

André Vaz da Silva Fidalgo

**Avaliação de Confiabilidade em Sistemas  
Baseados em Microprocessadores  
por Injecção de Falhas  
via Infra-Estruturas de Depuração**

Dissertação submetida para obtenção do grau de Doutor  
em Engenharia Electrotécnica e de Computadores

**Faculdade de Engenharia da Universidade do Porto**

Departamento de Engenharia Electrotécnica e de Computadores

Julho de 2007

Tese realizada sob supervisão de  
Prof. Doutor José Manuel Martins Ferreira  
Professor Associado com Agregação do  
Departamento de Engenharia Electrotécnica e de Computadores da  
Faculdade de Engenharia da Universidade do Porto  
e  
Prof. Doutor Gustavo Ribeiro Costa Alves  
Professor Adjunto do  
Departamento de Engenharia Electrotécnica do  
Instituto Superior de Engenharia do Porto do  
Instituto Politécnico do Porto

À Sofia.



*“Engineering is not merely knowing and being knowledgeable, like a walking encyclopedia; engineering is not merely analysis; engineering is not merely the possession of the capacity to get elegant solutions to non-existent engineering problems; engineering is practicing the art of the organized forcing of technological change... Engineers operate at the interface between science and society...”*

Gordon Brown  
(Reitor do MIT entre 1959 e 1968)



## Resumo

**Palavras-Chave:** Sistemas computacionais, Microprocessadores, Confiabilidade, Injecção de falhas em tempo real, Infra-estruturas de depuração embutidas, Proposta de norma NEXUS 5001

A confiabilidade dos sistemas computacionais torna-se cada vez mais importante, à medida que estes aumentam a sua implantação e se atingem níveis crescentes de miniaturização. Com a diminuição de tamanho dos diversos componentes, cresce a sua sensibilidade ao meio ambiente e torna-se simultaneamente mais difícil o acesso a recurso internos, aumentando a complexidade do teste, da depuração e da análise de avarias e suas origens.

Esta tese começa com uma introdução geral aos sistemas computacionais confiáveis, com particular enfoque na injeção de falhas, como forma de verificar e validar mecanismos de apoio à confiabilidade. São comparadas diferentes técnicas de injeção, sendo analisada a utilização de infra-estruturas de depuração como forma de ultrapassar algumas limitações existentes. É também apresentada a proposta de norma IEEE-ISTO 5001 NEXUS, que visa fornecer uma base para o conjunto de funcionalidades e a interface deste tipo de infra-estruturas. São também estudadas a motivação e as dificuldades inerentes à injeção de falhas em tempo real em microprocessadores, bem como algumas soluções existentes. É proposta uma nova abordagem a este problema, na forma de uma metodologia de injeção de falhas escalável, baseada na utilização de infra-estruturas de depuração, em conformidade com a proposta NEXUS, e de um depurador otimizado. São propostas alterações ao nível da infra-estrutura, como forma de obter maior eficiência, com elevada controlabilidade e observabilidade. Estas incluem modificações da interface com o exterior, a implementação de módulos dedicados para controlo do processo de injeção e modificações da lógica de interligação ao microprocessador alvo. Todas as soluções desenvolvidas foram validadas em simulação, permitindo os resultados obtidos comparar as diferentes variantes, em termos de funcionalidades, desempenho e sobrecarga imposta.

Concluiu-se que a metodologia proposta é uma alternativa eficiente para a injeção de falhas em tempo real em microprocessadores, sendo a variante específica a utilizar dependente dos requisitos de confiabilidade e níveis aceitáveis de intrusividade. Especificamente, a utilização de uma infra-estrutura modificada, designada por OCD-FI (*On Chip Debug and Fault Injection*), permite a injeção de falhas em tempo real de forma eficiente, sem intrusividade, com sobrecarga mínima, e sem perda de conformidade com a norma.



## Abstract

**Keywords:** Computer systems, Microprocessors, Dependability, Real time fault injection, On chip debug infrastructures, NEXUS 5001 standard proposal

Computer systems dependability is becoming increasingly important, as their application range increases and new levels of miniaturization become possible. Accompanying the reduction in component size, their sensibility to environmental factors increases and access to internal resources becomes more complex, making testing, debugging and failure analysis much harder to accomplish.

This thesis starts with a general introduction to dependable computer systems, with emphasis on fault injection as a way to verify and validate dependability support mechanisms. Different injection techniques are compared, and the debug infrastructures are analyzed to look for ways of overcoming some of their limitations. The IEEE-ISTO 5001 NEXUS standard proposal is also presented, as a basis for debug infrastructures interface and functionality set.

The motivation and difficulties inherent to real time fault injection on microprocessors are also studied, as well as some of the existent solutions. A new approach to this problem is proposed, in the form of a scalable methodology, based on the use of NEXUS compliant debug infrastructures, and a compatible optimized debugger. Specific modifications to these infrastructures are also presented, designed to achieve better efficiency, with high observability and controllability. The proposed modifications include the interface port, the control of the injection process through dedicated modules, and the interconnection logic with the target microprocessor. All the proposed solutions were validated in simulation, allowing the comparison of different variants in terms of functionalities, performance and overheads.

The proposed methodology is an efficient alternative for real time fault injection on microprocessors, and the best solution for each case is defined by the dependability requirements and acceptable overhead levels. The proposed OCD-FI (On Chip Debug and Fault Injection) infrastructure remains compatible with the NEXUS proposal and allows efficient non-intrusive fault injection in real time, with minimal overhead.



## Résumé

**Mots-clés:** Systèmes computationnels, Microprocesseurs, Sureté, Injection de fautes en temps réel, Infrastructures de dépuración embarquées, Proposition de Norme NEXUS 5001

La sureté des systèmes computationnels devient de plus en plus importante au fur et à mesure que ceux-ci augmentent son implantation et atteignent des niveaux croissants de miniaturisation. La diminution du volume des divers composants a accru sa sensibilité à l'environnement et a rendu plus difficile l'accès à des ressources internes, augmentant la complexité du test, de la dépuración et de l'analyse des pannes et de son origine.

Cette thèse commence par une introduction générale aux systèmes computationnels surs, en insistant particulièrement sur l'injection de fautes en tant que moyen de vérification et de validation des processus d'appui à la sureté. On compare plusieurs techniques d'injection et on analyse l'emploi d'infrastructures de dépuración envisageant le dépassement de quelques unes des limitations trouvées. On présente aussi une proposition de norme IEEE-ISTO 5001 NEXUS dont le but est de fournir une base pour l'ensemble de fonctionnalités et l'interface de ce genre d'infrastructures.

La motivation et les difficultés inhérentes à l'injection de fautes en temps réel dans des microprocesseurs sont également étudiées, ainsi que quelques unes des solutions déjà envisagées. On propose une solution pour ce problème à travers une méthodologie d'injection de fautes scalaire, basée sur l'utilisation d'infrastructures de dépuración, en conformité avec la proposition NEXUS, et un dépurateur optimisé. On propose encore des altérations au niveau de l'infrastructure pour atteindre une plus grande efficacité, avec une contrôlabilité et observabilité élevées. Celles-ci incluent des modifications de l'interface avec l'extérieur, l'implémentation des modules dédiés pour le contrôle du processus d'injection et des modifications de la logique d'interliaison vis-à-vis le microprocesseur cible. Toutes les solutions proposées ont été validées moyennant le processus de simulation et les résultats obtenus ont permis la comparaison de différentes variantes par rapport à leurs fonctionnalités, à leur accomplissement et à la surcharge imposée.

On a conclu que la méthodologie proposée représente une option efficace pour l'injection de fautes en temps réel dans des microprocesseurs. La variante spécifique qui doit être utilisée dépend des conditions de sureté et des niveaux acceptables d'intrusivité. En particulier, l'emploi d'une infrastructure modifiée, nommée OCD-FI (On Chip Debug and Fault Injection), accepte l'injection de fautes en temps réel et d'une façon efficace, sans intrusivité, avec une surcharge minimale et sans la perte de conformité à la proposition de la norme.



## **Agradecimentos**

A realização deste trabalho não seria possível sem a colaboração de várias pessoas que me incentivaram e apoiaram ao longo da sua execução. Sem querer omitir ou ser injusto para ninguém, gostaria de reconhecer particularmente algumas pessoas e instituições.

Começo por agradecer aos meus orientadores, Prof. Dr. José Martins Ferreira e Prof. Dr. Gustavo Ribeiro Alves pela ajuda na escolha e definição do tema, importantes conselhos relativos ao caminho a seguir e conversas muito úteis para a resolução dos problemas que foram surgindo.

Ao ISEP/IPP pela oportunidade de realizar este trabalho e por me dar as condições necessárias para o levar a bom termo. Aos colegas do LABORIS, pelo excelente ambiente e pelo apoio constante, quer em termos morais como no esclarecimento de dúvidas e na troca de ideias.

Também uma palavra de agradecimento ao Dr. Pedro Yuste, ao Dr. Juan Carlos Ruiz e ao Dr. Raul Velazco pela disponibilidade e interesse demonstrados durante as minhas visitas às suas instituições.

Por último, não poderia deixar de agradecer à minha família e amigos, que estiveram presentes quando era preciso. Em especial aos meus pais, que muito fizeram para que eu aqui chegasse, à Joana, apoio insubstituível e companheira de vida, e à Sofia, a quem nem sempre pude dar as devidas atenções.



## Nota ao Leitor

Um documento desta natureza, que abrange áreas científicas onde predomina o uso de língua inglesa, obriga a um equilíbrio entre a utilização de termos estrangeiros e a clareza de exposição. Certos conceitos, em particular, não apresentam uma designação uniforme em língua portuguesa, por vezes fruto de traduções diversas de um mesmo termo em inglês.

Optamos por uma estratégia mista, tendo por prioridade a clareza de exposição. Procuramos utilizar sempre que possível os termos em português, privilegiando as variantes de aceitação mais generalizada na comunidade científica, nos casos em que existem alternativas. O recurso a termos em inglês foi feito quando necessário para a compreensão do conceito em questão. Termos estrangeiros são apresentados em *itálico*, sendo utilizado o **negrito** para realçar os termos importantes em português, sendo os equivalentes em inglês apresentados em seguida entre parêntesis. Os acrónimos são sempre apresentados em maiúsculas, em estilo normal, sendo incluída uma listagem para facilitar a sua identificação. Certos termos estrangeiros originaram de acrónimos, e.g. o termo bit (*BI*nary *di*giT), que não podem ser traduzidos e que são frequentemente utilizados em língua portuguesa, pelo que se optou por manter o texto em estilo normal.



# Índice

|          |                                                       |           |
|----------|-------------------------------------------------------|-----------|
| <b>1</b> | <b>Introdução .....</b>                               | <b>3</b>  |
| 1.1      | Motivação .....                                       | 3         |
| 1.2      | Enquadramento e Contribuição Original .....           | 4         |
| 1.3      | Objectivos .....                                      | 6         |
| 1.4      | Organização da Tese .....                             | 6         |
| <b>2</b> | <b>Sistemas Computacionais Confiáveis .....</b>       | <b>11</b> |
| 2.1      | Motivação e Historial .....                           | 11        |
| 2.1.1    | Motivação .....                                       | 11        |
| 2.1.2    | Historial Resumido .....                              | 12        |
| 2.2      | Confiabilidade .....                                  | 15        |
| 2.2.1    | Conceitos Fundamentais .....                          | 15        |
| 2.2.1.1  | Sistemas e suas propriedades .....                    | 15        |
| 2.2.1.2  | Erros e Avarias .....                                 | 20        |
| 2.2.1.3  | Falhas .....                                          | 22        |
| 2.2.2    | Técnicas para lidar com falhas .....                  | 26        |
| 2.2.3    | Verificação e Validação .....                         | 29        |
| 2.3      | Injecção de Falhas .....                              | 32        |
| 2.3.1    | Objectivos .....                                      | 32        |
| 2.3.2    | Estrutura Genérica .....                              | 33        |
| 2.3.3    | Modelo de Falhas .....                                | 35        |
| 2.3.4    | Técnicas .....                                        | 36        |
| 2.3.4.1  | Injecção de Falhas por Simulação .....                | 37        |
| 2.3.4.2  | Injecção de Falhas por <i>Software</i> .....          | 40        |
| 2.3.4.3  | Injecção Física de Falhas .....                       | 43        |
| 2.4      | Resumo e Conclusões .....                             | 49        |
| <b>3</b> | <b>Depuração e Confiabilidade .....</b>               | <b>53</b> |
| 3.1      | Depuração Embutida .....                              | 53        |
| 3.1.1    | Evolução .....                                        | 53        |
| 3.1.2    | Implementações e Funcionalidades .....                | 58        |
| 3.1.3    | Esforços de Normalização .....                        | 63        |
| 3.2      | A Proposta de Norma NEXUS 5001 .....                  | 66        |
| 3.2.1    | Historial .....                                       | 67        |
| 3.2.2    | Motivação .....                                       | 68        |
| 3.2.3    | Classificação e Conformidade .....                    | 69        |
| 3.2.4    | Conjunto de Funcionalidades .....                     | 72        |
| 3.2.5    | Interface Física e Lógica .....                       | 74        |
| 3.2.6    | Mensagens .....                                       | 76        |
| 3.2.7    | Sinais e Protocolos .....                             | 78        |
| 3.2.8    | Aceitação .....                                       | 82        |
| 3.3      | Injecção de Falhas via infra-estruturas OCD .....     | 83        |
| 3.4      | Resumo e Conclusões .....                             | 88        |
| <b>4</b> | <b>Injecção de Falhas em Tempo Real via OCD .....</b> | <b>93</b> |
| 4.1      | Campo de Aplicação .....                              | 93        |
| 4.1.1    | Sistemas de Tempo Real .....                          | 93        |
| 4.1.2    | Injecção de Falhas em Tempo Real .....                | 96        |

|          |                                                               |            |
|----------|---------------------------------------------------------------|------------|
| 4.2      | Injecção de Falhas via Infra-estruturas NEXUS.....            | 100        |
| 4.2.1    | Ambiente e Metodologia .....                                  | 100        |
| 4.2.2    | Sistema Alvo .....                                            | 109        |
| 4.2.3    | Infra-estrutura OCD.....                                      | 110        |
| 4.2.4    | Ambientes de Depuração.....                                   | 111        |
| 4.2.5    | Realização das Experiências.....                              | 113        |
| 4.2.6    | Análise de Resultados.....                                    | 117        |
| 4.3      | Propostas de Aperfeiçoamento .....                            | 118        |
| 4.3.1    | Objectivos e Abordagem .....                                  | 118        |
| 4.3.2    | Depuração Aperfeiçoada .....                                  | 120        |
| 4.3.3    | OCD-FI.....                                                   | 124        |
| 4.4      | Resumo e Conclusões.....                                      | 126        |
| <b>5</b> | <b>Solução Proposta .....</b>                                 | <b>131</b> |
| 5.1      | Metodologia.....                                              | 131        |
| 5.2      | Sistema Alvo .....                                            | 135        |
| 5.2.1    | Microprocessador e Periféricos .....                          | 135        |
| 5.2.2    | Aplicações .....                                              | 138        |
| 5.2.3    | Infra-Estrutura OCD .....                                     | 140        |
| 5.3      | Depurador Optimizado .....                                    | 149        |
| 5.4      | Alterações à infra-estrutura OCD .....                        | 153        |
| 5.5      | Injecção de Falhas Embutida (OCD-FI).....                     | 155        |
| 5.6      | Extensões.....                                                | 157        |
| 5.6.1    | Injecção em mecanismos de detecção e correcção de erros ..... | 158        |
| 5.6.2    | Injecção em registos em tempo real .....                      | 161        |
| 5.7      | Resumo e Conclusões.....                                      | 164        |
| <b>6</b> | <b>Trabalho Experimental.....</b>                             | <b>167</b> |
| 6.1      | Condicionantes do Trabalho Experimental .....                 | 167        |
| 6.2      | Análise de Exequibilidade .....                               | 170        |
| 6.2.1    | Execução das Experiências.....                                | 170        |
| 6.2.2    | Implementações Normalizadas.....                              | 172        |
| 6.2.3    | Infra-estruturas modificadas .....                            | 177        |
| 6.2.4    | Sobrecarga Lógica .....                                       | 179        |
| 6.3      | Utilização Operacional .....                                  | 181        |
| 6.3.1    | Execução das Experiências.....                                | 182        |
| 6.3.2    | Resultados Experimentais .....                                | 185        |
| 6.3.3    | Análise de Resultados Inconclusivos .....                     | 188        |
| 6.3.4    | Falhas Múltiplas .....                                        | 190        |
| 6.4      | OCD-FI com Extensões.....                                     | 191        |
| 6.4.1    | Injecção em mecanismos de detecção e correcção de erros ..... | 191        |
| 6.4.2    | Injecção em registos em tempo real .....                      | 193        |
| 6.5      | Resumo e Conclusões.....                                      | 196        |
| <b>7</b> | <b>Conclusão .....</b>                                        | <b>201</b> |
| 7.1      | Apreciação Crítica .....                                      | 201        |
| 7.2      | Trabalho Futuro .....                                         | 203        |
| <b>8</b> | <b>Referências .....</b>                                      | <b>207</b> |
| <b>9</b> | <b>Anexo .....</b>                                            | <b>221</b> |

# Lista de Figuras

|                                                                                             |     |
|---------------------------------------------------------------------------------------------|-----|
| FIGURA 2.1 – MANUTENÇÃO DO ENIAC .....                                                      | 13  |
| FIGURA 2.2 – DA FALHA À AVARIA.....                                                         | 19  |
| FIGURA 2.3 – MODOS DE AVARIA .....                                                          | 20  |
| FIGURA 2.4 – CLASSES DE FALHA .....                                                         | 23  |
| FIGURA 2.5 – AMBIENTE DE INJEÇÃO DE FALHAS.....                                             | 33  |
| FIGURA 2.6 – TÉCNICAS DE INJEÇÃO DE FALHAS .....                                            | 37  |
| FIGURA 3.1 – INFRA-ESTRUTURA OCD TÍPICA .....                                               | 57  |
| FIGURA 3.2 – PRINCIPAIS FUNCIONALIDADES NEXUS POR CLASSE .....                              | 70  |
| FIGURA 3.3 – DIAGRAMAS EXPLICATIVOS DO PROTOCOLO DE COMUNICAÇÃO .....                       | 81  |
| FIGURA 4.1 – SISTEMA DE TEMPO REAL E SEU MEIO AMBIENTE .....                                | 94  |
| FIGURA 4.2 – RESTRIÇÕES DE TEMPO REAL .....                                                 | 95  |
| FIGURA 4.3 – AMBIENTE DE INJEÇÃO DE FALHAS (MPC565).....                                    | 100 |
| FIGURA 4.4 – INJEÇÃO DE FALHAS NUM ÚNICO BIT (EXEMPLO) .....                                | 101 |
| FIGURA 4.5 – CAMPANHA E EXPERIÊNCIA DE INJEÇÃO DE FALHAS .....                              | 102 |
| FIGURA 4.6 – FLUXO DE UMA EXPERIÊNCIA DE INJEÇÃO DE FALHAS .....                            | 104 |
| FIGURA 4.7 – CLASSIFICAÇÃO DOS RESULTADOS DAS EXPERIÊNCIAS DE INJEÇÃO DE FALHAS .....       | 106 |
| FIGURA 4.8 – AMBIENTE DE INJEÇÃO DE FALHAS UTILIZANDO A COMBINAÇÃO AXIOM E LAUTERBACH. .... | 112 |
| FIGURA 4.9 – AMBIENTE DE INJEÇÃO DE FALHAS UTILIZANDO A COMBINAÇÃO PHYTEC E ISYSTEM.....    | 113 |
| FIGURA 4.10 – DEPURADOR AUTÓNOMO OPTIMIZADO PARA INJEÇÃO DE FALHAS.....                     | 121 |
| FIGURA 4.11 – INFRA-ESTRUTURA OCD E MÓDULO FI .....                                         | 124 |
| FIGURA 5.1 – ARQUITECTURA DO SISTEMA ALVO .....                                             | 135 |
| FIGURA 5.2 – INTERFACE DO <i>CPUGENERATOR 2.0</i> .....                                     | 136 |
| FIGURA 5.3 – ARQUITECTURA INTERNA DO MICROPROCESSADOR .....                                 | 136 |
| FIGURA 5.4 – DESIGNAÇÃO DO MODELO DO MICROPROCESSADOR E INFRA-ESTRUTURA OCD .....           | 137 |
| FIGURA 5.5 – DESIGNAÇÃO DAS APLICAÇÕES.....                                                 | 138 |
| FIGURA 5.6 – TOLERÂNCIA A FALHAS POR <i>SOFTWARE</i> .....                                  | 139 |
| FIGURA 5.7 – INFRA-ESTRUTURA DE DEPURAÇÃO EMBUTIDA (OCD).....                               | 141 |
| FIGURA 5.8 – ARQUITECTURA DO MÓDULO MQM (SIMPLIFICADA) .....                                | 143 |
| FIGURA 5.9 – ARQUITECTURA DO MÓDULO RCT (SIMPLIFICADA) .....                                | 144 |
| FIGURA 5.10 – ARQUITECTURA DO MÓDULO RWA (SIMPLIFICADA).....                                | 147 |
| FIGURA 5.11 – ARQUITECTURA DO DEPURADOR (SIMPLIFICADA) .....                                | 149 |

|                                                                                       |     |
|---------------------------------------------------------------------------------------|-----|
| FIGURA 5.12 – LIGAÇÃO DIRECTA ENTRE DEPURADOR E INFRA-ESTRUTURA OCD.....              | 154 |
| FIGURA 5.13 – VERSÃO BASE DO MÓDULO FI .....                                          | 155 |
| FIGURA 5.14 – INFRA-ESTRUTURA OCD-FI (SIMPLIFICADA) .....                             | 156 |
| FIGURA 5.15 – EXEMPLO DE CODIFICAÇÃO COM EDAC.....                                    | 159 |
| FIGURA 5.16 – ARQUITECTURA EDAC (EXEMPLO PARA 32 BITS).....                           | 159 |
| FIGURA 5.17 – ARQUITECTURA EDAC MODIFICADA COM OCD-FI .....                           | 161 |
| FIGURA 5.18 – INFRA-ESTRUTURA OCD-FI (RTREG) .....                                    | 162 |
| FIGURA 6.1 – AMBIENTE DE INJEÇÃO DE FALHAS.....                                       | 168 |
| FIGURA 6.2 – OCORRÊNCIA DE RESULTADOS DO TIPO INC POR METODOLOGIA (COM PREDET.).....  | 189 |
| FIGURA 6.3 – OCORRÊNCIA DE RESULTADOS DO TIPO INC POR METODOLOGIA (SEM PREDET.) ..... | 189 |

