permanent failure mode. The TetraMAX ATPG suite is used to inject SAT faults in order to validate the initially estimated hardware architecture metrics what refers to the permanent failure mode. The paper provides a snippet from the SN 29500, which is used to perform the base failure rate estimation of the microprocessor subparts. Table 13 contains the data from the SN 29500 snippet.

Table 13 – SN 29500 hardware failure rate estimation. (Adapted from [41])

# Gates	≤ 1K	≤ 10K	≤ 100K	≤ 1M	in °C
CMOS	25				50
CMOS		30			60
CMOS			50		70
CMOS				80	80

Each subpart of the MIPS-like component gets a base failure rate derived from Table 13 according to their number of gates. However, the

2004 version of the SN 29500 standard provides more conservative estimations than the reliability data found in more recent standards [28]. Therefore, deriving the failure rate for each subpart from Table 13 can result in an overall failure rate too conservative. Alternatively, the SN 29500 could be used to calculate the overall base failure rate, which then is allocated to the subparts according to their percentage of gates relative to the whole component. Table 14 was prepared to highlight the difference between the two mentioned approaches of estimating the base failure rate. Even using the former calculation approach, the numbers are still ten times more conservative than the failure rates utilized in a similar example found in the ISO 26262 [28].

Table 14 – Two different approaches for calculating the base failure rate using the conservative SN 29500 reliability data.

Block	**Gates**	**Base Failure Rate (FIT)**	
Source data [41]		Estimated for each subpart	Derived from the overall estimation
Register File	1,209	*30* [a]	1.066
Instruction memory	27,192	*50* [a]	23.977
Decode unit	128	*25* [a]	0.113
ALU	208	*25* [a]	0.183
Data memory	27,186	*50* [a]	23.972
Write-back unit	35	*25* [a]	0.031
Pipeline register	746	*25* [a]	0.658
Totals	**56,704**	**230**	***50*** [a]

[a] failure rates calculated based on the SN 92500 data.

Another usage of fault injection in the safety context is the evaluation of dependent failures [43]. For example, faults on the clock or reset signals can result in a “common cause failure” (CCF) of the redundant parts of a system. Therefore, the possible CCFs must be identified since the associated faults can have a global impact on a large area of the component [38]. Fault injection using higher level design description can support the identification of CCF [44]. However, this is out of the scope here.

3.5. CHAPTER REMARKS

This chapter presented how the fault injection appears in the ISO 26262 context. After the component's FMEDA is complete, the subparts base failure rates are allocated, and the expected SMs' DC are defined, it is possible to calculate the hardware architectural metrics. Fault injection is then used to validate the FMEDA results by evaluating the DC estimated. However, ISO 26262 does not provide technical details on how this evaluation shall be performed. Additionally, this process is typically tight to the design application and the verification environment available. Therefore, details about the fault injection based on DC evaluations at semiconductor level on real designs are not common in the literature since it would disclose too much information about the design and the setup used. In the next Chapter, the dedicated functional verification machine is introduced. The DC evaluation approach proposed leverages this dedicated platform to accelerate the fault injections. However, not all faults can be accelerated without modifying the design description, so the next chapter also describes the algorithm developed to select suitable faults to run on the dedicated platform.

4. FAULT INJECTION CONCEPTS AND TECHNIQUES

As already mentioned, safety – main focus on this research – is just one of the attributes that are encompassed by the dependability concept [45]. The attributes together with the threats and means are the three elements that compose the tree shown in Figure 19, which is traditionally used to summarize the dependability concept even nowadays [46].

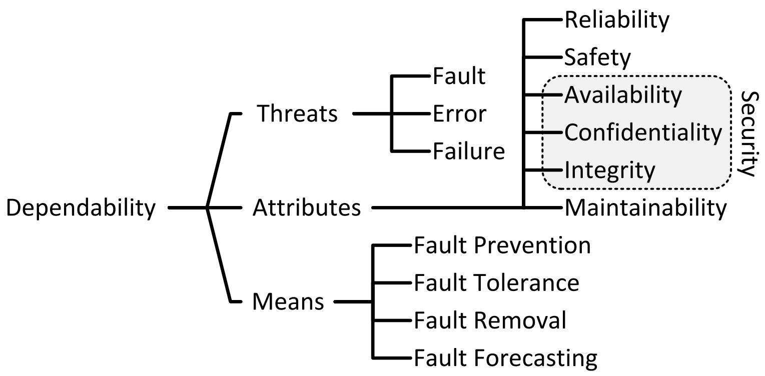


Figure 19 – Dependability tree. (Sources: [45] [46]).

In order to avoid unreasonable risk and hence provide functional safety, SMs are added to the design as means of mitigating the dependability threats, which are: faults, errors, and failures. Figure 20 illustrates how the threats correlate, and it also contains a synthesis of the most common definition for fault, error, and failure found in the literature [47] [45] [48] [13] [46].

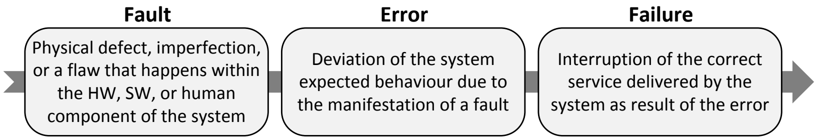

Fault
Physical defect, imperfection, or a flaw that happens within the HW, SW, or human component of the system
Error
Deviation of the system expected behaviour due to the manifestation of a fault
Failure
Interruption of the correct service delivered by the system as result of the error

Figure 20 – Summarized dependability threats explanation found on [47] [45] [48] [13] [46].

The correlation between fault, error, and failure presented in Figure 20 can be replicated for each abstraction level in order to exemplify how faults in the component level can lead to vehicle failures, as shown in Figure 21. Common practices of systematic faults prevention are coding guidelines, specification review, code/functional verification coverage, manufacturer tests, among others means. Memory protection and SMs based on some kind of redundancy are examples of techniques used to tolerate random faults during the operation lifetime. When there is an error that could not be corrected, then it is important that the system fails in a safe manner thus reaching a predictable, and safe state – e.g., a redundant shut-off of the cruise control system [49].

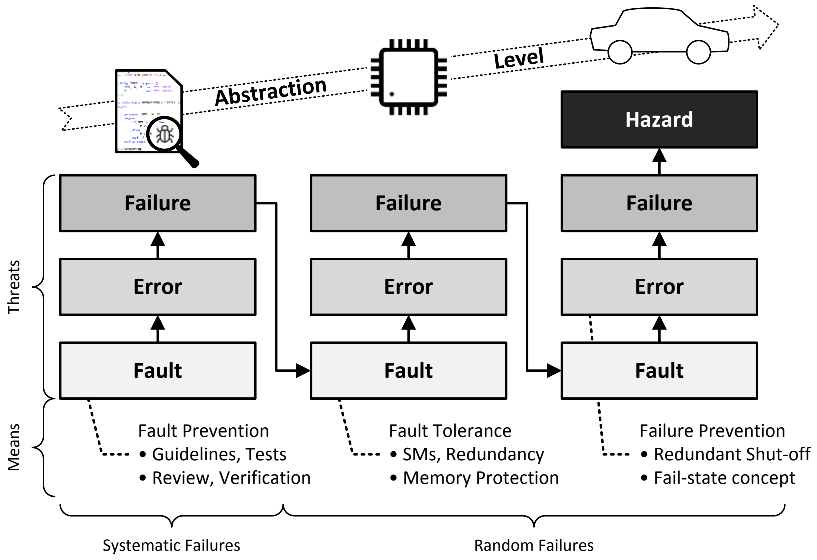


Figure 21 – Dependability threats at different level (adapted from [50]).

4.1. TRANSIENT FAULTS

At digital semiconductor level, permanent and transient are the two failures modes that most appear in the ISO 26262 hence also considered here – as pointed out before. The history shows that the technology scaling has a significant impact on the circuit vulnerability against single-event-effects (SEE) caused by radioactive particles even for applications operating at sea level [51]. Differently, of a hard error that results in permanent damage, a soft-error is a reversible upset of a circuit element, which is the consequence of a SEE fault, such as, SEU and SET – hence the failure mode name “transient” that encompasses these kinds of faults. Figure 22 shows the electron-hole pairs generated by the particle strike at a sensitive node such as the drain of an OFF-transistor in a CMOS circuit [52] [53] [54] [55]. Therefore, the charge collected by the particle strike can generate a positive or negative voltage pulse depending whether the particle hits an OFF-PMOS or an OFF-NMOS, respectively. Figure 22 illustrates a CMOS inverter with the input connected to the ground, hence its output remains high until a SET fault hits the OFF-transistor of the combinatorial gate producing a voltage glitch.

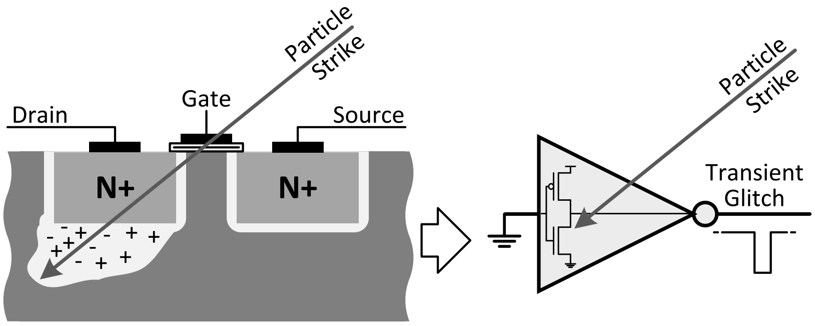


Figure 22 – Voltage glitch at the output of logical gate caused by a SET fault (adapted from [52] [53] [54]).

The glitch duration, for example, can determine whether a SET can become a soft-error via the fault effect capture by a storage element – e.g., memory cell, flip-flop, latch. Not all particles have the energy to cause transient faults and not all SET can result in a soft-error since its propagation can be masked due to logical, electrical or timing reasons – e.g., SET propagation falls outside the flip-flop latching window [52] [56]. On the other hand, an SEU fault corresponds to a particle that hits a storage element directly instead and bit-flip (upset) its content. Therefore, an SEU fault always results into a soft-error provided its effect deviates the correct behavior of a sequential circuit element. For that reason, the SEU term is often found in the literature being ambiguously used as a synonym for soft-errors [51].

4.2. PERMANENT FAULTS

There are SEEs which may persist until the circuit is reset or even cause irreversible damage, thus resulting in a hard-error – e.g., single event latchup (SEL). In addition to radiation effects, manufacturing defects and circuit aging are also related to permanent – and intermittent – faults that can lead to hard-errors [35]. With the continuous technology scaling, new IC designs do not only need to cope with the vulnerability to transient faults, but also consider the device lifetime, which tends to wear-out early, and the residual defects that escape manufacturing tests due to the transistor density [57] [58] [59]. Among the permanent faults, the most well-known fault model is SAT, which covers many physical defects [60] [61]. Safety-related researches and the ISO 26262 standard suggest that some techniques for SAT faults – e.g., N-detect and gate-exhaustive – are used to detect other common permanent fault models like bridging or stuck-open at transistor switch level [28] [37]. Therefore, any optimization of the SAT fault injection can be beneficial to other SAT based techniques.

SAT faults account for many transistor level faults that can be modeled at GL by making an input, or an output of a gate stuck at a constant value, either logic 0 (SA0) or logic 1 (SA1) [62]. Figure 23 illustrates a multiplexer (MUX) at GL in order to highlight the locations where the SAT faults shall be considered according to the industry compatible fault model [63]. This compatibility mode includes SAT faults in the primary inputs (PI) and primary outputs (PO) of the circuit. The MUX example presented has 15 fault sites (or nodes) thus corresponding 30 SAT faults in total. In Figure 23, the circuit considered has a MUX implemented connected directly to the PIs and POs, hence eight SAT faults are associated with the primary ports of the circuit.

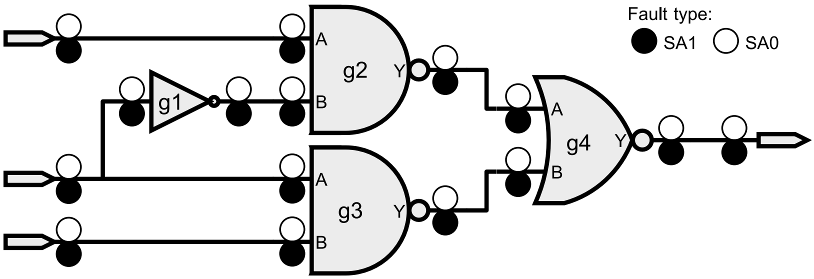


Figure 23 – MUX circuit with all possible SA0 and SA1 faults indicated.

4.2.1. Fault activation and propagation concepts

Fault coverage corresponds to the number of faults that can be detected by the manufacturing test, thus providing the quality of the selected test set. However, even after the simplification achieved with SAT fault model in comparison to transient faults, the number of faults to be detected can be too many for a complex circuit. Collapsing and testability analysis are optimizations used to reduce the SAT fault-set. Testability analysis statically checks if there are SAT faults which always produce the same output as a fault-free circuit – i.e., there is no possible test that can detect these faults. For a fault to be testable, it must be possible to activate and then propagate its effect to an observation point – e.g., a primary output. A fault can be activated when there is a test capable of driving the value opposite to the one which the fault node is stuck-at. After activation, the fault effect must not be masked in order to propagate until a detection point. Figure 24 illustrates the activation and propagation concepts.

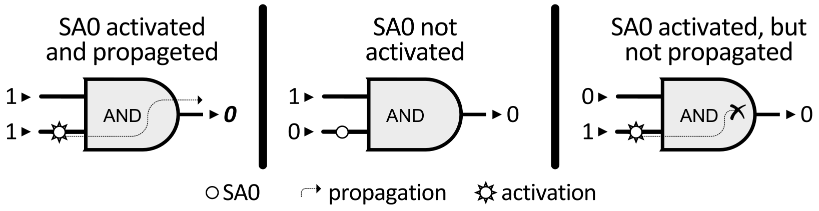


Figure 24 – Activation and propagation concepts.

In Figure 24, an SA0 fault occurs on one input of the AND gate, and consequently, the gate’s output gets fixed to ‘0’. The SA0 effect at the AND gate output can only be noticed when the inputs are set to ‘1’, what would make a fault-free AND gate to output ‘1’, and not zero as shown in the example. For this to happen, the fault must be activated by driving ‘1’ to the gate’s input where the SA0 is located. Additionally, the fault effect shall be able to propagate, which means that the stimuli to the other inputs of the gate should not block the SA0 effect. On the other hand, there are some cases which the activation or propagation are not possible as shown in Figure 25.

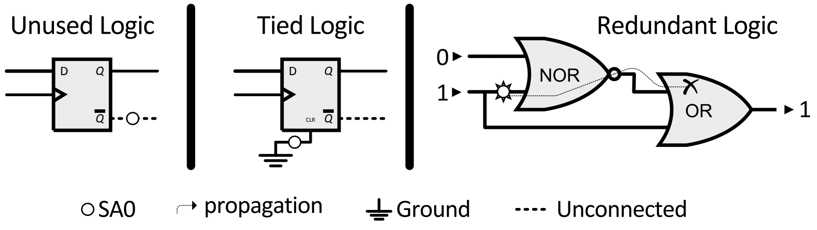


Figure 25 – Untestable faults (adapted from [63]).

4.2.2. Fault collapsing and testability

The untestable faults correspond to possible faults in the circuit model which no test that can make the fault detected at an observation point. The reasons for that can be: SAT with the same value of a tied node, hence never activate; a SAT located on an unused logic thus cannot propagate to anywhere; and the existence of redundant logic, which can block the propagation of the SAT fault. In the end, untestable can be deleted from the list of faults to detect [63].

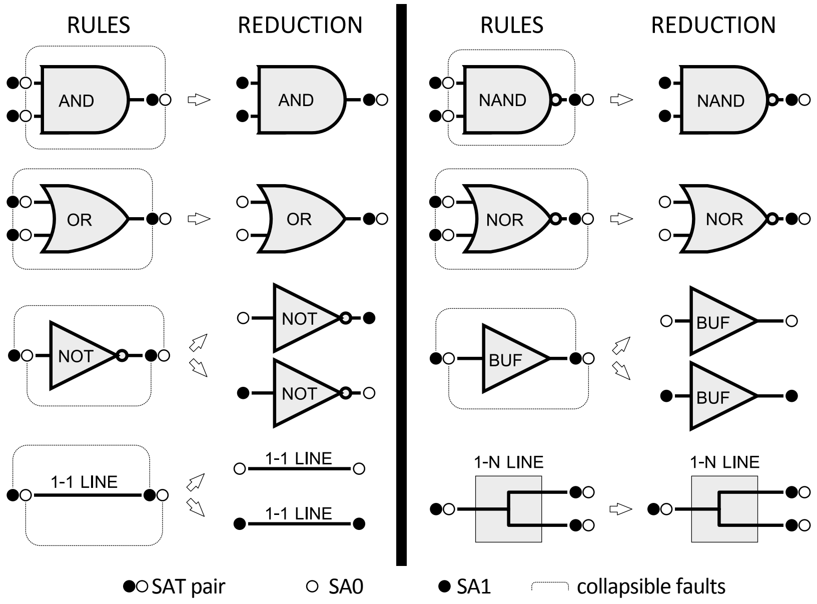


Figure 26 – Local fault collapsing rules at GL (adapted from [62]).

Also looking to reduce the fault set, collapsing techniques group faults that produce identical (or equivalent) effect at the output thus being indistinguishable from each other. In other words, there is no test that can independently detect each fault from the equivalent group. For instance, if one fault of the group is detected, then it means that all equivalent faults are also detected. Collapsing can reduce the fault set by 50 to 60% [64]. Figure 26 illustrates the gate-oriented fault collapsing rules.

Figure 26 identifies which faults are collapsible – i.e., equivalent faults are connected by a dashed line – within the basic element of a circuit described at GL. One of the equivalent faults must be selected to represent its group after collapsing. Therefore, after reduction of the faults due to collapsing, the fault towards the circuit output is selected from the equivalent group. The selected fault to represent its equivalent group is called the prime fault.

4.2.3. Collapsing exercise and the backward propagation requisite

The rules in Figure 26 applicable to gates, highlight which faults are logically equivalent. These rules are different from the "1-1 LINE" rule, which collapses "redundant" faults added by the industry compatible SAT fault model. For that reason, the gate-oriented collapsing starts with the "1-1 LINE" being applied. The result of the "1-1 LINE" rule in the MUX example is presented in Figure 27.

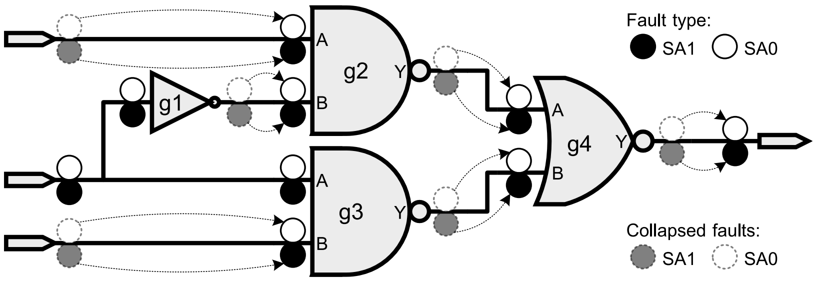


Figure 27 – Applying "1-1 LINE" collapsing rule in the MUX example.

After removing the "redundant" faults, each line segment in the circuit – where "1-N lines" is equal to N+1 segments – has a fault pair thus totalizing 18 SAT faults as shown in Figure 28. Next, the logic collapsing rules can be applied to the MUX circuit thus further reducing the fault set.