# Lista de Tabelas

|                                                                                                       |     |
|-------------------------------------------------------------------------------------------------------|-----|
| TABELA 2.1 – VERIFICAÇÃO E VALIDAÇÃO .....                                                            | 29  |
| TABELA 2.2 – PARÂMETROS DE FALHA .....                                                                | 35  |
| TABELA 2.3 – VANTAGENS / DESVANTAGENS DA INJEÇÃO DE FALHAS POR SIMULAÇÃO .....                        | 39  |
| TABELA 2.4 – VANTAGENS / DESVANTAGENS DA INJEÇÃO DO SWIFI .....                                       | 42  |
| TABELA 2.5 – VANTAGENS / DESVANTAGENS DA INJEÇÃO DE FALHAS AO NÍVEL DO PINO .....                     | 44  |
| TABELA 2.6 – VANTAGENS / DESVANTAGENS DO SCIFI .....                                                  | 45  |
| TABELA 2.7 – VANTAGENS / DESVANTAGENS DA INJEÇÃO DE FALHAS POR INTERFERÊNCIA MAGNÉTICA .....          | 46  |
| TABELA 2.8 – VANTAGENS / DESVANTAGENS DA INJEÇÃO DE FALHAS AO NÍVEL POR IÕES PESADOS.....             | 46  |
| TABELA 2.9 – VANTAGENS / DESVANTAGENS DA INJEÇÃO DE FALHAS POR LASER .....                            | 47  |
| TABELA 3.1 – FUNCIONALIDADES ESTÁTICAS POR CLASSE DE CONFORMIDADE .....                               | 71  |
| TABELA 3.2 – FUNCIONALIDADES DINÂMICAS POR CLASSE DE CONFORMIDADE .....                               | 71  |
| TABELA 3.3 – NÚMERO DE PINOS DE DADOS PROPOSTO NA NORMA NEXUS (32 BITS).....                          | 75  |
| TABELA 3.4 – MENSAGENS PÚBLICAS DEFINIDAS NA PROPOSTA NEXUS .....                                     | 77  |
| TABELA 3.5 – PINOS OBRIGATÓRIOS E OPCIONAIS DO PORTO AUX .....                                        | 79  |
| TABELA 3.6 – EXEMPLOS DE PORTOS AUX .....                                                             | 80  |
| TABELA 3.7 – CENÁRIOS DE INJEÇÃO DE FALHAS VIA INFRA-ESTRUTURAS OCD, TESTADOS NESTE<br>TRABALHO ..... | 86  |
| TABELA 4.1 – ANÁLISE COMPARATIVA DE DIFERENTES TÉCNICAS DE INJEÇÃO DE FALHAS.....                     | 98  |
| TABELA 4.2 – FUNCIONALIDADES NEXUS REQUERIDAS PARA INJEÇÃO DE FALHAS (POR CLASSE) .....               | 99  |
| TABELA 4.3 – PASSOS PARA A INJEÇÃO DE FALHAS NO MPC565 .....                                          | 105 |
| TABELA 4.4 – CARACTERÍSTICAS PRINCIPAIS DO MPC565.....                                                | 109 |
| TABELA 4.5 – CARACTERÍSTICAS DAS PLACAS DE DESENVOLVIMENTO UTILIZADAS.....                            | 110 |
| TABELA 5.1 – VARIANTES DA METODOLOGIA PROPOSTA E SEUS EFEITOS .....                                   | 133 |
| TABELA 5.2 – PROCEDIMENTO PARA INJEÇÃO DE FALHAS COM SUSPENSÃO DO SISTEMA ALVO.....                   | 134 |
| TABELA 5.3 – PROCEDIMENTO PARA INJEÇÃO DE FALHAS EM TEMPO REAL VIA OCD.....                           | 134 |
| TABELA 5.4 – PROCEDIMENTO PARA INJEÇÃO DE FALHAS EM TEMPO REAL VIA OCD-FI.....                        | 134 |
| TABELA 5.5 – APLICAÇÕES DE BASE UTILIZADAS.....                                                       | 138 |
| TABELA 5.6 – SOBRECARGA IMPOSTA PELA TOLERÂNCIA A FALHAS (EXEMPLOS) .....                             | 139 |
| TABELA 5.7 – COMANDOS DO DEPURADOR .....                                                              | 151 |
| TABELA 5.8 – CARACTERÍSTICAS DAS DUAS VERSÕES PROPOSTAS PARA A INFRA-ESTRUTURA OCD-FI<br>.....        | 155 |

|                                                                                                         |     |
|---------------------------------------------------------------------------------------------------------|-----|
| TABELA 6.1 – CENÁRIOS OPERACIONAIS E SUAS CARACTERÍSTICAS .....                                         | 169 |
| TABELA 6.2 – CENÁRIOS EXPERIMENTAIS COM INFRA-ESTRUTURA OCD SIMPLES .....                               | 172 |
| TABELA 6.3 – CAMPANHAS EXECUTADAS POR APLICAÇÃO (I) .....                                               | 173 |
| TABELA 6.4 – INTERVALOS DE TEMPO PARA AS OPERAÇÕES DE INICIALIZAÇÃO E INJEÇÃO DE FALHAS (I)<br>.....    | 174 |
| TABELA 6.5 – RESULTADOS EXPERIMENTAIS COM INFRA-ESTRUTURA OCD SIMPLES .....                             | 175 |
| TABELA 6.6 – CENÁRIOS EXPERIMENTAIS COM INFRA-ESTRUTURAS OCD E OCD-FI .....                             | 177 |
| TABELA 6.7 – INTERVALOS DE TEMPO PARA AS OPERAÇÕES DE INICIALIZAÇÃO E INJEÇÃO DE FALHAS (II)<br>.....   | 178 |
| TABELA 6.8 – RESULTADOS EXPERIMENTAIS COM INFRA-ESTRUTURA OCD E OCD-FI .....                            | 178 |
| TABELA 6.9 – RESULTADOS EXPERIMENTAIS COM LIGAÇÃO PARALELA DEPURADOR-OCD .....                          | 179 |
| TABELA 6.10 – SOBRECARGA LÓGICA (MICROPROCESSADORES 8 BITS) .....                                       | 180 |
| TABELA 6.11 – SOBRECARGA LÓGICA (MICROPROCESSADORES 16 E 32 BITS) .....                                 | 180 |
| TABELA 6.12 – SOBRECARGA LÓGICA (INFRA-ESTRUTURA OCD-FI) .....                                          | 180 |
| TABELA 6.13 – ÁREA LÓGICA DOS DEPURADORES UTILIZADOS .....                                              | 181 |
| TABELA 6.14 – CENÁRIOS EXPERIMENTAIS PARA MICROPROCESSADORES DE 32 BITS .....                           | 183 |
| TABELA 6.15 – INTERVALOS DE TEMPO PARA AS OPERAÇÕES DE INICIALIZAÇÃO E INJEÇÃO DE FALHAS (III)<br>..... | 184 |
| TABELA 6.16 – CAMPANHAS EXECUTADAS POR APLICAÇÃO (II) .....                                             | 185 |
| TABELA 6.17 – RESULTADOS DA INJEÇÃO DE FALHAS NO MICROPROCESSADOR DE 32 BITS (EM %) .                   | 186 |
| TABELA 6.18 – OCORRÊNCIA DE RESULTADOS INCONCLUSIVOS (EM %) .....                                       | 188 |
| TABELA 6.19 – TAXAS MÁXIMAS DE FALHAS POR SEGUNDO .....                                                 | 190 |
| TABELA 6.20 – RESULTADOS DA INJEÇÃO DE FALHAS NUM SISTEMA COM EDAC (EM %) .....                         | 192 |
| TABELA 6.21 – TEMPO DE SUSPENSÃO PARA A INJEÇÃO DE FALHAS EM REGISTOS .....                             | 193 |
| TABELA 6.22 – RESULTADOS DA INJEÇÃO DE FALHAS COM EXTENSÃO OCD-FI (RTREG) (EM %) ...                    | 194 |
| TABELA 6.23 – SOBRECARGA LÓGICA E EFEITOS NO DESEMPENHO DINÂMICO .....                                  | 195 |

# Lista de Acrónimos

|               |                                                     |
|---------------|-----------------------------------------------------|
| <b>API</b>    | Application Protocol Interface                      |
| <b>ASIC</b>   | Application-Specific Integrated Circuit             |
| <b>BDM</b>    | Background Debug Module                             |
| <b>BPU</b>    | Branch Processing Unit                              |
| <b>BRT</b>    | Basic Real Time                                     |
| <b>BTM</b>    | Branch Trace Message                                |
| <b>CAN</b>    | Controller Area Network                             |
| <b>CI</b>     | Circuito Integrado                                  |
| <b>COTS</b>   | Commercial-Off-The-Shelf                            |
| <b>CPU</b>    | Central Processing Unit                             |
| <b>DSP</b>    | Digital Signal Processor                            |
| <b>DTM</b>    | Data Trace Message                                  |
| <b>ECU</b>    | Engine Control Unit                                 |
| <b>EDAC</b>   | Error Detection And Correction                      |
| <b>EDC</b>    | Error Detection and Correction (alternativa a EDAC) |
| <b>ENIAC</b>  | Electronic Numerical Integrator And Computer        |
| <b>ERT</b>    | Enhanced Real Time                                  |
| <b>ESA</b>    | European Space Agency                               |
| <b>ETM</b>    | Embedded Trace Macrocell                            |
| <b>EVTI</b>   | Event In                                            |
| <b>EVTO</b>   | Event Out                                           |
| <b>FIFO</b>   | First-In First-Out                                  |
| <b>FPGA</b>   | Field Programmable Gate-Array                       |
| <b>FPU</b>    | Floating Point Unit                                 |
| <b>GEPDIS</b> | Global Embedded Processor Debug Interface Standard  |
| <b>HDL</b>    | Hardware Description Language                       |
| <b>I/O</b>    | Input / Output                                      |
| <b>ICE</b>    | In-Circuit Emulator                                 |
| <b>IEEE</b>   | Institute of Electrical and Electronics Engineers   |
| <b>IFIP</b>   | International Federation for Information Processing |
| <b>IMD</b>    | Integer Multiplier/Divider                          |
| <b>ISA</b>    | Industry Standard Architecture                      |
| <b>ISTO</b>   | Industry Standards and Technology Organization      |

|               |                                                      |
|---------------|------------------------------------------------------|
| <b>IU</b>     | Integer Unit                                         |
| <b>JTAG</b>   | Joint Test Action Group                              |
| <b>MCKI</b>   | Message Clock In                                     |
| <b>MCKO</b>   | Message Clock Out                                    |
| <b>MDI</b>    | Message Data In                                      |
| <b>MDO</b>    | Message Data Out                                     |
| <b>MQM</b>    | Message Queuing and Management                       |
| <b>MSEI</b>   | Message Start/End In                                 |
| <b>MSEO</b>   | Message Start/End Out                                |
| <b>OCD</b>    | On Chip Debug                                        |
| <b>OCD-FI</b> | On Chip Debug and Fault Injection                    |
| <b>OTM</b>    | Ownership Trace Message                              |
| <b>PBGA</b>   | Plastic Ball Grid Array                              |
| <b>PC</b>     | Personal Computer OU Program Counter                 |
| <b>QSPI</b>   | Queued Serial Peripheral Interface                   |
| <b>RAM</b>    | Random Access Memory                                 |
| <b>RCT</b>    | Run Control and Trace                                |
| <b>READI</b>  | Real-time Embedded Application Development Interface |
| <b>RISC</b>   | Reduced Instruction Set Computer                     |
| <b>ROM</b>    | Read Only Memory                                     |
| <b>RSTI</b>   | Reset In                                             |
| <b>RTREG</b>  | Real-Time Register access                            |
| <b>RWA</b>    | Read and Write Access                                |
| <b>SCIFI</b>  | Scan-ChaIn based Fault Injection                     |
| <b>SEU</b>    | Single Event Upset                                   |
| <b>SOC</b>    | System-On-a-Chip                                     |
| <b>SUT</b>    | System Under Test                                    |
| <b>SWIFI</b>  | Software Induced Fault Injection                     |
| <b>TAP</b>    | Test Access Port                                     |
| <b>TCP/IP</b> | Transmission Control Protocol / Internet Protocol    |
| <b>UART</b>   | Universal Asynchronous Receiver / Transmitter        |
| <b>UML</b>    | Unified Modeling Language                            |
| <b>VHDL</b>   | VHSIC Hardware Description Language                  |
| <b>VHSIC</b>  | Very High Speed Integrated Circuits                  |

*“A journey of a thousand  
miles must begin with a  
single step.”*

Provérbio Chinês

**Capítulo 1**

**Capítulo 2**

**Capítulo 3**

**Capítulo 4**

**Capítulo 5**

**Capítulo 6**

**Capítulo 7**

**Referências**

**Anexo**

# Introdução



# 1 Introdução

Este capítulo introduz o tema da tese, a motivação que lhe deu origem e os objectivos que se pretendem atingir. Inclui também o enquadramento do trabalho desenvolvido na área científica em que se insere, com referência à sua contribuição original. Termina com uma apresentação da estrutura da tese e dos seus elementos constituintes.

## 1.1 Motivação

Na sociedade actual, os sistemas computacionais são cada vez mais importantes, estando presentes na maioria das acções quotidianas. Os microprocessadores fazem parte de grande parte do equipamento indispensável à nossa vida, particularmente nas telecomunicações, transportes e electrónica de consumo. Mesmo os artigos que não incluem qualquer componente electrónico são geralmente fabricados e/ou distribuídos com recurso a equipamento ou aplicações computacionais.

É usual designar como sistemas críticos aqueles em que do seu correcto funcionamento dependem bens de elevado valor, o ambiente ou mesmo vidas humanas, como a electrónica automóvel, o controlo de tráfego aéreo ou o equipamento médico. Mesmo em sistemas aparentemente mais simples, como por exemplo no controlo de redes informáticas, a interrupção ou degradação do funcionamento pode ter consequências potencialmente catastróficas, tornando a confiabilidade de sistemas computacionais uma característica de crescente importância.

Esta evolução constitui também uma razão para o aumento das ameaças à confiabilidade, como é exemplo o crescente risco de corrupção de conteúdos de memória por acção de radiação, motivado pela progressiva miniaturização dos componentes electrónicos. Este tipo de problemas, usualmente considerados apenas para equipamento utilizado em missões espaciais ou em grandes altitudes, é hoje uma possibilidade a considerar em aplicações ditas convencionais.

Estes factos levam a uma crescente preocupação com a confiabilidade dos equipamentos electrónicos em geral e dos sistemas computacionais em particular. Questões como a probabilidade de avarias, as razões da sua ocorrência e os efeitos no serviço prestado têm ganho relevância e, como tal, as técnicas empregues para lidar com elas adquirem importância acrescida. Os sistemas críticos incluem frequentemente mecanismos para lidar com falhas ou minimizar os seus efeitos. Torna-se assim importante que existam

metodologias que permitam testar a ocorrência das referidas falhas e dos seus efeitos. De facto, este tipo de questão pode ser, e muitas vezes é, importante para a generalidade dos sistemas computacionais, mesmo que estes não incluam características específicas de confiabilidade. Mesmo neste caso, pode ser importante analisar a robustez intrínseca do sistema ou simplesmente quantificar as consequências das referidas falhas. A confiabilidade apresenta assim diversos atributos e ameaças, requerendo estas mecanismos apropriados de defesa, sendo o seu teste e validação extremamente importantes.

A injeção de falhas é uma técnica frequentemente empregue para verificar e validar a tolerância a falhas de sistemas confiáveis. Sendo realizada de forma controlada e num ambiente em que é possível monitorizar a resposta do sistema alvo, permite avaliar as suas consequências e analisar o funcionamento dos mecanismos de protecção.

Existem presentemente muitas técnicas para a injeção de falhas, cuja utilização depende de variados factores, como o sistema alvo, o tipo de falha que se pretende inserir, restrições tecnológicas e muito frequentemente também limitações orçamentais ou temporais. Em geral, o objectivo é a análise da confiabilidade de um determinado sistema alvo, face a um conjunto de ameaças específico e em determinadas condições operacionais.

## **1.2 Enquadramento e Contribuição Original**

O trabalho apresentado enquadra-se na área dos sistemas confiáveis, mais especificamente na injeção de falhas em sistemas computacionais. Segue naturalmente trabalho anterior, em que se identificaram algumas limitações nas técnicas convencionais, geralmente utilizadas nesta área, nomeadamente ao nível do desempenho e controlabilidade. Entre os antecedentes destacam-se o projecto BSCAN4FI (*Boundary Scan For Fault Injection*), no qual se utilizou uma infra-estrutura de depuração, acessível por varrimento periférico, para injectar falhas num microprocessador, destinado a utilização em missões espaciais [Rela06]. Este projecto forneceu a motivação inicial, bem como um enquadramento científico e tecnológico do problema.

O estudo de tecnologias mais recentes na área da depuração, permitiu identificar situações operacionais onde seria possível reutilizar as funcionalidades disponíveis para a avaliação de confiabilidade, especificamente através da injeção de falhas. Neste âmbito foram efectuadas visitas de estudo e intercâmbio a duas instituições com trabalho importante e reconhecido nesta área. Especificamente fez-se uma visita ao *Grupo de Sistemas Tolerantes a Fallos*

[GSTF07] da *Universidad Politecnica de Valencia* onde foi possível utilizar o ambiente de injeção de falhas em tempo real aí desenvolvido, o qual se baseia em equipamento (i.e. depurador e sistema alvo) em conformidade com a proposta de norma NEXUS 5001 [IEEE03]. Adicionalmente foi feita uma visita ao grupo QLF (*QuaLification of Circuits*) [QLF07], parte do laboratório TIMA (*Techniques of Informatics and Microelectronics for computer Architecture*), onde foi possível trocar experiências na área de injeção de falhas em microprocessadores, tendo sido particularmente interessante tomar conhecimento em primeira mão sobre as experiências realizadas através de radiação [Velazco05].

O estudo dos requisitos actuais de avaliação de confiabilidade dos sistemas computacionais, permitiu concluir que a injeção de falhas em tempo real, em elementos de memória de microprocessadores, é potencialmente muito útil e pode ser realizada através das infra-estruturas de depuração disponíveis. Para maximizar a universalidade da solução a desenvolver, adoptou-se o requisito de que ela deveria ser aplicável a infra-estruturas conformes com uma proposta de norma em fase avançada de desenvolvimento, com opção NEXUS 5001 a constituir a escolha natural.

É proposto um ambiente e uma metodologia de injeção de falhas em tempo real, para sistemas baseados em microprocessadores equipados com infra-estruturas de depuração embutidas. A metodologia proposta é escalável, configurável e visa tirar partido das funcionalidades presentes nessas infra-estruturas, de modo a fornecer controlo acrescido e melhor desempenho, mantendo um nível baixo de intrusividade. São propostas variantes, dependendo da sobrecarga e intrusividade consideradas aceitáveis e requisitos de confiabilidade do sistema alvo, incluindo a utilização de um depurador optimizado para a injeção de falhas [Fidalgo06d], alterações às infra-estruturas visando melhorar o seu desempenho de comunicações [Fidalgo06a] e modificações mais abrangentes para disponibilizar funcionalidades adicionais [Fidalgo06b][Fidalgo06f]. A principal contribuição original centra-se na inclusão na infra-estrutura de um módulo dedicado a injeção de falhas que reutiliza funcionalidades já existentes para depuração, para fornecer capacidades e desempenho acrescido para o processo de injeção de falhas. A utilização operacional das várias variantes permitiu validar as propostas, identificar lacunas e avaliar a sua eficiência e ganhos de desempenho.

### **1.3 Objectivos**

Os objectivos específicos que se pretendem atingir com o trabalho descrito neste documento são os seguintes:

- Fazer um estudo das diferentes técnicas de injeção de falhas em sistemas computacionais, incluindo o seu enquadramento no campo da confiabilidade em geral, identificando os campos de aplicação de cada técnica e avaliando as respectivas vantagens e desvantagens.
- Identificar as limitações das metodologias actuais de injeção de falhas, via infra-estruturas de depuração, e propor soluções para as minimizar ou eliminar.
- Verificar a exequibilidade de executar injeção de falhas em tempo real, em sistemas computacionais, via infra-estruturas de depuração.
- Analisar de que forma as crescentes tendências de normalização na área da depuração poderão apoiar a injeção de falhas e discutir a viabilidade de alterar a norma, de modo a melhor suportar a injeção de falhas.
- Desenvolver um caso de estudo para validar, por simulação, as metodologias e soluções propostas.

### **1.4 Organização da Tese**

Esta tese encontra-se dividida em sete capítulos, de forma a estruturar adequadamente os temas abordados. Este primeiro capítulo faz a introdução geral do documento, apresenta a motivação para o trabalho descrito, bem como os principais objectivos que nos propomos atingir. O segundo capítulo apresenta as motivações para o desenvolvimento de sistemas confiáveis, um breve historial da confiabilidade em microprocessadores e os conceitos fundamentais envolvidos. São abordados com particular detalhe, os conceitos de falha, erro e avaria, sendo também incluída uma explicação pormenorizada das técnicas para lidar com falhas em sistemas computacionais. São clarificados os conceitos de verificação e validação, para sistemas confiáveis em geral e tolerância a falhas em particular. Neste âmbito, apresentam-se as razões para injectar falhas em microprocessadores e é descrito um sistema genérico de injeção de falhas e os seus elementos constituintes, bem como as técnicas clássicas utilizadas, com exemplos práticos

O terceiro capítulo aborda a relação da depuração com a confiabilidade, sendo introduzida a depuração embutida (OCD – *On Chip Debug*) e apresentadas as razões para a sua utilização

em microprocessadores. São também descritas as metodologias clássicas para injeção de falhas utilizando infra-estruturas OCD e os esforços em curso para a sua normalização. É apresentada em algum detalhe a proposta de norma IEEE-ISTO NEXUS 5001, por servir de base para as metodologias e resultados apresentados nos capítulos subsequentes.

O capítulo quatro aborda em maior detalhe os problemas da injeção de falhas em tempo real e as soluções baseadas em infra-estruturas OCD, incluindo os problemas específicos derivados dos requisitos temporais mais restritos. São descritos os resultados de investigação anteriores, utilizando depuradores comerciais conformes com a proposta NEXUS, para a injeção de falhas em tempo real. São discutidas as limitações identificadas nesse trabalho, apresentado um conjunto de possíveis soluções e descrito um ambiente de injeção de falhas desenvolvido para a sua utilização.

O capítulo cinco faz a apresentação detalhada da metodologia de injeção de falhas, incluindo o depurador desenvolvido e os seus elementos constituintes, variantes e linguagem de programação. É explicada a infra-estrutura OCD proposta, projectada para estar em conformidade com as recomendações NEXUS, e as alterações que visam aumentar o seu desempenho e funcionalidade para injeção de falhas em tempo real. É dado particular destaque à proposta de uma infra-estrutura modificada, designada OCD-FI (*On Chip Debug and Fault Injection*), para a qual são também apresentadas duas extensões, desenvolvidas para a verificar a sua expansibilidade e adequabilidade a cenários mais complexos.

O capítulo seis faz a apresentação do sistema utilizado como alvo, incluindo o microprocessador e as aplicações. São descritas as experiências realizadas e os resultados obtidos utilizando diferentes variantes da metodologia proposta e é feita uma análise detalhada dos resultados experimentais, comparando-os com outras técnicas de injeção de falhas, sendo resumidas as vantagens e limitações das propostas feitas ao longo deste documento.



*“Basic research is what I am  
doing when I don't know  
what I am doing”*

Werner Von Braun

**Capítulo 1**

**Capítulo 2**

**Capítulo 3**

**Capítulo 4**

**Capítulo 5**

**Capítulo 6**

**Capítulo 7**

**Referências**

**Anexo**

# **Sistemas Computacionais Confiáveis**



## 2 Sistemas Computacionais Confiáveis

Este capítulo faz uma apresentação dos sistemas computacionais confiáveis, das ameaças à sua confiabilidade e das formas de lidar com essas ameaças. As falhas são analisadas com particular detalhe, incluindo as suas causas, efeitos e as técnicas para minimizar o problema. É discutida a sua injeção, de forma deliberada e controlada, bem como as dificuldades envolvidas neste processo. Numa primeira secção é apresentada a motivação para a utilização de sistemas computacionais confiáveis e um breve historial da sua evolução ao longo dos tempos. Em seguida são apresentados e classificados os conceitos de base, utilizados em confiabilidade, com particular atenção para os conceitos de falha, erro e avaria. São também apresentadas as formas de lidar com falhas neste tipo de sistema e explicadas as razões para a verificação e validação da confiabilidade, bem como as diferenças entre estes conceitos e as formas de os concretizar. A motivação e as diferentes metodologias de injeção de falhas são apresentadas em detalhe, sendo também feito um resumo do estado da arte relativo à sua utilização em sistemas computacionais confiáveis.

### 2.1 *Motivação e Historial*

#### 2.1.1 *Motivação*

Os sistemas computacionais cumprem actualmente um papel central em diversas funções críticas, como por exemplo equipamento de controlo de veículos, gestão de transacções financeiras e equipamento médico. A dependência crescente deste tipo de sistemas aumenta os seus requisitos de confiabilidade e segurança, para que possam suportar os efeitos de falhas, avarias acidentais e acessos não autorizados. Em geral, não é possível garantir que não ocorrem situações indesejáveis, sendo no entanto fundamental assegurar que as suas consequências se mantêm dentro de limites aceitáveis .

Adicionalmente, a complexidade dos sistemas computacionais tem vindo a aumentar. Os blocos de *software* utilizados duplicam em extensão a cada 18 meses e os microprocessadores actuais, com dezenas ou mesmo centenas de milhões de transístores, são projectados por equipas com milhares de engenheiros [Lehman03]. As metodologias de desenvolvimento clássicas não conseguem acompanhar este padrão de crescimento, originando avarias potencialmente graves, algumas das quais com ampla expressão mediática, como o sejam o acidente do Ariane 5 no seu voo inicial (e último), ou o erro

presente na unidade aritmética de vírgula flutuante da versão original do microprocessador Intel Pentium. Este tipo de problema origina prejuízos por vezes avultados, em danos directos ou em custos de substituição e/ou reparação.

Quanto mais crítica for a função de um sistema, mais importante é a avaliação da sua confiabilidade e, conseqüentemente, maior o esforço necessário para tal. O investimento em confiabilidade é mais facilmente justificável em sistemas onde estão em jogo vidas humanas, equipamento de grande valor ou a conservação do meio ambiente. A título de exemplo, um equipamento para utilização espacial pode apresentar múltiplos processadores em paralelo, utilizando componentes resistentes a radiação e mecanismos integrados de tolerância a falhas. Por outro lado, um PC de secretária inclui tolerância a falhas mínima, normalmente integrada nas aplicações ou no sistema operativo e utilizando uma estratégia simplista de reinicialização em caso de avaria. Simultaneamente, o esforço necessário para verificar e validar a tolerância a falhas é muito maior em sistemas críticos, sendo razoavelmente frequente o desenvolvimento de protótipos e ferramentas dedicadas exclusivamente a essa actividade, utilizando injeção de falhas ou outros meios. O sistema alvo, para o qual se pretende garantir confiabilidade, é também um factor importante. Arquitecturas e tecnologias diferentes apresentam normalmente problemas diversos para a avaliação da sua confiabilidade, pelo que as soluções a utilizar serão necessariamente diferentes.

### 2.1.2 Historial Resumido

O estudo do processamento aritmético e computacional confiável e dos seus usos para comunicações e criptografia é um desafio antigo. Há quase dois séculos, Lardner publicou o artigo *“Babbages’s calculating engine”*, no qual escreve:

*“The most certain and effectual check upon errors which arise in the process of computation, is to cause the same computations to be made by separate and independent computers; and this check is rendered still more decisive if they make their computations by different methods”* [Lardner34].