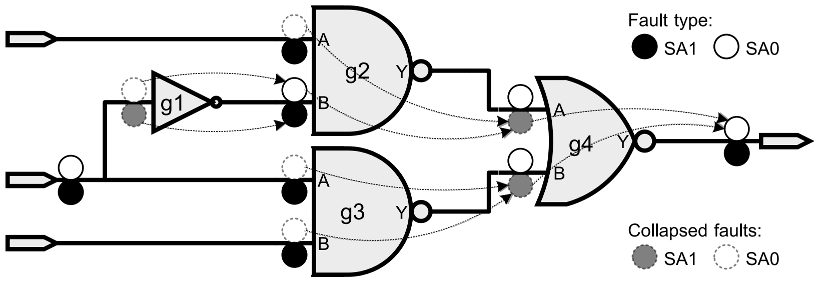


Figure 28 – Applying the logic collapsing rules in the MUX example.

Figure 29 shows that after collapsing, the final fault set for the MUX example contains only ten faults, representing the 30 faults initially considered.

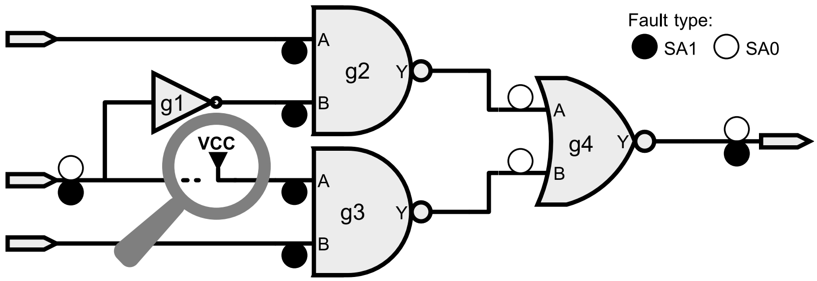


Figure 29 – Final set of SAT faults after collapsing.

In Figure 29, it is important to notice also that the SAT fault pair located in between the circuit’s PI and the line split, is prevented from being collapsed by the “1-N LINE” rule provided in Figure 26. This rule assures that the SAT fault model is observed, that is, the effect of the SA1 fault at *g3.A* shall not backward propagate and reach the gate *g1*. In other words, the line driving a SAT fault node must be disconnected – or isolated – as illustrated in Figure 29 [63]. The magnifier in this figure highlights an analogy of an SA1 fault isolation done by disconnecting the SAT fault input from the original driver and then connecting to VCC. The fault isolation concept is relevant for the methodology presented.

4.3. FAULT INJECTION TECHNIQUES

The semiconductor industry has extensively accepted fault injection over the years, and its tradition comes either from the DfT field or the know-how of robust circuit designs [40] [35]. Fault grading or fault simulation is a well-known fault injection application, where the goal is to determine the fault coverage of a given test set in order to guarantee that most defective devices do not escape the manufacturing tests. Another common usage of fault injection is for the verification of fault-tolerant techniques used for hardening the circuit design. Today, there are many fault injection solutions not just targeting this two most common application, but different use cases also. For example, testbench (TB) qualification (TB-Q) explores fault injection in order to expose bugs in the checking – e.g., assertions, coverage bins, transactions – implemented in the verification environment [65]. Functional safety is another use case, where the ISO 26262 recommends fault injection with different semantics across the safety lifecycle. In this former usage of fault injection, the goals can be for instance evaluation of the hardware architectural metrics and DC assessment as already discussed in this research.

The different fault injections techniques found in the literature are commonly classified by the design representation type that underlies the fault injection approach [66] [48] [35] [14] [67]. Therefore, the fault injection can be based on:

- Hardware – it relies on the existence of the product prototype where the faults are physically inserted by forcing faulty conditions through the device pins or at specialized facilities with access to radiation or laser equipment to reproduce hostile environments;
- Software – applicable to data processing designs, where some hardware faults can be imitated as software errors like incorrect instruction code, wrong transaction sequence, invalid message payload, etc.;
- Simulation – widely used technique, where a simulation tool is employed to inject faults in the design model at different abstraction levels (e.g., RTL and GL) thus not requiring a prototype available;
- Emulation – commonly based on configurable devices like FPGAs, with the objective of delivering performance rates that are not feasible on simulation-based techniques;

- Hybrid – mixture of two or more techniques trying to leverage and combine the best features of each of them. For example, fault injection using on-chip debug access available on the microprocessor to mimic hardware faults at software level.

4.3.1. Fault injection aspects to consider for safety assessment

Different goals require support to different requirements that associated with the target use-case. For example, hardware-based fault injection is not suitable for TB-Q since it is available far too late for when it is needed. Therefore, not all fault injection techniques are applicable in the functional safety context. For functional verification of SMs for example, different abstraction levels can be used. However, for the evaluation of the hardware architectural metrics, the latest design model before sign-off shall be used – e.g., GL netlist post-layout. The confidence of the results achieved with a fault injection campaign is proportional to the completeness of the stimulus used, the total of faults injected, and level of detail of the design description [28]. On the other hand, the ISO 26262 does not require an exhaustive injection neither is entirely rigid about the used abstraction level. So, there is some flexibility, but it can be leveraged only if there is adequate justification. For example, fault injection based on RTL is acceptable, provided sufficient correlation with GL. However, this may be adequate for SM verification or initial DC assessment only. For the final evaluation of the SPFM and LFM, an auditor may not accept results based on RTL without an irrefutable argumentation. Such argumentation may not be feasible if there are solutions already available that provide more reliable results – e.g., that inject more faults on the detailed design model. Therefore, the best-effort concept is not just important for convincing a safety certification auditor. It can also result in a commercial advantage for a company that invests in a new solution, which might force the competition to accommodate the state-of-the-art in order to avoid future liability.

In the functional safety context, the general requisites for fault injection targeting SM verification and DC assessment at semiconductor level, are:

- Feature the injection of fault models associated with the permanent and transient failure modes including the definition of their parameters – e.g., fault injection time and fault duration;

- Support detailed design models, but also abstract descriptions for early assessment – e.g., GL and RTL as mentioned in the ISO 26262;
- Definition of different observation points in order to distinguish faults detected by SMs from faults that propagate through the functional output without prevention;
- Provide timing information with respect to fault observation thus allowing to check if the fault tolerant time interval is respected;
- Multiple fault injection thus enabling the verification of SM that can handle more than one fault – e.g., Error Detection and Correction (EDC) mechanisms;
- Usage of functional tests thus allowing an assessment closer to the real application.

4.3.2. Optimizations and available tools

Considering the general requisites, there are optimizations for reducing the overall time of fault injection campaigns, such as:

- Fault collapsing and testability analysis for SAT faults and equivalent approaches for other fault models;
- Reusability of the functional tests and the existing verification environment;
- Ranking of the most suitable test for the fault injection campaign;
- Optimal fault injection time based on the application;
- Fault injection criteria to stop the injection run when it occurs;
- Campaign threshold to conclude the campaign as soon as the goal is reached;
- Distribution, parallelization, acceleration, and many others;

Recently, the major EDA vendors have noticed that fault-injection solutions featuring these characteristics were missing in the market. Tailored extension of existing tools started to be released in the EDA business aiming to cope with the challenges of the functional safety assessment. Incisive Functional Safety Simulator (IFSS) is a perfect example [20] [21]. As an add-on to the vendor’s main HDL simulator, IFSS provides specific features reflecting the standard requisites. Competing with IFSS, there is Certitude, which was originally created for

TB-qualification based on fault mutation-technique, and now also support to the ISO 26262 fault models [68] [65] [8].

IFSS is one of the tools used in this research. Since it is integrated to an event-driven simulator, IFSS can benefit from the latest verification methodologies and TB languages by reusing the existing functional verification environment. It can be employed throughout the design development due to the fault injection support at RTL and GL hence suitable for a functional safety flow. Even more important is the reduction of the fault campaign setup achieved by not requiring the generation of test vectors or being restricted to structural GL netlist. These two characteristics are common restrictions imposed by the most popular fault simulation algorithm in commercial DfT tools, which takes advantage of the GL netlist modularity to simulate only the parts affected by each fault, concurrently [69] [40] [70]. Additional to concurrent fault simulation, there are others DfT-oriented algorithms for performance improvement compared to standard event-driven simulation, but still with limitations to support behavioral models and transient faults. On the other hand, IFSS can leverage the most sophisticated distributed resource management (DRM) tools to run many faults simulation at the same time. However, despite the simulation-based technique utilized, an exhaustive safety assessment is practically impossible given the today's design complexity and the total of faults to consider. Statistical methods like fault sampling permit to randomly select a feasible number of faults from the population and yet obtain relevant metrics with sufficient accuracy [71]. The ISO 26262 suggests statistics for many other situations including fault sampling, which are out of scope for this study and will not be covered.

4.4. FAULT SIMULATION ACCELERATION

4.4.1. FPGA Techniques

Emulation-based fault injection is typically proposed to overcome the long runtime of techniques based on simulation [13] [14]. In order to speed-up the fault injection run, FPGA-based techniques are often proposed as they can efficiently emulate the circuit. In the FPGA-based flow, the design must be synthesized to the specific target device, which is later configured with the resulting bitstream – i.e., the "synthesis image". If the reproduction of the design copy containing a fault requires modification of the design description, then the synthesis process naturally becomes a bottleneck for the FPGA-based fault injection emulation. Different approaches have been proposed over the years in order to avoid this bottleneck thus enabling the benefit of emulation-acceleration [72] [73] [74] [75]. The general ideas of these approaches are:

- Substitution of GL cells by highly controllable blocks that allows to enable or disable the error condition by fault injection manager running on the platform with minimal interaction of the computer host. This approach is suitable when intrusion is not a concern;
- Dynamically partial reconfiguration featured in some FPGAs is explored aiming to reduce the intrusiveness of the previous approach. The fault injection controller is also implemented in the FPGA or the emulation platform thus minimizing the interaction with the host. However, these approaches rely on the controllability provided by the FPGA configuration resource, thus limiting the support to fault models;
- By manipulating the bitstream, it is possible to reconfigure part of the circuit and replace it with its faulty version. In order to do that, the design running in the FPGA must be correctly stopped at the fault-injection instant and then resumed after the bitstream portion has been overwritten. The bitstream manipulation and reconfiguration must be executed in the platform hence close or even inside of the FPGA due to the communication bottleneck with the host. Additionally, a full mapping between the synthesis image and the circuit implemented in the FPGA must be disclosed by the manufacturer to permit proper bitstream manipulation.

The application of FPGA-based approaches is mentioned in the ISO 26262. They are suitable for SM verification for example in order to maximize the performance of the fault campaign execution. However, the validation of the hardware architectural metrics shall be done using the most detailed model viable of the design. It might not be possible to show sufficient correlation between the synthesized model running in the FPGA and the latest GL netlist targeting another technology. Even if feasible, the collection of correlation evidence to convince the auditor is not a trivial task. Therefore, intrusive approaches like the replacement of GL cells to enable fault injection should be avoided. Less intrusive FPGA-based techniques can still be helpful as an additional methodology to corroborate the results achieved.

4.4.2. Abstract Model Approaches

There are other acceleration approaches that explore the usage of higher level description models to enable efficient fault analyses at the system level [76] [77] [78]. Such methodology can be used to inject errors at earlier stages of the development process to quantitatively assess SMs selected to compose the design architecture. However, for more detailed analysis, the abstracts models must be further refined in order to keep sufficient level of details with respect to the actual design model. Consequently, the effort of maintaining a parallel development flow just for the high-level model with enough accuracy to the real implementation must be considered. Similar as in FPGA-based approaches, yet sufficient correlation between the models must be provided.

4.4.3. Hardware-Assisted Platform

4.4.3.1. Characteristics of the Available Platforms

Hardware-assisted platforms are proved by the major EDA vendors as an acceleration solution for verification [10] [11] [12]. These specialized machines are typically installed in secured and acclimatized rooms given its valuable asset to the companies' verification teams [79]. The most powerful model within the latest versions can emulate up to 9.2 billion gates, which can correspond to 40 billion transistors depending on the design [80]. These platforms share some common features including multi-users and simulation acceleration among others. It is not surprising that these commercialized platforms also have in common the lack of fault injection support since they were built targeting functional verification speedup.

This research explores Palladium XP (PXP) from Cadence Design Systems, Inc. in order to accelerate fault simulation. This hardware-assisted verification platform can be employed for different verification purposes in order improve the turnaround time [81]. In the following, some of these purposes are listed:

- Virtual prototyping – mixed-accuracy abstract models are emulated maybe together with RTL models eventually available in order to enable architecture decision making or closer software and hardware development and others;
- Co-simulation – runs thousand times faster than simulation can be reached with the DUT mapped in the platform and the

TB residing in the host. Hence, it can still leverage the capabilities of advanced verification languages.

- In-circuit emulation (ICE) – the full design is mapped into the verification platform with the possibility of real-world interaction devices through rate adapters for interfaces like Ethernet, USB, and others.

A variation of the ICE mode exists, which supports the commonly called the synthesizable TBs (STB) that runs together with the DUT in the platform. The ICE mode with STB permits much higher simulation acceleration since there is no communication with the host, which naturally reduces the run performance.

Run-time debugging features available in any commercial simulator are standard requirements for any acceleration solution. These features can be: waveform generation; run-time control/access to signal values; assertions; user customized logging; functional/code coverage collection; and so on. Such standard features are not exclusive to one hardware-acceleration platform they are supported by all of them. However, regarding their utilization in the fault injection context, there are specific characteristics that can be leveraged, which are not common to all solutions.

4.4.3.2. The Application of the Hardware-Assisted Platform Used

Since PXP is a processor-based platform, it contains a distinctive advantage towards the FPGA-based fault injection techniques discussed. The same design representation – later referred as snapshot – that is used by the simulation tool can also be mapped to run on PXP without requiring any synthesis step as in the FPGA flow. The snapshot resulted from the compilation of the HDL files is automatically mapped to the PXP. Hot-swap between the simulator and accelerator platform enables seamless execution of the snapshot on both domains. The processor-based architecture of PXP also delivers equivalent performance regardless design model being RTL and GL thus providing significant acceleration for GL simulation. The development history of the three most important hardware-assisted verification platforms indicates that PXP is the only solution not based on FPGA, which seems to allow these unique features [82] [83] [84].

Considering the functional safety context, all these characteristics made PXP an appealing candidate for fault injection. However, the actual feasibility of emulating faults using the hardware-assisted platform had to be investigated. The potential benefit of using such kind of platforms for fault injection has recently been acknowledged as suggested future work [77] [85]. However, they either suggest an intrusive modification of the design model to enable fault emulation or recommend the definition of accurate fault models to be used with high-level abstractions models. Actually, it has been found only one research that proposes a solution based on the hardware-assisted platforms to accelerate fault simulation [86] – continuation of the work presented in [87]. The approach substitutes the flip-flop cells by equivalent saboteur cells that are used to enable SEU fault injection. Different from other intrusive techniques, the fault injection manager resides in the computers host thus resulting in more communication with between the host and the platform.

There is no other similar approach, to the best of the author's knowledge, which leverages from hardware-assisted platforms to accelerate the injection of the ISO 26262 models fault injection.

4.5. CHAPTER REMARKS

This chapter presented the main concepts of fault injection including various techniques available in the literature. Additionally, the hardware-assisted verification platforms were introduced and their characteristics discussed. There are unique features available in the PXP platform that are valuable for fault injection within the safety domain. Related works based on the hardware-assisted platforms are scarce and do not leverage the characteristics highlighter. In the next chapter the approach developed in this research to accelerate the fault simulation is presented.

5. LITERATURE REVIEW ON FAULT INJECTION ACCELERATION

Fault injection is a widely accepted technique to assess design robustness [88]. Moreover, the injection of faults plays a key role in the manufacturing test domain [40]. Adding to that, fault injection is also employed for functional safety assessment and TB-Qualification. Across the use cases, there are many aspects that impact on fault injection definition. The abstraction level will define the fault models that can be applied and also the applicable optimization. For instance, the duration of the campaign, and the level of details in the design description also have influence in the number of faults to inject. Therefore, the phase of the product lifecycle where the fault injection is considered contributes to the decision regarding the applicable injection technique.

5.1. SCOPE OF THE LITERATURE REVIEW

With so many aspects associated with fault injection, innumerous researches have been conducted addressing several challenges that are intrinsic to each aspect. Therefore, it is important to draw the borderline in order to be able to cover one, or a couple of the possible topics properly. As already discussed, this Thesis' goal is to propose a methodology to accelerate fault injection considering the functional safety challenges. To be more precise, the objective is to provide better performance than fault simulation when assessing the hardware architectural metrics before sign-off. Therefore, the design abstraction level focused here is the gate-level netlist. RTL is also applicable, and it is planned to be covered in the future. The targeted fault model is Stuck-at (SAT), but SET and SEU faults are also discussed since they appear in the ISO 26262.

The literature review considers fault injection acceleration solutions which match the provided scope definition. The functional safety requirements drive the related work discussion. Given the broader application of the proposed solution, it is possible that the presented contribution can be leveraged for other use cases different than functional safety. However, functional safety is focused by this Thesis.

5.2. FPGA-BASED ACCELERATION VIA INSTRUMENTATION

FPGA technology is vastly explored to accelerate fault injection campaigns. In 2001, *Civera et al.* [89] [90] presented an instrumentation approach in order to provide the controllability and observability required to enable the fault injection emulation. In the next year, the more recent published results have shown a slight increase in the area overhead – i.e., between 8% and 42% – imposed by the presented solution [91]. Figure 30 indicates the amount of instrumentation required per flip-flop to enable the injection of SEU faults. The reported acceleration was up to 60 times faster fault injection when compared to fault simulation.

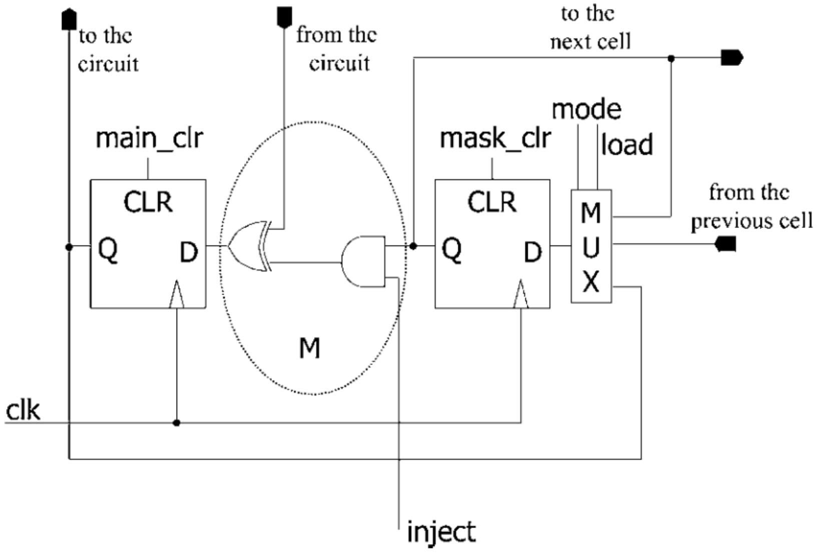


Figure 30 – Instrumented flip-flop. (Source [91]).

Lopez-Ongil *et al.* [92] in 2007 proposed an approach also based on instrumentation to enable the control of the design atomic parts in order to inject the faults. However, the presented “time-multiplexed” technique seems to be the main difference of this solution, which allows having the fault-free and the faulty version of the design running in alternate clock cycles. Figure 31 shows the flip-flop instrumentation required to implement “time-multiplexed” solution. Another two instrumentation techniques are presented with less area overhead, but consuming more memory resources of the FPGA. The claimed performance is about 10^6 SEU fault injections per second. However, such performance is achieved by quadruplicating the FPGA area consumed by the user design.

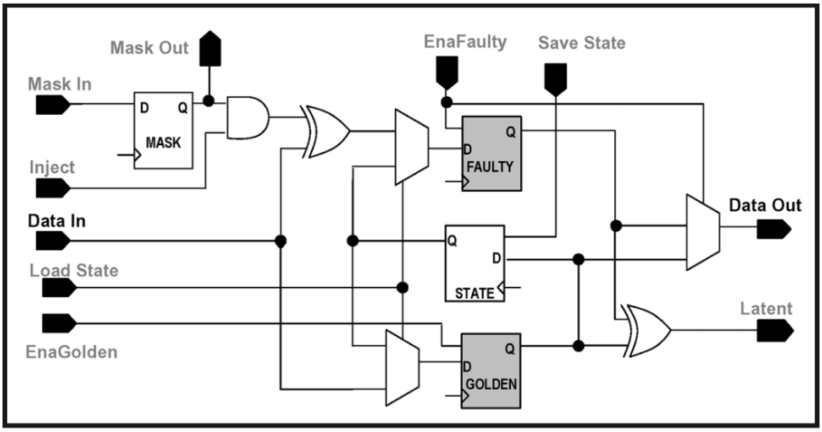


Figure 31 – Flip-flop instrumentation enabling the “time-multiplexed” technique. (Source [92]).

Entrena *et al.* [72] in 2010 published the work presented by Lopez-Ongil in 2007. The diagram block with the proposed approach is shown in Figure 32

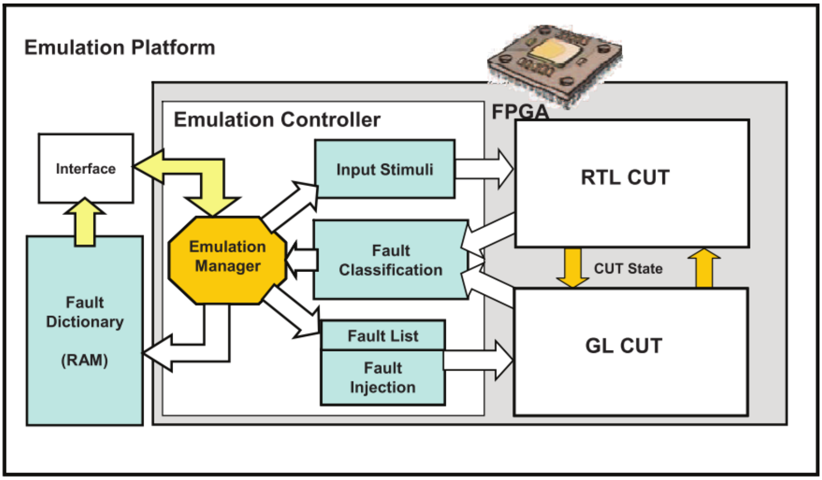


Figure 32 – Emulation platform diagram. (Source [72]).

In this version, SET faults are supported due to the modeling of delays inside of the cells at gate-level netlist. A second design module is generated with only the FF instrumented. The idea is to inject SET faults using in delay-enabled module implemented using shift registers resources available in the used FPGA. Whenever, a sequential element captures the fault, then the design state – i.e., flip-flop logic values at a given moment – is copied to the module version with the instrumented flip-flops only. This second module can run faster since it has not implemented the delay. Both modules run on the FPGA, and the "emulation manager" – shown in Figure 32 – selects the module to execute. To enable SET fault injection acceleration, the proposed solution requires a massive instrumentation. Additionally, the fault injection campaign is executed over two generated models, which are derived from the original design description.

5.3. FPGA-BASED ACCELERATION VIA RECONFIGURATION

Kenterlis *et al.* [93] in 2006 presents a platform, which automates the fault injection of SEU faults by using the JBits API to tweak the bitstream image used to configure the FPGA. With the API and full control of the bitstream generation, only bits actually disturbing configurable logic blocks could be selected to inject the faults, thus reducing the fault space. Even if the faults were injected on the occupied FPGA resources, the proposed methodology still was more verifying the device itself than the fault tolerant design configured in the FPGA. Hence, the approach was overdoing the vendor's work, which already provided reliability experiments information at that time [94]. Additionally, significant interaction with the host is required, and a faster communication link had to be used in order to reduce the bottleneck. Up to two order of magnitude have been observed between simulation and the proposed solution. Kuuhn *et al.* [95] in 2013 presented a similar approach, but instead of using the JBits API, a tool was developed to make the link. The developed tool provides the correlation between the fault selected at circuit description – e.g., VHDL, Verilog – level, and the generated bitstream for the injection of the fault. However, the research focus was fault tolerance hence no fault injection performance figures were discussed.

Aguirre *et al.* [96] [97] presented a non-intrusive FPGA solution, where the fault are injected by the manipulating the configuration memory of the device. Instead of corrupting the bitstream, FT-

UNSHADES leverages the dynamic partial reconfiguration available in some FPGAs to inject the faults. Figure 33 shows the implemented approach. The fault-free and faulty version share the same inputs, and their outputs compared in order to check if the injected fault has propagated out of the design.

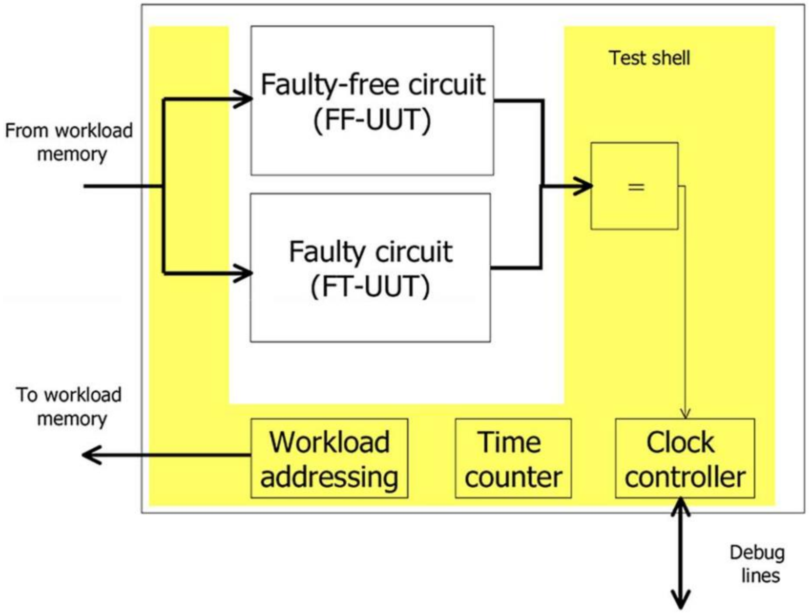


Figure 33 – FT-UNSHADES Emulation approach presented in [97].

Mogollon *et al.* [74] in 2011 presented the second version of the FT-UNSHADES solution. The new release promises to have eliminated the communication bottleneck by processing all data management in the developed platform. Experiments showing performance ration about 100k faults per second are claimed. However, the presented results achieve up to 1980 fault injection runs per second.

5.4. SIMULATION-BASED ACCELERATION VIA INSTRUMENTATION

Rohani and Kerkhoff [98] in 2011 presented the experiment results achieved with the proposed simulation-based approach. First, the design is modified in order to add the saboteurs shown in Figure 34. These saboteurs allow to inject two SET and the SEU transient faults, and it also permits to instrument a delay fault model. Each saboteur has an enable signal which is controlled through the simulator tool commands. A pre-analysis is performed utilizing a mathematical tool to configure the fault campaign by defining the fault target, injection time, and injection duration. The detection is checked as post-process step by comparing the logged data generate during the fault-free run after each fault simulation. The authors claim between 27% and 67% CPU time reduction against other two considered.

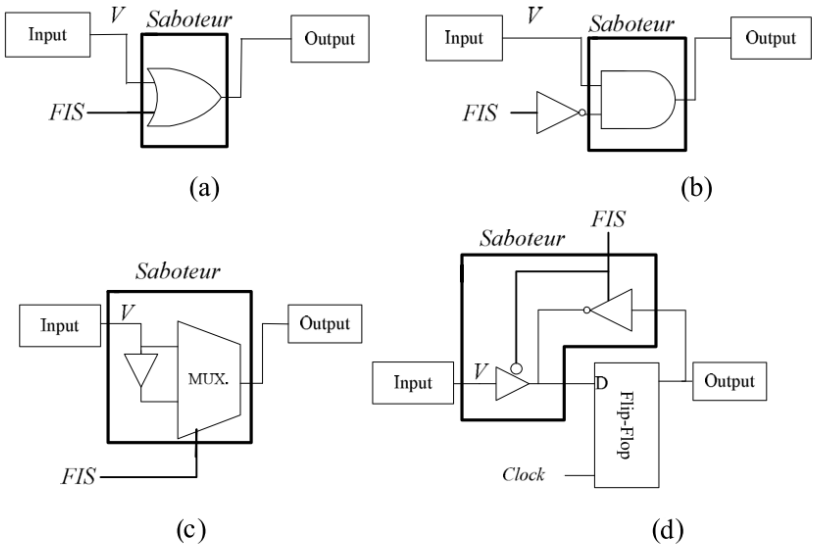


Figure 34 – (a) SET 0→1. (b) SET 1→0. (c) Delay fault. (d) SEU. (Adapted from [98])

5.5. ACCELERATION VIA FAULT CAMPAIGN OPTIMIZATION

Ebrahimi *et al.* [99] in 2015 presents a fault injection solution applicable to the fault campaign pre-analysis in order to avoid wasting time by ineffective fault injection. The proposed solution does not consider an injection technique per se. Instead, it highlights the potential benefit achieved with sampling by using the proposed analytical analysis thus providing campaign speedup factor up to thirteen. Such contribution can be leveraged by the workload profiling technique mentioned in the functional safety assessment methodology presented in [39]. In general, the proposed approach applies to any fault injection technique.

5.6. ACCELERATION VIA COMPLEXITY ABSTRACTION

Bombieri *et al.* [100] in 2011 utilize Transaction-Level Modeling vastly used for functional verification at the system level in order to optimize the fault simulation. The proposed approach claims automatic extraction of the TLM models from the RTL description. Additionally, an ATPG implemented with TLM models is used to generate the stimulus for the DUT automatically. Even stating initially that by using TLM, the performance gain can get up to a factor of thousand when compared to standard RTL simulation, the results show a speedup between 6.3 and 68.8 times faster runs. Moreover, an interesting contribution corresponds to the possibility of reusing the test vectors generated by TLM-ATPG back into the RTL simulation.

5.7. ACCELERATION VIA HARDWARE-ASSISTED VERIFICATION PLATFORMS

Daveau *et al.* [87] in 2009 proposed a fault injection acceleration methodology using the hardware-assisted verification platforms. In the following year, Bailan *et al.* [86] seemed to have moved forward with the research, and published more experimental results. Similar to other approaches, the acceleration is based on the instrumentation of the flip-flops to enable the controllability required by SEU fault injection. About 20% of area overhead in addition to the fault injection controller that also runs on the platform. Given the massive parallelism implemented, a significant fault injection runtime reduction is achieved. The selected fault target is a Leon2 IP core which can be replicated 19 times into the same hardware-assisted platform domain. Using the 16 domains, 304 faults could be injected at the same time. Figure 35 shows that to achieve such parallelism, one controller per domain is required in addition to the master controller running in the host.

To the best of the author’s knowledge, this research is the only fault injection acceleration technique, which is similar to the MADC solution that is proposed in this Thesis. Therefore, a discussion comparing the results achieved is carried out in the experiments chapter.

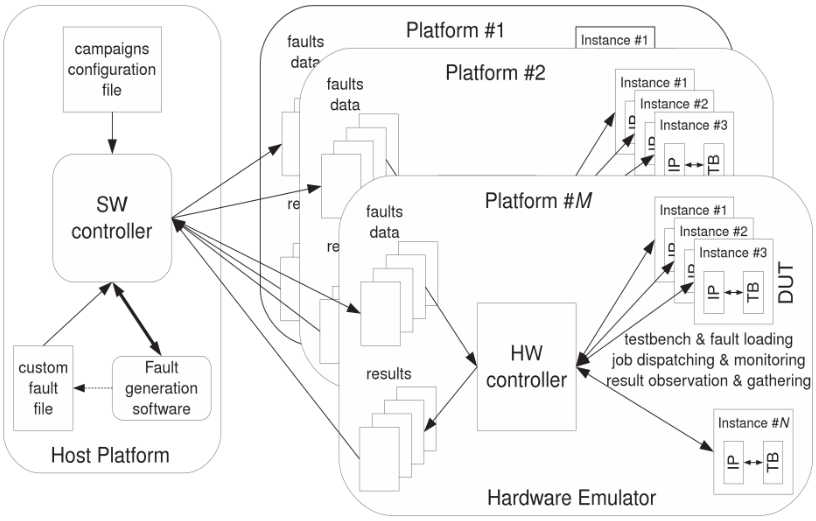


Figure 35 – Fault injection platform architecture highlighting the implemented parallelism in [86] [87].

5.8. COMMENTS ON THE REVIEWED RELATED WORK

The presented MADC solution is different from any work found in the literature review, as it does not require changing the design description neither to provide the required controllability/observability nor to synthesize for a different technology than the targeted one. Table 15 lists some comments on the related work considering their application to functional safety.

Table 15 – Related work comments.

Related work	**Comments**
Civera et al. [89] [90]	Instrumentation required with significant area overhead; performance gain not as high as more modern solutions.

Lopez-Ongil *et al.* [92] Entrena *et al.* [72]	Massive instrumentation needed in order to provide sufficient controllability and testability in order to enable fault injection; given the amount of instrumentation, it may not be possible to justify a safety assessment done using such different model. A SET fault injection solution is later presented. However, a massive instrumentation is required, and the campaign is executed over two modules extracted from the original design description. Three thousand SET faults per second was the performance achieved in the latest version commented.
Kenterlis *et al.* [93]	FPGA-based approach where the faults are emulated by external manipulating the device configuration to mimic the fault; it is questionable whether there is any value injecting faults in the device instead of the user logic configured in the device; Even if not requiring modification of the design description, the safety assessment is done in the synthesized model targeting a device different from the actually aimed application;
Kuuhn *et al.* [95]	
Aguirre et al. [96] [97] Mogollon *et al.* [74]	FT-UNSHADES work is already in the second version given its successful application in the aerospace and academic domains. However, the external manipulation of the FPGA's configuration memory seems to limit fault injection ration. Approaches like in [75] presented by the author, which leverage the internal access to the FPGA's configuration memory could be used to improve the performance. FT-UNSHADES supports the injection of SEU and SET faults.
Rohani and Kerkhoff [98]	Simulation-based approach combined with instrumentation; the pre-analysis for the campaign configuration is something applicable to any fault injection solution, which can be considered;
Ebrahimi *et al.* [99]	In general, it is applicable to any fault injection technique; Applicable to the safety domain;
Bombieri *et al.* [100]	The safety assessment would be performed based on a high-level generated model, and not in the actual design description.

Daveau *et al.* [87] Bailan *et al.* [86]	Massive parallelism, which can be leveraged by MADC; Outstanding fault injection runtime achieved; Based on instrumentation, which is not desired from the safety point of view;

5.9. CHAPTER REMARKS

In Chapter 3, the related work containing relevant description about the fault injection application to the functional safety domain is presented. In Chapter 0, some of the state-of-the-art fault injection approaches are cited. However, the related work discussion up to Chapter 5 was not sufficient, given the generic aspect of the proposed acceleration solution. Therefore, in this chapter, a more broad literature review is performed in order to allow putting the Thesis' contribution in perspective to the state-of-the-art. Next chapters present the proposed methodology details and the obtained results.

6. PROPOSED FAULT INJECTION ACCELERATION STRATEGY

As already mentioned, the proposed approach is based on the PXP hardware-assisted platform, which is originally intended for general functional verification speedup. The methodology implemented leverages this platform to accelerated fault injection campaign execution. In order to inject faults, the approach uses the common simulation debugging features that are supported by PXP and other hardware-assisted solutions thus not being restricted to one vendor. However, PXP has unique characteristics and features that are leveraged here in order to achieve better turnaround time.

Typical emulation-based techniques and other intrusive approaches can demand a significant effort to convince the auditor by showing the correlation between the latest design model and the one used for the safety assessment. Therefore, such techniques may not be applicable for the hardware architecture metrics evaluation using the pre-silicon design model. The proposed approach avoid any kind of design modification since it can run the simulation and the emulation sharing the same snapshot, i.e., the HDL design compilation image.

MADC is the shortening for Methodology to Accelerated the Diagnostic Coverage assessment. MADC is split into three main parts in order to explain the proposed methodology. These three parts are:

- <u>Enablement</u>: the faults that can be accelerated are identified. Collapsing is used to extend fault-set of identified faults. Testability is used to avoid injection of untestable fault on simulation or emulation;
- <u>Flow</u>: includes the required data input, the used tools, the storage and manipulation before and during the fault injection, the tools used, and the generated results;
- <u>Execution</u>: control of the runs and fault detection were implemented to enable fault injection using the hardware-assisted platform;

6.1. MADC ENABLEMENT

Force, *deposit*, and *release* are common features found in commercial RTL simulation tools. They permit to control internal signal values directly from the simulator console. The *force* command makes a design signal remain stuck at the value predefined, while the release can undo this. Similar to the *force* behavior, the *deposit* command also sets an internal design signal to a certain value. However, the value set by the deposit is overridden by the normal operation of the circuit whenever the signal is driven again. The same set of commands are possible on PXP, and they can be used to insert a fault condition in the design without modifying the description. The *force* command has similar characteristics as the SAT faults. On the other hand, the *deposit* feature is analogous to the SEU fault behavior. In the proposed methodology, the *force* command is used in order to replicate the fault injection effect of a SAT fault without modifying the design model.

As already seen, the effect of a SAT fault shall not propagate backward. However, the *force* command is applicable to the whole line thus generating an incompatible behavior towards the SAT fault model. Figure 36 highlights the backward propagation problem when using the *force* command to inject a SAT fault in the *g3.A* pin. On the other hand, there are many faults in the design, which the back propagation has no impact on any other part of the design – e.g., cell outputs. Therefore, a structural analysis has been developed in order to identify those nodes where the SAT faults can be injected regardless if the fault effect backward propagates or not.

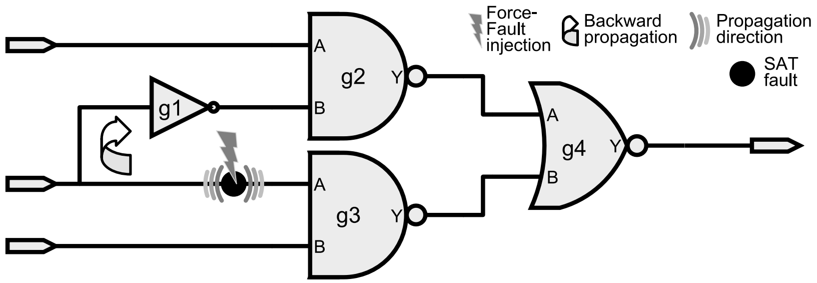


Figure 36 – Backward propagation problem associated with the *force* command.

Structural information extraction from the GL netlist is required to identify the faults that can be executed in the hardware-assisted platform. The extraction was implemented using two different synthesis tools in order to compare and validate the information collected [101] [102]. Redundant tool flows when considering the tools' confidence level required in the ISO 26262 [103]. Additionally, an ATPG tool was utilized to generate the fault list with SAT industry compatible fault model together with the fault equivalent groups and testability information [104].

Figure 37 shows the MUX once more, but this time, the circuit is not directly connected to the PIs and POs. Hence only the SAT faults in the gates are considered. Figure 37 is used to explain which faults can actually be selected for injection utilizing the *force* command, and how the number of faults suitable for acceleration can be optimized leveraging the collapsing and the testability information.