Este tipo de problema apresenta portanto um longo historial [Iyer95]. No século XX, os anos 40 viram surgir a tolerância a falhas em sistemas digitais, com o aparecimento dos retransmissores de comunicações e o uso generalizado de sistemas baseados em válvulas para análise criptográfica e aplicações balísticas, encorajados pela segunda guerra mundial. Os sistemas electrónicos desta geração (de meados dos anos 40 até meados dos anos 50) utilizavam componentes pouco fiáveis, impondo a necessidade de técnicas operacionais para

melhorar a sua confiabilidade, como por exemplo códigos de controlo, duplicação com comparação, redundância tripla com votadores, diagnóstico de componentes avariados, etc. Como exemplo prático, o computador ENIAC (*Electronic Numerical Integrator And Computer*) [Burks97], desenvolvido durante a segunda guerra mundial, possuía quase 18.000 válvulas, as quais eram consideravelmente vulneráveis a falhas (explosão). A técnica utilizada para resolver o problema consistia simplesmente na substituição dos tubos o mais rapidamente possível.

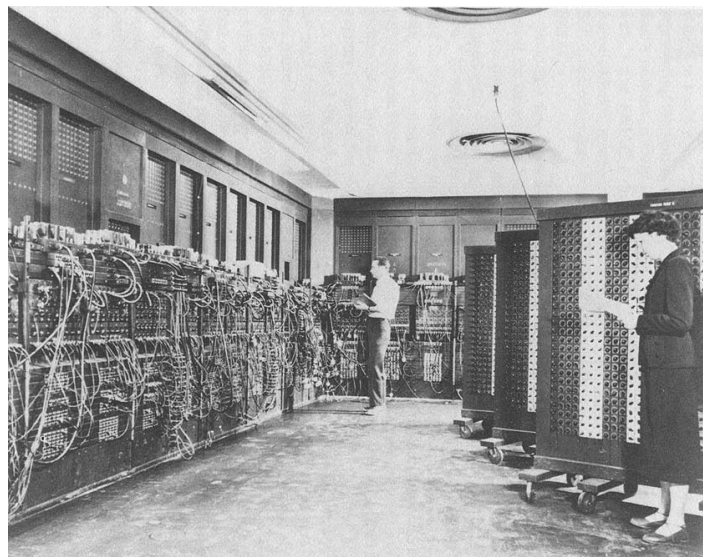


Figura 2.1 – Manutenção do ENIAC

Nos anos 50, J. Von Neumann, E. F. Moore e C. E. Shannon [Neumann52] [Moore56] iniciaram o desenvolvimento de teorias relativas à utilização de redundância para a construção de estruturas lógicas fiáveis a partir de componentes com baixa confiabilidade, cujas falhas eram mascaradas pela utilização de redundância. As teorias de mascaramento por redundância foram depois unificadas por W.H. Pierce [Pierce65] através do conceito de tolerância a falhas, tendo pouco depois A. Avizienis [Avizienis67] integrado técnicas práticas de detecção de erros, mascaramento, diagnóstico e recuperação, num conceito mais abrangente de sistemas tolerantes a falhas.

A invenção do transistor em 1948 [Maiden07] e do circuito integrado (CI) em 1959 aumentaram consideravelmente a confiabilidade dos sistemas electrónicos, para o que contribuiu também a redução do número de componentes, permitida por estas tecnologias. Assim, a confiabilidade dos equipamentos melhorou consideravelmente até aos anos 60, embora sem progressos apreciáveis, em termos de tolerância a falhas.

Com a evolução dos sistemas computacionais, tornou-se usual a utilização de serviços em pacotes (*batches*) para processamento, com os operadores a planificar a sua execução de forma a maximizar a taxa de ocupação das máquinas. Como era normal que cada pacote demorasse horas a completar, a ocorrência de eventuais erros, que obrigavam à reinicialização do processamento do pacote, provocavam inevitavelmente grandes reduções de desempenho. O tempo de execução (*turnaround time*) não era um problema importante nessa época.

O aparecimento de computadores disponíveis em rede, como o TX-2 [Clark57], desenvolvido no laboratório Lincoln do MIT (*Massachusetts Institute of Technology*) em 1958, elevou a importância da confiabilidade a um novo patamar. Os utilizadores começaram a esperar que o serviço estivesse disponível sempre que lhe pretendiam aceder. Ao mesmo tempo, o programa espacial americano introduziu novos requisitos, visto que a abordagem baseada em manutenção deixou de ser viável. A confiabilidade tornou-se subitamente fundamental e o interesse em investimento na área aumentou consideravelmente.

Nos anos 70, a expansão dos computadores aumentou as preocupações com o custo da indisponibilidade do serviço, e o investimento em confiabilidade começou a ser visto como uma utilização rentável de recursos, pelo mercado industrial.

No início dos anos 80, o grupo de trabalho do IEEE (*Institute of Electrical and Electronics Engineers*) designado por IFIP (*International Federation for Information Processing*) WG 10.4 *Dependable Computing and Fault Tolerance*, ele próprio derivado de um grupo anterior, promoveu o desenvolvimento de um conjunto unificador e consistente de conceitos, sumariados e publicados em 1985, quando J. Laprie [Laprie85] apresentou as conclusões do grupo. Os conceitos propostos foram revistos e expandidos desde a sua formulação inicial mas, no essencial, mantêm-se válidos até à actualidade, apesar de diversas expansões necessárias para lidar com problemas emergentes. Por exemplo, as falhas intencionais<sup>1</sup> não eram abordadas no documento original, mas à medida que se tornaram um problema mais sério, com o alargamento da comunicação e armazenamento de dados, o conceito de segurança foi introduzido numa revisão subsequente.

Apesar de não existir uma norma oficial para definir a terminologia de confiabilidade, o conjunto de conceitos apresentados no documento referido, e expandido nas suas revisões subsequentes, é geralmente aceite como uma norma “de facto”.

---

<sup>1</sup> Falhas intencionais representam aquelas que são deliberadamente introduzidas no sistema por um utilizador humano, sendo explicadas na subsecção 2.2.1.

No início dos anos 90, as aplicações passaram a constituir o estrangulamento (*bottleneck*) da confiabilidade em sistemas computacionais, com estudos contemporâneos a mostrar que o equipamento disponível na época era já adequadamente tolerante a falhas físicas. Esta tendência manteve-se até recentemente, com o consequente aumento do interesse em aplicações tolerantes a falhas.

Simultaneamente, o crescimento dos serviços distribuídos e o aparecimento da Internet, converteram a segurança num assunto chave. À medida que aumenta a informação disponível e em circulação *on-line*, tornam-se potencialmente mais graves os riscos de danos, provocados por acessos não autorizados.

Actualmente, os avanços nas tecnologias de fabrico de sistemas electrónicos e a previsível evolução da miniaturização dos respectivos componentes, obriga a considerar o aumento dos riscos de perturbações provocadas pelo meio ambiente. A elevada sensibilidade a alterações de estado de elementos de memória, por acção de elementos exteriores (e.g. radiação, influência electromagnética), que geralmente só era considerada para sistemas utilizados em ambientes hostis (i.e. espaço, altas altitudes), passou a ter de ser considerada também para equipamento convencional.

## 2.2 Confiabilidade

### 2.2.1 Conceitos Fundamentais

O estudo dos sistemas computacionais em geral, e dos sistemas computacionais confiáveis em particular, requer não só uma terminologia adequada, mas principalmente a utilização de um conjunto de conceitos, que permita descrever o seu funcionamento e os problemas que os afectam, sem ambiguidade.

#### 2.2.1.1 Sistemas e suas propriedades

O conceito base para este tipo de estudo é o próprio **sistema**, que pode ser definido como uma entidade que interage com outras (i.e. outros sistemas), incluindo aplicações, equipamento, seres humanos e o meio ambiente (fenómenos naturais) [Avizenis04].

Um sistema diz-se **computacional** se for baseado numa unidade de processamento (CPU – *Central Processing Unit*), que executa um conjunto pré-definido de instruções, e por um banco de memória, onde são armazenados o programa a executar e eventualmente os dados a

utilizar. O funcionamento deste tipo de sistema consiste na execução de uma série de instruções (i.e. programa), que definem como evolui a relação entre entradas e saídas. O conceito de sistema tem associados outros conceitos que importa clarificar, nomeadamente:

- O **ambiente** (*environment*), constituído pelo conjunto dos sistemas que envolvem e comunicam com o sistema sob observação.
- A sua **fronteira** (*boundary*), definida pelos limites entre o sistema sob observação e o seu ambiente.
- A sua **função** (*function*), que define o que o sistema deve fazer e é descrita em detalhe numa especificação funcional.
- O seu **comportamento** (*behavior*), que representa o que o sistema realmente faz para executar a sua função e é descrito por uma sequência de estados.
- Os **estados** (*states*) do sistema, sendo cada um deles composto pela informação relativa ao processamento, comunicação, informação armazenada, interligações e condição física, num determinado instante de tempo.
- O **serviço** (*service*) definido pelo comportamento do sistema, tal como é percebido pelos seus utilizadores.
- O **utilizador** (*user*), por sua vez um outro sistema, que recebe serviço do sistema sob observação.
- A **interface** (*interface*) do sistema consiste na parte da fronteira onde é entregue o serviço ao utilizador (ou utilizadores). Esta interface pode ser subdividida em entradas e saídas, consoante o sentido do fluxo de informação.

O serviço fornecido por um sistema pode ser classificado da seguinte forma:

- **Serviço correcto** (*correct service*), quando fornecido de acordo com a especificação.
- **Serviço incorrecto** (*incorrect service*), quando fornecido diferentemente da especificação.
- **Serviço indisponível** (*service unavailable*), quando nenhum serviço é fornecido. Por vezes não é percéptivel aos utilizadores a diferença entre esta classificação e a do serviço incorrecto.

Os sistemas computacionais podem ser caracterizados por quatro propriedades intrínsecas, nomeadamente:

- A **funcionalidade** (*functionality*), que representa o conjunto de funções que devem ser executadas pelo sistema, tal como descritas na sua especificação.
- O **desempenho** (*performance*), que representa os resultados quantificáveis que descrevem a progressão do sistema e as suas respostas.
- O **custo** (*cost*), que inclui todos os encargos, financeiros e outros, associados à instalação e manutenção do sistema.
- A **confiabilidade** (*dependability*), que é definida como a capacidade de fornecer um serviço em que se pode justificadamente acreditar.

A confiabilidade é particularmente importante neste tipo de sistemas, pois é frequente que um serviço incorrecto apresente consequências potenciais mais graves do que as que resultam da sua indisponibilidade. Em linguagem corrente, pode-se dizer que algo (ou alguém) é confiável, se se puder acreditar que irá executar a sua função de forma adequada. Para lidar com as diferentes aplicações em sistemas computacionais, o conceito de confiabilidade é caracterizado por um conjunto de atributos, que inicialmente incluía apenas dois elementos [Laprie85], tendo sido posteriormente expandido, para os seguintes seis (não necessariamente todos em conjunto) [Avizenis04]:

- A **disponibilidade** (*availability*), que consiste na probabilidade de um sistema apresentar um serviço correcto, num determinado instante de tempo.
- A **fiabilidade** (*reliability*), que consiste na probabilidade de um sistema apresentar um serviço correcto continuamente, durante um determinado intervalo de tempo.
- A **segurança** (*safety*), que consiste na probabilidade do sistema não apresentar uma avaria que acarrete consequências catastróficas para os seus utilizadores, num determinado intervalo de tempo.
- A **confidencialidade** (*confidentiality*), que consiste na probabilidade de não ser divulgada informação indevida, num determinado intervalo de tempo.
- A **integridade** (*integrity*), que consiste na probabilidade de não ocorrerem alterações impróprias de estado, num determinado intervalo de tempo.
- A **manutenabilidade** (*maintenability*), que representa a capacidade de um sistema poder ser reparado, retornando ao estado de serviço correcto, num determinado intervalo de tempo, após a detecção de uma avaria.

Dependendo da funcionalidade do sistema, vários atributos podem ser enfatizados, de forma diferente, com os requisitos de confiabilidade do sistema a definir os objectivos a atingir para

cada um. Estes requisitos podem ser definidos como limites para a frequência com que ocorre serviço incorrecto e para a gravidade das suas consequências (usualmente em termos do tempo de inactividade), sendo válidos para um determinado conjunto de problemas e um ambiente operacional bem definido.

Outros atributos de confiabilidade são frequentemente utilizados, mas estes normalmente não são mais do que combinações ou especializações dos seis atributos básicos apresentados. Por exemplo, a seguridade (*security*) não é mais do que a existência simultânea de disponibilidade (para utilizadores autorizados), confidencialidade e integridade, com o termo “acidental” a ser interpretado como “não autorizado”.

A importância dos diversos atributos depende da aplicação específica considerada para o sistema alvo. A disponibilidade é sempre necessária, se bem que em grau variável, enquanto que a fiabilidade, a segurança e a confidencialidade são opcionais. Por exemplo, um microcontrolador, utilizado em electrónica automóvel, não tem requisitos elevados de confidencialidade (no sentido em que os dados em processamento não são alvo de acessos indevidos), mas a fiabilidade é importante e a segurança é fundamental. A importância de cada atributo da confiabilidade deve ser interpretada num sentido relativo e probabilístico, e não como um valor absoluto e determinístico. É importante notar que, devido à inevitável presença e ocorrência de falhas, os sistemas computacionais nunca são 100% disponíveis, fiáveis ou seguros.

Um sistema pode avariar porque não cumpre a sua especificação ou porque a especificação não descreve adequadamente a sua funcionalidade, ou a interacção com o ambiente. Apesar da razão básica para uma avaria ser uma falha, os seus efeitos não são imediatamente perceptíveis e poderão continuar a propagar-se até que um utilizador se aperceba das consequências. A descrição deste processo requer a clarificação de alguns conceitos adicionais, nomeadamente falha, erro e avaria:

- Uma **avaria** (*failure*) é um desvio do serviço fornecido em relação à funcionalidade prevista, e ocorre quando um erro atinge a interface do sistema e afecta o serviço fornecido.
- Um **erro** (*error*) é uma modificação de um elemento do sistema, que pode causar uma subsequente avaria.
- Uma **falha** (*fault*) é a provável ou hipotética causa de um erro.

Uma falha ou erro dizem-se **activos** (*active*) quando produzem um erro ou avaria, respectivamente, e designam-se como **latentes** (*latent*) em caso contrário. É importante notar

que uma falha não origina necessariamente um erro, nem este uma subsequente avaria, como será explicado em maior detalhe na subsecção 2.2.2. No entanto, o contrário já não se verifica, pois uma avaria tem sempre origem num erro, sendo ele próprio causado por uma falha. A Figura 2.2 apresenta a sequência típica que conduz a uma avaria, a partir da ocorrência de uma falha.

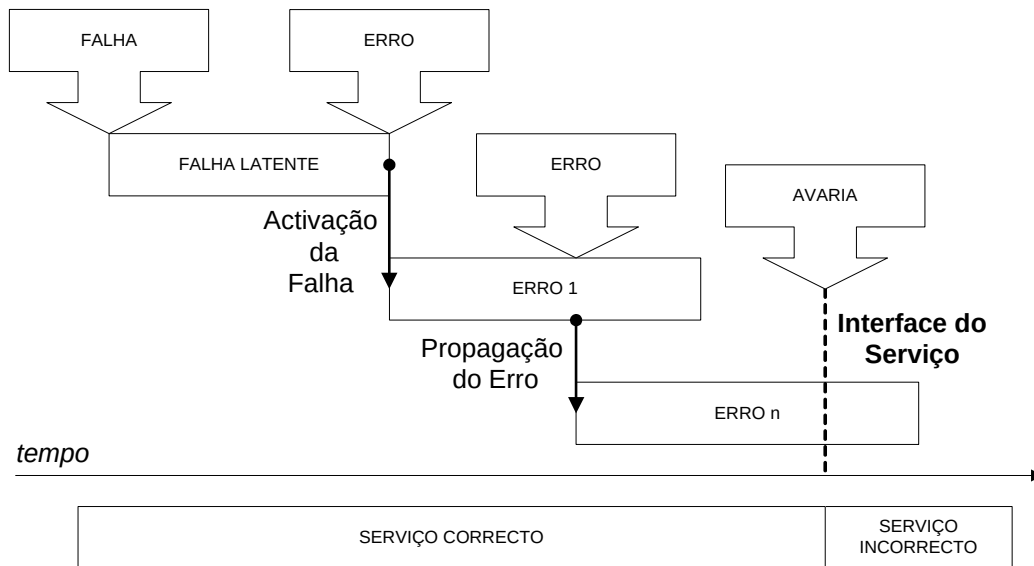


Figura 2.2 – Da Falha à Avaria

Para a classificação das falhas é importante considerar o ciclo de vida do sistema, que se divide em duas fases principais, designadas como **fase de desenvolvimento** (*development phase*) e **fase operacional** (*deployment phase*). A fase de desenvolvimento engloba todas as tarefas até à entrega do sistema para utilização no seu ambiente operacional. A fase operacional inicia-se com a aceitação do sistema para funcionamento. Esta fase consiste em períodos de serviço correcto, incorrecto e indisponível. O serviço incorrecto inclui períodos em que este deveria ser disponibilizado, mas é fornecido incorrectamente, ou não é fornecido. O serviço indisponível representa situações em que este é deliberadamente interrompido por uma entidade supervisora. Durante a fase operacional, um sistema pode ser submetido a manutenção, incluindo reparações e modificações, as quais podem ser executadas num ou vários dos três períodos referidos, dependendo do sistema.

### 2.2.1.2 Erros e Avarias

Os modos de avaria são definidos pelas diferentes formas em que ela se pode manifestar, podendo cada um deles apresentar diversos graus de gravidade. A ocorrência de uma avaria é definida relativamente à funcionalidade do sistema e não à descrição incluída na sua especificação funcional. É perfeitamente possível que um serviço que esteja de acordo com a especificação funcional seja inadequado para o utilizador final, se aquela não representar adequadamente a funcionalidade pretendida, constituindo portanto um erro de especificação. Nestas circunstâncias, o facto de ter ocorrido um evento indesejável (sendo de facto uma avaria), poderá ser reconhecido apenas através das suas consequências (e.g. uma reacção inesperada a uma sequência de comandos) e, eventualmente, bastante depois da sua ocorrência. Em termos gerais, as avarias podem ser algo subjectivas, requerendo discernimento para as identificar e caracterizar.

O serviço incorrecto pode ser caracterizado pelos modos de avaria, de acordo com as classificações apresentadas na Figura 2.3.

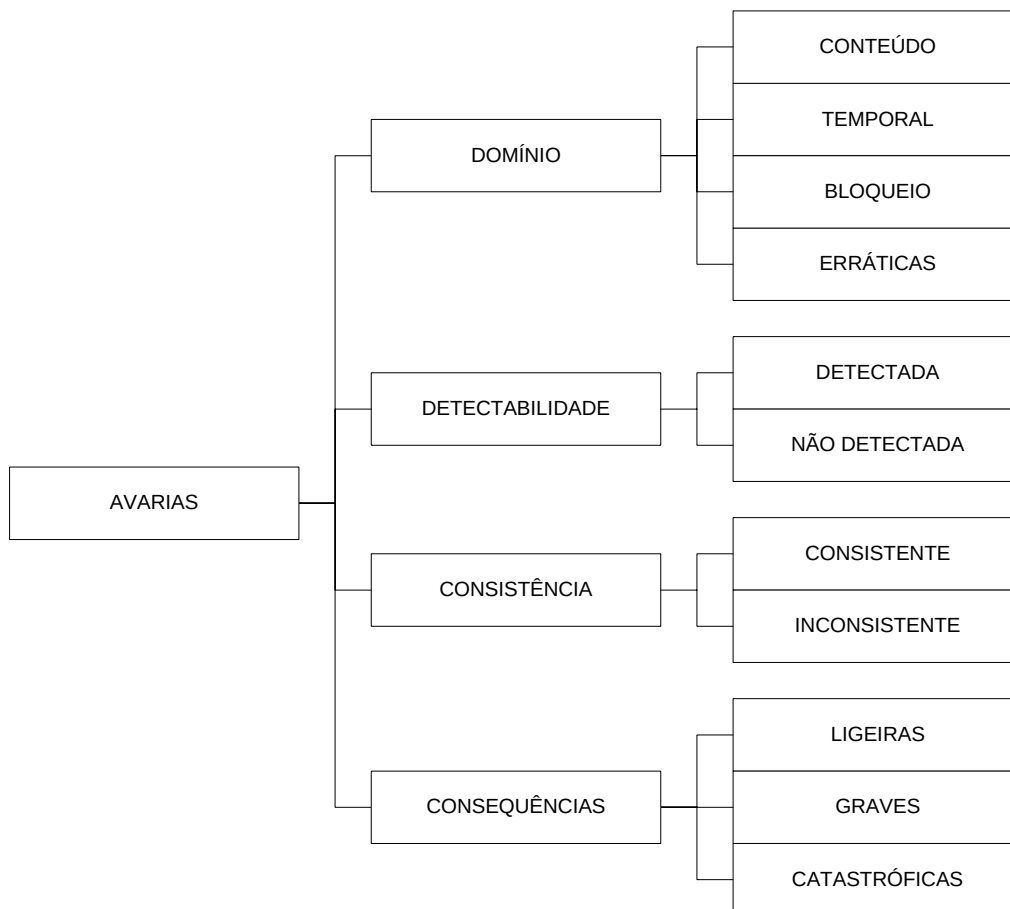


Figura 2.3 – Modos de Avaria

O **domínio** (*domain*) da avaria distingue principalmente entre avarias de **conteúdo** (*content*) e **temporais** (*temporal*), com o primeiro caso representando situações em que a informação entregue na interface do serviço é diferente da pretendida e o segundo representando situações em que a duração ou instante em que a informação é disponibilizada, difere da funcionalidade pretendida. Uma avaria temporal pode ser de atraso, adiantamento ou duração incorrecta.

Avarias em que o conteúdo e o comportamento temporal estão simultaneamente errados podem ser classificadas como avarias de **bloqueio** (*halt*) ou **erráticas** (*erratic*). As primeiras verificam-se quando o estado perceptível do sistema se torna constante (a actividade deixa de ser perceptível para os utilizadores) ou quando nenhum serviço é disponibilizado (avaría silenciosa). Uma avaria errática ocorre quando serviço é disponibilizado, mas é completamente inconsistente com a funcionalidade pretendida.

A **detectabilidade** (*detectability*) refere-se à capacidade do sistema sinalizar a avaria aos utilizadores, i.e. proporcionar algum tipo de sinal para indicar a sua ocorrência, podendo assim ser classificada como **detectada** (*detected*) ou **não detectada** (*undetected*). Os mecanismos de detecção de avarias podem eles mesmos avariar, gerando falsos alarmes ou omitindo a sinalização. Quando a ocorrência de uma avaria resulta em modos de serviço com desempenho reduzido, o sistema deverá informar o utilizador de que se encontra em modo degradado. O serviço assim fornecido pode variar entre uma redução aceitável, o fornecimento de serviços mínimos (ou de emergência), ou mesmo a desactivação automática (e segura).

A **consistência** (*consistency*) das avarias é relevante apenas quando um sistema tem dois ou mais utilizadores, ou quando são consideradas execuções diferentes do mesmo serviço em condições semelhantes. As avarias são consideradas **consistentes** (*consistent*) quando todos os utilizadores do sistema se apercebem do serviço incorrecto da mesma forma, ou quando este falha de forma idêntica em diferentes execuções, sendo portanto determinístico. As avarias são consideradas **inconsistentes** (*inconsistent*) quando alguns ou todos os utilizadores se apercebem de serviço incorrecto de forma diversa (alguns podem mesmo aperceber-se de serviço correcto), ou o sistema avaria de forma diferente, mesmo quando submetido a condições idênticas<sup>2</sup>.

---

<sup>2</sup> Avarias inconsistentes são também designadas por avarias bizantinas (Byzantine failures), a partir da histórica corrupção dos generais do império bizantino, que fazia com que nunca se tivesse a certeza de dizerem a verdade.

No que toca às **consequências** das avarias, a classificação é mais aberta. Estas consequências dependem da funcionalidade do sistema (i.e. uma avaria grave numa máquina de café tem muito menos relevância do que uma avaria ligeira num sistema de controlo aeroportuário). A classificação depende portanto do tipo de serviço fornecido pelo sistema. De uma forma geral, avarias ligeiras (*minor*) são aquelas cujas consequências apresentam um custo semelhante aos benefícios fornecidos por um serviço correcto, e avarias catastróficas (*catastrophic*) são aquelas cujo custo é várias ordens de grandeza superior. Diversos graus intermédios são também possíveis.

Um erro considera-se detectado quando a sua presença se manifesta através de uma subsequente avaria ou quando é indicado por um sinal (ou mensagem) de erro. Se um erro se encontra presente mas não está detectado, diz-se latente. Se um erro origina ou não uma avaria, depende do funcionamento do sistema e da sua redundância natural. Por exemplo, um erro numa operação de escrita em memória, pode ser eliminado por um mecanismo adequado no barramento, ou simplesmente quando a aplicação escrever novamente na mesma posição, sem ter entretanto utilizado o valor errado. Exceptuando esta distinção entre erros latentes e detectados, a classificação é feita em termos da avaria que originam, sendo usados os mesmos atributos<sup>3</sup>.

### 2.2.1.3 Falhas

O desenvolvimento de infra-estruturas e de técnicas dedicadas para lidar com falhas requer uma prévia classificação do seu universo e origens. O conhecimento de todas as origens possíveis permite decidir adequadamente quais as falhas a incluir numa especificação de confiabilidade (ou segurança). Uma definição bastante completa e universal é apresentada em [Aviezenis04] e inclui oito classes elementares, tal como representado na Figura 2.4. Cada falha possível pode ser classificada por uma combinação das classes apresentadas. O limite máximo teórico de classificações seria 256, mas não é possível combinar alguns critérios (i.e. não é possível ter falhas naturais intencionais), pelo que o número de combinações efectivamente identificadas em ambiente operacional é de apenas 31.

---

<sup>3</sup> Por exemplo, um erro que origina uma avaria temporal consistente, será um erro temporal consistente.

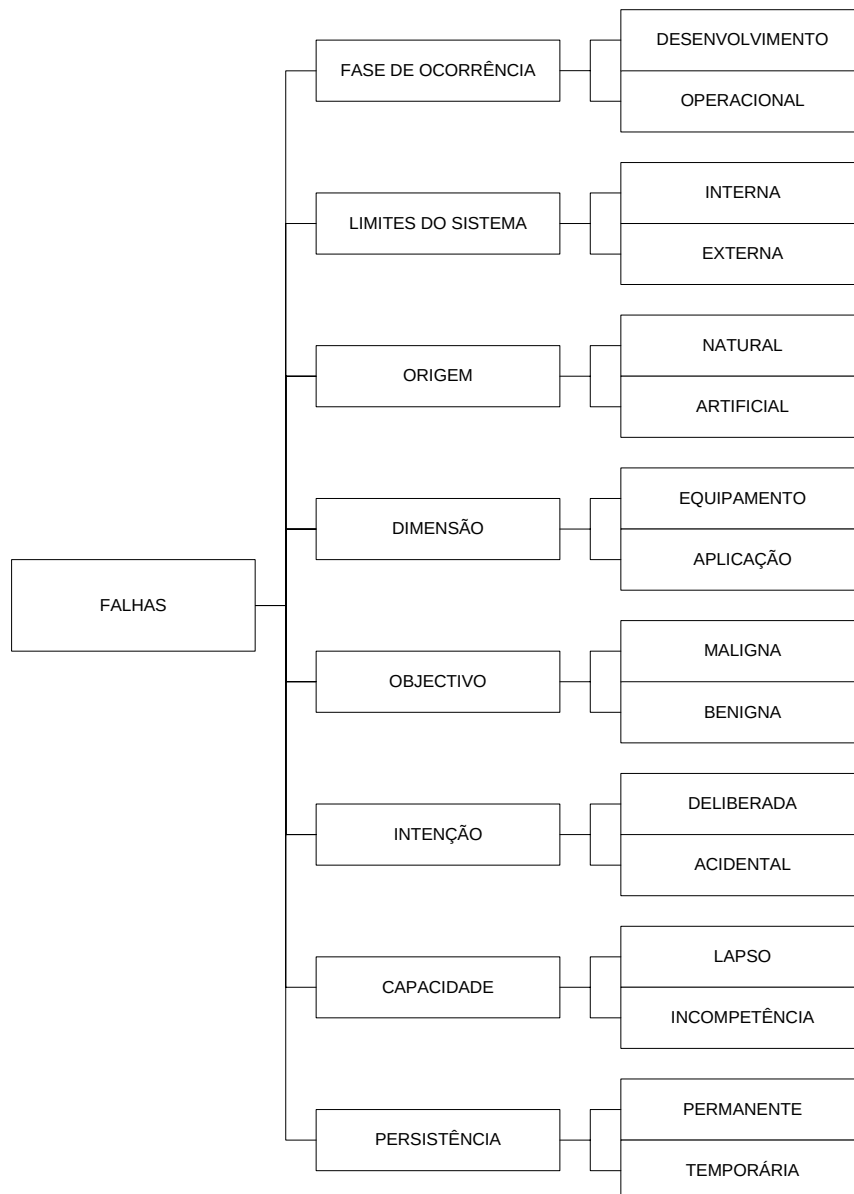


Figura 2.4 – Classes de Falha

Em termos da **fase de ocorrência** (*phase of occurrence*), uma falha considera-se de **desenvolvimento** (*development*) se ocorrer durante o projecto, fabrico, manutenção (mesmo que o sistema já esteja operacional), ou na geração de procedimentos para o operar ou manter. A falha considera-se **operacional** (*operational*) apenas se ocorrer enquanto estiver a ser fornecido serviço. Esta distinção é feita relativamente à ocorrência da falha e não à sua manifestação. Apesar de ser normal que uma falha gerada na fase de desenvolvimento apenas se manifeste na utilização operacional, é mesmo assim classificada como falha de desenvolvimento. Para ser classificada como operacional, uma falha terá de ocorrer depois de o sistema estar em utilização (por exemplo uma falha provocada por radiação electromagnética).

Uma falha considera-se **interna** (*internal*) se ocorrer completamente dentro dos **limites do sistema** (*system boundaries*) e **externa** (*external*) se ocorrer no seu exterior, propagando, no entanto, erros para o seu interior, por interacção ou interferência. Para uma falha externa afectar o sistema, terá de existir uma vulnerabilidade, i.e. uma característica interna que permite que um factor externo o danifique. Por exemplo, quando um pico de corrente (falha externa) danifica um CI, a falta de protecção adequada é uma vulnerabilidade. A diferença entre uma vulnerabilidade e uma falha interna decorre da primeira só por si não provocar nenhum erro (em certos casos a distinção é argumentativa).

Relativamente à **origem** (*phenomological cause*), as falhas **naturais** (*natural*) representam fenómenos físicos sem influência humana, correspondendo todas elas também a falhas do equipamento. As falhas naturais mais frequentes são os defeitos introduzidos no processo produtivo, a deterioração dos componentes ou fenómenos físicos originados no exterior do sistema, mas que interferem com o seu funcionamento (e.g. radiação, picos de tensão). Falhas **artificiais** (*human-made*) resultam de acção humana (falhas de comissão) ou de ausência de acção necessária (falhas de omissão).

Em termos de **dimensão** (*dimension*) uma falha pode afectar o **equipamento** (*hardware*) ou a **aplicação** (*software*), apesar dos recentes avanços na área da lógica programável tornarem esta classificação algo difusa. Uma falha é classificada como de aplicação se afectar programas ou dados e como de equipamento se afectar componentes físicos<sup>4</sup>.

Falhas artificiais requerem uma classificação adicional em termos de **objectivo** (*objective*), **intenção** (*intent*) e **capacidade** (*capability*). As falhas podem ser **malignas** (*malicious*) se introduzidas com o objectivo de causar danos ao sistema, ou **benignas** (*non-malicious*) nos restantes casos. As benignas podem ainda ser classificadas em **acidentais** (*non-deliberate*), se devidas a acções não intencionais, ou **deliberadas** (*deliberate*), se devidas a más decisões (sem intenção maligna). Nas soluções de compromisso, por vezes necessárias no desenvolvimento de sistemas computacionais, é frequente encontrar funcionalidades incompletamente especificadas, que vêm a causar comportamentos incorrectos, em circunstâncias não consideradas durante o desenvolvimento. Este tipo de falha pode ser classificado como falha de desenvolvimento artificial, deliberada e benigna.

Mesmo nos casos em que as falhas são introduzidas por seres humanos, não intencionalmente, é necessário distinguir a capacidade dos intervenientes, de forma a identificar os casos de negligência grosseira devida à falta de competência dos indivíduos

---

<sup>4</sup> Clarificação: Falhas de aplicação representam código errado e falhas de equipamento representam implementações erradas de código correcto (o que também é aplicável a lógica programável).

envolvidos. A diferença entre este tipo de falhas e os lapsos cometidos por pessoal qualificado, é a base para a distinção entre falhas por **lapso** (*accidental*) e por **incompetência** (*incompetence*).

No que toca à **persistência** (*persistence*) das falhas, elas podem ser **permanentes** (*permanent*) se permanecem constantes após a sua ocorrência, ou **temporárias** (*transient*), se tiverem duração limitada ou forem passíveis de eliminação, durante a operação normal do sistema afectado.

Os exemplos seguintes permitem um melhor entendimento dos conceitos apresentados:

- Um estado errado num elemento lógico (*flip-flop* ou *gate*), causado por um curto-circuito, é um erro. A causa para o curto-circuito é uma falha, ocorrendo a avaria quando uma saída do circuito, do qual o elemento faz parte, apresentar um valor errado.
- A inversão de um bit da memória de um microprocessador que calcula a posição de um satélite, devido à influência de radiação espacial, representa uma falha latente, até que esta posição de memória seja lida. A falha é então activada e o valor lógico incorrecto torna-se um erro (ou não). Ao ser lido e utilizado o valor errado, vão-se criar novos erros que se propagarão no interior do microprocessador, provocando decisões inesperadas e erros noutras posições de memória. Eventualmente ocorrerá uma avaria, quando o microprocessador enviar valores errados ao sistema central. Se a posição de memória onde ocorreu a inversão de bit for acedida para escrita, antes da sua propagação, não ocorrerá nenhum erro ou avaria.
- Um erro humano, que resulte numa instrução inválida no código, converte o programa resultante num componente com uma falha latente. Uma vez iniciado o programa, a execução da instrução errada origina um erro, que eventualmente se propagará para uma avaria.

Em paralelo com as referidas classes, podem ser considerados três outros grupos de falhas (parcialmente sobrepostos), que são utilizados frequentemente como forma mais prática de classificação. Temos assim as falhas de desenvolvimento, que incluem todas aquelas que ocorrem durante esta fase, as falhas físicas, que compreendem todas as que afectam o equipamento, e as falhas de interacção, que incluem todas as falhas externas.

### 2.2.2 Técnicas para lidar com falhas

O desenvolvimento de sistemas computacionais confiáveis recorre a quatro técnicas para lidar com falhas (*fault handling*), que podem ser utilizadas individualmente de forma combinada:

- **Prevenção de falhas** (*fault prevention*), com o objectivo de prevenir a sua ocorrência ou introdução.
- **Tolerância a falhas** (*fault tolerance*), com o objectivo de fornecer serviço correcto, mesmo na sua presença.
- **Remoção de falhas** (*fault removal*), com o objectivo de reduzir o seu número e/ou gravidade.
- **Previsão de falhas** (*fault forecasting*), para estimar o número presente, ocorrência futura e suas consequências prováveis.

A prevenção de falhas é frequentemente considerada como um recurso da engenharia em geral e não exclusiva da área da confiabilidade. Todos os problemas relacionados com o desenvolvimento de um sistema robusto (seja equipamento, aplicações ou ambos) estão englobados nesta disciplina. O objectivo é evitar as falhas de projecto e de produção, e eliminar as vulnerabilidades nestas etapas do ciclo de vida do sistema.

O uso correcto de metodologias e tecnologias adequadas é a forma indicada para melhorar os processos de desenvolvimento, mantendo a mais elevada qualidade. O uso de modularidade e linguagens tipadas<sup>5</sup> (para aplicações), e de regras de projecto e controlo de qualidade (para equipamento), são exemplos possíveis de técnicas de prevenção de falhas.

A tolerância a falhas tem como objectivo evitar avarias e é executada através da detecção de erros e da recuperação do sistema [Santos98]. Como a designação indica, o objectivo não é evitar as falhas, mas antes manuseá-las, de forma a que não ocorra uma avaria ou, se tal não for possível, reduzir a sua gravidade.

Um sistema tolerante a falhas pode incluir diversos elementos em diferentes combinações, sendo o elemento fundamental a **detecção de erros** (*error detection*). Por definição, o efeito visível de uma falha é o erro, que terá de ser detectado, para que outros elementos possam reagir adequadamente [Gericota01]. É conveniente notar que um sistema tolerante a falhas pode incluir apenas detecção de erros, podendo citar-se como exemplo a grande variedade de

---

<sup>5</sup> Linguagens tipadas são aquelas que usam variáveis com tipos específicos, geralmente de declaração obrigatória.

luzes de aviso presentes em vários equipamentos da electrónica de consumo (e.g. dispositivos multimédia, electrodomésticos).

A detecção de erros é feita de forma **concorrente** (*concurrent detection*), se executada durante o serviço normal, ou de forma **preventiva** (*preemptive detection*), se executada quando o serviço se encontra suspenso. Uma vez detectado um erro, num sistema tolerante a falhas, torna-se possível iniciar um processo de **recuperação** (*recovery*), que transforma um estado errado num estado correcto (sem erros detectados).

Alguns sistemas tolerantes a falhas lidam primariamente com erros, mas é desejável que as falhas originais sejam também eliminadas, tomando-se simultaneamente providências para que não voltem a ocorrer. É nestes dois níveis de abordagem que se localizam as diferenças entre **lidar com erros** (*error handling*) e **lidar com falhas** (*fault handling*).

O manuseamento de erros subdivide-se ainda em recuperação (*rollback*), restabelecimento (*rollforward*) e compensação (*compensation*):

- A **recuperação** consiste em recolocar o sistema num estado correcto, anterior à ocorrência do erro, sendo necessário que este tenha sido previamente guardado e que esteja disponível.
- O **restabelecimento** consiste em colocar o sistema num estado correcto, o qual é normalmente um estado novo e pré-definido.
- A **compensação** é baseada no uso de redundância para mascarar os erros e impedir a sua propagação. Existem diversas soluções deste tipo, recorrendo as mais frequentes a Redundância Modular Tripla (TMR – *Triple Modular Redundancy*) [Lala85][Kastensmidt05] e aos códigos de detecção de erros.

O manuseamento de falhas subdivide-se em diagnóstico (*diagnosis*), isolamento (*isolation*), reconfiguração (*reconfiguration*) e reinicialização (*reinitialization*):

- O **diagnóstico** consiste em registar a causa dos erros no tempo e no espaço, de forma a que possa ser analisada posteriormente.
- O **isolamento** consiste em confinar, física ou logicamente, os componentes avariados.
- A **reconfiguração** é a gestão de componentes sobressalentes ou o reescalamento de tarefas para componentes operacionais, de forma a manter um serviço correcto.
- A **reinicialização** verifica, actualiza e carrega uma configuração correcta (i.e. sem erros detectados) para retomar o serviço normal.

Para serem consideradas parte de um mecanismo de tolerância a falhas, todas as operações referidas deverão ser automáticas, não requerendo a acção de um agente externo. Se esta for necessária, então aplica-se o termo **manutenção** (*maintenance*). O cenário habitual consiste num sistema tolerante a falhas, a operar autonomamente, com tarefas de manutenção executadas periodicamente, ou na detecção de erros, baseadas em informação disponibilizada pelos componentes tolerantes a falhas. Em alguns casos muito específicos (i.e. aplicações espaciais), a manutenção não é exequível e os componentes tolerantes a falhas deverão manter o serviço pelo máximo período de tempo possível (eventualmente com alguma degradação).

Se a remoção de falhas ocorrer durante a fase operacional, é classificada como manutenção correctiva ou preventiva, podendo estas ser aplicadas a qualquer tipo de sistemas.

- A **manutenção correctiva** (*corrective maintenance*) remove falhas que originaram erros detectados.
- A **manutenção preventiva** (*preventive maintenance*) detecta e remove falhas, antes que causem erros.

Este último tipo de manutenção pode ser executado para localizar falhas que tenham ocorrido desde a manutenção anterior, ou para eliminar aquelas que tenham sido detectadas em sistemas similares.

Quando executada na fase de desenvolvimento de um sistema, a remoção de falhas consiste em três passos: verificação, diagnóstico e correcção. O primeiro confirma a conformidade com as propriedades pretendidas e é discutido em maior detalhe na subsecção 2.2.3. Se a verificação não tiver sucesso, é seguida por um diagnóstico, com o objectivo de identificar as falhas que impediram o resultado positivo, e depois por um processo de correcção. A verificação é repetida depois de todas as falhas identificadas serem corrigidas.

A previsão de falhas consiste em estimar o comportamento do sistema em termos da sua probabilidade de ocorrência e eventuais consequências. Em geral parte-se da identificação, classificação e análise dos modos de avaria detectados, sendo portanto uma avaliação qualitativa. Em alternativa, podem avaliar-se os atributos específicos de forma estatística, sendo então uma avaliação quantitativa. Neste último caso, os atributos medidos serão as métricas de confiabilidade do sistema. Ambas as abordagens são complementares e englobam diversos procedimentos, como sejam a análise de modos de avaria, os diagramas de fiabilidade, etc.

As variações na ênfase dos atributos da confiabilidade afectam directamente o equilíbrio das técnicas (prevenção, tolerância, remoção e previsão) que deverão ser utilizadas para tornar o sistema confiável. Este problema pode ser difícil de resolver, pois alguns atributos entram em conflito (e.g. disponibilidade com segurança), pelo que o equilíbrio entre as técnicas disponíveis tem de ser considerado no projecto inicial.

A aplicação da técnica pretendida é também um problema complexo e a sua exequibilidade deve ser adequadamente garantida. Como a maioria dos dispositivos computacionais, os mecanismos que lidam com falhas têm de ser devidamente verificados e validados.

### 2.2.3 Verificação e Validação

Os conceitos de verificação e validação são muito importantes na engenharia em geral e na ciência dos computadores em particular. Como estes conceitos estão consideravelmente relacionados, é frequente alguma confusão e indefinição na sua utilização. As definições adoptadas neste documento são apresentadas na Tabela 2.1.

Tabela 2.1 – Verificação e Validação

|                    | Definição                                                                                                                         | Pergunta-chave                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------|--------------------------------------|
| <b>Verificação</b> | Confirmar por análise e fornecimento de prova objectiva, de que a especificação foi cumprida.                                     | Construímos o sistema correctamente? |
| <b>Validação</b>   | Estabelecer por prova objectiva que a especificação do dispositivo está de acordo com as necessidades dos utilizadores previstos. | Construímos o sistema correcto?      |

No desenvolvimento de CIs, a verificação equivale a confirmar que os parâmetros, requerimentos ou especificações pré-estabelecidos foram cumpridos. Os procedimentos de verificação consistem em conferir diversos itens, incluindo aplicações, para a conformidade e consistência com uma especificação associada. Em sistemas computacionais, a verificação pode ser classificada como estática ou dinâmica. A verificação estática é feita sem exercitar o sistema e pode ser executada sobre o próprio ou sobre um seu modelo. A verificação dinâmica é executada de forma experimental sobre o sistema alvo e, dependendo das entradas utilizadas, pode ser subclassificada como execução simbólica ou de teste, sendo no segundo caso as entradas do sistema acedidas directamente.

O teste é uma parte importante da verificação, sendo possíveis diversos tipos, dependendo do sistema considerado (SUT – *System Under Test*) e dos objectivos propostos. Os tipos de teste mais frequentes são o teste estrutural e o teste funcional. O primeiro requer

conhecimento e acesso à infra-estrutura interna do SUT e é usualmente executado ao nível da porta lógica ou do transístor. O teste funcional utiliza apenas uma descrição funcional do SUT, sendo usualmente executado ao nível lógico ou do conjunto de instruções.

Nos sistemas computacionais actuais, o teste exaustivo, incluindo todas as combinações de entradas, é impossível (pelo menos em tempo útil). É frequente que a execução de uma metodologia de teste completa, com perto de 100% de cobertura dos padrões de entradas possíveis, demore várias décadas<sup>6</sup>.

A geração de padrões de teste é um problema por si só, sendo as duas abordagens mais comuns, o teste determinístico e o teste aleatório. No primeiro, os padrões de teste são pré-determinados de forma selectiva, segundo um critério bem definido. Esse critério pode ser sistemático, no sentido de que as entradas escolhidas seguem um padrão de comportamento, de forma a testarem completamente um conjunto de condições, ou pode ser adaptado a um objectivo mais complexo e/ou específico, geralmente de forma personalizada (i.e. manual). O teste pseudo-aleatório (ou estatístico) utiliza padrões seleccionados, de acordo com uma distribuição de probabilidades, definida por um determinado modelo.

Como exemplo da aplicação de diferentes padrões de teste, podemos considerar o trabalho descrito em [Velazco90], que apresenta a utilização de diferentes alternativas para um microprocessador do tipo 6800. Neste caso, o teste pseudo-aleatório foi executado através de uma sequência de instruções. O teste determinístico foi executado em duas versões, ambas utilizando programas estruturados, num caso através de uma ferramenta automática, com o objectivo de testar componentes individuais do microprocessador, e no outro recorrendo a um programa desenvolvido especificamente para testar todo o conjunto de instruções.

O diagnóstico dos resultados baseia-se usualmente na detecção de discrepâncias entre os valores presentes nas saídas do SUT e os valores correctos (ou esperados). Estes podem ser obtidos a partir de um **sistema de referência** (*gold unit*), através de simulação ou recorrendo à própria especificação. Caso existam diversas versões do SUT é possível omitir a referência, comparando as saídas de dois ou mais sistemas que correspondam a implementações diferentes da mesma especificação. O diagnóstico consiste na classificação de cada caso, de acordo com as divergências entre os resultados obtidos em cada experiência e os resultados do sistema de referência. Os métodos de verificação podem ser utilizados em combinação e executados em diferentes fases do desenvolvimento e operação de um sistema.

---

<sup>6</sup> Por exemplo, o teste de todas as combinações possíveis de vectores de entrada de uma ALU de 32 bits, com 4 entradas de controlo, realizado a 1 GHz demoraria mais de 9000 anos.

Um projecto é validado através da utilização prática de um produto real (parte de uma série inicial de produtos), produzido em condições normais, instalado e em utilização pelos utilizadores finais no seu ambiente operacional. É nesta fase que poderão ser identificados os defeitos e outros problemas que possam ter escapado aos procedimentos de verificação, podendo alguns daqueles ter sido inseridos posteriormente.

O desenvolvimento de qualquer sistema computacional deverá iniciar-se por uma especificação detalhada, para que os serviços fornecidos pelo sistema possam ser completamente definidos, e os valores de desempenho e tolerâncias precisamente determinados.

As técnicas de detecção e manuseamento de falhas e erros e as suas implementações, encontram-se fortemente relacionadas com as assunções de falhas. As classes que podem efectivamente ser toleradas dependem do enfoque da tolerância a falhas que foi considerado durante o desenvolvimento, em particular na fase de especificação.

A eficiência de sistemas tolerantes a falhas requer uma métrica, de forma a que um utilizador potencial possa avaliar o seu desempenho. A métrica mais utilizada é a cobertura de falhas, definida como a percentagem (e eventualmente o tipo) das que podem ser efectivamente toleradas pelo sistema. No contexto de teste de confiabilidade, a expressão “cobertura de falhas” é também utilizada para referir a percentagem das que podem ser detectadas por uma determinada metodologia de teste, dentro do modelo<sup>7</sup> considerado. O processo de validação endereça também a eventual implementação física da tolerância a falhas, requerendo portanto a análise dos resultados obtidos nas situações em que as entradas do sistema diverjam dos valores esperados (i.e. apresentam falhas). O mecanismo de validação é normalmente também um sistema e pode necessitar de ser ele próprio validado.

Na verificação e validação de sistemas tolerantes a falhas, a cobertura que é fornecida pelos mecanismos de manuseamento tem um influência acentuada nas métricas de confiabilidade. A análise das características de tolerância de um sistema pode normalmente ser executada sobre um modelo ou através de teste experimental. Se o procedimento de teste requer que sejam inseridos valores errados como parte dos padrões de teste inseridos, então designa-se por **injecção de falhas** [Hsueh97].

---

<sup>7</sup> Um modelo de falhas consiste numa representação abstracta de um conjunto de falhas, para um determinado sistema alvo, e é explicado e maior pormenor na subsecção 2.3.3.

## **2.3 Injecção de Falhas**

### **2.3.1 Objectivos**

A verificação e validação de sistemas confiáveis requer o estudo de avarias e erros. A natureza potencialmente destrutiva das avarias e a possibilidade de longas latências para os erros, em ambiente operacional, tornam difícil a identificação da causa das avarias. Para melhor classificar e compreender possíveis avarias é preferível fazer experiências em sistemas reais, de forma a melhor avaliar e aperfeiçoar a sua confiabilidade. Esta abordagem pode ser aplicada na fase de desenvolvimento, onde podem ser utilizados modelos ou protótipos, ou na fase operacional, onde falhas podem ser deliberadamente induzidas, sem risco de danos para o equipamento e considerando intervalos de tempo adequados.

A utilização de técnicas experimentais requer o conhecimento da arquitectura e comportamento do sistema alvo e, principalmente, dos mecanismos disponíveis para garantir a confiabilidade. A introdução controlada de falhas e a monitorização dos seus efeitos, requer instrumentos e ferramentas específicas. A tecnologia e correspondente metodologia indicada para cada caso depende do sistema alvo, do objectivo, da fase de ocorrência considerada e ainda de outros factores [Hsueh97].

A selecção da fase de ocorrência em que se efectua a injecção de falhas é uma opção fundamental, sendo radicalmente diferente na fase do desenvolvimento, em que é primariamente feita sobre modelos de simulação, e na fase operacional, em que são predominantes técnicas que incidem sobre o equipamento e aplicações. Estas diferentes abordagens não são mutuamente exclusivas, pois a validação de sistemas críticos inicia-se frequentemente na fase de desenvolvimento e prossegue na fase operacional (repetindo-se algumas vezes pelo meio). Adicionalmente, verificar a confiabilidade numa fase inicial, pode permitir a detecção de falhas de projecto, com os benefícios inerentes para a qualidade final e para o custo de desenvolvimento.