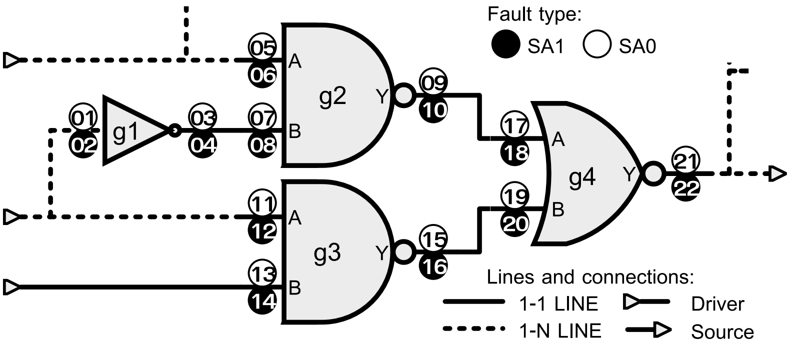


Figure 37 – SAT fault selection.

Naturally, faults located on cell outputs do not have the backward propagation issue since the effect of the *force* command cannot overpass the line driver. The remaining faults may be collapsible with the output faults thus allowing to maximize the ratio of faults verified by acceleration and those that can only be simulated. However, it is important to select the appropriate set of prime faults – equivalent group representatives – in order to guarantee the correctness of the fault injection executed in the emulator. Table 16 contains the equivalent groups (EGs) for the example of Figure 37, and the IDs of the prime faults are highlighted in bold – i.e., faults 6, 8, 12, 14, 19, 21, and 22.

The "*acceleratable*" (ACC) set of faults starts with those eight located on cell outputs (OUT). The ratio of ACC faults is then optimized by using logical collapsing (COL). Faults on input cells may not appear in any equivalent group but still fit for verified through acceleration. Such case can occur when the fault resides in an input cell that is connected to a 1-1 LINE (1-1L), and there is no fault instrumented in the driver side; otherwise, the two faults would have been collapsed. Fault 14 fits this situation hence it is the only fault on the 1-1L column that is added to

ACC list in Table 16. A similar case happens with faults on unconnected pins of registers or other cells, which also are not collapsible but still can be accelerated.

Table 16 – Fault list analysis for the MUX example.

Fault List				**Equivalent Groups**					**Ratio Optimization**				
Id	Node	Type	I/O	EG1	EG2	EG3	EG4	EGs	OUT	COL	1-1L	ACC	SEL
1	*g1.A*	sa0	I				8	8		✔		✔	8
2	*g1.A*	**sa1**	I	21				21		✔		✔	21
3	*g1.Y*	sa0	O	21				21	✔	✓	✓	✔	21
4	*g1.Y*	**sa1**	O				8	8	✔	✓	✓	✔	**8**
5	*g2.A*	sa0	I	21				21		✔		✔	21
6	*g2.A*	**sa1**	I					**6**					
7	*g2.B*	sa0	I	21				21		✔	✓	✔	21
8	*g2.B*	**sa1**	I				8	**8**		✔	✓	✔	8
9	*g2.Y*	sa0	O			17		17	✔	✓	✓	✔	**17**
10	*g2.Y*	**sa1**	O	21				21	✔	✓	✓	✔	21
11	*g3.A*	sa0	I	21				21		✔		✔	21
12	*g3.A*	**sa1**	I					**12**					
13	*g3.B*	sa0	I	21				21		✔		✔	21
14	*g3.B*	**sa1**	I					**14**			✔	✔	**14**
15	*g3.Y*	sa0	O		19			19	✔	✓	✓	✔	**19**
16	*g3.Y*	**sa1**	O	21				21	✔	✓	✓	✔	21
17	*g4.A*	sa0	I			17		**17**		✔	✓	✔	17
18	*g4.A*	**sa1**	I	21				21		✔	✓	✔	21
19	*g4.B*	sa0	I		19			**19**		✔	✓	✔	19
20	*g4.B*	**sa1**	I	21				21		✔	✓	✔	21
21	*g4.Y*	sa0	O	21				**21**	✔	✓		✔	**21**
22	*g4.Y*	**sa1**	O					**22**	✔			✔	**22**
22 faults				***11***	***2***	***2***	***3***	***8***	***8***	***11***	***1***	***20***	***6***

Faults: (✓) selected by the optimization; (✔) new to the acceleration set

From the 22 SAT faults in the MUX example, 20 can be accelerated (ACC) as shown in Table 16. However, the faults that can actually be injected must be located on a cell output, 1-1 line, or on a not driven pin. Therefore, the representative faults of the equivalent groups within the set of ACC faults need to be selected properly in order to avoid the backward propagation issue. The faults that are selected (SEL) to be injected via the hardware-assisted platform are marked in bold – i.e., 8, 14, 17, 19, 21 and 22. For that reason, the fault representatives of the EGs that are selected for acceleration can be different from the original prime faults.

To summarize the importance of the MADC fault selection, then the backward propagation related to the SAT fault injection via *force* command must be clear. Figure 38 shows a hypothetical circuit to explain the effect of the *force* command on the line in order to define when this effect can be used to mimic a SAT fault model. Considering this fault model, then the forward propagation is the expected behavior hence not represented in the circuit illustration. Although the backward propagation is not a valid SAT fault behavior, yet there are many circuit locations where it has no actual side effect thus allowing to employ the *force* feature.

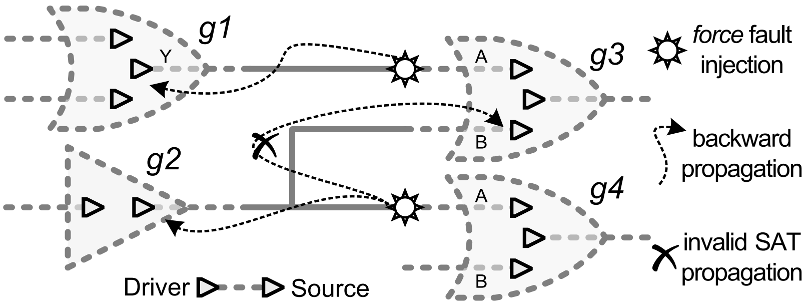


Figure 38 – Highlight the importance of the MADC fault selection.

The *force* command is used to inject one SAT fault in the input of the cell *g3* in Figure 38. Given the *force* semantics [105], then everything between the *drivers* and the *sources* of the line gets affected. Since the fault is injected close to the *source* (*g3* input pin) of the line, then the *force* behavior in the line can be seen as the backward propagation of the fault effect. For the fault injected in the *g3.A*, the *force* effect goes backward until it gets to the *driver* (*g1.Y* output) thus not disturbing any other part of the circuit. Therefore, this fault is a suitable candidate for acceleration. Notice that backward propagation concern is not applicable to fault located at the *drivers*.

As already mentioned, there are some situations where the fault effect does not only propagates towards the *driver*, but it also affects other cells. For example, the *force* applied to the *g4* input pin in Figure 38 generates a side effect by disturbing the *g3* cell as well. Therefore, whenever a line connects one *driver* to multiple *sources*, then the *force* command is not suitable for the injection of the SAT faults residing on cell inputs. For those cases, MADC checks if there is any equivalent fault located in a driver, which then can be selected for acceleration. Table 17 lists the faults located between the sources and the *drivers* illustrated in Figure 38 and comments whether they can be selected for acceleration or not.

Table 17 – Faults suitable for acceleration considering the hypothetical circuit example.

Cell	**Pin**	**Tie**	**Is the fault suitable for fault acceleration?**
g1	output (Y)	*driver*	Yes. Output faults are always suitable
g2	output (Y)	*driver*	Yes. Output faults are always suitable
g3	input (A)	*source*	Yes. Same effect as injecting an output fault
g3	input (B)	*source*	[a] No, since it would disturb g4 as well
g4	input (A)	*source*	[a] No, since it would disturb g3 as well
g4	input (B)	*source*	Yes, assuming it is unconnected.

[a] although this fault itself can not be accelerated, still it may be collapsible with another fault located in the cell output.

6.1.1. Enablement Algorithm

The fault selection described using the Table 16 can be seen as a three steps procedure. The *first-step* corresponds to the selection of the faults located in the output. The *second-step* and *third-step* are related to the ratio optimization of faults that can be accelerated. These procedure steps are illustrated by the pseudo-algorithm shown in Figure 39. MADC relies on the structural circuit analysis developed to explore the SAT fault injection in the hardware-assisted platform by discovering the faults that do not require isolation – i.e.; its effect does not backward propagate. The

“Algorithm 1” in Figure 39 represents this structural analysis, where the function *c.faults*(out,SAT) prints the SAT (SA0 and SA1) faults on the output of cell *c*, while *c.faults*(in,SA0) prints the SA0 faults on inputs of *c*. The *p.faults*(SA1) prints the SA1 fault of the corresponding cell pin *p*. The *p.connections*() returns the number or cell pins interconnected by the wire that is also connected to pin *p*.

Algorithm 1 Prints faults selected to be accelerated

```
 1: procedure DISCOVERFAULTSTOACCEL
 2:     Let $C$ be an array with all cell instances
 3: first-step: # all output faults are selected
 4:     for each $c$ in $C$ do :
 5:         $c.faults$(out,SAT)
 6: second-step: # get input faults that are collapsed with an output fault
 7:     for each $c$ in $C$ do :
 8:         if $c \in \{FF, XOR, Complex\}$ then next
 9:         if $c \in \{INV, BUF\}$ then $c.faults$(in,SAT)
10:         if $c \in \{N/AND\}$ then $c.faults$(in,SA0)
11:         if $c \in \{N/OR\}$ then $c.faults$(in,SA1)
12: third-step: # get (not selected) faults on ports only wired to the driver
13:     for each $c$ in $C$ do :
14:         for each $p$ in $c.inputs()$ do :
15:             if $c \in \{INV, BUF\} \parallel p.connections() > 2$ then next
16:             if $c \in \{FF, XOR, Complex\}$ then $p.faults$(SAT)
17:             if $c \in \{N/AND\}$ then $p.faults$(SA1)
18:             if $c \in \{N/OR\}$ then $p.faults$(SA0)
```

Figure 39 – Pseudo-algorithm for printing the suitable faults to execute in the hardware-assisted platform.

As already mentioned, the pseudo-algorithm shown in Figure 39 starts with the execution of the *first-step* to print all SAT faults located at cell outputs, which are suitable for acceleration by default. As a *second-step*, the procedure prints the faults that are collapsible to the output faults thus being indirectly enabled for acceleration. After considering the equivalent groups to increase the ratio of classified faults via acceleration, then another optimization step is executed.

The *third-step* searches for faults in the same condition as the ones located at the *g3.A* and *g4.B* pins of the example in Figure 38. Therefore, if the function *p.connections*() returns a value greater than two, then it means that a *g3.B* or *g4.A* kind of fault has been found. In this case, the fault is not selected for acceleration, and the next fault is analyzed. When the *p.connections*() is equal to one, then the fault is located at an unconnected input pin – similar to the *g4.B* example. To not print the same fault twice, MADC algorithm checks if the fault has not been collapsed or marked as untestable already. Faults located on an input pin connected to a 1-1 line can also be selected – the same situation as for the *g3.A* fault in Figure 38. The faults not collapsed before are printed by the MADC algorithm.

The DiscoverFaultsToAccel procedure shown in Figure 39 prints all faults to be classified using the hardware-assisted platform. However, only the prime faults need to be executed. In order to avoid choosing a fault with side effects, MADC selects from the equivalent group, always one fault that is located on a cell output pin. The faults printed during the *third-step* are the only faults residing in cell input pin that are executed on the verification platform.

All faults not printed by the MADC algorithm can only be simulated, or some kind of design instrumentation is required in order to isolate the fault node from the rest of the circuit thus avoiding the backward propagation problem. Since the goal is to use the same design representation for the fault injection, then instrumentation is not considered this Thesis.

It is important to notice that the information printed by the structural analysis procedure illustrated in Figure 39 must permit the mapping between the selected faults and the original fault list generated by the ATPG tool. This information includes the original fault IDs, the EGs indication, the direction (I/O), the fault type, and the fault node. Table 18 demonstrates the fault list that should be generated by the pseudo-algorithm presented in Figure 39 with the minimum content necessary to allow identifying the faults that can have its classified via fault injection acceleration and also permitting back annotate the results into the original fault list. The shaded rows in Table 18 indicate the faults that actually need to be injected to be able classify all 20 faults in the list. Each equivalence group (EG) has one bolded fault IDs to indicate the selection. Among those six fault IDs selected, five are located in the first cell output of the EG. Notice that fault 01 would backward propagate thus affecting gate *g2* as well. Therefore, it is important to select fault in the outputs whenever possible. Only fault 14 resides in a cell input since it is not collapsed with an output port. Fault 14 is selected by the 1-1 LINE rule.

Table 18 – Fault list generated by the "Algorithm 1".

ID	Type	I/O	EGs	Node
1	sa0	I	8	*g1.A*
4	sa1	O	8	*g1.Y*
8	sa1	I	8	*g2.B*
14	sa1	I	14	*g3.B*
9	sa0	O	17	*g2.Y*
17	sa0	I	17	*g4.A*
15	sa0	O	19	*g3.Y*
19	sa0	I	19	*g4.B*
2	sa1	I	21	*g1.A*
3	sa0	O	21	*g1.Y*
5	sa0	I	21	*g2.A*
7	sa0	I	21	*g2.B*
10	sa1	O	21	*g2.Y*
11	sa0	I	21	*g3.A*
13	sa0	I	21	*g3.B*
16	sa1	O	21	*g3.Y*
18	sa1	I	21	*g4.A*
20	sa1	I	21	*g4.B*
21	sa0	O	21	*g4.Y*
22	sa1	O	22	*g4.Y*

The handling of the PIs, POs, and untestable faults are not shown in Figure 39 since they are not part of the MUX example illustrated in Figure 37. However, they must be considered for the correct calculation of the metrics hence they are covered by the MADC structural analysis.

6.1.2. The Ratio of Faults Suitable for Acceleration.

The number of faults that can be accelerated within a GL netlist is given by $\boldsymbol{F_{ACCEL}}$ in (12). $\boldsymbol{F_{ACCEL}}$ consists of three addends: $\mathbf{2} * |\boldsymbol{C}|$ and the summation of the functions $\boldsymbol{f}(\boldsymbol{c})$ in (13) and $\boldsymbol{g}(\boldsymbol{c},\boldsymbol{p})$ in (14), which are associated with the steps in the **Algorithm 1** in Figure 39.

$$\boldsymbol{F_{ACCEL}} = \mathbf{2} * |\boldsymbol{C}| + \sum_{\boldsymbol{c} \in \boldsymbol{C}} \boldsymbol{f}(\boldsymbol{c}) + \sum_{\boldsymbol{c} \in \boldsymbol{C}} \sum_{\boldsymbol{p} \in \boldsymbol{P}} \boldsymbol{g}(\boldsymbol{c},\boldsymbol{p}) \qquad (12)$$

$$\boldsymbol{C} = \{\boldsymbol{c} \mid \boldsymbol{c} \text{ is a cell instance}\},\ \boldsymbol{P} = \{\boldsymbol{p} \mid \boldsymbol{p} \text{ is an input pin of a given cell } \boldsymbol{c}\}$$

$$\boldsymbol{f} = \begin{cases} \mathbf{0}, & \boldsymbol{c} \in \{\boldsymbol{FF}, \boldsymbol{XOR}, \boldsymbol{Complex}\} \\ \mathbf{2}, & \boldsymbol{c} \in \{\boldsymbol{INV}, \boldsymbol{BUF}\} \\ |\boldsymbol{P}|, & \boldsymbol{c} \in \{\boldsymbol{N/OR}, \boldsymbol{N/AND}\} \end{cases} \quad (13)$$

$$\boldsymbol{g} = \begin{cases} \mathbf{0}, & (\boldsymbol{p.connections}() > \mathbf{2}) \vee (\boldsymbol{c} \in \{\boldsymbol{INV}, \boldsymbol{BUF}\}) \\ \mathbf{1}, & \boldsymbol{c} \in \{\boldsymbol{N/OR}, \boldsymbol{N/AND}\} \\ \mathbf{2}, & \boldsymbol{c} \in \{\boldsymbol{FF}, \boldsymbol{XOR}, \boldsymbol{Complex}\} \end{cases} \quad (14)$$

The $\mathbf{2} * |\boldsymbol{C}|$ results in two times the number of cells belonging to $\boldsymbol{C}$ – e.g., all cells in the netlist – thus corresponding to the amount of SA0 and SA1 faults located at the cell outputs. The function $\boldsymbol{f}(\boldsymbol{c})$ is equal to the number of collapsible input faults for each cell $\boldsymbol{c}$ according to the logical collapsible rules presented in Figure 26 except the "1-1 LINE" rule. For instance, $\boldsymbol{f}(\boldsymbol{c})$ equals to the number of inputs $|\boldsymbol{P}|$ of $\boldsymbol{c}$ when it is an AND, OR, NAND, or a NOR gate. On the other hand, the function $\boldsymbol{g}(\boldsymbol{c}, \boldsymbol{p})$ is equal to the number of faults on the input pins $\boldsymbol{p}$ of a cell $\boldsymbol{c}$ that are collapsible according to the "1-1 LINE" rule – i.e., the line where the fault resides has no more than two connection ends. Therefore, the summations $\sum \boldsymbol{f}(\boldsymbol{c})$ and $\sum \boldsymbol{g}(\boldsymbol{c}, \boldsymbol{p})$ provide the total number of faults that can also be accelerated due to the collapsing optimization. Table 19 shows the results of applying (12) for the MUX example illustrated in Figure 37.

Table 19 – Results of each step of the MADC analysis

Faults that can be:	**Ratio of accelerated faults per step**			
	No$\boldsymbol{F_{ACCEL}}$	$\mathbf{2} * |\boldsymbol{C}|$	$\sum \boldsymbol{f}(\boldsymbol{c})$	$\sum \boldsymbol{g}(\boldsymbol{c}, \boldsymbol{p})$
Accelerated	0	8	16	20
Only simulated	22	14	6	2
Accel. Ratio	**0.00%**	**36.36%**	**72.72%**	**90.91%**

The accumulated result of each term in (12) is shown by the "Accelerated" row in Table 19. After each term is calculated, the number of faults that can be accelerated increases hence reducing the number of faults that can only be simulated. For the MUX example shown in Figure 37, up to 90.91% of all faults can be accelerated by just selecting the appropriate set of prime faults to be used for the injection campaign. In other words, less than 10% of the faults require design model adaption to provide the isolation essential for the SAT model. To keep MADC as a zero intrusion solution, these two faults are simulated using IFSS – or any fault simulator. Meanwhile, the hardware-assisted platform can accelerate 90.91% of the faults hence enabling the parallel execution of fault injection on both engines thus reducing the overall execution time of the fault campaign.