Por fim, existem as questões não técnicas, como as restrições temporais e orçamentais. A maioria dos sistemas electrónicos tem de ser desenvolvida e/ou produzida dentro de um determinado orçamento e intervalo de tempo, e como os demais processos, a verificação e validação da confiabilidade tem de se manter dentro de certos limites. É frequente ajustar o esforço de validação da confiabilidade, de forma a obter uma solução adequada (não necessariamente ideal) pois, em muitos casos, um sistema marginalmente melhor requer um investimento temporal ou financeiro consideravelmente maior. Adicionalmente, por vezes o

planeamento não atribui recursos suficientes para a área da confiabilidade, frequentemente considerada “não-vital”, com resultados potencialmente catastróficos.<sup>8</sup>

### 2.3.2 Estrutura Genérica

Um ambiente de injeção de falhas requer um conjunto de componentes (equipamento e aplicações), usualmente organizado de acordo com a estrutura genérica apresentada na Figura 2.5.

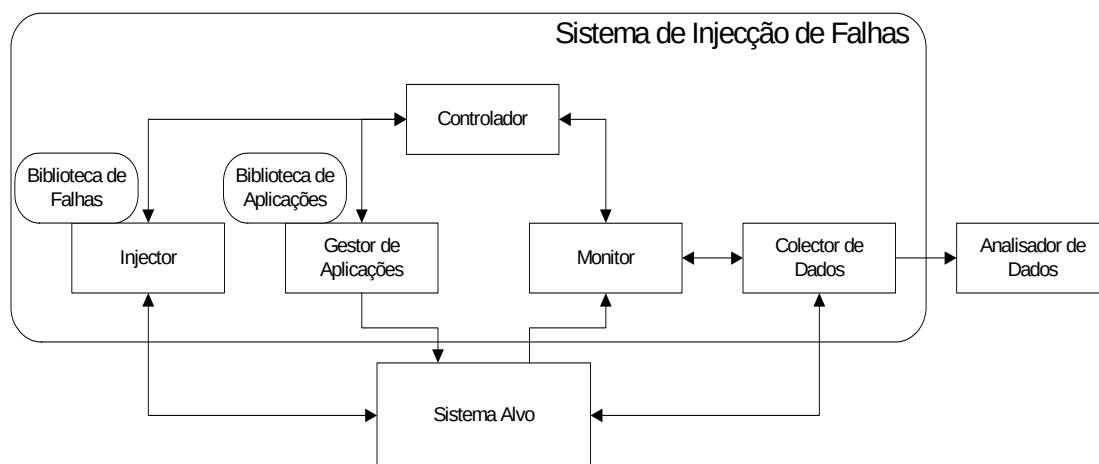


Figura 2.5 – Ambiente de Injecção de Falhas

Qualquer processo de injeção de falhas necessita de um sistema alvo, que pode existir em diversas formas, desde um modelo de um componente básico, até à realização física de um sistema computacional completo. Na maioria dos casos o sistema alvo inclui características de tolerância a falhas, mas estas também podem ser injectadas em sistemas não tolerantes, para identificar as suas vulnerabilidades e possíveis consequências.

O sistema de injeção de falhas inclui normalmente todos os blocos representados na Figura 2.5, embora alguns sejam facultativos, podendo igualmente serem necessários blocos adicionais:

- O **injector** (*injector*) é o componente utilizado para a efectiva injeção de falhas e requer um canal de comunicação com o sistema alvo. Este canal poderá ou não ser físico, tal como explicado na subsecção 2.3.4, e em certas técnicas poderá ser bidirecional, retornando algum tipo de informação. Este componente é normalmente capaz de injectar diferentes tipos de falhas, de acordo com um

<sup>8</sup> Como exemplo, o referido acidente do Ariane 5 foi provocado por uma rotina de manuseamento de erros incorrecta, reutilizada de uma versão anterior da aplicação de controlo do veículo de lançamento, a qual foi inadequadamente verificada por várias vezes [Lions96].

determinado modelo, previamente definido. As falhas que podem ser inseridas e os procedimentos e condições utilizados para a sua injeção, são armazenados na **biblioteca de falhas**.

- O **gestor de aplicações** (*workload manager*) gera os comandos e dados que serão utilizados pelo sistema alvo, durante a injeção de falhas. Aqueles podem compreender sinais, código sintético, algoritmos padrão (*benchmarks*) ou mesmo aplicações completas, armazenados na **biblioteca de aplicações** (*workload library*).
- O **monitor** (*monitor*) acompanha a evolução dos estados do sistema alvo, mantendo informação actualizada sobre os seus elementos fundamentais. Este processo pode ser executado por acesso directo, por inferência a partir de dados disponíveis na interface, ou por uma mistura de ambas as técnicas. As possibilidades de acesso e a velocidade de transferência de dados, disponibilizada pelo sistema alvo, permitem seleccionar a abordagem mais adequada.
- O **colector de dados** (*data collector*) regista a informação necessária, para caracterizar o processo de injeção de falhas e a resposta do sistema alvo.
- O **analizador de dados** (*data analyzer*) associado é utilizado quando os dados registados requerem posterior processamento, sendo na maioria dos casos esta análise executada *a posteriori* (*offline*).
- O **controlador** (*controller*) gere toda a experiência, activando os elementos necessários, quando requerido, a partir da informação por eles transmitida. Normalmente, o controlador consiste num programa a correr num sistema anfitrião, que aloja ou se encontra ligado aos restantes elementos.

O cenário mais comum é constituído por um sistema alvo e um sistema anfitrião, compreendendo o primeiro um microprocessador e a memória com dados e aplicações a executar. O sistema anfitrião é normalmente um computador externo, onde são alojadas as aplicações de injeção de falhas e os dados das experiências. Uma aplicação alvo típica é pré-carregada no sistema alvo e um dispositivo electrónico é ligado entre este e o sistema anfitrião, para actuar como injectores de falhas. A biblioteca de falhas é representada por uma série de guiões (*scripts*) relacionados com os tipos de falha e os métodos de injeção. O monitor e o colector de dados são também programas a correr no sistema anfitrião, ligados às saídas do sistema, de forma a serem capazes de determinar o seu estado e a registar a informação necessária numa base de dados. Finalmente, o analisador de dados é um outro

programa, executado após a conclusão do processo de injeção de falhas, que permite retirar conclusões a partir dos resultados registados na base de dados. Existem muitas variantes em relação a este cenário, mas a estrutura de base é comum à maioria dos casos.

### 2.3.3 Modelo de Falhas

As falhas a injectar são geralmente seleccionadas de forma a emular os efeitos esperados sobre o alvo, da mesma forma que ocorrem no seu ambiente operacional. Um modelo de falhas consiste numa representação abstracta de um conjunto de falhas que podem afectar um determinado sistema alvo [Anghel05]. Este modelo deve replicar as falhas existentes no mundo real, de forma a que os efeitos induzidos sejam o mais realistas possível [Rebaudengo02]. O universo de falhas a considerar depende dos objectivos pretendidos, do âmbito de aplicação e de outras considerações técnicas. Limitações em termos de equipamento podem afectar o modelo, impondo restrições e/ou simplificações. O tempo é também um factor a ter em conta, pois cada experiência, e inerentemente, todo o processo de injeção de falhas, devem ser planeados de forma a serem exequíveis num intervalo de tempo aceitável. Na maioria dos casos, é necessário algum tipo de compromisso, pois muitas vezes é impossível testar completamente o modelo pretendido em tempo útil [Berrojo02]. Nestas condições é frequente limitar o modelo em termos espaciais ou temporais, ou recorrer a métodos estatísticos, para tirar conclusões a partir de resultados parciais. Os modelos de falhas geralmente utilizados em sistemas computacionais utilizam falhas individuais, caracterizadas pelos quatro parâmetros presentes na Tabela 2.2.

Tabela 2.2 – Parâmetros de falha

| Parâmetro                        | Descrição                                                                                                                                                                                               | Exemplos                                                    |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|
| <b>Tipo</b><br>(Type)            | As consequências da falha para o sistema alvo ou a sua representação lógica.                                                                                                                            | Inversão de Bit<br>(Bit-flip)<br>Curto-Circuito<br>(Bridge) |
| <b>Localização</b><br>(Location) | O ponto preciso do sistema alvo onde a falha é inserida                                                                                                                                                 | Pino<br>Memória<br>Registo                                  |
| <b>Activação</b><br>(Trigger)    | A condição exacta que origina a ocorrência de uma falha. Pode ser gerada pelo próprio sistema alvo ou por um sinal externo e, em ambos os casos, pode ser dependente de um evento temporal ou espacial. | Tempo<br>Condição Lógica<br>Acção do Utilizador             |
| <b>Duração</b><br>(Duration)     | O intervalo de tempo durante o qual a falha está activa. Pode também ser representada por um padrão temporal.                                                                                           | Permanente<br>Temporária<br>Periódica                       |

Em algumas das técnicas aplicáveis a microprocessadores, a injeção de falhas é executada directamente sobre um elemento lógico (e.g. registo), correspondendo portanto à inserção directa de um erro (e não de uma falha).

A localização da falha identifica o elemento afectado, e poderá ser tão exacta quanto o pretendido (e tecnicamente possível). A acção sobre um único bit é a mais utilizada e designa-se por falha individual (*single fault*), sendo a acção simultânea sobre diversos bits designada por falha múltipla (*multiple fault*).

A condição de activação, que origina a inserção da falha, pode ser mais ou menos complexa e incluir considerações lógicas, temporais, aleatórias ou estatísticas.

Finalmente, a duração da falha é um valor temporal, que usualmente define o intervalo durante o qual a falha se encontra activa, podendo ser representada por um valor finito ou por uma função temporal, podendo mesmo não ser limitada, representando portanto uma falha permanente. É sempre necessário definir se o sistema alvo pode interferir com este valor, i.e. se a falha se mantém mesmo quando a aplicação em execução tenta alterar os dados em questão.

A injeção de falhas é normalmente estruturada em **campanhas**, que compreendem uma série de experiências onde o sistema alvo é utilizado (uma aplicação é carregada e activada) e uma falha (ou conjunto de falhas) é inserida. Uma experiência de injeção de falhas requer usualmente uma fase de inicialização (*setup phase*), onde o sistema alvo e o sistema de injeção são configurados. Antes da execução das campanhas de injeção de falhas é usual efectuar uma ou mais execuções da aplicação alvo, sem falhas, de forma a determinar o comportamento correcto do sistema, sendo estas execuções habitualmente designadas por *gold runs*. Durante todo o processo o comportamento do sistema é monitorizado e a informação relevante é registada, da forma mais completa possível, para possibilitar o estudo e avaliação dos efeitos das falhas. As campanhas de injeção são definidas tendo em conta o objectivo pretendido, o equipamento disponível e as restrições temporais.

### 2.3.4 Técnicas

As técnicas existentes de injeção de falhas podem ser classificadas em três grupos principais, nomeadamente:

- Simulação.
- Aplicações que executam no sistema alvo (*Software*).
- Injeção física.

Estas técnicas e as suas subclassificações são apresentadas na Figura 2.6.

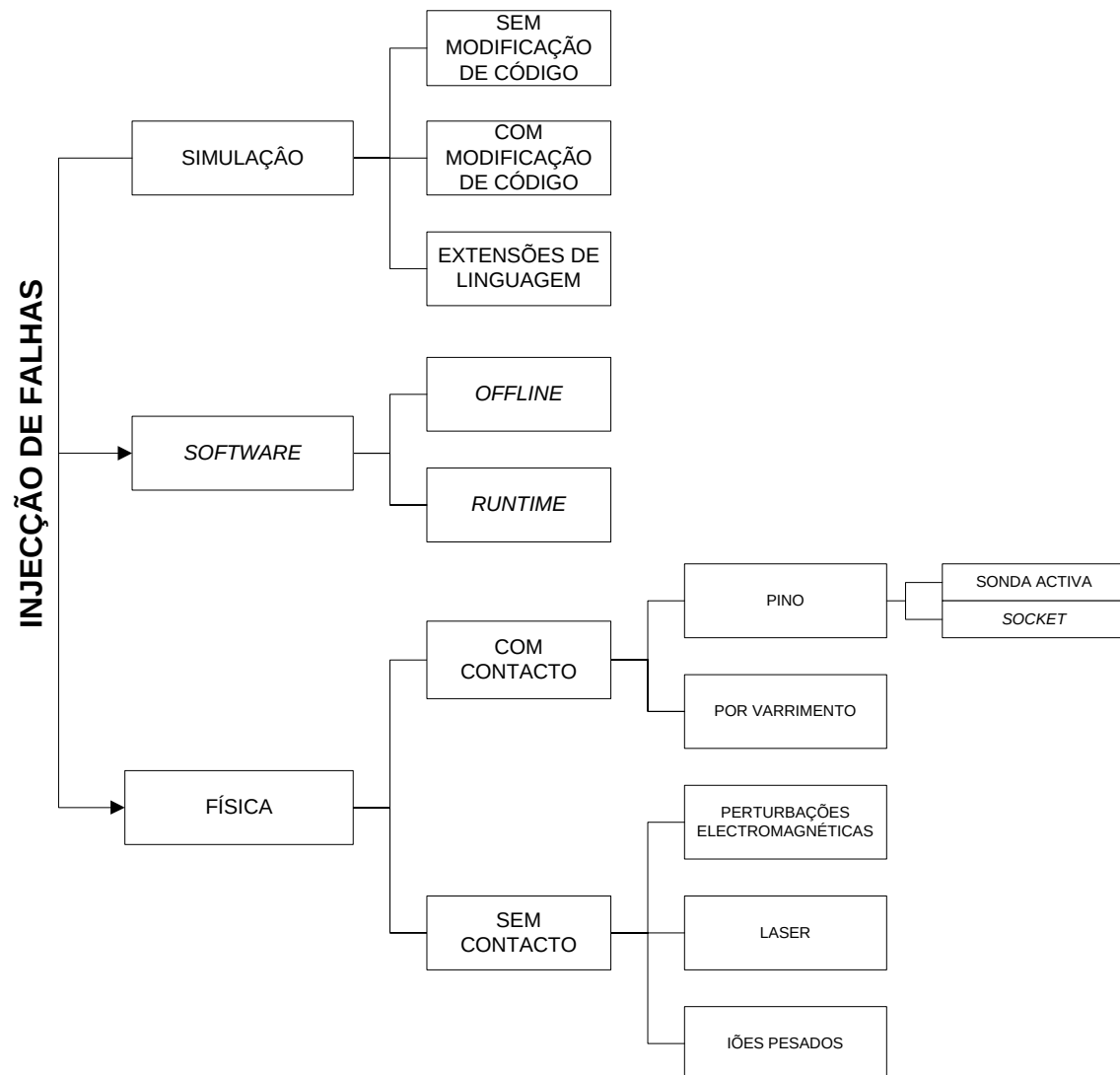


Figura 2.6 – Técnicas de Injecção de Falhas

#### 2.3.4.1 Injecção de Falhas por Simulação

A injeção de falhas por simulação é particularmente indicada para a fase inicial de um projecto, onde o sistema alvo existe apenas em modelo, que pode corresponder ao sistema computacional completo, ou apenas a uma unidade funcional [Folkesson98] [Leveugle02].

São necessários o modelo, normalmente num código de descrição de *hardware* (HDL – *Hardware Description Language*), sendo VHDL [IEEE00] e Verilog [IEEE01] os mais

usuais<sup>9</sup>, ferramentas de simulação adequadas ao caso em análise e equipamento com suficiente capacidade de processamento. Este último é de extrema importância, pois os modelos de sistemas computacionais são normalmente muito complexos e as campanhas de injeção de falhas requerem um elevado número de simulações. Cada simulação deve abranger o funcionamento do sistema até ao instante de activação e inserção da falha pretendida, compreendendo também a subsequente avaliação dos seus efeitos, revelando-se portanto potencialmente exigente em termos de necessidades de processamento e/ou tempo de execução.

O nível ao qual a simulação de falhas é executada depende do modelo disponível, variando desde o CI (caixa negra) até ao transístor. Para a injeção de falhas é necessária uma ferramenta que possa modificar os parâmetros indicados e emular os efeitos da falha, de acordo com o modelo definido. Esta ferramenta deve possuir também capacidades de sincronização temporal, para permitir que as falhas sejam inseridas no instante pretendido.

Como foi referido na subsecção 2.3.2, as campanhas de injeção de falhas seguem uma estratégia de alto nível, que resulta na geração de algum tipo de base de dados com a informação registada durante a sua execução. Esta informação terá de ser analisada e, mesmo sendo teoricamente possível efectuar esta análise em tempo real (i.e. simultaneamente com a simulação), na prática as limitações em capacidade de processamento e o volume de dados que têm de ser analisados geralmente impõe que esta análise seja efectuada *a posteriori*.

A injeção de falhas por simulação pode ser dividida em três tipos fundamentais:

- Sem modificação do modelo do alvo.
- Com modificação do modelo do alvo.
- Utilizando extensões da linguagem de modelização.

A injeção de falhas sem modificação do modelo do alvo é executada através de comandos específicos, durante a simulação ou durante a compilação do modelo.

A modificação do modelo do sistema alvo pode ser executada através de duas técnicas diferentes, usualmente designadas por sabotadores (*saboteurs*) ou mutantes (*mutants*) [Gracia01]. Os sabotadores são componentes específicos adicionados ao modelo, com o único propósito de injectar falhas no seu interior. Os mutantes são versões modificadas de

---

<sup>9</sup> Apesar de não serem classificadas como HDLs, o SystemC [IEEE05] e a UML [UML07] também podem ser considerados neste âmbito. A UML é mais indicada para descrições a um nível de abstracção mais elevado e o SystemC é geralmente utilizado para descrever componentes simples, i.e. interfaces entre o equipamento e as aplicações.

componentes previamente existentes, que permitem a injeção de falhas no seu interior ou em componentes com os quais têm ligações .

A terceira possibilidade é o recurso a extensões específicas da linguagem de modelização, principalmente utilizadas durante a compilação, de forma a efectuar a injeção de falhas pretendida. Estas extensões não são executadas ao nível do modelo e dependem da linguagem utilizada. No caso da linguagem VHDL, as técnicas recentes [Gil02] baseiam-se na adição de novos tipos de dados, de sinais que definem o comportamento das falhas e na alteração das funções de resolução do modelo VHDL. Estas técnicas requerem compiladores específicos e algoritmos de controlo para gerir as extensões de linguagem, sendo portanto de aplicação limitada.

Tabela 2.3 – Vantagens / desvantagens da injeção de falhas por simulação

| Vantagens                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Desvantagens                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>▪ Pode ser utilizada antes do primeiro protótipo estar disponível.</li> <li>▪ O sistema alvo não pode ser danificado ou destruído.</li> <li>▪ O custo da infra-estrutura de teste é relativamente baixo.</li> <li>▪ As falhas são facilmente reproduzíveis.</li> <li>▪ As latências de detecção e recuperação de erros podem ser facilmente medidas (a precisão depende da qualidade do modelo).</li> <li>▪ A controlabilidade e observabilidade pode ser muito boa (dependendo da qualidade do modelo).</li> </ul> | <ul style="list-style-type: none"> <li>▪ Os tempos de execução e os requisitos de capacidade de processamento podem ser extremamente elevados.</li> <li>▪ Os resultados obtidos dependem fortemente da qualidade do modelo utilizado.</li> <li>▪ Não é geralmente possível executar injeção de falhas em tempo real.</li> <li>▪ Problemas relacionados com atrasos temporais podem ser mascarados pelo ambiente de simulação.</li> </ul> |

A literatura disponível a este respeito permite citar diversos exemplos de injeção de falhas por simulação:

#### **VERIFY** [Sieh96][Sieh97]

O VERIFY (*VHDL-based Evaluation of Reliability by Injecting Faults efficiently*) é uma ferramenta de *software* que estende a linguagem VHDL, de forma a introduzir uma forma de descrever o comportamento dos componentes físicos, quando sujeitos a falhas. A precisão dos resultados é garantida através do uso dos mesmos modelos VHDL que são desenvolvidos durante a fase convencional de projecto do equipamento. Para demonstrar as capacidades do VERIFY, foi utilizado um modelo de um microprocessador de 32 bits (DP32), que permite ilustrar os diferentes passos da avaliação da confiabilidade.

**MEFISTO** [Jenn94][Boue98]

O MEFISTO (*Multi-level Error and Fault Injection Simulation Tool*) é uma ferramenta para a injeção de falhas simuladas em modelos VHDL, a diversos níveis de abstracção. Permite inserir falhas de diferentes tipos, i.e falhas do tipo curto-circuito, permanentes ou temporárias, falhas do tipo inversão de bit e falhas funcionais configuráveis pelo utilizador. Uma campanha de injeção de falhas é dividida em três fases, nomeadamente inicialização, simulação e análise. A fase de inicialização envolve a selecção dos objectos VHDL aos quais deve ser aplicado um modelo de falhas. Também nesta fase são definidas as experiências, seleccionando as falhas a partir de uma biblioteca pré-definida e incluindo no modelo o momento de activação. A fase de simulação inicia-se com uma execução de referência, seguida pelas experiências de injeção de falhas, executadas em diversas máquinas em paralelo. O intervalo de tempo necessário para cada simulação é consideravelmente reduzido por intermédio de *checkpointing*. O processamento dos dados é executado posteriormente à execução das campanhas de injeção de falhas.

**Ferramenta VHDL do GSTF** [Baraza00][Gil02]

O GSTF (*Grupo de Sistemas Tolerantes a Fallas*) do Instituto Politécnico de Valência desenvolveu uma ferramenta de injeção de falhas baseada na linguagem VHDL, simples, facilmente portátil e adequada para a injeção de falhas em portas lógicas, registos e CIs. A ferramenta é composta por uma série de elementos, desenvolvidos para uso com o simulador comercial *Modelsim*. Esta ferramenta pode operar através de três técnicas, nomeadamente comandos do simulador, sabotadores ou mutantes. Podem ser injectadas falhas permanentes ou temporárias e podem ser utilizadas diferentes funções de distribuição de probabilidade (Uniforme, Exponencial, *Weibull* e Gaussiana), para determinar o instante e duração de cada falha. A ferramenta permite a utilização de um variado leque de modelos de falhas, incluindo os modelos tradicionais de inversão de bit e curto-circuito.

**2.3.4.2 Injeção de Falhas por Software**

O objectivo da injeção de falhas por *software* é reproduzir ao nível lógico os erros introduzidos pela falhas físicas [Velazco05], utilizando comandos previamente disponíveis no sistema alvo. A designação “injeção de erros” seria neste caso mais precisa, mas a analogia com as restantes técnicas levou à sua designação presente, dando também origem

ao acrónimo SWIFI (*Software Induced Fault Injection*) [Han95]. Em alguns casos, esta técnica é também utilizada para reproduzir erros de projecto, não directamente relacionados com falhas físicas, como por exemplo erros de desenvolvimento.

O SWIFI permite a injeção de erros em todos os recursos acessíveis por *software*, como os registos, memórias de dados e programa, a maioria dos periféricos e alguns temporizadores. Recursos como registos especiais e alguns sinais de controlo não são usualmente acessíveis por esta técnica. Este tipo de injeção requer que o alvo seja um sistema computacional a executar uma aplicação. O injector consiste num conjunto de rotinas que são executadas em conjunto com a aplicação alvo ou com uma sua versão modificada. Em ambos os casos, o código de injeção de falhas pode ser incluído manualmente ou utilizando ferramentas automáticas [Some01]. Este tipo de ferramentas é normalmente muito específico, sendo o seu uso limitado a um determinado sistema alvo.

A inexistência de equipamento adicional para a injeção de falhas torna o SWIFI uma técnica pouco dispendiosa e portanto muito utilizada. No entanto, as capacidades de monitorização do alvo, necessárias para activação de falhas e/ou recolha de dados relacionados com os seus efeitos, estão também limitadas ao que é possível fazer por *software*. Técnicas de SWIFI requerem normalmente equipamento adicional, aplicações do tipo monitor ou a inserção de interrupções em pontos da execução, para efeitos de diagnóstico. Esta última opção não permite monitorização detalhada, enquanto que os outros métodos requerem uma sobrecarga (*overhead*) considerável, em termos de equipamento ou tempo de execução [Arlat03].

As técnicas de injeção de falhas por *software* podem ser subdivididas em *offline* e *runtime*. SWIFI *offline* consiste em modificar o código da aplicação (onde o erro deverá ser inserido) antes da sua execução. A vantagem principal é a reduzida intrusividade, em termos de atraso na execução, já que não é necessária nenhuma aplicação adicional. Por esta razão esta técnica é recomendada para sistemas de tempo real. Esta abordagem é simples, mas apresenta limitações em termos do modelo de falhas. As falhas inseridas no código alvo são permanentes, pois uma vez a aplicação em execução, o código não pode ser modificado. Adicionalmente, é difícil introduzir falhas temporárias em memória de dados, e a activação das falhas é limitada a instantes da execução do programa. As falhas *runtime* são inseridas com a aplicação a executar, sendo o código responsável pela execução armazenado separadamente do programa alvo. Como neste caso é necessária a execução de uma rotina de injeção de falhas e algum controlo adicional, é inevitável a introdução de alguma sobrecarga. É possível utilizar SWIFI em sistemas de tempo real, mas geralmente com

limitações em termos de cobertura e/ou desempenho [Rodriguez02]. A variante *runtime* oferece a possibilidade de utilizar um modelo de falhas mais abrangente, permitindo activação espacial ou maior diversidade de activação temporal. As falhas são inseridas pela corrupção do espaço de memória ocupado pela aplicação ou dados, pela inserção de interrupções / excepções, ou substituindo uma ou mais instruções. Qualquer um destes métodos pode ser activado pela execução de uma instrução ou por activação temporal, não sendo possível recorrer à activação relacionada com a manipulação dos dados.

Tabela 2.4 – Vantagens / desvantagens da injeção do SWIFI

| Vantagens                                                                                                                                                                                                                                                                 | Desvantagens                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>▪ É virtualmente impossível danificar o sistema alvo.</li> <li>▪ As falhas são facilmente reproduzíveis.</li> <li>▪ Baixo custo de desenvolvimento.</li> <li>▪ Baixo custo se não for necessário equipamento adicional.</li> </ul> | <ul style="list-style-type: none"> <li>▪ A cobertura de falhas é limitada aos recursos acessíveis por <i>software</i>.</li> <li>▪ Na maioria dos casos inserem-se erros e não falhas, não sendo possível replicar os efeitos de uma falha natural.</li> <li>▪ As ferramentas utilizadas têm um impacto considerável no comportamento do sistema alvo.</li> <li>▪ O conjunto de condições de activação das falhas é limitado (particularmente para SWIFI <i>offline</i>).</li> <li>▪ A monitorização do sistema alvo é consideravelmente difícil se não se utilizar equipamento adicional.</li> </ul> |

Exemplos de SWIFI :

#### **FIAT** [Barton90]

O FIAT (*Fault-Injection-based Automated Testing*) é capaz de emular uma considerável variedade de arquitecturas de sistemas distribuídos, fornecendo capacidades de inserir falhas e de monitorizar o comportamento do sistema. As experiências consistem em injectar três tipos de falhas, em diversas localizações, abrangendo memória de dados e de programa. Os resultados mostram que a distribuição das manifestações das falhas e latências na detecção de erros, segue uma distribuição normal para cada tipo de falha. Esta metodologia pode ser utilizada para prever as respostas a falhas, ao nível do sistema, na fase de desenvolvimento.

#### **FERRARI** [Kanawati92][Kanawati95][Ghani95]

O FERRARI (*Fault and ERROR Automatic Real-time Injection*) utiliza interrupções (*traps*) de *software* para inserir falhas na memória ou nos barramentos do CPU, e compreende

quatro componentes: o inicializador / activador, a informação do utilizador, o injector de falhas / erros e o colector / analisador de dados. O injector utiliza rotinas de inserção e manuseamento de interrupções, activadas pelo contador do programa (*program counter*) ou por um temporizador externo. Quando a interrupção é activada, a rotina de manuseamento é chamada e insere uma falha na localização pretendida, tipicamente através da modificação do conteúdo de um registo ou posição de memória.

#### **XCEPTION** [Carreira98][Maia02]

O Xception utiliza uma base de dados do tipo SQL para armazenar toda a informação relativa à experiência, permitindo automatizar procedimentos. As falhas podem ser inseridas em código de utilizador ou do sistema operativo, e é possível afinar a emulação de falhas para os processos pretendidos, em diferentes elementos de um sistema distribuído. O Xception já foi utilizado em diversos campos desde OLTP (*Online Transaction Processing Systems*) até aplicações para uso espacial, suportando sistemas alvo variados, incluindo plataformas PowerPC, Intel Pentium e SPARC.

### **2.3.4.3 Injecção Física de Falhas**

Como a designação indica, a injecção física de falhas é mais realista, no sentido em que tenta replicar falhas de forma directa, e todas as técnicas deste tipo efectuem a inserção no circuito alvo ou emulam o seu efeito ao nível lógico (erro), através de acção interna ou externa.

A injecção física de falhas pode ser com contacto (*contact*) ou sem contacto (*contactless*). A primeira utiliza uma ligação física entre o injector e o sistema alvo, utilizando directamente os terminais disponíveis, ou através de infra-estruturas específicas para o efeito. Em alternativa, a injecção pode ser executada sem contacto, utilizando campos electromagnéticos, íões pesados, radiação ou laser. Alguns autores referem uma terceira classificação referente à injecção de falhas por varrimento periférico, mas pela sua natureza (utiliza uma interface física com o alvo), esta é uma técnica com contacto.

A injecção de falhas no pino (*pin-level*) é executada utilizando uma ferramenta que controla o valor lógico presente nos pinos de um CI que faz parte do sistema alvo. As falhas podem então ser injectadas directamente nos pinos acessíveis, de forma a replicar uma falha natural. Esta técnica pode utilizar diferentes abordagens, dependendo da arquitectura do sistema alvo e do modelo de falhas pretendido. É frequente que os protótipos forneçam pinos extra (ou

outros métodos), para permitir o acesso a recursos não disponíveis no produto final. As duas técnicas mais comuns de injeção de falhas ao nível do pino, consistem no uso de sondas activas (*active probes*) e inserção de *sockets*. A primeira é utilizada para injectar directamente (i.e. forçar) um determinado valor lógico num pino de um CI, numa linha ou num conector, sem desligar nenhum componente. Deve ser utilizada com precaução pois o forçar de valores lógicos nas saídas pode provocar correntes indesejáveis (*backdriving*), o que poderá afectar o sistema alvo de forma diferente do pretendido, ou danificar alguns componentes mais sensíveis. O *socket* introduz um dispositivo especial entre um componente e o seu encaixe normal. Este dispositivo actua como uma ponte entre o componente e o resto do sistema, possibilitando a modificação dos valores que são escritos/lidos nos diversos pinos. Como é possível incluir estados de alta impedância nos pinos do *socket*, esta abordagem é potencialmente menos arriscada que a da sonda activa.

Tabela 2.5 – Vantagens / desvantagens da injeção de falhas ao nível do pino

| Vantagens                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Desvantagens                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>▪ Os resultados obtidos são representativos, pois as falhas são injectadas no sistema real.</li> <li>▪ Não se sobrecarrega o sistema alvo (e aplicações a executar).</li> <li>▪ Pode ser utilizada activação temporal ou espacial.</li> <li>▪ A injeção de falhas pode ser efectuada sem parar o sistema alvo.</li> <li>▪ Utilizando ferramentas automáticas, as campanhas de injeção de falhas podem ser executadas relativamente depressa (comparadas com as técnicas que usam simulação ou SWIFI).</li> <li>▪ As falhas podem ser reproduzidas.</li> <li>▪ As latências de detecção e recuperação de falhas podem ser medidas.</li> </ul> | <ul style="list-style-type: none"> <li>▪ A controlabilidade é limitada aos pinos disponibilizados ou a CIs inseridos em <i>sockets</i>.</li> <li>▪ A observabilidade depende da complexidade do sistema alvo e pode ser limitada, se não se tiverem seguido regras de projecto para a testabilidade.</li> <li>▪ Existe alguma possibilidade de se danificar o sistema alvo, principalmente se for utilizada a técnica da sonda activa.</li> <li>▪ A infra-estrutura necessária pode apresentar custos elevados, principalmente se for utilizada a técnica de inserção de <i>socket</i>.</li> <li>▪ A ferramenta de injeção de falhas pode interferir com o sistema alvo, principalmente a altas frequências (capacidades parasitas).</li> </ul> |

Os sistemas electrónicos modernos encontram-se frequentemente equipados com recursos de teste e depuração que fornecem acesso à maior parte dos seus elementos, os quais podem ser utilizados para outros fins, como por exemplo a injeção de falhas [Steininger03].

As infra-estruturas de teste e depuração seguem habitualmente a norma IEEE 1149.1 [IEEE90], desenvolvida para realizar testes estruturais por varrimento periférico (*boundary scan test*). Tendo sido desenvolvida entre 1985 e 1990 pelo JTAG (*Joint Test Action Group*)

[Oshana02], tornou-se também comum a sua designação por infra-estrutura JTAG. A correspondente arquitectura de teste baseia-se em células de teste incluídas na periferia dos componentes, estruturadas como registos, que podem ser acedidos (*scanned*) em série e controlados através de um mínimo de quatro pinos (entrada e saída série, controlo e relógio). Estas cadeias de varrimento permitem por vezes o acesso a recursos internos de forma pública ou privada (necessitando de informação proprietária) [Alves99]. Alguma sobrecarga temporal é inevitável, dada a natureza série das cadeias de varrimento. A utilização desta técnica requer que o sistema alvo possua algum tipo de infra-estrutura de teste e depuração, e um meio para que o injectores lhe consiga aceder (i.e. um controlador da infra-estrutura). É comum referir as técnicas de injeção que utilizam infra-estruturas de varrimento periférico por SCIFI (*Scan-ChaIn based Fault Injection*) [Folkesson98][Aidemark01].

Tabela 2.6 – Vantagens / desvantagens do SCIFI

| Vantagens                                                                                                                                                                                                                                                                                                                                                                                                                                                | Desvantagens                                                                                                                                                                                                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>▪ Os resultados obtidos são representativos pois as falhas são inseridas no sistema real.</li> <li>▪ Boa observabilidade, pois estas infra-estruturas foram desenvolvidas para funções de teste.</li> <li>▪ É virtualmente impossível danificar o sistema alvo.</li> <li>▪ Boa portabilidade, pois utiliza infra-estruturas normalizadas (IEEE 1149.1)</li> <li>▪ A falhas são facilmente replicáveis.</li> </ul> | <ul style="list-style-type: none"> <li>▪ Os recursos onde as falhas podem ser introduzidas são limitados aqueles acessíveis por varrimento periférico.</li> <li>▪ Cada acesso requer uma paragem relativamente prolongada do sistema alvo.</li> <li>▪ Esta técnica só é aplicável a sistemas equipados com infra-estruturas de teste normalizadas.</li> </ul> |

A injeção de falhas por perturbações electromagnéticas é uma técnica muito específica, utilizada principalmente para analisar a vulnerabilidade de sistemas computacionais a fenómenos físicos deste tipo. Baseia-se na utilização de impulsos electromagnéticos de diferentes durações, frequência e potência. As falhas resultam da interferência com as linhas e elementos de memória dos CIs alvo.

Tabela 2.7 – Vantagens / desvantagens da injeção de falhas por interferência magnética

| Vantagens                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Desvantagens                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>▪ Os resultados obtidos são representativos pois as falhas são inseridas no sistema real.</li> <li>▪ A intrusividade é pequena (dependente da técnica de monitorização utilizada).</li> <li>▪ A injeção de falhas pode ser executada sem parar o sistema alvo.</li> <li>▪ Com ferramentas automáticas, a execução das campanhas de injeção de falhas pode ser feita relativamente depressa.</li> <li>▪ Não requer contacto físico com o sistema alvo.</li> </ul> | <ul style="list-style-type: none"> <li>▪ Controlabilidade espacial muito limitada.</li> <li>▪ Observabilidade dependente da utilização de meios complementares.</li> <li>▪ Possibilidade relativamente alta de inserção de falhas múltiplas indesejadas.</li> <li>▪ Difícil replicar experiências.</li> <li>▪ Modelo de falhas limitado.</li> <li>▪ Custo do equipamento experimental elevado.</li> <li>▪ Latências difíceis de medir.</li> </ul> |

A injeção de falhas por iões pesados é utilizada para replicar as falhas causadas pela exposição de sistemas electrónicos a radiação. Baseia-se na utilização de uma fonte radioactiva, para bombardear o sistema alvo de forma controlada (i.e. utilizando câmaras de vácuo para eliminar o efeito do ar), e inserir falhas directamente no silício. No entanto, esta técnica não permite um controlo preciso sobre o ponto exacto onde a radiação atinge o alvo e é limitada às falhas induzidas por radiação. A técnica dos iões pesados permite inserir falhas directamente no interior do CI, o que não é possível com injeção de falhas ao nível do pino (embora possa ser feito de forma limitada com SCIFI).

Tabela 2.8 – Vantagens / desvantagens da injeção de falhas ao nível por iões pesados

| Vantagens                                                                                                                                                                                                                                                                                                                                                                                                                                  | Desvantagens                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>▪ Os resultados obtidos são representativos pois as falhas são inseridas no sistema real.</li> <li>▪ A intrusividade é pequena (dependente da técnica de monitorização utilizada).</li> <li>▪ A injeção de falhas pode ser executada sem parar o sistema alvo.</li> <li>▪ As falhas podem ser injectadas em toda a área do CI.</li> <li>▪ Não requer contacto físico com o sistema alvo.</li> </ul> | <ul style="list-style-type: none"> <li>▪ As experiências requerem um elevado tempo de preparação.</li> <li>▪ Os componentes degradam-se durante as experiências.</li> <li>▪ É praticamente impossível replicar experiências.</li> <li>▪ O modelo de falhas é limitado.</li> <li>▪ O equipamento de injeção de falhas é extremamente dispendioso.</li> <li>▪ A observabilidade depende da utilização de meios complementares.</li> <li>▪ É possível danificar o sistema alvo, particularmente se este incluir tecnologia CMOS.</li> </ul> |

A injeção de falhas por laser é relativamente similar à utilização de iões pesados, no sentido em que a falha é injectada directamente na superfície do silício. No entanto, o laser é um

dispositivo muito mais preciso, o que torna possível a inserção de falhas em localizações específicas. A injeção é efectuada submetendo o elemento onde se pretende inserir a falha a um raio laser, provocando uma alteração no estado lógico.

Tabela 2.9 – Vantagens / desvantagens da injeção de falhas por laser

| Vantagens                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Desvantagens                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>▪ Os resultados obtidos são representativos, pois as falhas são inseridas no sistema real.</li> <li>▪ A intrusividade é pequena (dependente da técnica de monitorização utilizada).</li> <li>▪ A injeção de falhas pode ser executada sem parar o sistema alvo.</li> <li>▪ Com ferramentas automáticas, a execução das campanhas de injeção de falhas pode ser feita relativamente depressa.</li> <li>▪ As falhas podem ser injectadas em toda a área do CI.</li> <li>▪ Apresenta boa controlabilidade espacial, pois o ponto de injeção pode ser determinado com precisão.</li> <li>▪ As experiências podem ser replicadas, pois o laser pode ser aplicado ao mesmo ponto repetidamente.</li> <li>▪ Não requer contacto físico com o sistema alvo.</li> </ul> | <ul style="list-style-type: none"> <li>▪ As experiências requerem um elevado tempo de preparação.</li> <li>▪ A controlabilidade temporal é baixa, devido ao tempo de preparação do laser.</li> <li>▪ O equipamento de injeção de falhas é extremamente dispendioso.</li> <li>▪ A observabilidade depende da utilização de meios complementares.</li> <li>▪ É possível danificar o sistema alvo, particularmente se este incluir tecnologia CMOS.</li> </ul> |

Exemplos de injeção de falhas física:

#### **FIST** [Karlsson89]

O FIST (*Fault Injection System for study of Transient fault effects*) utiliza técnicas com e sem contacto. Esta ferramenta utiliza injeção por iões pesados para criar falhas temporárias, em localizações aleatórias no interior do CI, por exposição a radiação, podendo injectar falhas simples ou múltiplas, do tipo inversão de bit. A fonte da radiação é montada no interior de uma câmara de vácuo, em conjunto com um sistema de dois processadores. O sistema é posicionado de forma a que apenas um deles seja submetido a radiação, sendo o processador restante utilizado como referência para diagnosticar os efeitos das falhas injectadas. Em adição à radiação, o FIST permite a injeção de perturbações da alimentação, que é efectuada colocando um transístor MOS entre a fonte de alimentação e o pino correspondente do processador, para controlar a amplitude da queda de tensão. A injeção

por iões pesados é principalmente utilizada para inserir erros nos barramentos, enquanto a perturbação da alimentação afecta principalmente os sinais de controlo.

#### **MESSALINE** [Arlat90]

Esta ferramenta utiliza sondas activas e inserção de *sockets*, para realizar a injeção de falhas nos pinos. Podem ser injectadas falhas do tipo circuito aberto, curto-circuito e falhas lógicas complexas, sendo possível controlar a duração das falhas e a sua frequência. O controlo das experiências é centralizado e pode incluir até 32 pontos de injeção em simultâneo, podendo o injectador receber retorno a partir de sinais monitorizados no sistema alvo. A cada ponto de injeção é associado um dispositivo, de forma a detectar a activação das falhas. Esta ferramenta foi utilizada numa aplicação computacional de controlo ferroviário e num sistema de comunicações distribuído (projecto ESPRIT Delta-4).

#### **MARS** [Karlsson95][Arlat03]

O MARS (*Maintainable Real-Time System*) é uma arquitectura distribuída e tolerante a falhas, validada através de injeção de falhas sem contacto, por iões pesados e campos electromagnéticos. Neste último caso a injeção de falhas é feita colocando o sistema alvo entre duas placas carregadas, ou próximo de uma sonda com carga eléctrica elevada. Condutores eléctricos são aplicados em pinos do CI alvo, de forma a actuarem como antenas, intensificando o efeito do campo electromagnético nesses pontos. Os resultados publicados fazem a comparação de três métodos (injeção de falhas por iões pesados, campo electromagnético e nos pinos), no que respeita à capacidade dos mecanismos de tolerância a falhas para lidar com as perturbações induzidas. Os resultados mostraram que as técnicas utilizadas se revelaram complementares, gerando diferentes tipos de erros. A injeção de falhas ao nível dos pinos é detectada principalmente pelos mecanismos de tolerância exteriores ao CPU, enquanto que nas restantes técnicas a detecção ocorre principalmente no seu interior, o que torna estas mais adequadas para testar mecanismos de detecção ao nível das aplicações.

#### **FIMBUL** [Folkesson97][Aidemark03]

O FIMBUL (*Fault Injection and Monitoring using Built in Logic*) é uma ferramenta que utiliza a infra-estrutura de varrimento periférico presente no microprocessador Thor. Falhas temporárias podem ser inseridas em todas as localizações acessíveis pelo registos *Boundary Scan* e *Internal Scan* do CPU Thor, por inversão de bit. Os instantes em que as falhas devem

ocorrer são relacionados com a aplicação em execução, utilizando pontos de paragem através do registo *Debug Scan*. A injeção de falhas é activada quando um determinado endereço de memória for lido ou escrito  $n$  vezes, ou quando o *Program Counter* apontou para o mesmo endereço  $n$  vezes. A selecção de quando e onde injectar as falhas pode ser feita de forma aleatória ou manual (dentro das limitações dos recursos acessíveis do Thor).

#### **LFI [Sampson98] [Moreno99]**

A injeção de falhas por laser (LFI – *Laser Fault Injection*) avalia a confiabilidade, por emulação de erros temporários, durante o uso operacional, de um sistema computacional. A técnica utilizada extrai as coordenadas dos pontos de interesse a partir de informação disponível na base de dados CAD (*Computer Aided Design*) utilizada para o seu fabrico. O laser é sincronizado com a aplicação em execução, sendo utilizado equipamento adicional para diagnóstico do sistema alvo. Uma funcionalidade de tolerância a falhas (recuperação automática de instrução na ocorrência de erros detectados), incluída num sistema para aplicações espaciais, foi validada utilizando uma metodologia baseada em LFI.

## **2.4 Resumo e Conclusões**

Os sistemas computacionais confiáveis apresentam actualmente uma forte implantação na sociedade, tornando fundamental a existência de técnicas uniformes para a sua implementação, verificação e validação. A utilização de uma terminologia consistente é também importante, de forma a garantir uma adequada documentação e expansão das soluções disponíveis. Este tipo de sistemas está em constante evolução, encontrando-se os conceitos utilizados já adequadamente uniformizados e documentados. Como na maioria dos assuntos transversais a diferentes campos da engenharia, existem algumas divergências entre termos e significados, e consequentes variações em termos de prioridades e abordagens utilizadas.

A garantia de confiabilidade é possível utilizando diferentes técnicas, não necessariamente incompatíveis, cada qual adaptada a cenários operacionais específicos. A tolerância a falhas é particularmente utilizada em sistemas computacionais críticos recentes, onde as falhas são inevitáveis e potencialmente desastrosas.

A injeção de falhas é uma solução para verificar e validar os mecanismos de tolerância, sendo também utilizável, de forma mais limitada, para a avaliação de diferentes técnicas de manuseamento, ou para a avaliação das suas consequências.

Existem diversas metodologias de injeção de falhas, com diferentes aplicações e desempenho. Variam entre soluções relativamente simples, para utilização em sistemas de baixo risco, e sistemas de avaliação de confiabilidade muito completos, utilizando simultaneamente diferentes metodologias, de forma a garantir uma cobertura de falhas adequada a sistemas críticos complexos.

Qualquer técnica de injeção recorre inevitavelmente a alguns compromissos, como por exemplo entre desempenho e intrusividade, tempo de execução e cobertura, ou custos e benefícios. Todas as técnicas apresentam vantagens e limitações, sendo cada cenário de injeção de falhas definido por uma combinação entre os requisitos da análise de confiabilidade, as características do sistema alvo e a tecnologia de injeção disponível.

*“To err is human, but for a  
real disaster you need a  
computer.”*

Autor Desconhecido

**Capítulo 1**

**Capítulo 2**

**Capítulo 3**

**Capítulo 4**

**Capítulo 5**

**Capítulo 6**

**Capítulo 7**

**Referências**

**Anexo**

# Depuração e Confiabilidade



## 3 Depuração e Confiabilidade

Este capítulo tem como objectivos apresentar as tecnologias que servem de suporte ao trabalho desenvolvido e enquadrá-lo no estado actual das infra-estruturas e normas de depuração. Numa primeira secção, explica-se em detalhe o que é e para que serve a depuração embutida, incluindo a sua evolução histórica, variantes, exemplos, funcionalidades e previsões de aperfeiçoamento. Seguem-se uma apresentação resumida dos esforços de normalização das infra-estruturas de depuração e a proposta de norma NEXUS 5001, que serviu de base ao trabalho desenvolvido. Para terminar, é feita uma análise da utilização de infra-estruturas de depuração para a injeção de falhas, recorrendo a alguns casos de estudo.

### 3.1 Depuração Embutida

#### 3.1.1 Evolução

Inicialmente, a depuração de sistemas computacionais consistia simplesmente na observação das consequências da execução de um programa específico. Apesar de não ser necessariamente um mau método, esta abordagem pode ser muito dificultada pela falta de informação útil e revelar-se extremamente morosa. Em certos casos, como por exemplo na fase inicial de desenvolvimento, pode representar a única forma de depuração possível. Na maioria das situações, existem diversas formas de melhorar este método, fazendo algum tipo de depuração “informada”. Podem ser utilizadas mensagens enviadas para o exterior em certas partes do código, utilizando um dispositivo de saída, como um monitor ou painel de cristais líquidos, ou simplesmente recorrer a díodos luminosos, que se podem ligar ou piscar em determinados instantes da execução. As limitações deste tipo de depuração são principalmente tempo e custos. É mais dispendioso empregar uma grande quantidade de tempo (e recursos) a localizar cada erro, do que melhorar a metodologia de depuração, utilizando equipamento e aplicações específicas para este fim [Berger03].

A necessidade de melhores técnicas de depuração levou ao desenvolvimento da execução passo-a-passo<sup>10</sup> (*single-stepping*) por *hardware*. Esta abordagem permite que o processador

---

<sup>10</sup> Em microprocessadores, a execução passo-a-passo consiste na execução de uma instrução (ou ciclo de relógio) de cada vez, com paragem do processador após cada execução.

execute uma única instrução e pare em seguida. Neste ponto é possível aceder a diversos sinais do sistema alvo e eventualmente verificar qual o código que foi executado. O microprocessador Intel 8041 foi um dos primeiros componentes a permitir execução passo-a-passo [Haller96], utilizando para tal um pequeno circuito externo.

O passo seguinte foi o monitor ROM (*Read Only Memory*), ou monitor de depuração, o qual consiste em *firmware*, especificamente desenvolvido para ser usado como ferramenta de depuração. Este *firmware* comunica com um depurador externo, através de um porto dedicado (e.g. porto série) ou partilhado (e.g. JTAG). O monitor ROM fornece um conjunto de comandos simples para ler e modificar posições de memória, criar e remover pontos de paragem, e executar programas. Um depurador externo com conhecimento do formato de comandos a utilizar comunica com o monitor ROM e combina as diversas funcionalidades, para cumprir requisitos de nível mais elevado, como carregamento de programas (*program upload*) e execução passo-a-passo. Alguns monitores podem permitir opções mais elaboradas como análise de código executado (*code profiling*) e pontos de paragem condicionais, entre outras.

Uma alternativa possível, subsequentemente introduzida, foi o emulador ROM, que substitui externamente a memória ROM presente nos microprocessadores. Este dispositivo é também ligado a um computador anfitrião (*host*) externo, através de uma ligação série, paralela ou mais recentemente TCP/IP. Na sua forma mais simples, o emulador ROM permite o carregamento rápido de código (substituindo a programação num dispositivo separado). As versões mais sofisticadas permitem incluir um monitor ROM, comunicando com ele através de uma ligação ao depurador externo.