6.2. MADC FLOW

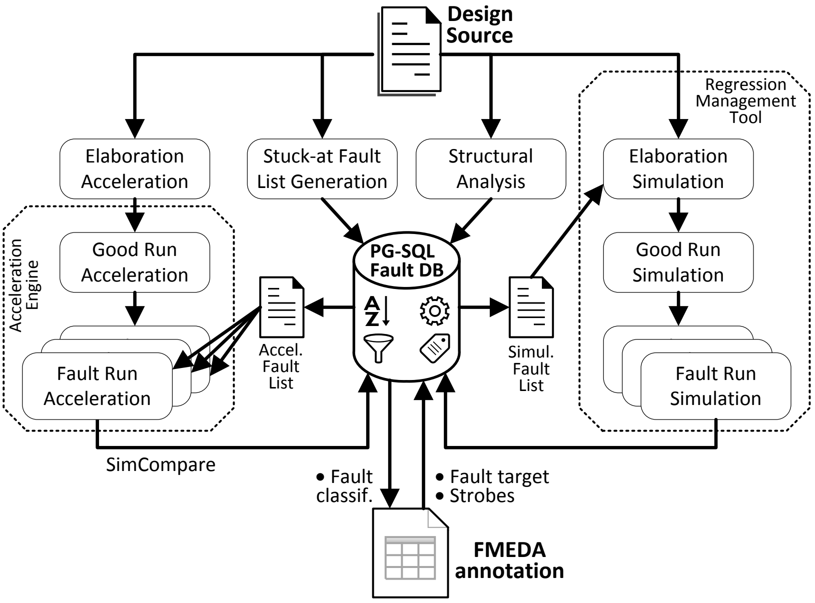


Figure 40 – MADC flow diagram.

The proposed MADC flow diagram is presented in Figure 40. In addition to the design model (Design Source), the flow requires the definition for the faults to be injected (Fault target) and also the definition for the strobes (Strobes) that are the detection points. The fault definition can be the instance path of a subpart of the design model. Currently, MADC consider only SAT faults, but it can be extended to support other fault models. A time different than zero or a specific condition – e.g., 100ns after reset sequence – to trigger the injection can be set in order to permit the evaluation of faults occurring during normal operation conditions – i.e., random faults. The strobe information provided to

MADC contains a list of detection points and their location meaning towards the fault classification defined by the ISO 26262.

6.2.1. Mapping the campaign results to the standard classification

The inputs for the MADC flow follow similar semantics as for the IFSS's inputs, and they are translated by MADC to the PXP language and interface in order to provide transparent integration between the two engines. IFSS permits the definition of two strobe groups thus allowing a more advanced fault classification than typically supported by standard DFT fault simulators, such as detected, undetected or potentially detected. The former classification causes an unknown value – i.e., "*X value*" or "*don't care*" – at an observation point hence not giving a definitive answer about its detection. It must be remembered that the ISO 26262 compliance demands the assessment of faults that negatively impact the SPFM and LFM, which are the SPFs or RFs and the LFs, respectively. Whenever the required metrics are not achieved, SMs can be added to detect these faults and turn their classification into DFs – or MPF_D. Using the IFSS terminology, the detection points can be grouped as functional or checker strobes to support ISO 26262 fault classification. For instance, as many faults as possible should be prevented from getting to a functional strobe. In other words, the functional strobes correspond to where the failure mode is considered activated. On the other hand, the checker strobes are located in the observation points that indicate when a fault was diagnosed.

The fault classification done by IFSS depends on which strobe group was triggered by the fault propagation. A fault that only propagates to a functional observation point is classified as dangerous undetected (DU) according to the IFSS syntax. The faults detected by SMs hence triggering the checker strobes can either be classified as safe detected (SD) or dangerous detected (DD) whether they also propagate or not through a functional strobe. Safe undetected (SU) is the classification of faults that either remain latent or are masked by the circuit and hence no observation point within the strobes groups is triggered. The mapping between IFSS and ISO 26262 fault classifications is shown in Table 20.

A fault that propagates through functional output can be classified as DF if an SM later detects the fault thus triggering a checker strobe observing the error detection time specified. IFSS permits to specify this time window between triggering of functional strobe until the detection at checker strobe. Additionally, the simulation can be configured to stop whenever a checker strobe is activated or the error detection time window ends. The stopping feature can reduce the overall execution time of a fault campaign by only running the test completely when strictly necessary – e.g., safe-undetected (SU) faults are not detected at any observation point hence the whole test is executed. However, the optimization achieved with this feature varies according to the number of faults injected that can quickly propagate to an observation point thus stopping the simulation.

Table 20 – IFSS and ISO 26262 fault classification mapping.

Fault that propagates	**Strobe Groups**		**Fault Classification**	
	Functional	***Checker***	***IFSS***	***ISO 26262***
only to:	X		DU	SPF/RF
to both:	X	X	DD	DF [a]
only to:		X	SD	DF
to none.			SU	LF

[a] assuming the error detection time stipulated is observed.

6.2.2. The MADC interface definition

Additional to the fault list format, the MADC supports the IFSS fault classification syntax thus allowing the translation to the ISO 26262 terminology. This includes the untestable (UT) faults that are classified as SFs since they cannot violate an SG. The definition of functional and checker strobes are also supported by the MADC flow as well as the set of time window between the two groups in order to proper classify the faults. The MADC support here means that the proposed methodology enables equivalent features already provided in the simulator, on the hardware-assisted platform.

The MADC controls the ATPG tool in order to generate the SAT fault list with the collapsing and testability results. The synthesis tools are used to collect the number of connections and the direction for each pin where the faults are located. All this information is committed to a PostgreSQL (PG) database that is used to store, sort, and filter the faults in order to generate the lists with those suitable for acceleration according to the algorithm shown in Figure 39. Another list is generated for the faults that can only be simulated – in this case, the IFSS. Both lists contain only the primes hence the minimum set of faults to be injected by each engine. After injecting the selected faults, MADC collects the reports generated by IFSS and the results achieved with PXP in order to annotate the classification back to all faults stored in the PG database. Finally, the resulting fault annotation can be translated into the ISO 26262 classification using the mapping from Table 20.

6.2.3. Link to the FMEDA

A technical FMEDA contains the relevant information needed for the definition of the fault campaign inputs. The targeting area where to inject the faults can be specified based on the part/subpart name found in the spreadsheet first columns like in the example shown in Table 11. The failure mode defines the fault model. Functional strobes can be mapped to the part/subpart outputs. The SM references can be used to find the observation points to be selected as checker strobes.

The MADC fault classification results provide the information required to calculate the percentages that must be annotated back to the FMEDA in order to allow the evaluation of the SPFM and LFM. The failure rate fraction associated with the safe faults is entered in the FMEDA as the percentage calculated in (15) where the number of untestable faults ($\#UT$) is divided by the size of the fault set. The amount of dangerous undetected faults ($\#DU$) is used in (16) to find the percentage of the RFs failure rate that is covered by an SM. The same happens with the total of safe undetected faults ($\#SU$) in (17) that is equal to the DC_{LF}, which also corresponds to the LF failure rate percentage.

$$F_{SF}(\%) = \frac{\#UT}{(\#UT + \#DU + \#SU + \#DD + \#SD)} \times 100 \tag{15}$$

$$DC_{RF}(\%) = \frac{\#DU}{(\#DU + \#SU + \#DD + \#SD)} \times 100 \tag{16}$$

$$DC_{LF}(\%) = \frac{\#SU}{(\#SU + \#DD + \#SD)} \times 100 \tag{17}$$

There are innumerous spreadsheet formats as well as requirement management tools that provide support to FMEDAs. For that reason, one-size-fits-all solution for interfacing all kind of FMEDA tools is unrealistic. However, the precise definition of the MADC input and output allows the development of an interface to any solution available in order to automate the link to the FMEDA.

6.3. MADC FAULT INJECTION EXECUTION

As mentioned before, the fault injection is performed by two different engines. Therefore, the MADC is split into two sub-flows for the fault campaign execution. In the left side of the flow, there are the steps perform the fault injection using the hardware-assisted platform while on the right side of Figure 40, the three IFSS's steps are shown. Regardless the engine, three common steps are executed, which are:

- Elaboration: corresponds to the design model compilation into a snapshot that is loaded by the tool when starting the simulation.
- Good Run: fault-free execution of the snapshot to collect the reference values on the functional and checker strobes defined;
- Fault Run: fault injection execution until the test completes or a checker strobe is triggered thus deviating from the reference value.

As already mentioned, the snapshot generated for a standard simulation can also be executed in the hardware-assisted platform. However, the fault injection feature provided by IFSS is only enabled within Incisive simulator flow. Since IFSS feature is not available in the PXP flow, then the simulation snapshot can not be recognized by the hardware-assisted platform. Therefore, two snapshots are generated from the elaboration executed for each sub-flow (Elaboration Acceleration and Simulation) as shown in Figure 40. In other words, the snapshot resulted from the "Elaboration Acceleration" step can be simulated too, but without the IFSS features enabled. However, the inverse is not true for the snapshot output of the "Simulation Elaboration", which cannot be loaded in the hardware-assisted platform.

In the IFSS flow, the faults to be injected must be defined at elaboration while the strobes are passed to the tool during the good run. The strobe definition includes the instance path of each node selected as an observation point, and the strobe type – i.e., function or checker. During the fault-free run (Good Run Acceleration in Figure 40), IFSS saves the traced data of the selected strobe signals. This trace information is constantly compared during fault injection runs (Fault Run Simulation), and a detection notification is issued if any discrepancy occurs. A similar approach was implemented using the capability available in the hardware-assisted platform to store waveform data of selected signals. After each accelerated fault injection (Fault Run Acceleration), the waveform data is compared with the one generated by the fault free-run (Good Run Acceleration) using a simulator utility that informs the timestamp and the signals of each existing mismatch. This post-run check procedure has a drawback compared to IFSS, which can stop the simulation as soon as the fault effect is detected at a checker strobe for example. Figure 41 highlights the possible impact on the overall campaign execution time by the "stop at detection" and the "post-run check".

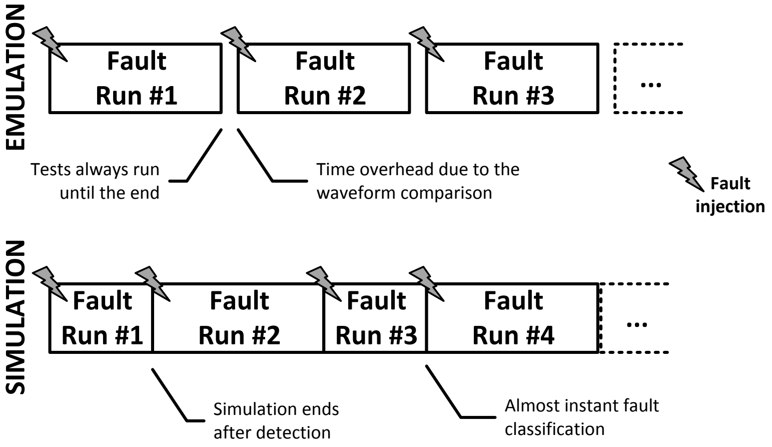

EMULATION
Fault Run #1
Fault Run #2
Fault Run #3
...
Tests always run until the end
Time overhead due to the waveform comparison
Fault injection
SIMULATION
Fault Run #1
Fault Run #2
Fault Run #3
Fault Run #4
...
Simulation ends after detection
Almost instant fault classification

Figure 41 – The possible negative impact on the MADC performance due to the lack of support of "stop at detection" or because of the "post-run check".

It is important to notice that the possible negative impact on the performance of the MADC varies according to the fault campaign profile. The profile encompasses of the stimulus quality, the fault set, and the techniques used by the SMs being evaluated. Insufficient stimulus or specific SMs may require most of the simulation runs to be executed for longer periods thus reducing the performance difference between simulation and emulation achieved by the "stop at detection" feature. On the other hand, the acceleration provided by the hardware-assisted platform often compensates the simulation gain achieved with "stop at detection" even when many faults are shortly detected.

The control of when and which faults are injected is performed by the implemented scripts with commands executed during the fault run for each engine. The script created to control the fault injection on PXP executes the good run (Good Run Acceleration in Figure 40) to generate reference-data trace, which is stored in the host computer. Next, the snapshot is reset, and a fault is injected by using the *force* command. The trace data generated during the fault run (Fault Run Acceleration) is also copied to the host where the simulator utility is used to compare the waveform databases in order to classify the fault according to its detection status. The communication between the hardware-assisted platform and the host can become a bottleneck depending on network quality and the amount of strobe data generated. Many flow steps executed in the PXP fault injection script are not needed in the script created to control IFSS since the fault injection and the classification are featured by the simulator. The summary of the fault injection commands executed on each engine is shown in Figure 42.

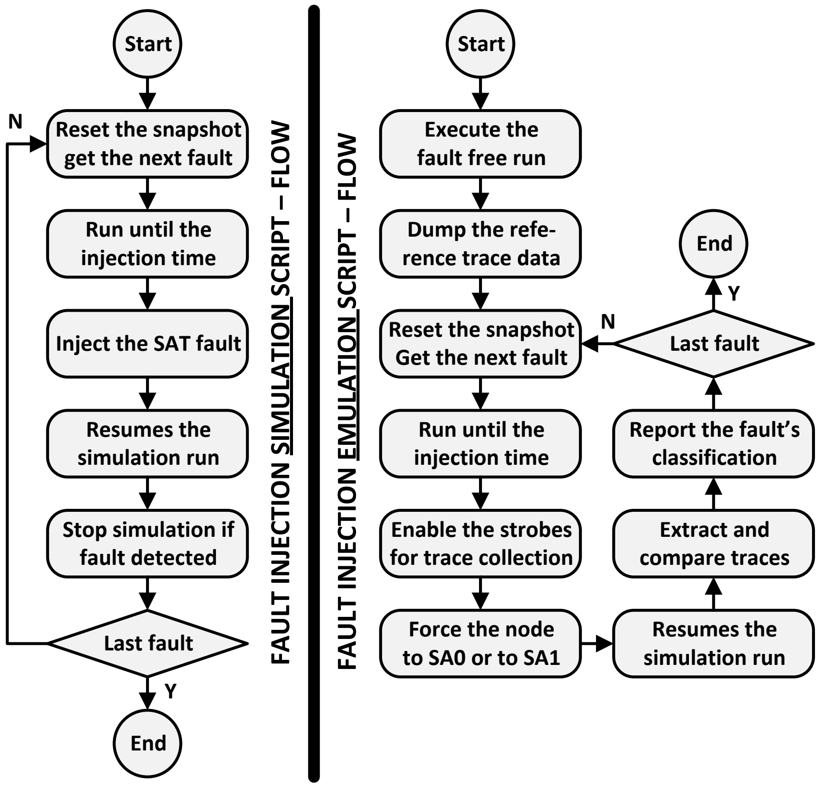

Start
Reset the snapshot get the next fault
N
Run until the injection time
Inject the SAT fault
Resumes the simulation run
Stop simulation if fault detected
Last fault
Y
End
FAULT INJECTION SIMULATION SCRIPT – FLOW
FAULT INJECTION EMULATION SCRIPT – FLOW
Start
Execute the fault free run
Dump the reference trace data
Reset the snapshot Get the next fault
Run until the injection time
Enable the strobes for trace collection
Force the node to SA0 or to SA1
Resumes the simulation run
Extract and compare traces
Report the fault's classification
Last fault
N
Y
End

Figure 42 – The execution flow of the developed scripts for the simulation and the emulation platform.

An important optimization, applicable to simulation in general thus including IFSS, is the possibility to leverage a computer farm available to run multiple fault simulations in parallel. Similarly, PXP has many “domains” which are resources where snapshots can be loaded thus allowing parallel fault injection. The DRM for the distributed fault simulation and usage multi PXP domain could not be investigated due to the lack of resources available. These techniques can be explored later to optimize the MADC execution flow.

6.4. CHAPTER REMARKS

This chapter presented the methodology to accelerate DC assessment, which is the Thesis main contribution. The methodology consists on leveraging the PXP hardware-assisted verification platform to boost the performance of the fault injection campaign execution. To enable that, the GL-netlist is analyzed by the algorithm developed, which identifies the SAT faults that can be correctly emulated. This algorithm enables a non-intrusive approach to emulate most of the faults, and also observing the ISO 26262 guidelines. The remaining faults still have to be simulated using IFSS. The MUX example is used again to explain the proposed approach. In the next chapter, the results achieved using a more meaningful design is discussed.

7. EXPERIMENTS AND RESULTS

7.1. CASE STUDY OVERVIEW

To confirm the MADC results, faults selected for acceleration – by the proposed strategy – were also simulated and the equal fault detection status was obtained with both engines. As already mentioned, the safety related works available in the literature do not share details about the implementations used probably to avoid infringing NDAs since the safety analysis is closely related to the design. For instance, “MIPS-like” is the term used to refer to the design underlying the safety analysis presented in [41]. Another example can be found in [39] where the authors briefly discuss the results achieved “during the validation of a real safety critical system based on a 32-bit RISC processor” without further information on the design. On the other hand, there are safety related works which cover the topic at different abstraction levels thus not providing suitable test cases to be explored in this Thesis. For example, the authors of the DBW example used to introduce functional safety and the ISO 26262, perform the safety analysis over an existing prototype [31]. In this research, an open source design is used to show the MADC achieved results. The design is based on an OpenRISC architecture as illustrated in Figure 43 [106] [107].

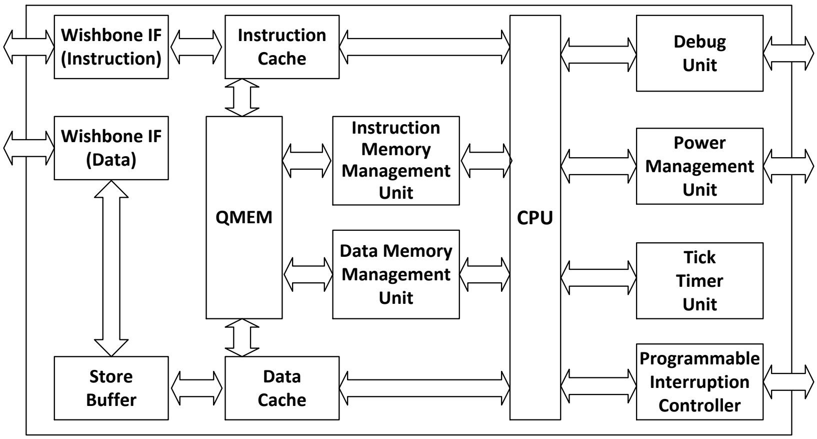

Wishbone IF (Instruction)
Instruction Cache
Debug Unit
Wishbone IF (Data)
QMEM
Instruction Memory Management Unit
CPU
Power Management Unit
Data Memory Management Unit
Tick Timer Unit
Store Buffer
Data Cache
Programmable Interruption Controller

Figure 43 – OpenRISC block diagram (source [106]).

A leaf block of the OpenRISC design hierarchy was used as the fault target due to its small number of faults, thus fitting to the purpose of feasibility checking. The leaf block, in this case, is an unsigned carry adder (CA) automatically inserted by the synthesis tool. The adder composes the ALU (Arithmetic Logic Unit) of the OpenRISC CPU shown in Figure 44.

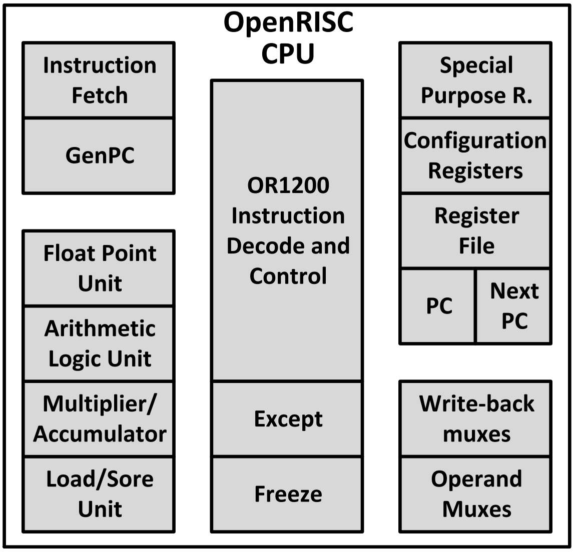


Figure 44 – OpenRISC CPU.