Os monitores e emuladores ROM fornecem capacidades de depuração consideráveis, tendo no entanto a desvantagem de ser geralmente impossível suspender todas as funcionalidades do sistema alvo. Este problema surge devido à inclusão na maioria dos microprocessadores, particularmente nos mais pequenos, de periféricos e/ou funcionalidades impossíveis de suspender por *software*. Por exemplo, o temporizador presente no microprocessador 68HC05 da Motorola [Motorola98] corre continuamente, mesmo quando o monitor estiver a executar alguma operação. Desta forma, a depuração de problemas relacionados com tempo, atrasos ou interrupções, torna-se complicada e por vezes impossível. Um monitor ou emulador ROM típico também só permite depurar código residente em memória RAM e os pontos de paragem são introduzidos modificando o código da aplicação. A execução passo a passo é frequentemente realizada através da introdução sequencial de pontos de paragem, em diversos pontos do programa. Adicionalmente, são necessários recursos consideráveis do

sistema alvo para este tipo de depuração, incluindo memória (RAM e ROM) e um canal de comunicações próprio. Cada solução tem de ser adaptada ao sistema alvo, sendo portanto um processo complexo e moroso.

Depois dos monitores e emuladores ROM surgiu o emulador embutido (ICE – *In-Circuit Emulator*), o qual consiste numa ferramenta de depuração que substitui (ou emula) o microprocessador na placa do alvo. O ICE representa uma ponte entre o sistema alvo e um depurador por *software* a correr num sistema anfitrião, fornecendo os recursos necessários para acesso ao microprocessador. É vulgarmente aceite que a Intel inventou o ICE nos anos 70, com a primeira das suas famosas caixas azuis para o microprocessador 8080, e desde então os emuladores e microprocessadores evoluíram a par.

Um ICE liga-se ao CPU, ou substitui-o completamente, através de um cabo que direcciona todos os sinais relevantes para a unidade ICE, e que geralmente requer um elevado número de linhas e pinos de contacto. É frequente que os ICE utilizem uma versão especial do sistema alvo, designada por *bond-out*, que permite o acesso a sinais indisponíveis no encapsulamento normal, utilizando para isso um número maior de sinais de comunicação. As funcionalidades mais poderosas de um ICE incluem pontos de paragem complexos (mesmo em ROM), monitorização (*trace*) em tempo real da actividade do microprocessador, intrusividade mínima e, em certos casos, a depuração sem utilizar o equipamento alvo.

Estas funcionalidades adicionais incorrem num custo, podendo este tipo de emuladores ser extremamente dispendioso. De facto, a complexidade, a integração e o desempenho cada vez maior dos microprocessadores modernos, limitam a disponibilidade de versões *bond-out*, tornando os emuladores difíceis de projectar e ainda mais dispendiosos. Como resultado, algumas funcionalidades de depuração, que distinguiam este tipo de depurador, não são possíveis em microprocessadores mais recentes.

Os ICE também apresentam outras desvantagens. Tipicamente, um ICE representa um conjunto complexo de equipamento e aplicações, sendo consideravelmente mais dispendioso do que um monitor ou emulador ROM. A maior parte utiliza uma interface com o exterior (o sistema alvo) que não é 100% idêntica à interface do componente real. Por exemplo, o circuito oscilador utilizado para depuração encontra-se normalmente no emulador e não no sistema alvo.

Muitos dos microprocessadores mais recentes eliminaram a necessidade de utilização dos métodos referidos, através da incorporação de elementos de depuração nos seus circuitos integrados. Esta solução é designada por **depuração embutida** (OCD – *On Chip Debug*) e usualmente não necessita utilizar recursos do microprocessador ou sistema onde se insere

[Berger03]. A maioria dos fabricantes de microprocessadores designa esta tecnologia de forma diferente (e proprietária). Quanto à implementação, existem versões dedicadas para depuração e outras que, simplesmente, acrescentam funcionalidades de depuração a infra-estruturas de teste existentes.

As versões iniciais de infra-estruturas OCD consistiam basicamente em monitores de depuração escritos no microcódigo do microprocessador alvo (e.g. família CPU32 [Motorola96]). As soluções mais avançadas acrescentaram outras funcionalidades, como leitura em tempo real do *Program Counter* (microprocessador SHARC [AnalogD04]) e leitura em quase tempo real<sup>11</sup> de posições de memória (microprocessador ColdFire [Freescale06]). Uma infra-estrutura OCD básica permite carregamento de código, leitura e escrita de memória e recursos do microprocessador, pontos de paragem, execução passo a passo, reinicialização do microprocessador e acesso a informação de estado. Tipicamente, os periféricos existentes deverão ser suspensos durante a depuração.

Alguns microprocessadores enriquecem as suas infra-estruturas OCD com recursos adicionais, de forma a permitir a utilização de depuradores mais complexos (e capazes). A série de microcontroladores PowerPC 4xx [IBM98] da IBM disponibiliza uma interface de sete linhas, adicionalmente aos sinais de controlo da depuração, para monitorizar a execução dos programas. A captura em tempo real dos valores neste barramento permite a um depurador reconstruir a sequência de instruções executada. Apesar das capacidades básicas da depuração embutida serem semelhantes às de um monitor de depuração, também estão presentes algumas capacidades de um ICE.

As infra-estruturas OCD utilizam o seu próprio canal de comunicações com o interior do CPU, existindo alguns casos em que este é partilhado com uma infra-estrutura de teste, já que a depuração e o teste não são normalmente executados em simultâneo. Desta forma, os dois canais de comunicação mais utilizados actualmente para acesso a OCD são o *Test Access Port* (TAP), definido na norma IEEE 1149.1 [IEEE90], e o BDM (*Background Debug Mode*) [Motorola06]. O TAP compreende quatro linhas (mais uma opcional) para comunicação bi-direccional, permitindo o acesso aos diversos componentes por ligação série (*daisy-chain*). O BDM é uma especificação proprietária da Motorola, utilizada em vários dos seus microprocessadores. A Figura 3.1 representa uma infra-estrutura OCD típica.

---

<sup>11</sup> Neste caso “quase tempo real” significa que a leitura do conteúdo de células de memória é feita com algum atraso temporal.



### 3.1.2 Implementações e Funcionalidades

A implementação de infra-estruturas OCD, em diferentes famílias de microprocessadores, partilha algumas características comuns, que formam um conjunto fulcral de funcionalidades. Em geral, uma infra-estrutura OCD consiste numa combinação de *firmware* e *hardware* presentes no microprocessador.

Uma infra-estrutura OCD permite diferentes configurações, como por exemplo um simples monitor em micro código (e.g. Motorola CPU32), ou recursos específicos implementados em *hardware* (e.g. IBM PPC403). É também possível recorrer a funcionalidades disponíveis via código de utilizador (executado no sistema alvo), como registos de ponto de paragem (e.g. algumas implementações da família PowerPC) ou *hardware* dedicado, utilizado apenas pelo OCD, como sejam os *buffers* de instruções (também presente na família PowerPC).

A depuração via infra-estruturas OCD normalmente requer também equipamento externo, ainda que mínimo. O requisito básico é um canal de comunicações entre o sistema alvo e um sistema anfitrião, sendo frequente a existência de um depurador dedicado entre estes dois sistemas. Por exemplo, o BDM utiliza normalmente cinco pinos dedicados (ocasionalmente multiplexados com outras funções) e a família IBM 403 reutiliza os pinos do TAP, com ambas as soluções a necessitarem ainda de um sinal adicional para reinicialização. Muitos processadores digitais de sinal (DSP – *Digital Signal Processor*) utilizam uma interface baseada num porto TAP (na maioria baseados numa versão da Texas Instruments), ou numa variante do porto BDM para DSPs, designada por OnCE (*On Chip Emulation*).

Uma infra-estrutura OCD é inútil sem um sistema anfitrião, que execute as aplicações de interface com o utilizador e controlo do equipamento de depuração e do sistema alvo, gerindo os recursos e disponibilizando os dados necessários à depuração. A ligação entre o sistema anfitrião e a infra-estrutura OCD varia consideravelmente de caso para caso. Na sua versão mais simples, pode consistir numa ligação directa do porto de comunicações da infra-estrutura OCD à porta paralela de um PC comum, sendo esta solução simultaneamente económica e lenta. As opções mais complexas incluem uma ligação da infra-estrutura OCD via porta do tipo RS-232 (e.g. Cygnus), porta paralela de alto desempenho (e.g. Macraigor Systems), TCP/IP (e.g. Cygnus, Macraigor Systems), barramento ISA (e.g. Nohau), ou outras. A crescente importância da depuração originou muitas e variadas implementações de infra-estruturas OCD, sendo apresentados em seguida alguns exemplos.

**BDM (CPU16, CPU32, ColdFire) [Motorola96] [Motorola06]**

As famílias de microprocessadores CPU16 e CPU32 da Motorola estiveram entre as primeiras a incorporar infra-estruturas OCD. A abordagem seleccionada foi a de incluir um monitor ROM no microcódigo do próprio microprocessador. Originalmente, um conjunto de oito pinos foi definido para a interface, sendo mais tarde expandido para dez. As implementações mais recentes incluem suporte para múltiplos pontos de paragem, por *hardware*, e a capacidade de leitura do conteúdo de posições de memória, enquanto o microprocessador executa o código do utilizador. Actualmente, o BDM é a infra-estrutura OCD proprietária que é mais utilizada. Esta solução define uma norma de comunicações para a interface com o sistema de depuração, utilizando um conector de 10 ou 26 pinos. Um sistema anfitrião corre uma aplicação de depuração remota, de forma a comunicar e controlar a infra-estrutura OCD, utilizando a referida interface. Os comandos BDM básicos são geralmente os mesmos para as diversas famílias de microprocessadores compatíveis, existindo algumas diferenças devido a pormenores de cada arquitectura. Estas diferenças são endereçadas pelo depurador específico que controla o BDM.

Os comandos BDM têm 17 bits de extensão e são comunicados através de um canal de inserção de dados em série, presente na interface. As respostas são extraídas do sistema alvo, através de um canal de leitura de dados, também em série. Um sinal de relógio gerado pelo depurador permite sincronizar ambas as transmissões de dados. O processo em si é pouco intrusivo, desde que não seja necessário suspender o microprocessador alvo, podendo ser utilizado um sinal adicional para este efeito.

O BDM dos CPU16, CPU32 e Coldfire é bastante similar, e o conjunto de comandos é consistentemente o mesmo, desde a sua inclusão inicial em monitores ROM. A execução passo a passo pode ser realizada através do controlo do porto BDM, por *hardware*, ou utilizando pontos de paragem por *software* no fluxo de instruções.

O microprocessador alvo não é afectado pelo BDM, i.e. a sua operação não provoca nenhum tipo de excepção ou interrupção. Existe uma instrução disponível para o código de utilizador, que permite aceder aos recursos do BDM (e suspender a execução).

**BDM embutido em PowerPC (MPC5xx, MPC8xx) [Motorola06] [Motorola02b]**

Este tipo de BDM funciona de forma bastante diferente do anterior, devido à inexistência de um conjunto específico de comandos. Qualquer vector de dados inserido deverá ter 7 ou 32 bits de comprimento (sem contar os bits de controlo). Os vectores de 32 bits são inseridos num registo de instrução e os de 7 bits são utilizados para controlar o sistema de pontos de

paragem, a execução passo-a-passo e outras funcionalidades de depuração. Os comandos são interpretados como códigos de instrução PowerPC, sendo portanto possível executar qualquer instrução do seu conjunto normal. Esta solução é flexível, permitindo o acesso à maioria dos recursos, pois a infra-estrutura OCD tem as mesmas capacidades do microprocessador. Os pinos do BDM são partilhados com a interface de uma infra-estrutura de teste normalizada 1149.1.

Neste caso, o microprocessador detecta a actividade do BDM como uma excepção do CPU. Tal como acontece no caso geral, diversos eventos podem originar este tipo de excepção (*invalid opcode, address bus misalignment, non-maskable interrupt*, etc.). Para resumir a execução normal basta que o depurador insira um comando de retorno de interrupção no registo de instruções do microprocessador alvo.

### **Depuração via JTAG (PPC6xx, IBM 4xx, TI C90, SHARC) [AnalogD04]**

O termo JTAG indica os portos compatíveis com a norma IEEE 1149.1, que foi originalmente desenvolvida para realizar o teste de ligações, em cartas de circuitos impresso. No entanto, apesar de não existir nenhuma especificação que oriente o seu uso para depuração, também não existe nada que o limite, sendo este tipo de utilização encorajado na norma.

Vários microprocessadores implementam a depuração via porto JTAG. Na série 600 dos microprocessadores PowerPC, a depuração limitava-se a utilizar a cadeia de varrimento e a aceder aos recursos disponíveis de forma idêntica à utilizada para teste. A maioria dos recursos importantes, como o acesso à memória e aos registos internos, estão disponíveis algures nas longas cadeias de varrimento. A depuração torna-se assim consideravelmente morosa, pois cada acção (modificação dos dados numa posição de memória, por exemplo) requer várias operações de leitura e escrita em série, ao longo de uma longa cadeia de varrimento (e.g. centenas ou milhares de bits), apesar da operação em execução requerer apenas o acesso a uma pequena parte (e.g. 8/32 bits).

Uma alternativa é a utilização de cadeias de varrimento mais pequenas, com acesso directo aos recursos relevantes para a depuração. A utilização deste método torna possível manter a conformidade com a norma IEEE 1149.1 e várias cadeias podem ser utilizadas para teste e depuração. Esta solução foi implementada na série IBM 400 de PowerPC, bem como nos DSP SHARC da Analog Devices. Estas cadeias alternativas permitem acesso mais directo a registos especificamente utilizados para depuração, bastando normalmente duas ou três cadeias para esse fim. Nos dispositivos da IBM, o porto de depuração permite aceder a um

*buffer* para inserção de instruções, a um registo de controlo e a um registo de estado da depuração. A execução das instruções adequadas permite efectuar qualquer acção de depuração necessária. Os registos de controlo e de estado permitem realizar comandos típicos, como o controlo de execução e a execução passo a passo. O acesso à memória é realizado utilizando instruções convencionais, inseridas individualmente no registo de instrução. O comprimento destas cadeias de depuração não excede os 50 bits, sendo portanto de acesso muito mais rápido.

Algumas versões recentes acrescentam nove pinos para monitorização em tempo real. O acesso a este barramento permite a um depurador externo reconstituir a execução do código a partir do conhecimento do ponto inicial e do código fonte.

### **EJTAG (MIPS)** [Zilmer99][MIPS06]

As famílias de microprocessadores MIPS 32 e MIPS 64 utilizam um modo de depuração designado por EJTAG (Extended JTAG), suportando a monitorização em tempo real, com diferentes soluções e larguras de banda. A infra-estrutura OCD funciona de forma análoga a um monitor ROM, no sentido em que, durante a depuração, o microprocessador retira as instruções a executar de um espaço de memória separado do código de utilizador. Estas instruções são inseridas pelo depurador através da interface JTAG.

A especificação EJTAG (v2.x) destina-se à utilização em todos os dispositivos MIPS e foi proposta tendo em vista a sua evolução para norma industrial. O EJTAG reutiliza o porto JTAG e o microprocessador alvo deve disponibilizar um modo de depuração, bem como registos e instruções que o suportem. Uma das funcionalidades mais importantes é o suporte para uma excepção de alta prioridade, que coloca o microprocessador neste modo.

Outras funcionalidades incluem pontos de paragem por *hardware* (instruções e dados), que permitem suspender a execução normal e colocar o microprocessador em modo de depuração. A monitorização em tempo real é também possível, utilizando pinos adicionais, cujo número depende da frequência de operação do microprocessador.

### **OnCE (On Chip Emulation)** [Freescale05]

A interface OnCE está presente na família de DSPs da Motorola e permite o mesmo tipo de depuração disponibilizado pela interface BDM, sendo normalmente acedida através de pinos dedicados, se bem que em versões mais recentes o acesso é partilhado com um porto JTAG. A solução OnCE é mais complexa do que a BDM pois utiliza uma máquina de estados controlável através de um depurador externo.

**Intel (Xscale) [Intel05]**

A Intel acrescentou uma infra-estrutura OCD na sua família de microprocessadores XScale e optou por implementá-la de forma análoga a um monitor ROM, no sentido em que o depurador carrega uma imagem do monitor para um *buffer* de instruções específico. A partir daí, é utilizado um porto JTAG de forma similar a uma porta série, i.e como um simples canal de comunicações para comandos. Não existe nenhuma especificação para o monitor, que é feito à medida de cada infra-estrutura.

Como a família Xscale é baseada no microprocessador ARM, estão disponíveis células ETM (*Embedded Trace Macrocell*) em alguns dispositivos, que permitem a monitorização em tempo real através de um registo dedicado e a compactação do fluxo de programa executado. O conhecimento do código de compactação e o acesso ao código fonte permitem a um depurador reconstruir o percurso seguido pelo microprocessador.

Como pode ser inferido dos exemplos apresentados, as infra-estruturas OCD podem ser implementadas de formas muito diferentes, cada variante apresentando as suas vantagens e desvantagens. Na maioria dos casos as diferenças encontram-se mais no desempenho do que nas funcionalidades disponíveis, as quais são razoavelmente invariantes entre arquitecturas.

Obviamente, as funcionalidades de depuração embutida são inúteis sem *software* e *hardware* adequado para controlar o seu funcionamento e interpretar os dados obtidos. Existem diversas ferramentas para o apoio da depuração, teste e programação do sistema alvo, variando não só em termos de custo e desempenho, mas também em características como o suporte para pontos de paragem de *hardware*, a monitorização e a operação em tempo real. É frequente encontrar diferenças significativas entre ambientes de depuração indicados para o mesmo sistema alvo e utilizando a mesma interface.

Um número considerável de depuradores, actualmente disponíveis, baseiam-se em infra-estruturas OCD. As aplicações utilizadas normalmente incluem as funcionalidades clássicas de depuração, incluindo reconstrução (*disassembly*) de código fonte, visualização de registos, memória, pontos de paragem, etc. As funcionalidades mais avançadas, como pontos de paragem por *hardware*, monitorização, execução de guiões (*scripts*), depuração multi-tarefa e análise de execução (*profiling*), estão disponíveis apenas em alguns casos. A depuração por infra-estrutura OCD é disponibilizada por um grande número de fabricantes e adaptável a vários sistemas alvo. O suporte para sistemas operativos específicos é também possível, incluindo sistemas de tempo real. Existem muitas variantes, desde ambientes de desenvolvimento e depuração completos e extremamente dispendiosos, passando por

ambientes muito específicos, dedicados e personalizados para um determinado sistema alvo, até ferramentas gratuitas para aplicações e equipamento mais convencional.

Uma infra-estrutura OCD também pode ser utilizada para o teste de CIs e no desenvolvimento de aplicações, sendo a programação de memórias *flash* (ou outras) também uma utilização frequente. Hoje em dia, os componentes com memória deste tipo são disponibilizados em encapsulamento sofisticado, incompatível com programadores convencionais, necessitando por vezes de incorporação na placa, antes de serem programados. A utilização de infra-estruturas OCD permite a verificação das ligações e a programação da memória, com esta já inserida no circuito, inclusive em SoCs (*System-on-a-chip*).

Outra possível utilização de infra-estruturas OCD é o teste funcional na linha de produção, sendo a sua interface utilizada para executar um teste completo durante a montagem, incluindo inicialização, calibração e programação. Aplicações específicas podem ser utilizadas para um teste ao espaço de memória e de execução de instruções. A calibração de dispositivos presentes num CI pode ser automatizada e executada imediatamente a seguir à sua programação. Qualquer recurso presente na placa, que possa ser acedido pelo microprocessador, estará também disponível através da interface OCD, permitindo vários tipos de teste, complementares ao teste por varrimento periférico normal.

À medida que o uso de infra-estruturas OCD se generaliza, a sua utilização em áreas menos evidentes também se intensifica. Alguns campos de aplicação têm requisitos especiais relativos ao equipamento, que podem ser satisfeitos utilizando infra-estruturas OCD. Por exemplo, a indústria automóvel utiliza unidades de controlo de motor (ECU – *Engine Control Unit*) baseadas em microprocessadores, que requerem calibração via dados do veículo e algoritmos específicos. Este processo requer a leitura de valores internos da ECU e a alteração de parâmetros do algoritmo, com este em execução. Apesar da calibração ser tradicionalmente executada utilizando ferramentas de *software*, as soluções baseadas em *hardware* (i.e. OCD) são menos intrusivas e mais adequadas, quer em capacidades quer em desempenho.

### 3.1.3 Esforços de Normalização

A depuração de sistemas computacionais é um requisito importante, cuja abordagem tem sido feita de forma segmentada pelos vários fabricantes de microprocessadores. Até recentemente, as infra-estruturas OCD podiam apresentar variações consideráveis mesmo

entre CIs do mesmo fabricante. Este facto impõe a necessidade de utilizar equipamento diferente para cada sistema alvo, com o inerente aumento dos custos de produção e desenvolvimento. Actualmente, é já frequente a utilização do mesmo ambiente para a depuração de diferentes sistemas alvo, partilhando a aplicação executada no sistema anfitrião e parte do equipamento utilizado. No entanto, é também frequente que sejam necessários depuradores ou adaptadores específicos para cada sistema, em consequência da utilização de interfaces diferentes e implementações distintas de funcionalidades de depuração semelhantes.

A crescente preocupação com a redução dos custos de desenvolvimento, depuração e teste de sistema electrónicos, tem originado o aparecimento de grupos de trabalho com o objectivo de normalizar os processos de desenvolvimento e fabrico de componentes electrónicos. Os alvos deste esforço de normalização incluem o projecto de componentes, as interfaces, os barramentos internos e os mecanismos de depuração.

Nesta área específica ainda não existe nenhuma norma formalmente aceite, tendo diversos grupos iniciado trabalhos que incluem sugestões concretas para a normalização parcial de infra-estruturas OCD. Em geral, as funcionalidades implementadas são propostas de forma mais ou menos aberta, os detalhes de funcionamento não são especificados, mas a interface e parâmetros de comunicação com o exterior (i.e. depurador) são sugeridos ou mesmo explicitamente definidos. Para além da proposta de norma NEXUS, referida no capítulo anterior e descrita em detalhe mais adiante, existem algumas outras que são resumidas em seguida.

### **SPIRIT Debug WG**

O consórcio SPIRIT [SPIRIT07a] foi anunciado na DAC2003 e apresentou como objectivo a criação de uma norma aberta para o processo de desenvolvimento de SoCs (*System-on-a-Chip*), desde os componentes até aos CIs, permitindo a utilização de núcleos IP (*Intellectual Property*) de fontes variadas, bem como a escolha livre das ferramentas a utilizar. O enfoque assenta na normalização da integração de ferramentas e IPs, de modo a permitir uma reutilização e automatização eficientes. O projecto proposto visa a criação de uma norma IEEE, baseada num esquema proposto pelo consórcio, independente do processo de projecto específico.

Em 2006 o consórcio transformou-se numa organização sem fins lucrativos e anunciou os objectivos futuros, em que se inclui a criação de um grupo de trabalho orientado a problemas de depuração e a publicação do conjunto oficial de especificações de IPs e interfaces, que se

encontra actualmente em revisão pelo IEEE Working Group P1685 [SPIRIT07b] [SPIRIT07c].

Os objectivos do grupo de trabalho para depuração incluem incentivar os fabricantes de ferramentas de depuração a aderir ao consórcio, definir requisitos para a descrição de componentes de *hardware* que devam ser submetidos a depuração, identificar extensões, clarificações e regras semânticas necessárias para que a especificação do SPIRIT atinja os requisitos adicionais, e demonstrar que os IPs, em conformidade com a especificação, podem ser gerados por ferramentas de desenvolvimento e utilizados por ferramentas de depuração distintas. Este grupo deverá propor extensões à especificação, que permitam incluir a informação adicional requerida por ferramentas de depuração, a descrição do acesso pelo depurador (e.g. JTAG) e a identificação de arquitecturas e versões do microprocessador alvo.

### **IEEE P1687 IJTAG**

A iniciativa IJTAG (*Internal JTAG*) [Rearick05] resultou de uma reunião do IEEE *Test Technology Standards Group* na ITC (*International Test Conference*) de 2004 com o objectivo de normalizar o acesso e implementação de infra-estruturas de teste interno, existentes em muitos microprocessadores recentes. A proposta IEEE P1687 [IEEE07a], que originou da iniciativa IJTAG, tem por âmbito o desenvolvimento de uma metodologia de acesso a funcionalidades de teste e depuração (mas não as funcionalidades propriamente ditas) através do TAP e sinais adicionais, eventualmente necessários. Os elementos da metodologia incluem uma linguagem de descrição para as características das funcionalidades e para o seu acesso, bem como requisitos para a respectiva interface.

Foram consideradas desde o início duas abordagens, baseadas na utilização do TAP para o acesso a funcionalidades de teste interno e, inerentemente, de depuração. A primeira abordagem visa normalizar o teste interno, definindo explicitamente como objectivos, descrever o tipo e protocolo de acesso às referidas funcionalidades, e permitir um acesso de elevada largura de banda às cadeias de varrimento. Não se pretende normalizar as funcionalidades propriamente ditas, nem redefinir a operação do TAP. A segunda abordagem é mais ambiciosa e endereça os problemas de teste interno via TAP, pretendendo assegurar o acesso a cadeias de varrimento em alta velocidade, melhorar a largura de banda através de múltiplos portos de acesso e modificar o desenho das células, de forma a permitir funcionalidades de teste acrescidas. Foi considerado que estas propostas não eram mutuamente exclusivas e a sua inclusão está presentemente em estudo como extensão à norma IEEE 1149.1, de forma independente ou em conjunto.