7.2. USING MADC IN A COMBINATORIAL CIRCUIT

The OpenRISC adder contains 1,280 fault candidates from which 980 – i.e., 72.5% – are selected by MADC to be executed in the hardware-assisted platform. After the collapsing and testability analysis, the set of faults to run is reduced to 640 prime faults. Table 21 has the outcome of "MADC analysis" showing that most of SAT faults can be accelerated. Naturally, only prime faults need to be run, i.e., 288 or 45% of all prime faults. The 288 faults were injected using both engines – PXP and IFSS – in order to validate the proposed MADC solution. This validation check consists in the comparison between the fault detection status and the detection time achieved with each tool. This checking stage exposed many challenges to be overcome by MADC.

Table 21 – MADC analysis results.

Faults that can be:	**MADC Analysis**			
	Faults	**Ratio (%)**	**Prime**	**Ratio (%)**
Accelerated	928	72.5	***288***	45.0
Simulated	352	27.5	352	55.0
Total	**1,280**		**640**	

To allow the advanced classification available on IFSS, not just the detection status had to be reported, but also the strobes that were triggered in order to permit identification of the fault propagation through the functional, or the checker strobes. Additionally, the detection time had to be aligned with the time reported by the simulator. To achieve that, the instants when the strobes occur had to match on both engines. Initially, it was developed a script based approach to collect the strobe timestamps from simulation, and then use them during acceleration. However, the comparison overhead was prohibitive due to the amount of data handled in ASCII format hence impacting on performance. Although inefficient, the experience gained with the hardware-assisted platform was essential to further develop the MADC flow especially regarding storage and export of the trace data from the acceleration platform.

Similar to other acceleration solutions, PXP permits to probe signals and stores the trace data in many formats. The format can be either customized using specific tool language or predefined standard waveform formats like value change dump (VCD). Naturally, PXP also supports the waveform database format used by the Incisive simulators. An Incisive utility named SimCompare permits to compare waveform databases. Signals to compare, time difference tolerance, maximum errors and many other comparison characteristics can be configured with SimCompare. Within the MADC flow, the waveform database generated during Good Run contains the same signals that are probed during the Fault Run, and they are all compared. In the fault injection context, only the earliest signals mismatch is interesting and no time difference tolerance is considered. Next, there is an example on how MADC configures SimCompare.

```
1. maxerrors 1
2. database g <reference waveform database>
3. database t <compared waveform database>
```

```
4. compare .
5. report -detail errorsonly \
         -style comparescan \
         -values -output simcompare.rpt
```

With this configuration, SimCompare stops the comparison at the first mismatch and report the details to *simcompare.rpt*. The details include the mismatching signals, their reference and compared values, and the timestamp of the earliest discrepancy found. Leveraging the SimCompare capability, it was possible to check if each detected fault in PXP had the same detection time as reported in IFSS. From the 288 injected faults, only 46 were not detected as shown in Table 22. All other faults presented same detection time on both engines thus achieving the objective of validating the MADC for this initial test case. The detection information resulted from SimCompare is parsed and annotated back in the PG database as illustrated in Figure 40. The execution time required for the simulation and by the hardware-assisted platform is shown in Table 22 subdivided according to the fault detection status.

Table 22 – MADC performance results.

Status	**Faults**	**Acceleration**	**Simulation**	**Ratio**
Detected	242 (84.03%)	6,372.67s	1,285.67s	▼ 4.96
Undetected	46 (15.97%)	1,211.33s	3,954.33s	▲ 3.46
Total	**288**	**7,584.00s**	**5,240.00s**	**▼ 1.45**

The fault runs were executed sequentially on both platforms. The overall time results on Table 22 shows that the overall execution time achieved with the hardware-assisted platform was 1.45 times slower than simulation. This negative performance is mainly related to the time overhead for uploading the design to the acceleration platform and for dumping the waveform data at the end of each fault injection. Additionally, IFSS was configured to stop as soon as a fault is detected, but such capability for the acceleration platform is not yet implemented, and the post-run comparison based on SimCompare is the solution used within MADC flow.

Considering only the undetected faults, which require the test to be fully executed, the acceleration gain has been 3.46 in comparison to simulation according to the results in Table 20. The acceleration factor per fault achieved by the hardware-assisted platform with the undetected faults portion (i.e., only 15.97% of the faults) is higher than the factor reached by simulation considering the detected faults. In other words, if the ratio of undetected faults increases, then the overall execution time with the acceleration engine is likely to be better than with simulation.

It is important to notice that the overall performance gain is associated with the detection profile of the fault set under consideration. Additionally, the strobes were placed on the boundary of the fault target instance. So, the location of the selected detection points means they are sequentially close to where the faults are injected. Since an SM, external to the ALU or even to the CPU, would result in strobes defined sequentially more distant, thus requiring a longer run until detection hence reducing the simulation performance associated with the group of detected faults – that stops the simulation at detection.

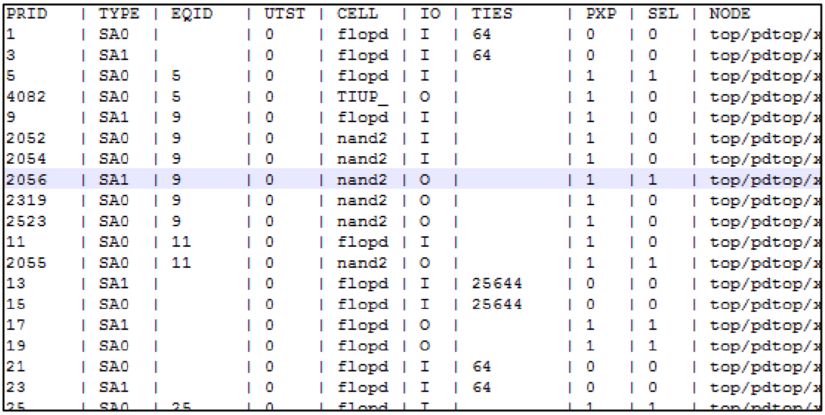
7.3. USING MADC IN A SEQUENTIAL CIRCUIT

The unsigned carry adder taken as fault target is purely combinatorial thus increasing the chances to rapidly detecting the injected faults. The technique used by the SM must also be taken into account. For instance, the evaluation of an SBST as an SM – that executes diagnostic routines during boot up or periodically for example – would naturally require longer simulations. Therefore, in such case, the usage of acceleration could be extremely beneficial for the performance of the SM evaluation.

7.3.1. Tick Timer - Peripheral

To validate such argument, a block with sequential elements at a higher level of the OpenRISC architecture was selected for the evaluation of the MADC fault target. The Tick Timer (TT) block illustrated in Figure 43 was chosen as the fault target of the injection campaign for a sequential circuit. The Tick Timer unit provides a programmable counter at the clock frequency that can be used to interrupt the CPU once after a counter threshold is reached or periodically according to the configured time interval. Even if the software application running on OpenRISC does not use the Timer, a fault can still propagate out the Tick Timer block and cause an unexpected CPU interruption, for example. For this to happen, a fault may propagate through many registers until triggering an interruption or any other disturbance to the CPU. Therefore, the detection profile of the faults injected in the Tick Timer unit is expected to be different from the one presented with the unsigned carry adder (combinational circuit).

Again the structural analysis developed is utilized to select the faults that are going to be injected using the hardware-assisted platform. By providing the instance path of the Tick Timer unit, the SAT fault list is generated using the ATPG tool and the fault node information is collected via the synthesis tool. The node information corresponds to the details of each cell pin where the faults reported by the ATPG tool are located. These details are the number of ports connected to the pin (TIES), the pin direction (IO), and the cell name (CELL) back annotated to the PG database as illustrated in Figure 45. The prime fault ID (PRID), the equivalent grouped ID (EQID), the fault type (TYPE), and the testability (UTST) information are extracted from the reports generated by the ATPG tool. Notice that the faults which share the same EQID are members of the same equivalence group from where one fault, preferably located in a cell output, is marked as selected (SEL) for acceleration – see example highlighted in Figure 45. Although, faults residing in a cell input can be picked for acceleration when the collapsing group contains only two faults located in different cells – i.e., due to “1-1 LINE” collapsing rule. For instance, from the first two faults sharing the same EQID in Figure 45, the fault with PRID equals to five is selected instead to the other fault which its IO value indicates output.



```
PRID    | TYPE | EQID   | UTST | CELL  | IO | TIES    | PXP | SEL | NODE
1       | SA0  |        | 0    | flopd | I  | 64      | 0   | 0   | top/pdtop/x
3       | SA1  |        | 0    | flopd | I  | 64      | 0   | 0   | top/pdtop/x
5       | SA0  | 5      | 0    | flopd | I  |         | 1   | 1   | top/pdtop/x
4082    | SA0  | 5      | 0    | TIUP_ | O  |         | 1   | 0   | top/pdtop/x
9       | SA1  | 9      | 0    | flopd | I  |         | 1   | 0   | top/pdtop/x
2052    | SA0  | 9      | 0    | nand2 | I  |         | 1   | 0   | top/pdtop/x
2054    | SA0  | 9      | 0    | nand2 | I  |         | 1   | 0   | top/pdtop/x
2056    | SA1  | 9      | 0    | nand2 | O  |         | 1   | 1   | top/pdtop/x
2319    | SA0  | 9      | 0    | nand2 | O  |         | 1   | 0   | top/pdtop/x
2523    | SA0  | 9      | 0    | nand2 | O  |         | 1   | 0   | top/pdtop/x
11      | SA0  | 11     | 0    | flopd | I  |         | 1   | 0   | top/pdtop/x
2055    | SA0  | 11     | 0    | nand2 | O  |         | 1   | 1   | top/pdtop/x
13      | SA1  |        | 0    | flopd | I  | 25644   | 0   | 0   | top/pdtop/x
15      | SA0  |        | 0    | flopd | I  | 25644   | 0   | 0   | top/pdtop/x
17      | SA1  |        | 0    | flopd | O  |         | 1   | 1   | top/pdtop/x
19      | SA0  |        | 0    | flopd | O  |         | 1   | 1   | top/pdtop/x
21      | SA0  |        | 0    | flopd | I  | 64      | 0   | 0   | top/pdtop/x
23      | SA1  |        | 0    | flopd | I  | 64      | 0   | 0   | top/pdtop/x
25      | SA0  | 25     | 0    | flopd | I  |         | 1   | 1   | top/pdtop/x
```

Figure 45 – MADC PG database content illustration.

Actually, any of the faults with EQID equals to nine in Figure 45 can be selected for acceleration. These faults are marked in Figure 46.

This flexibility can happen when the cell inputs where the faults are located have no more than two connections indicated by an empty value in the TIES column – i.e., faults with PRID equals to 9, 2052, and 2054. However, this is not always the case as highlighted in Figure 47, where some of the faults sharing the same EQID are located in cell input pins with ties greater than two. If instead of using the fault from the highlighted line, the fault with PRID equals to 3954 would have been injected using the *force* command, then 30 other cells would be affected from the moment the fault is inserted. At least one input of these 30 cells is tied to the same line that the injected cell input is tied as well. The remaining tie corresponds to line driver, which is naturally a cell output.

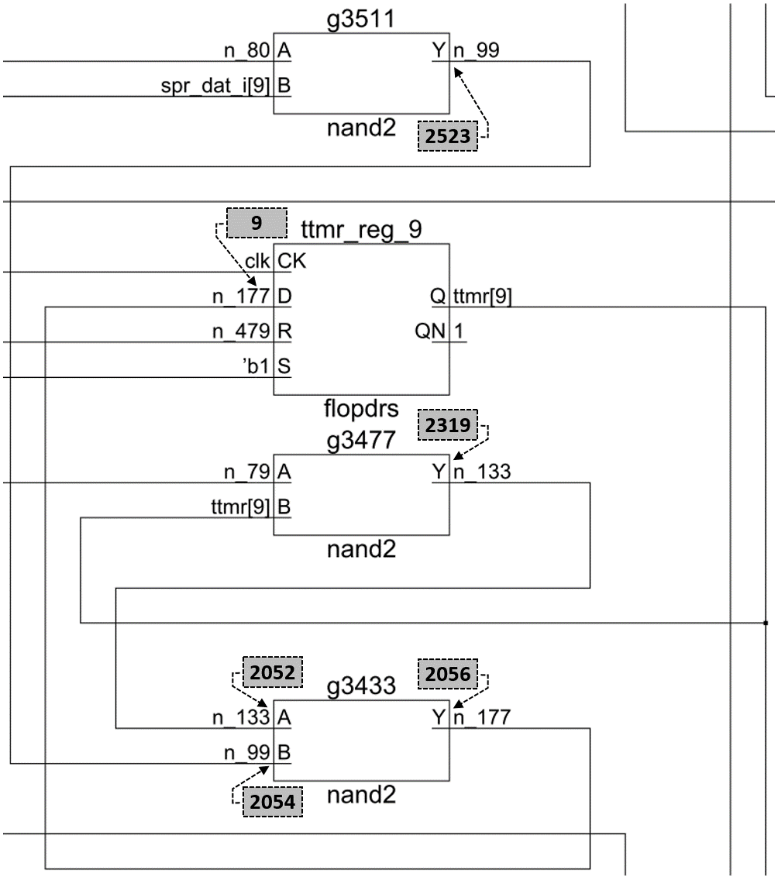
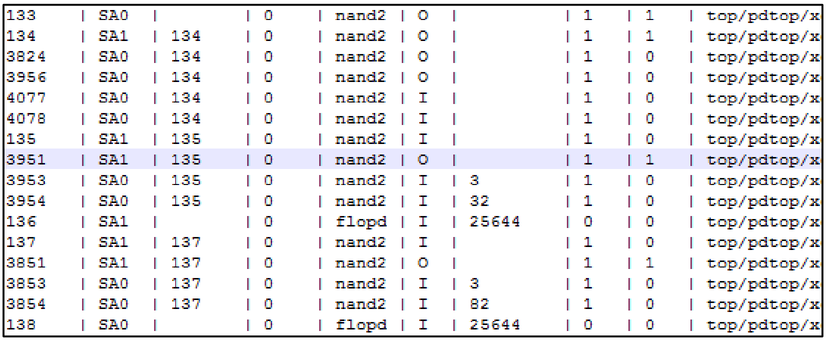


Figure 46 – Fault equivalence group number nine illustrated in Figure 46.

Actually, any of the faults with EQID equals to nine in Figure 45 can be selected for acceleration. This flexibility can happen when the cell inputs where the faults are located have no more than two connections indicated by an empty value in the TIES column – i.e., faults with PRID equals to 9, 2052, and 2054. However, this is not always the case as highlighted in Figure 47, where some of the faults sharing the same EQID are located in cell input pins with ties greater than two. Figure 48 shows how the faults with EQID equals to 135 are distribute in the TickTimer circuit. If instead of using the fault from the highlighted line, the fault with PRID equals to 3954 would have been injected using the *force* command, then 30 other cells would be affected from the moment the fault is inserted. At least one input of these 30 cells is tied to the same line that the injected cell input is tied as well. The remaining tie corresponds to line driver, which is naturally a cell output.



```
133    | SA0  |       | 0   | nand2 | O |       | 1  | 1  | top/pdtop/x
134    | SA1  | 134   | 0   | nand2 | O |       | 1  | 1  | top/pdtop/x
3824   | SA0  | 134   | 0   | nand2 | O |       | 1  | 0  | top/pdtop/x
3956   | SA0  | 134   | 0   | nand2 | O |       | 1  | 0  | top/pdtop/x
4077   | SA0  | 134   | 0   | nand2 | I |       | 1  | 0  | top/pdtop/x
4078   | SA0  | 134   | 0   | nand2 | I |       | 1  | 0  | top/pdtop/x
135    | SA1  | 135   | 0   | nand2 | I |       | 1  | 0  | top/pdtop/x
3951   | SA1  | 135   | 0   | nand2 | O |       | 1  | 1  | top/pdtop/x
3953   | SA0  | 135   | 0   | nand2 | I | 3     | 1  | 0  | top/pdtop/x
3954   | SA0  | 135   | 0   | nand2 | I | 32    | 1  | 0  | top/pdtop/x
136    | SA1  |       | 0   | flopd | I | 25644 | 0  | 0  | top/pdtop/x
137    | SA1  | 137   | 0   | nand2 | I |       | 1  | 0  | top/pdtop/x
3851   | SA1  | 137   | 0   | nand2 | O |       | 1  | 1  | top/pdtop/x
3853   | SA0  | 137   | 0   | nand2 | I | 3     | 1  | 0  | top/pdtop/x
3854   | SA0  | 137   | 0   | nand2 | I | 82    | 1  | 0  | top/pdtop/x
138    | SA0  |       | 0   | flopd | I | 25644 | 0  | 0  | top/pdtop/x
```

Figure 47 – Importance of the right fault selection from an equivalent group.

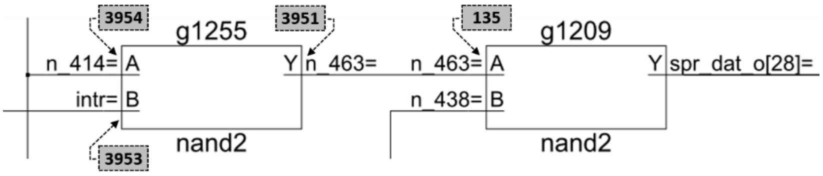


Figure 48 – Fault equivalence group number 135 illustrated in Figure 47.

The example highlighted in Figure 47 exposes how important is to select the proper fault from each collapsed group in order to guarantee the valid fault injection results. Consequently, this emphasizes the value in the contribution of the proposed MADC which automatically generates a list of the selected faults.

The Tick Timer Unit has 4,014 SAT fault candidates according to the results from the ATPG tool. Additional 128 faults located in TIEUP logic are marked as untestable hence not considered for acceleration. The ratio of faults suitable for acceleration is shown in Table 23. The percentages of faults that can be accelerated before and after collapsing are similar to the numbers found for the unsigned carry adder shown in Table 21.

Table 23 – Tick Timer fault candidates that are suitable for acceleration.

Faults that can be:	**MADC Analysis**			
	Faults	**Ratio (%)**	**Prime**	**Ratio (%)**
Accelerated	3,031	~75.51	909	~48.04
Simulated	983	~24.49	983	~51.95
Total	**4,014**		**1,892**	

Table 24 compares the results from the CA and TT test cases regarding the average run time required for each fault injection.

Table 24 – Runtime average per fault injection execution.

Engine	**Run time per fault injection**			
	CA	Time/#Faults	**TT**	Time/#Faults
Acceleration	7,584s	~26.33s	22,025s	~24.23
Simulation	5,240s	~18.19s	56,042s	~61.65
#Faults / **Ratio**	288	**▼ 1.45**	909	**▲ 2.54**

For the CA test case, the average execution time of each fault run was 1.45 times better than the acceleration. However, the experiments done with the TT instance, which has less than four times the number of faults in CA, already presented acceleration gain over simulation. As explained earlier, this alteration in the results was expected due to the fault injection profile. The TT block has registers thus making more difficult to propagate the fault. This affects the number of faults that are undetected, which for the TT test case corresponded to 74% of the injected faults. Another factor that impacts the average injection time is the test duration. The test duration for the TT example is four times longer than the one used for the CA, in order to run at least up to the point where the software running in the OpenRISC can initialize TT block. Given that no SM is used, the outputs of the TT were selected as strobe points thus not contributing to a longer detection time.

The acceleration achieved with the TT test case points to an opposite direction from the numbers obtained in the first experiment due to the profile differences between the two fault campaigns executed. Table 24 shows that average execution time for the fault injection on TT was 2.54 times faster when comparing with the injection of the same faults via simulation. This result corroborates to the idea that the campaign profile has a significant impact on the benefit potential of MADC. Fault campaigns on complex designs with not so many shortly detected faults, running complex TBs, with specific SMs – e.g., SBST – are likely to benefit from the proposed MADC approach. The benefit potential becomes more evident when separating the amount of time spent on each fault injection step executed in the hardware-assisted platform. Table 25 shows the average time required for loading the snapshot on PXP, injecting the fault while running the test, and comparing the waveform databases.

Table 25 – Runtime average of each acceleration step for the TT test case.

Engine	**Total Runtime**	**Injection Time**	**Strobes Check Time**	**Snapshot Load Time**
Acceleration	24.23s	7.87s (32.47%)	10.76s (44.42%)	5.6s (23.11%)
Simulation	61.65s	61.65 (100%)	–	–
Ratio	**▲ 2.54**	**▲ 7.83**	**–**	**–**

If equivalent fault injection features supported on IFSS – i.e., stop at detection, and runtime comparison between good and fault run – would be available on the hardware-assisted platform, then the time spent with the strobes comparison could be saved. In the TT fault campaign, the waveform databases comparison consumed almost half of the fault injection run time as indicated in Table 25. Other approaches are being investigated in order to cope with this bottleneck.

By only considering the actual time spent for a fault injection execution on PXP, and comparing with simulation, the difference would be almost eight times (▲ 7.83) faster for the TT test case as shown in Table 25. However, even if the comparison time could be neglected, there is also the time consumed for loading the snapshot in the acceleration platform that must be considered. The loading time seems to be constant and mostly associated with the size of the snapshot and the quality of the network access to the area where the generated snapshot is located. Therefore, additional to the campaign profile, the snapshot size and the network quality needs to be counted when considering the trade-off between acceleration and simulation. In case DRM is available, the number of fault simulations in parallel that can be executed, must be examined. The same applies to the possibility of using multiple PXP domains for the fault injection – as it is used in [87].

7.3.2. Exception Handler – CPU Block

The Exception Handler (EH) test case corresponds to one of the CPU blocks (Except) shown in Figure 44. The EH block has almost three times more gates than the TT unit. The difference in the number of faults between the EH and TT has similar ratio. The TT unit has 4,014 faults almost a third of the 11,798 faults found in the EH instance. Table 26 summarizes the number of faults located in the analyzed CPU block.

Table 26 – Exception handler fault candidates that are suitable for acceleration.

Faults that can be:	**MADC Analysis**			
	Faults	**Ratio (%)**	**Prime**	**Ratio (%)**
Accelerated	10,030	85.01%	3,472	66.26%
Simulated	1,768	14.99%	1,768	33.74%
Total	**11,798**		**5,240**	

The fault set for the EH test case presented higher collapsed ratio if compared to the other two test cases. This fault set reduction due to collapsing has a positive impact on the MADC algorithm, which could then select more than 85% of all faults for acceleration. When considering only the prime faults, then 66.26% of the faults are suitable for acceleration. This percentage corresponds to a campaign where 3,742 faults run on the hardware-assisted platform meanwhile the other 1,768 are simulated.

Table 27 compares the runtime average for each fault injection between the TT and the EH test cases. Once again, the size of the of the block has influence on the campaign profiles thus highlighting even more the performance gain achieved with acceleration in comparison to simulation. It is important to notice that the runtime average per fault injection using the hardware-assisted platform ranges between 24.23 and 26.33 considering the three test cases analyzed. With the runtime average not increasing, then acceleration shows great advantage over simulation, especially when there are many undetected faults among the MADC selection thus requiring to execute the whole test.

Table 27 – Runtime average per fault injection execution.

Engine	**Run time per fault injection**			
	TT	Time/#Faults	**EH**	Time/#Faults
Acceleration	22,025s	~24.23	88,408.4s	~25.46
Simulation	56,042s	~61.65	357,133.0s	~102.86
#Faults / **Ratio**	909	**▲ 2.54**	3472	**▲ 4.04**

Table 28 shows the runtime average comparison between acceleration and simulation in total and the stepwise. The EH test case shows almost 60% performance ratio increase (▲ 4.04) than in the TT campaign. By analyzing the time spent on each step, and considering only the injection runtime, then the acceleration difference goes as high as 12.54 times faster than simulation.

Table 28 – Runtime average of each acceleration step for the EH test case.

Engine	**Total Runtime**	**Injection Time**	**Strobes Check Time**	**Snapshot Load Time**
Acceleration	25.46s	8.20s (32.21%)	11.82s (46.43%)	5.44s (21.36%)
Simulation	102.86s	102.86 (100%)	–	–
Ratio	**▲ 4.04**	**▲ 12.54**	**–**	**–**

Basically the same percentage (60%), related to the performance ratio increase, is achieved when comparing the gains resulted from the TT (▲ 4.04) and the EH (▲ 12.54) test cases, when considering the fault injection runtime only. The same workload and test duration were utilized in both fault injection campaigns. The performance difference is believed to be associated to the number of observations points selected for each test case – all outputs of the corresponding module. The number of single bits selected as strobe points for the EH test case is 271, which corresponds to 8.21 times more bits than the 33 output signals found in the TT block. By not comparing the strobe data generated during the run, but instead at once as post-run process, then the amount of computation to check if the fault was detected, done with MADC, is not affected as much as for simulation.

7.4. OPENRISC TEST CASE

Together with the OpenRISC CPU, some peripherals, and a Wishbone Bus compose the SoC illustrated in Figure 49. A former version of the OpenRISC SoC already running on PXP was used to gain experience with the hardware-assisted platform. Many peripherals like PS2 and Ethernet Interface come along to that OpenRISC SoC version, and they were used together with a Linux distribution which can execute over the HDL of the design running on PXP and simulation as well. While the Linux boot up on PXP takes few minutes, over simulation the same procedure takes more than a couple of hours. Since such long test case was not suitable for the fault simulation at beginning, then a bare metal code application had to be used instead. However, only the source code and the image was available without the toolchain to compile a new one. Moreover, the toolchain available in the OpenRISC project website [106] could not be used since the executable code generated was not aligned to the outdated OpenRISC HDL version used. Therefore, all modules of the OpenRISC SoC were updated to the latest version available in the OpenRISC website repository. Hence the toolchain could be used to generate the memory image for each modification in the software code. This task was essential to get more familiar with the OpenRISC HDL description.

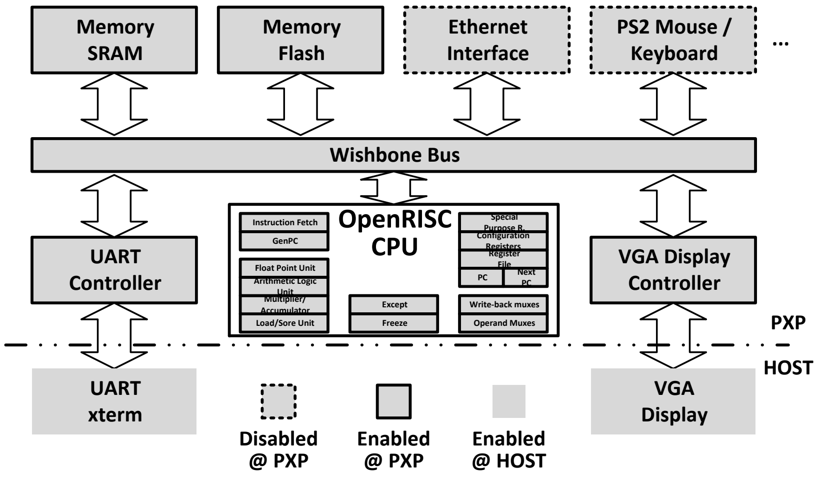


Figure 49 – OpenRISC SoC.

The bare metal code executed by the OpenRISC generates image data that is copied to the memory address range used by the VGA Controller. A pseudo display window (VGA Display) is emulated in the host – it could be a physical display connected directly to PXP instead – where the VGA controller constantly updates with the image being generated. Meanwhile, a pseudo terminal (UART xterm) also running on the host shows the output printed by the program being executed in the OpenRISC. Notice that Ethernet, PS2, and I2C peripherals are not used in this application hence they were commented off from the OpenRISC HDL description. A routine interacting with the Tick Timer was added to the VGA bare metal code example used. At least two milliseconds of simulation is required to initialize and configure the VGA and TT modules, and start to execute few iterations of the image generation routine. The good simulation for the TT experiment took 331s to execute two milliseconds of the VGA test at GL. Given the long runtime, the VGA test seemed to be a suitable candidate to explore the MADC potential.

SM based in software like SBST can be scheduled to execute when the vehicle starts, during the operation phase or before turning off [108]. Use fault injection to assess the DC of such mechanism is recommended by the ISO 26262 as already mention. However, an SM routine scheduled to execute at power-off of the vehicle requires the simulation of at least the complete start sequence, which may be prohibitive due to the given simulation time needed, especially at GL. Therefore, the proposed MADC has significant contribution optimizing such scenarios. The complex workload prepared for the safety RISC example cited in [39] resulted in an average of 56.6s for each injected fault. Hence it is another candidate for acceleration leveraging the presented MADC.

7.5. MADC PERFORMANCE COMPARISON

As already mentioned, a similar approach based on design intrusion was proposed in a research developed at STMicroelectronics Inc. (ST) [87]. The proposed approach has an area overhead equivalent to eight gates per instrumented cell that is selected as fault target plus the fault injection controller developed. The controller occupies almost three times more resources on PXP than the tested IP itself – a Leon2 processor. Only SEU faults are considered, but the authors claim that SAT fault model is also supported by using the same instrumentation method. To cope with the massive area overhead to instrument both fault models on all cells, only few flip-flops are selected for the campaign, and they are grouped whenever possible in order to share the instrumentation logic. This optimization leads to area overhead minimization around 20% of the fault target.

The results presented in the ST research are extracted from a fault injection campaign composed of 65,380,350 SEU faults injected in 2,631 flip-flops at 24,850 different times. Given the small size of Leon2 IP, 19 instances could be mapped per PXP domain. Using 16 domains, the developed approach could inject 304 fault in parallel. The full campaign execution last ten hours, which corresponds to more than 1.8k faults per second.

Table 29 brings the ST's Leon2 and the EH test case fault injection campaigns to a similar perspective in order to allow comparing the performance results. The ST's technique achieves minimized runtime average per fault injection due to the substantial parallelism. Even considering the same number of runs in parallel, the proposed MADC would still result in a performance about 152 times lower than the intrusive method from the research developed at ST. Although, there are some difference in the campaigns that may explain such performance discrepancy. For example, the Leon2 design is approximately four and an half times smaller than the OpenRISC, which impacts in the snapshot loading time to PXP. The gate count number is no considering the OpenRISC memories since it is not clear whether they are accounted in ST's example. Additionally, by injecting many faults in the same node, considerably reduces the amount of instrumentation required by the proposed approach developed at ST. To inject faults in more candidate, the area overhead would limit the number of possible instances per domain, thus reducing the parallelism.

Table 29 – MADC comparison.

Characteristics	**Campaigns**			**Ratio**
	ST	**EH**		
Design	Leon2	OpenRISC		–
#equ. nand 2	56,565	259,149		**~4.58**
Fault target	flip-flops	All cells		–
#candidates	2,631	10,030		
#injections	65,380,350	3,471		
#Parallelism	304	1	304[a]	–
Time / Fault	0.0005506s	24.46s	0.08376s	**~152.12**

[a] assuming the same number of instance × domains used for the EH campaign.

The workload used in the research at ST's executes the boot sequence, the program initialization plus 59,283 cycles, which corresponds to the fault injection time window. One million cycles are executed in the EH test case. Without knowing how many cycles are needed to boot up the Leon2 processor and initialize the program, it is not possible to perform a fair comparison. However, the test duration must also be considered when comparing the performance achieved in the two researches.

The proposed non-intrusive approach presented in this Thesis has room for improvement, especially considering the implemented comparison technique which consumes up to 46.43% of the fault injection runtime. In addition, similar parallelization approach leveraged in [87] can boost MADC performance in many times. Another important aspect of the MADC approach is that around 50% of the faults can be simulated using a computer farm while the rest of the faults is accelerated in the hardware-assisted platform thus giving another parallelism factor. Such performance potential combined with the non-intrusive characteristic makes MADC more suitable for the semiconductor industry when targeting safety automotive applications.

7.6. CHAPTER REMARKS

This chapter presents the experiments performed to guarantee the correctness of the results obtained with MADC regardless the achieved optimization. A second experiment was presented in order to demonstrate the potential benefit of employing MADC for the safety assessment of SMs in complex semiconductor design where long simulation runs are required, and hence acceleration can minimize the fault campaign execution time. The test cases used are discussed, and the importance of being an open-source design is highlighted.

Different from any other solution found in the literature, MADC provides SAT fault injection acceleration without imposing the modification of the design model to enable the fault campaign. Therefore, the proposed solution fits the requirements specified in the ISO 26262 as already discussed.

8. CONCLUSION

In this Thesis, we presented the MADC approach that enables the acceleration of DC assessment by leveraging the most advanced functional verification solutions including the latest fault simulator tools that already target functional safety. The verification environment (e.g., TB) can be seamlessly reused for the stimulus generation and therefore limitations from DfT oriented fault simulators (e.g., lack of TB support) do not apply to MADC. Benefiting from the cutting-edge emulators like those used in [86], MADC provides a non-intrusive solution, thus not requiring any design modification, and hence it is more likely to satisfy an ISO 26262 auditor. The intrusion characteristic is what makes MADC different from any other fault injection acceleration solution. Especially when comparing MADC to approaches based on FPGAs, where the design model is synthesized to a different technology, and sometimes also modified to enable acceleration. Moreover, the only solution using the same technology as MADC – i.e., hardware-assisted platform – to accelerate the fault injection campaigns, also relies on design instrumentation in order to enable the flow.

MADC is part of a methodology being developed that extracts data from an FMEDA to assess the initially estimated DC numbers. This assessment must be done to provide enough confidence in the results. The results are annotated back to the FMEDA, and the hardware architecture metrics can be recalculated based on the actual design data. The conservative estimations and the refined metrics shall match. Moreover, to comply with the ISO 26262, it is fundamental considering the state-of-the-art. Therefore, MADC is in evidence since it leverages the latest verification solutions to allow a more thorough safety assessment by reusing the functional verification environment and providing significant acceleration.

An OpenRISC architecture has been used as test case for validating the developed MADC flow. Negative performance results were observed when injecting faults in a small combinatorial block of the OpenRISC ALU. However, considerable acceleration gain was observed when analyzing a larger block containing sequential cells. The existence of a threshold defining whether the MADC can be leveraged is discussed. This threshold seems to be influenced by the fault campaign profile. The profile includes the ratio of undetected faults, the test length, the sequential distance of the strobe points in relation to the fault injection locations, the SMs being evaluated, among other aspects. The obtained results using the MADC show great potential benefit when considering its application to assess SMs based on software that naturally require long runs regardless the proportion of undetected faults.

The results achieved could confirm the initial hypothesis that by selecting higher instances within the OpenRISC design, an actual acceleration gain can be obtained by employing MADC. Although, the implemented mechanism to enable fault detection on the verification platform imposes a significant bottleneck to the MADC performance, yet positive results could be observed. Therefore, the proof-of-concept built allowed to confirm the aimed contribution of this Thesis. Meanwhile, a solution for the bottleneck is being investigated to maximize the potential of the proposed MADC solution.

A comparison between MADC and the most similar research work found in the literature was presented. The design model intrusion required by the compared approach makes the usage justification of such solution much more difficult, thus highlighting the MADC main contribution. Despite the intrusion aspect, the fault injection performance was compared. The massive parallelism implemented by the intrusive solution allowed achieving remarkable performance results. MADC could not get the same performance by considering if the same amount of parallelism would have been implemented in this Thesis. However, many arguments were discussed in order to provide a rational explanation for the performance difference. From this discussion, it was possible to identify some optimizations including the parallelism strategy, which can be leveraged by MADC.

In this Thesis, the developed research targets the functional safety for car applications. Therefore, a thorough introduction to the ISO 26262 standard, which drives the functional safety requirements in the automotive segment, is presented. However, MADC as a generic fault injection acceleration solution can be leveraged by other industry segments. For example, in avionics, the functional safety standards make MADC a possible solution. Additionally, the scope defined in the first edition of the ISO 26262 will expand with the new release. Not just the car mass restriction is going to be dropped, but the scope will include motorcycles and different series production vehicles as well.

As future work, the MADC can be extended to be used to support SEU faults or to accelerate fault injection at RTL. The *deposit* command – standard simulator feature and also available in the hardware-assisted platforms – can be used to mimic the SEU faults. The backward propagation is not a problem for SEU injection since the fault model applies to sequential elements outputs only. Fault injection at RTL can only be done at outputs of processes or assignments since the SAT fault model has no clear meaning at the behavioral level. Therefore, MADC can be leveraged also to inject faults at RTL, thus allowing early verification of the DC and hardware architectural metrics.

REFERENCES

[1] Infineon Technologies AG, “Sensor Solutions for Automotive, and Industrial Applications,” 2015. [Online]. Available: www.infineon.com/dgdl/Infineon-Sensor_Solutions_for_Automotive_Industrial_and+Customer_Appl_BR-2015.pdf?fileId=5546d4614937379a01495212845c039f. [Accessed: 10-Jan-2016].

[2] L. Reger, “1.4 The road ahead for securely-connected cars,” in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 29–33.

[3] J. Gorzelany, “The Hottest 2016 New-Car Features,” 2016. [Online]. Available: www.forbes.com/forbes/welcome/#539c3c263e57. [Accessed: 11-Jul-2016].

[4] R. Nazaretian and G. M. Molen, “Reducing Vehicle Weight and Improving Security by Using Plastic Optical Fiber,” in *2015 IEEE Vehicle Power and Propulsion Conference (VPPC)*, 2015, pp. 1–6.

[5] Freescale Semiconductor Inc., “Future Advances in Body Electronics,” 2013. [Online]. Available: www.nxp.com/files/automotive/doc/white_paper/BODYDELECTRWP.pdf. [Accessed: 20-Jan-2016].

[6] International Organization for Standardization, “ISO26262 - Road Veichles - Functional Safety,” 2012.

[7] Cadence Design Systems, “Incisive Functional Safety Simulator,” 2015.

[8] Synopsys, “Certitude.” [Online]. Available: www.synopsys.com/Tools/Verification/FunctionalVerification/Pages/certitude-ds.aspx. [Accessed: 15-Dec-2016].

[9] I. Synopsys, “Z01X Functional Safety Assurance,” 2016. [Online]. Available: www.synopsys.com/Tools/Verification/FunctionalVerification/Pages/z01x-functional-safety.aspx. [Accessed: 01-Jul-2016].

[10] Mentor Graphics, “Veloce2,” *2013*. [Online]. Available: www.mentor.com/products/fv/emulation-systems/veloce. [Accessed: 16-Jan-2016].

[11] Synopsys, “Zebu.” [Online]. Available: www.synopsys.com/Tools/Verification/hardware-verification/emulation/Pages/default.aspx. [Accessed: 16-Jan-2016].

[12] Cadence Design Systems, “Palladium XP.”

[13] M. Kooli and G. Di Natale, “A survey on simulation-based fault injection tools for complex systems,” in *Proceedings - 2014 9th IEEE International Conference on Design and Technology of Integrated Systems in Nanoscale Era, DTIS 2014*, 2014.

[14] Y. S. Jeong, S. M. Lee, and S. E. Lee, “A Survey of Fault-Injection Methodologies for Soft Error Rate Modeling in Systems-on-Chips,” *Bull. Electr. Eng. Informatics*, vol. 5, no. 2, pp. 169–177, 2016.
[15] International Organization for Standardization, “ISO26262 - Road Veichles - Functional Safety - Part 1: Vocabulary,” techreport, 2011.
[16] International Organization for Standardization, “ISO26262/CD - Road Veichles - Functional Safety - Part 12: Adaptation for motorcycles,” techreport, 2016.
[17] International Organization for Standardization, “ISO26262/CD - Road Veichles - Functional Safety - Part 11: Application of concepts for semiconductors,” techreport, 2016.
[18] L. Rierson, “History of DO-178,” in *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance*, 1st editio., CRC Press, 2013, pp. 51–55.
[19] A. J. Kornecki and J. Zalewski, “Hardware certification for real-time safety-critical systems: State of the art,” *Annu. Rev. Control*, vol. 34, no. 1, pp. 163–174, 2010.
[20] A. G. Foord, W. G. Gulland, and C. E. Howard, “Ten Years of IEC 61508; Has It Made Any Difference?,” *IChemE Symp.*, no. 156, pp. 232–237, 2011.
[21] MTL Instruments Group, “An introduction to Functional Safety and IEC 61508,” *MTL Instruments Gr.*, no. HCSS++, 2002.
[22] P. Koopman, “A Case Study of Toyota Unintended Acceleration and Software Safety,” Pittsburgh, PA, 2014.
[23] B. Vlasic and M. Apuzzo, “Toyota Is Fined $1.2 Billion for Concealing Safety Defects,” *The New York Times*, New York, NY, pp. 1–4, 14-Mar-2014.
[24] D. J. Smith and K. G. L. Simpson, *Safety Critical Systems Handbook: A Straightforward Guide to Functional Safety, IEC 61508 and Related Standards*, 3rd Edtion. Oxford: Elsevier Ltd, 2010.
[25] J. Schwarz and J. Buechl, “Preparing the future for functional safety of automotive E/E-systems,” in *21st (ESV) International Technical Conference on the Enhanced Safety of Vehicles*, 2009, pp. 1–3.
[26] M. Schmidt, M. Rau, E. Helmig, and B. Bauer, “Functional Safety – Dealing with Independency, Legal Framework Conditions and Liability Issues,” 2011.
[27] “ISO 26262 Academy Guide,” in *International Conference ISO 26262*, 2016, pp. 1–22.
[28] International Organization for Standardization, “ISO26262 - Road Veichles - Functional Safety - Part 10: Guidelines on ISO 26262,” techreport, 2012.
[29] R. Mariani, “The impact of functional safety standards in the design and test of reliable and available integrated circuits,” in *2012 17TH*

IEEE EUROPEAN TEST SYMPOSIUM (ETS), 2012, vol. 2011, pp. 1–1.

[30] International Organization for Standardization, “ISO26262 - Road Veichles - Functional Safety - Part 2: Managment of functional safety,” techreport, 2011.

[31] A. Altby and D. Majdandzic, “Design and implementation of a fault-tolerant drive-by-wire system,” Chalmers University of Technology, 2014.

[32] International Organization for Standardization, “ISO26262 - Road Veichles - Functional Safety - Part 5: Product development at the hardware level,” techreport, 2011.

[33] DoD-US, “Procedures for Performing a Failure Mode, Effects and Criticality Analysis,” Washington, 1998.

[34] M. Chaari, W. Ecker, C. Novello, B. Tabacaru, and T. Kruse, “A model-based and simulation-assisted FMEDA approach for safety-relevant E/E systems,” in *Proceedings of the 52nd Annual Design Automation Conference on - DAC ’15*, 2015, pp. 1–6.

[35] L. Entrena, C. López-Ongil, M. García-Valderas, M. Portela-García, and M. Nicolaidis, “Hardware Fault Injection,” in *Soft Errors in Modern Electronic Systems*, M. Nicolaidis, Ed. Boston, MA: Springer US, 2011, pp. 141–166.

[36] Yogitech Spa, “YOGITECH’s Smart Comparator for Lockstep Solution using ARM® Cortex®-R5,” 2015.

[37] R. Mariani, T. Kuschel, and H. Shigehara, “A flexible microcontroller architecture for fail-safe and fail-operational systems,” *2nd HiPEAC Work. Des. Reliab.*, 2010.

[38] R. Mariani, G. Boschi, and F. Colucci, “Using an innovative SoC-level FMEA methodology to design in compliance with IEC61508,” in *2007 Design, Automation & Test in Europe Conference & Exhibition*, 2007, pp. 1–6.

[39] A. Benso, A. Bosio, S. Di Carlo, and R. Mariani, “A Functional Verification based Fault Injection Environment,” in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, 2007, pp. 114–122.

[40] J.-L. Huang, J. C.-M. Li, and D. M. (Hank), “Logic and Fault Simulation,” in *VLSI Test Principles and Architectures: Design for Testability*, C.-W. Wu, L.-T. Wang, and X. Wen, Eds. Morgan Kaufmann, 2006, pp. 105–159.

[41] Y.-C. Chang, L.-R. Huang, H.-C. Liu, C.-J. Yang, and C.-T. Chiu, “Assessing automotive functional safety microprocessor with ISO 26262 hardware requirements,” in *Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test*, 2014, pp. 1–4.

[42] T. Instruments, “Safety Analysis Report,” *Technical Documents*, 2012. .

[43] International Organization for Standardization, “ISO26262 - Road Veichles - Functional Safety - Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses,” techreport, 2011.

[44] S. Reiter, M. Pressler, a. Viehl, O. Bringmann, and W. Rosenstiel, “Reliability assessment of safety-relevant automotive systems in a model-based design flow,” *2013 18th Asia South Pacific Des. Autom. Conf.*, pp. 417–422, Jan. 2013.

[45] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[46] B. E. Hallett, G. Corradi, and S. Mcneil, “Xilinx Reduces Risk and Increases Efficiency for IEC61508 and ISO26262 Certified Safety Applications,” 2015.

[47] Y. Yu and B. W. Johnson, “Fault Injection Techniques: A Perspective on the State of Research,” in *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*, 1st ed., A. Benso and P. Paolo, Eds. Boston, MA: Kluwer Academic Publishers, 2004, pp. 7–18.

[48] H. Ziade, R. Ayoubi, and R. Velazco, “A Survey on Fault Injection Techniques,” *Int. Arab J. Inf. Technol.*, vol. 1, no. 2, pp. 171–186, 2004.

[49] S. Wagner, B. Schatz, S. Puchner, and P. Kock, “A Case Study on Safety Cases in the Automotive Domain: Modules, Patterns, and Models,” in *2010 IEEE 21st International Symposium on Software Reliability Engineering*, 2010, pp. 269–278.

[50] C. Ebert and A. Braatz, “Functional Safety with ISO 26262: principles and practice,” *Webinars on Functional Safety / ISO 26262*, 2014. [Online]. Available: http://vector.com/portal/medien/cmc/events/Webinars/2014/Vector_Webinar_FunctionalSafety_Principles_and_Practice_20141117_EN.pdf. [Accessed: 05-Oct-2015].

[51] T. Heijmen, “Soft Errors from Space to Ground: Historical Overview, Empirical Evidence, and Future Trends,” in *Soft Errors in Modern Electronic Systems*, M. Nicolaidis, Ed. Boston, MA: Springer US, 2011, pp. 1–25.

[52] P. E. Dodd, M. R. Shaneyfelt, J. a. Felix, and J. R. Schwank, “Production and propagation of single-event transients in high-speed digital logic ICs,” *IEEE Trans. Nucl. Sci.*, vol. 51, no. 6, pp. 3278–3284, Dec. 2004.

[53] D. Xueyan, W. Liyun, and L. Jinmei, “Effect of charge sharing on the single event transient response of CMOS logic gates,” vol. 32, no. 9,

pp. 3–8, 2011.
[54] S. Sayil and J. Wang, “Coupling induced soft error mechanisms in nanoscale CMOS technologies,” *Analog Integr. Circuits Signal Process.*, vol. 79, no. 1, pp. 115–126, Nov. 2013.
[55] M. Saremi, A. Privat, H. J. Barnaby, and L. T. Clark, “Physically Based Predictive Model for Single Event Transients in CMOS Gates,” vol. 63, no. 6, pp. 2248–2254, 2016.
[56] R. H.-M. Huang, D. K.-H. Hsu, and C. H.-P. Wen, “A Determinate Radiation Hardened Technique for Safety-Critical CMOS Designs,” *J. Electron. Test.*, 2015.
[57] J. Abella, F. J. Cazorla, E. Quinones, A. Grasset, S. Yehia, P. Bonnot, D. Gizopoulos, R. Mariani, and G. Bernat, “Towards improved survivability in safety-critical systems,” in *2011 IEEE 17th International On-Line Testing Symposium*, 2011, pp. 240–245.
[58] J. Abraham, R. Iyer, D. Gizopoulos, D. Alexandrescu, and Y. Zorian, “The future of fault tolerant computing,” in *2015 IEEE 21st International On-Line Testing Symposium (IOLTS)*, 2015, vol. 5, pp. 108–109.
[59] C. Hernandez and J. Abella, “Timely Error Detection for Effective Recovery in Light-Lockstep Automotive Systems,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 34, no. 11, pp. 1718–1729, Nov. 2015.
[60] F. Hapke, W. Redemund, A. Glowatz, J. Rajski, M. Reese, M. Hustava, M. Keim, J. Schloeffel, and A. Fast, “Cell-Aware Test,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 33, no. 9, pp. 1396–1409, 2014.
[61] M. Stanisavljević, A. Schmid, and Y. Leblebici, “Reliability, Faults, and Fault Tolerance,” in *Reliability of Nanoscale Circuits and Systems*, New York, NY: Springer New York, 2011, pp. 7–18.
[62] Z. Navabi, “Fault and Defect Modeling,” in *Digital System Test and Testable Design*, Boston, MA: Springer US, 2011, pp. 63–101.
[63] Council Automotive Electronics, “AEC - Q100-007 Rev-B:FAULT SIMULATION AND FAULT GRADING,” 2007.
[64] Y. Min and C. Stroud, “Introduction,” in *VLSI Test Principles and Architectures: Design for Testability*, C.-W. Wu, L.-T. Wang, and X. Wen, Eds. Morgan Kaufmann, 2006, pp. 1–36.
[65] S. Rahkonen, “Mutation-Based Qualification of Module Verification Environments,” Tampere University of Technology, 2015.
[66] A. Benso and P. Paolo, *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*, 1st ed., vol. 23. Boston, MA: Kluwer Academic Publishers, 2004.
[67] R. Natella, D. Cotroneo, and H. S. Madeira, “Assessing Dependability with Software Fault Injection : A Survey,” *ACM Comput. Surv.*, vol.

48, no. 3, p. 44:1-44:55, 2016.
[68] Certress, “Certitude: Delivering Functional Qualification,” 2007. [Online]. Available: http://www.iss.se/files/4/Certess_datasheet.pdf. [Accessed: 07-May-2016].
[69] M. J. S. Smith, “Fault Simulation,” in *Application-Specific Integrated Circuits*, no. 1st, Addison-Wesley Professional, 1997.
[70] Z. Navabi, “Fault Simulation Applications and Methods,” in *Digital System Test and Testable Design*, Boston, MA: Springer US, 2011, pp. 103–142.
[71] A. Efody, “Whose fault is it? Advanced techniques for optimizing ISO 26262 fault analysis,” in *Design & Verification Conference (DVCon)*, 2016, pp. 1–9.
[72] L. Entrena, M. Garcia-Valderas, R. Fernandez-Cardenal, A. Lindoso, M. Portela, and C. Lopez-Ongil, “Soft Error Sensitivity Evaluation of Microprocessors by Multilevel Emulation-Based Fault Injection,” *IEEE Trans. Comput.*, vol. 61, no. 3, pp. 313–322, 2012.
[73] J. Napoles, H. Guzman-Miranda, M. Aguirre, J. Tombs, J. Mogollon, R. Palomo, and A. Vega-Leal, “A Complete Emulation System for Single Event Effects Analysis,” in *Programmable Logic, 2008 4th Southern Conference on*, 2008, vol. 41092, pp. 213–216.
[74] J. M. Mogollon, H. Guzmán-Miranda, J. Nápoles, J. Barrientos, and M. A. Aguirre, “FTUNSHADES2: A novel platform for early evaluation of robustness against SEE,” *Proc. Eur. Conf. Radiat. its Eff. Components Syst. RADECS*, pp. 169–174, 2011.
[75] F. Ferlini, *PLASER - Plataforma de Emulação de Soft Errors*, 1st ed. Novas Edições Acadêmicas, 2015.
[76] S. Misera, H. T. Vierhaus, and A. Sieber, “Fault Injection Techniques and their Accelerated Simulation in SystemC,” in *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*, 2007, no. Dsd, pp. 587–595.
[77] J.-H. Oetjens, K. Grüttner, T. Kruse, C. Kuznik, H. M. Le, A. Mauderer, W. Müller, D. Müller-Gritschneder, F. Poppen, H. Post, S. Reiter, N. Bannow, W. Rosenstiel, S. Roth, U. Schlichtmann, A. von Schwerin, B.-A. Tabacaru, A. Viehl, M. Becker, O. Bringmann, A. Burger, M. Chaari, S. Chakraborty, R. Drechsler, and W. Ecker, “Safety Evaluation of Automotive Electronics Using Virtual Prototypes,” in *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference - DAC ’14*, 2014, pp. 1–6.
[78] S. Reiter, A. Viehl, O. Bringmann, and W. Rosenstiel, “White-Box Error Effect Simulation for Assisted Safety Analysis,” in *2015 Euromicro Conference on Digital System Design*, 2015, pp. 534–538.
[79] H. Mackey, “Sneak Peak: Inside NVIDIA’s Emulation Lab,” *The*

Official NVIDIA Blog, 2011. [Online]. Available: https://blogs.nvidia.com/blog/2011/05/16/sneak-peak-inside-nvidia-emulation-lab. [Accessed: 13-Feb-2015].

[80] P. McLellan, “NVIDIA: Ten Months of Emulation on Palladium, Hours to Bring - Up,” *Breakfast Bytes- - Cadence Blogs*, 2016. .

[81] F. Schirrmeister, “Productivity, Predictability, and Use-Model Versatility,” *White Paper*, 2013. [Online]. Available: www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/system-design-verification/palladium-xp-ii-wp.pdf. [Accessed: 30-Apr-2016].

[82] L. Rizzatti, “Hardware Emulation: Three Decades of Evolution - Part I,” *Verification Horizons*, Wilsonville, OR, pp. 26–27, 2015.

[83] L. Rizzatti, “Hardware Emulation: Three Decades of Evolution - Part II,” *Verification Horizons*, Wilsonville, OR, pp. 40–42, Jun-2015.

[84] L. Rizzatti, “Hardware Emulation: Three Decades of Evolution - Part III,” *Verification Horizons*, pp. 15–18, 2015.

[85] David Kaushinsky, “Emulation Based Approach to ISO 26262 Compliant Processors Design,” *Verification Horizons*, vol. 11, no. 2, Wilsonville, OR, pp. 47–53, Jun-2015.

[86] O. Bailan, U. Rossi, A. Wantens, J.-M. Daveau, S. Nappi, and P. Roche, “Verification of soft error detection mechanism through fault injection on hardware emulation platform,” in *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2010, pp. 113–118.

[87] J. M. Daveau, A. Blampey, G. Gasiot, J. Bulone, and P. Roche, “An industrial fault injection platform for soft-error dependability analysis and hardening of complex system-on-a-chip,” *IEEE Int. Reliab. Phys. Symp. Proc.*, no. January 2016, pp. 212–220, 2009.

[88] L. Entrena, C. López-Ongil, M. García-Valderas, M. Portela-García, and M. Nicolaidis, “Hardware Fault Injection,” in *Soft Errors in Modern Electronic Systems*, Vol. 41., M. Nicolaidis, Ed. Boston, MA: Springer US, 2011, pp. 141–166.

[89] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and A. Violante, “Exploiting FPGA for accelerating fault injection experiments,” in *Proceedings Seventh International On-Line Testing Workshop*, 2001, pp. 9–13.

[90] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, and M. Violante, “Exploiting FPGA-based Techniques for Fault Injection Campaigns on VLSI Circuits,” in *Proceedings 2001 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2001, pp. 250–258.

[91] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante, “An FPGA-based approach for speeding-up fault injection

campaigns on safety-critical circuits," *J. Electron. Test.*, vol. 18, no. 3, pp. 261–271, 2002.

[92] C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia, and L. Entrena, "Autonomous Fault Emulation: A New FPGA-Based Acceleration System for Hardness Evaluation," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 1, pp. 252–261, Feb. 2007.

[93] P. Kenterlis, N. Kranitis, A. Paschalis, D. Gizopoulos, and M. Psarakis, "A Low-Cost SEU Fault Emulation Platform for SRAM-Based FPGAs," *12th IEEE Int. On-Line Test. Symp.*, pp. 235–241, 2006.

[94] A. Lesea, S. Drimer, J. Fabula, C. Carmichael, and P. Alfke, "The Rosetta Experiment: Atmospheric Soft Error Rate Testing in Fiffering Technology FPGAs," *IEEE Trans. Device Mater. Reliab.*, vol. 5, no. 3, pp. 317–328, Sep. 2005.

[95] J. M. Kuuhn, T. Schweizer, D. Peterson, T. Kuhn, and W. Rosenstiel, "Testing reliability techniques for SoCs with fault tolerant CGRA by using live FPGA fault injection," in *2013 International Conference on Field-Programmable Technology (FPT)*, 2013, pp. 462–465.

[96] M. Aguirre, J. Tombs, F. Muñoz, V. Baena, A. Torralba, A. Fernández-León, F. Tortosa, and D. González-Gutiérrez, "An FPGA based hardware emulator for the insertion and analysis of Single Event Upsets in VLSI Designs," in *Radiation Effects on Components and Systems Workshop (RADECS)*, 2004, pp. 1–5.

[97] M. a. Aguirre, V. Baena, J. Tombs, and M. Violante, "A New Approach to Estimate the Effect of Single Event Transients in Complex Circuits," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 4, pp. 1018–1024, Aug. 2007.

[98] A. Rohani and H. G. Kerkhoff, "A Technique for Accelerating Injection of Transient Faults in Complex SoCs," in *2011 14th Euromicro Conference on Digital System Design*, 2011, pp. 213–220.

[99] M. Ebrahimi, N. Sayed, M. Rashvand, and M. B. Tahoori, "Fault injection acceleration by architectural importance sampling," in *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2015, pp. 212–219.

[100] N. Bombieri, F. Fummi, and V. Guarnieri, "Accelerating RTL Fault Simulation through RTL-to-TLM Abstraction," in *2011 Sixteenth IEEE European Test Symposium*, 2011, pp. 117–122.

[101] Cadence Design Systems, "Encounter RTL Compiler," 2013.

[102] Cadence Design Systems, "Genus Synthesis Solution," 2015.

[103] International Organization for Standardization, "ISO26262 - Road Veichles - Functional Safety - Part 8: Supporting processes," techreport, 2011.

[104] Cadence Design Systems, "Encounter True-Time ATPG," 2011.

[105] IEEE Computer Society, *1364-2005 - IEEE Standard for Verilog Hardware Description Language*. New York, NY: IEEE Computer Society, 2006.

[106] OpenCores.org, “OR1200 OpenRISC Processor,” 2012. [Online]. Available: http://opencores.org/or1k/OR1200_OpenRISC_Processor. [Accessed: 05-Feb-2015].

[107] J. Baxter, “Open Source Hardware Development and the OpenRISC Project,” KTH ROYAL INSTITUTE OF TECHNOLOGY, 2011.

[108] M. de Carvalho, P. Bernardi, E. Sanchez, M. S. Reorda, and O. Ballan, “Increasing the Fault Coverage of Processor Devices during the Operational Phase Functional Test,” *J. Electron. Test.*, vol. 30, no. 3, pp. 317–328, 2014.