### **IEEE P1149.7 cJTAG**

O grupo MIPI (*Mobile Industry Processor Interface*) [MIPI07], mais especificamente o seu subgrupo designado como *Test and Debug Working Group*, está a desenvolver uma interface de depuração designada com cJTAG (*compact JTAG*) [Oshana06], construída sobre a norma IEEE 1149.1. O objectivo do cJTAG, proposto para norma como IEEE P1149.7, é fornecer melhores funcionalidades de teste e depuração, mantendo a compatibilidade com a IEEE 1149.1 e criando um super-conjunto da interface especificada nesta última. Desta forma, são mantidos os investimentos na norma original, sendo possível estender as ferramentas de depuração a sistemas alvo existentes, recorrendo a adaptadores para a interface. A proposta inclui mecanismos de monitorização de informação, variantes dos protocolos de comunicação, e prevê a utilização de múltiplos controladores TAP. O IEEE-SA (IEEE *Standards Association*) aprovou o projecto em Março de 2006, com conclusão prevista até final de 2010.

Adicionalmente, outras iniciativas [OCP-IP07] [Multicore07] [SPRINT07], com objectivos na área da normalização de componentes IP e suas interligações, têm criado grupos de trabalho independentes, para estudar a inclusão de requisitos e normas, destinados a especificar funcionalidades e infra-estruturas de depuração. Estes esforços têm decorrido com alguma interacção, visto diversas empresas da área fazerem parte de diferentes grupos, mas as múltiplas orientações e objectivos têm gerado soluções muito diversas e de difícil compatibilização. A curto prazo, poderão surgir componentes em conformidade com uma das propostas de norma, mas dificilmente existirão componentes compatíveis com diferentes normas, enquanto não for feito um esforço de uniformização e/ou classificação. O mais provável é que várias áreas tecnológicas adoptem diversas normas na área do projecto e interligação de componentes, podendo estas incluir uma componente de depuração.

Para o trabalho desenvolvido, optou-se por escolher uma alternativa já numa fase de maturidade razoável (à data de início dos trabalhos) e fortemente orientada para infra-estruturas de depuração, sendo a escolha natural a proposta de norma NEXUS 5001.

### **3.2 A Proposta de Norma NEXUS 5001**

Como tecnologia, o principal problema com a depuração embutida é a falta de um conjunto consistente de funcionalidades e de uma interface de comunicação de dados, comum a diversas arquitecturas de microprocessadores. Mesmo considerando que alguns sistemas

partilham portos JTAG ou BDM para a ligação física, os modos de funcionamento e as capacidades das infra-estruturas OCD variam consideravelmente. Apesar de a utilização de ferramentas de depuração comuns, com funcionalidades e interface normalizadas apresentar óbvias vantagens, a implantação de uma norma nesta área é dificultada pela diversidade de arquitecturas e limitações comerciais.

### 3.2.1 Historial

O consórcio GEPDIS (*Global Embedded Processor Debug Interface Standard*) foi formado em Abril de 1998 para definir e desenvolver uma interface de depuração para aplicações de controlo embutido. A 23 de Setembro de 1999 o consórcio escolheu o IEEE-ISTO (*IEEE Industry Standards and Technology Organization*) como o fórum operacional e legal para a continuação do seu trabalho. Durante a transição, o grupo também mudou a sua designação para NEXUS 5001 Forum™, de forma a reflectir a submissão da versão 1.0 da sua proposta de norma [IEEE99] ao IEEE-ISTO para publicação, distribuição e futuro desenvolvimento, como “*IEEE-ISTO 5001™ - 1999, The NEXUS 5001 Forum™ Standard for a Global Embedded Processor Debug Interface*”.

Após a apresentação de uma prova de conceito (*proof-of-concept*) em 2000 [Vetterli00], o microprocessador Motorola MPC565 “Spanish Oak” [Motorola02a], desenvolvido para uso em equipamento de controlo de motorizações (*powertrain*), tornou-se nesse mesmo ano o primeiro dispositivo a incorporar uma infra-estrutura OCD e o correspondente porto de acesso, de acordo com a proposta de norma NEXUS. Este processador é comercializado em diferentes versões e as aplicações propostas incluem também equipamento médico, instrumentação de aviação, controladores genéricos e outros [Freescale07]. Presentemente, pelo menos três fabricantes de equipamento [Lauterbach07] [Ashling07] [iSystem07] de depuração fornecem uma solução que utiliza acesso via porto NEXUS, para depuração e monitorização.

Em 2003 o Fórum NEXUS 5001 actualizou a sua especificação para a versão 2.0 [IEEE03]. As principais alterações consistiram na clarificação de alguns conceitos e na actualização das especificações dos conectores físicos. Os membros actuais do Fórum incluem as seguintes empresas: Alphamosaic, Analog Devices, Ashling Microsystems, Delphi, dSPACE, ETAS, First Silicon Solutions, Ford, General Motors, Hitex, IAR Systems, Infineon Technologies, iSystem, Lauterbach, Motorola, National Semiconductor, NEC Electronics Corporation,

Nohau, Renesas Technology, Siroyan, STMicroelectronics, Tensilica, Texas Instruments e Wind River.

### 3.2.2 Motivação

Os avanços na tecnologia de semicondutores e o projecto de sistemas electrónicos aumentam a utilização de microprocessadores embutidos de alto desempenho. O seu uso de forma eficiente requer ferramentas de desenvolvimento de *software* e *hardware*, que possam aceder facilmente a áreas críticas da arquitectura. A falta de uma norma unificadora para as diferentes infra-estruturas OCD existentes dificulta esta acessibilidade, impedindo que os fabricantes de ferramentas de depuração possam aplicar uma solução com funcionalidades consistentes, para uma gama abrangente de sistemas alvo. Em última análise, cada caso requer uma solução adaptada às necessidades e características de cada ferramenta e interface. O IEEE-ISTO é uma organização sem fins lucrativos, que fornece um fórum operacional e serviços de apoio a corporações industriais. O fórum está disponível, não só para o desenvolvimento de normas, mas também para auxiliar as actividades de apoio à implementação e aceitação das normas no mercado. A organização é afiliada do IEEE e da IEEE-SA (*IEEE Standards Association*) [IEEE07b]. Um dos principais objectivos do consórcio NEXUS é o apoio aos fabricantes de ferramentas de depuração, de forma a permitir o desenvolvimento de um conjunto de soluções que possam ser aplicadas em diferentes microprocessadores embutidos. Tendo sido previamente identificado que as infra-estruturas e ferramentas de depuração actualmente disponíveis cumprem adequadamente as necessidades de depuração estática, foi decidido concentrar esforços em características de depuração dinâmica<sup>12</sup>.

Na sequência desta decisão, foi definido como objectivo do consórcio fornecer um percurso de migração para um conjunto normalizado de funcionalidades a incluir em infra-estruturas de depuração, eficiente em termos de custos, mas simultaneamente poderoso nas suas capacidades. Sabendo que mais de 70% dos fabricantes de microprocessadores embutidos inclui nos seus componentes uma interface IEEE 1149.1, para teste ou depuração, o consórcio decidiu incluir esta interface como opção.

---

<sup>12</sup> A proposta NEXUS 5001 identifica como funcionalidades “dinâmicas” aquelas que podem ser executadas quando o microprocessador está a correr o código da aplicação. Existe uma distinção subtil em relação a funcionalidades de tempo real, no sentido em que o acesso a recursos exclusivos da infra-estrutura OCD (i.e. pontos de paragem) não é usualmente associado à expressão “tempo real”.

As necessidades para efeitos de depuração foram identificadas da seguinte forma, todas elas limitadas a provocar um impacto aceitável<sup>13</sup> no sistema alvo:

- Aceder à monitorização de instruções.
- Recuperar a informação relativa ao fluxo de dados, pelo interior do sistema.
- Verificar se as aplicações em execução cumprem os requisitos de desempenho.

Para controlo de execução as necessidades básicas identificadas foram:

- Ler e escrever dados em registos internos e memória, com o processador parado.
- Suportar funcionalidades de ponto de paragem / observação, em depuradores, por *hardware* ou *software*, dependendo da arquitectura do sistema alvo.

O objectivo final é uma especificação de uso geral, que enderece os desafios rigorosos das interfaces OCD. Para permitir a melhor universalidade da solução, a proposta NEXUS prevê não só o uso de uma interface IEEE 1149.1, mas também um porto auxiliar extensível (AUX), bem como os protocolos de comunicação e o necessário conjunto de funcionalidades. Os dois portos de acesso podem ser utilizados separadamente ou em conjunto.

### 3.2.3 Classificação e Conformidade

As capacidades das infra-estruturas compatíveis com a proposta NEXUS são classificadas de duas formas, nomeadamente em termos das funcionalidades suportadas e do desempenho do porto de comunicações, sendo esta última dividida em capacidades de *upload* e *download*.

Para acomodar diversos requisitos de desenvolvimento, a proposta NEXUS classifica as capacidades estáticas e dinâmicas de acordo com quatro classes de conformidade. Os microprocessadores são designados em conformidade com as classes 1 a 4 ou com subclasses previamente aprovadas (não definidas no momento em que este documento é escrito). Cada nível apresenta uma complexidade maior em termos de funcionalidades de depuração disponibilizadas.

- A classe 1 inclui funcionalidades de depuração estática normalmente implementadas em infra-estruturas OCD actuais e não requer modificações consideráveis aos blocos de depuração existentes.

---

<sup>13</sup> O significado exacto de “aceitável” não é explicitamente indicado na proposta, sendo referido que será dependente do contexto específico de cada microprocessador e da sua utilização operacional.

- A classe 2 endereça requisitos fundamentais para depuração dinâmica, com ênfase na monitorização de execução.
- A classe 3 acrescenta a monitorização de acesso a dados e o acesso em tempo real a registos e memória.
- A classe 4 acrescenta capacidades de acesso a memória virtual, sendo dirigida a futuras arquitecturas de microprocessadores.

Cada classe de conformidade requer um protocolo específico de comunicação com um sistema anfitrião (i.e. computador externo). Existe também um número mínimo de registos de depuração em cada classe, bem como o número de pinos necessário para cumprir os requisitos de desempenho. A Figura 3.2 apresenta as funcionalidades mais importantes fornecidas por cada classe de conformidade, as quais são explicadas em maior detalhe na subsecção 3.2.4.

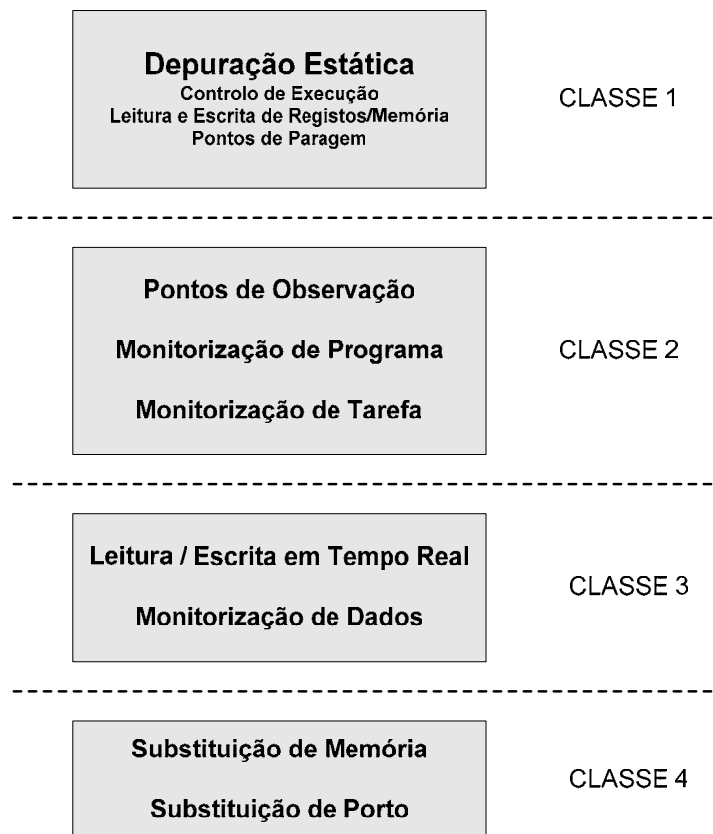


Figura 3.2 – Principais Funcionalidades NEXUS por classe

É importante notar que cada classe de conformidade suporta todas as funcionalidades incluídas nas classes precedentes e que todas as funcionalidades previstas na proposta NEXUS são incluídas em pelo menos uma das classes.

O desempenho é classificado como *full-duplex* ou *half-duplex* e incluindo a largura de banda, medida em megabits por segundo (Mbps) para *downloads* e *uploads*, de e para o microprocessador.

A classificação de uma infra-estrutura NEXUS deverá então ser feita de acordo com:

- A classe de conformidade 1, 2, 3 ou 4.
- A taxa máxima de *download* (Mbps/s)
- A taxa máxima de *upload* (Mbps/s)
- *Full-duplex* ou *half-duplex*

Para endereçar requisitos mais específicos, poderão ser definidas subclasses de conformidade a pedido, que terão de ser aprovadas por uma organização de normalização adequada. Uma listagem das funcionalidades presentes em cada classe de conformidade é apresentada na Tabela 3.1 e na Tabela 3.2. A letra F indica as implementadas de acordo com especificação do fabricante do dispositivo, a letra N as definidas na norma, a letra M as que utilizam especificamente as mensagens definidas na norma, e a letra O as de inclusão opcional.

Tabela 3.1 – Funcionalidades estáticas por classe de conformidade

| Funcionalidade                                                                              | Classe 1 | Classe 2 | Classe 3 | Classe 4 |
|---------------------------------------------------------------------------------------------|----------|----------|----------|----------|
| Escrita/Leitura de registos em modo de depuração                                            | F        | F        | F        | F        |
| Escrita/Leitura de memória em modo de depuração                                             | N        | N        | N        | N        |
| Entrada em modo de depuração a partir de reinicialização                                    | N        | N        | N        | N        |
| Saída de modo de depuração para modo normal                                                 | N        | N        | N        | N        |
| Execução de instruções passo-a-passo                                                        | N        | N        | N        | N        |
| Paragem de execução em ponto de paragem por instrução/dados (mínimo de 2 pontos de paragem) | N        | N        | N        | N        |

Tabela 3.2 – Funcionalidades dinâmicas por classe de conformidade

| Funcionalidade                                                                      | Classe 1 | Classe 2 | Classe 3 | Classe 4 |
|-------------------------------------------------------------------------------------|----------|----------|----------|----------|
| Inserção de ponto de paragem/observação                                             | N        | N        | N        | N        |
| Identificação do dispositivo                                                        | N        | N e M    | N e M    | N e M    |
| Sinalização da ocorrência de ponto de observação                                    | M        | M        | M        | M        |
| Monitorização de fluxo de programa em tempo real                                    | -        | M        | M        | M        |
| Monitorização de acesso a dados para escrita, em tempo real                         | -        | -        | M        | M        |
| Leitura/Escrita de posições de memória em tempo real                                | -        | -        | N e M    | N e M    |
| Execução de instruções a partir de porto NEXUS                                      | -        | -        | -        | M        |
| Capacidade de iniciar monitorização na ocorrência de pontos de observação           | -        | -        | -        | N        |
| Capacidade de iniciar substituição de memória na ocorrência de pontos de observação | -        | -        | -        | O        |
| Monitorização de acesso a dados para leitura, em tempo real                         | -        | -        | O        | O        |
| Substituição de porto                                                               | -        | O        | O        | O        |
| Transmissão de valores de dados para aquisição                                      | -        | -        | O        | O        |

A taxa de transferência de dados para *download* deverá ser interpretada com a largura de banda nominal requerida na interface de entrada pelo dispositivo. Reciprocamente, a taxa de transferência para *upload* deverá ser interpretada como a largura de banda nominal requerida na interface de saída do dispositivo. Os requisitos de largura de banda são tipicamente determinados pelas funcionalidades em tempo real e terão impacto directo no número de pinos necessários no porto AUX e/ou na frequência de relógio.

Os portos AUX ou IEEE 1149.1 são utilizados de forma a cumprir os requisitos de largura de banda, cujo cálculo deverá considerar a dimensão dos blocos de dados a ler/escrever, bem como a frequência das respectivas operações. Não existe uma estrutura obrigatória de registos na proposta NEXUS, mas são incluídas alguma recomendações a este respeito, cuja utilização é opcional.

### 3.2.4 Conjunto de Funcionalidades

O conjunto de funcionalidades definido na versão 2.0 da proposta de norma NEXUS 5001 foi baseado nas implementações recentes de infra-estruturas OCD, tendo por objectivo desenvolver o melhor conjunto possível, minimizando o número de pinos necessários e a área de silício utilizada. Simultaneamente, a proposta pretende-se independente das arquitecturas de microprocessador utilizadas e capaz de suportar dispositivos multi-processador. A sua descrição pode ser feita da seguinte forma:

**Controlo de execução** (*Run Control*) – Funcionalidade elementar comum a todas as implementações de depuração embutida. Permite a uma ferramenta de depuração arrancar e parar o microprocessador, ler e modificar registos, e executar instruções passo-a-passo.

**Acesso à Memória** (*Memory Access*) – A proposta NEXUS suporta acesso à memória com o microprocessador em funcionamento (*on-the-fly*), sem afectar as instruções em execução. O acesso em tempo real é uma funcionalidade útil para a depuração em geral e fundamental para sistemas deste tipo, particularmente em cenários experimentais onde não se pode parar o sistema alvo.

**Pontos de paragem / observação** (*Breakpoints / Watchpoints*) – A proposta NEXUS suporta pontos de paragem que permitem que o programa em execução seja suspenso num determinado evento, que pode corresponder à execução de uma instrução num endereço

específico ou ao acesso a dados (leitura ou escrita) presentes num determinado endereço e/ou com um determinado valor (e.g. suspender quando o valor 0x55 for escrito no endereço 0x123456). A suspensão é feita sem alterar os dados, em todos os elementos do microprocessador, de forma a que estes possam ser acedidos durante a paragem e a execução possa ser retomada exactamente no mesmo ponto e nas mesmas condições. Os pontos de observação funcionam de forma análoga, podendo inclusivamente partilhar a lógica de detecção de eventos. A diferença consiste em que neste caso o processador não suspende a execução, mas envia uma mensagem para a ferramenta de depuração.

**Monitorização de Programa** (*Program Trace*) – A proposta NEXUS usa a técnica designada por *Branch-Trace* para comprimir a informação que é necessária monitorizar a execução do programa. Uma mensagem (BTM – *Branch Trace Message*), com informação relativa ao código executado, é enviada pelo porto NEXUS na ocorrência de instruções de salto (*branch*) ou excepção (*exception*), incluindo o número de instruções sequenciais executadas desde a última mensagem. Uma ferramenta externa poderá então reconstruir o código executado, a partir de uma imagem local do conteúdo da memória (ou conhecimento do código fonte).

**Monitorização de Dados** (*Data Trace*) – Esta funcionalidade permite à ferramenta de depuração acompanhar em tempo real os acessos pelo microprocessador a posições de memória. Esta monitorização é efectuada através do envio de mensagens específicas (DTM – *Data Trace Message*) e pode ser filtrada, de forma a limitar a área e os tipos de acesso que devem ser monitorizados.

**Monitorização de Tarefa** (*Ownership Trace*) – A funcionalidade de monitorização de processo permite a um sistema operativo de tempo real (*RTOS – Real Time Operating System*) identificar a tarefa ou processo em execução e enviar essa informação à ferramenta de depuração. O RTOS escreve num registo próprio sempre que mudar de tarefa, o que força o envio de uma mensagem (OTM – *Ownership Trace Message*) pelo porto AUX. Esta mensagem envia os dados de identificação da nova tarefa à ferramenta de depuração.

**Substituição de Memória e Substituição de Porto** (*Memory Substitution and Port Replacement*) – Esta funcionalidade permite que o acesso do microprocessador à memória ou portos internos seja efectuada através do porto AUX. Desta forma é possível que o

microprocessador aceda a dados ou instruções em recursos externos e que emule a funcionalidade permite emular a funcionalidade dos portos de comunicação presentes no dispositivo, através do porto AUX.

**Aquisição de Dados (*Data Acquisition*)** – Esta funcionalidade foi incluída para suportar prototipagem rápida, proporcionando uma elevada taxa de transferência de dados através do porto AUX. Disponibiliza um protocolo de comunicação mais eficiente do aquele que é normalmente utilizado.

### 3.2.5 Interface Física e Lógica

Em termos físicos, a proposta NEXUS define um conjunto normalizado de conectores para a interface entre a ferramenta de depuração e o sistema alvo. Em termos lógicos, os dados são transmitidos utilizando um protocolo baseado em pacotes, que pode ser adaptado para um porto IEEE 1149.1, ou utilizar o porto AUX, de forma a suportar *full-duplex* e maior largura de banda. Os microprocessadores compatíveis com a classe 1 da proposta NEXUS devem incluir um porto IEEE 1149.1 para o acesso às funcionalidades mínimas previstas para esta classe. Microprocessadores em conformidade com as classes 2, 3 ou 4 devem incluir uma interface NEXUS, de acordo com a norma, para visibilidade externa (saídas do microprocessador), como requerido para acesso às funcionalidades mínimas, previstas para a estas classes. Adicionalmente, estes microprocessadores devem proporcionar acesso (entradas do microprocessador) às funcionalidades de cada classe através de um porto IEEE1149.1 ou AUX.

No caso da opção por um porto IEEE 1149.1, a implementação da interface requer apenas três pinos auxiliares. A classificação de desempenho será necessariamente mínima e apenas cumprirá os requisitos em termos de largura de banda para aplicações de baixo desempenho. Se for necessária a transferência mais rápida de dados, deverá ser utilizado um porto AUX.

A proposta NEXUS permite ajustar o desempenho de duas formas distintas - aumentando o número de pinos no porto AUX de forma a permitir transferir mais bits por impulso de relógio ou aumentando a frequência de relógio para transmitir mais bits por unidade de tempo. A Tabela 3.3 apresenta o número de pinos proposto na norma, para cada classe de conformidade.

Tabela 3.3 – Número de pinos de dados proposto na norma NEXUS (32 bits)

| Classe de conformidade | Tipo de porto | Número de pinos no barramento de dados |   |   |   |    |
|------------------------|---------------|----------------------------------------|---|---|---|----|
|                        |               | 1                                      | 2 | 4 | 8 | 16 |
| 2                      | Entrada       | X                                      | X |   |   |    |
| 2                      | Saída         | X                                      | X |   |   |    |
| 3 ou 4                 | Entrada       | X                                      | X | X |   |    |
| 3 ou 4                 | Saída         |                                        |   | X | X | X  |

A norma IEEE 1149.1 define o protocolo para acesso às funcionalidades mínimas da classe de conformidade 1. Adicionalmente, a norma IEEE 1149.1 define o protocolo para acesso às funcionalidades mínimas das classes 2, 3 e 4, se seleccionada para disponibilizar as entradas da infra-estrutura NEXUS.

A arquitectura que utilize uma interface IEEE 1149.1 deve assegurar a existência de instruções (sequências de bits) para:

- Aceder à identificação do microprocessador, a informação de controlo e estado, incluindo configuração dos recursos de depuração (e.g. pontos de paragem).
- Acesso a registos mapeados em memória com o processador parado e em tempo real.
- Acesso a mensagens da infra-estrutura (e.g. mensagens de monitorização).
- Acesso a todas as funcionalidades específicas do dispositivo (e.g. acesso a registos com o microprocessador parado).

A interface AUX deve fornecer a seguinte informação ao exterior, dependendo da classe de conformidade com a norma:

- Monitorização da execução de aplicações pelo sistema operativo via mensagem.
- Monitorização de programa via mensagem.
- Monitorização de acesso a dados via mensagem.
- Sinalização de eventos de ponto de paragem e ponto de observação.
- Substituição de memória em tempo real.
- Outras transferências de informação (definidas pelo fabricante do dispositivo).

Adicionalmente, a interface AUX deverá permitir o acesso aos seguintes elementos, dependendo da classe de conformidade com a norma e se a opção de interface via IEEE 1149.1 não for usada:

- Acesso à identificação do microprocessador, a informação de controlo e de estado.
- Acesso a registos mapeados em memória, com o processador parado ou em funcionamento.

- Acesso a todas as funcionalidades do microprocessador (e.g. registos de utilizador com o microprocessador parado).
- Acesso opcional a ferramentas que efectuem a substituição de porto.
- Acesso opcional de alto desempenho para permitir a aquisição de dados por uma ferramenta específica.

### 3.2.6 Mensagens

O porto AUX fornece uma ligação de alto desempenho entre a infra-estrutura OCD e uma ferramenta externa. Todas as comunicações que sejam efectuadas através desta ligação utilizam mensagens numa ou em ambas direcções. Opcionalmente, as mensagens podem também ser transmitidas através de um porto IEEE 1149.1, utilizando um método diferente (e menos eficiente), descrito na proposta de norma.

O formato e o significado de certas mensagens, classificadas como Mensagens Públicas (*Public Messages*), é definido na proposta. O projectista da infra-estrutura pode definir outras mensagens (*Vendor Defined Messages*), que deverão necessariamente ser suportadas pelas ferramentas de depuração utilizadas.

Podem ser implementadas duas categorias básicas de mensagens: solicitadas (*solicited messages*) e espontâneas (*unsolicited messages*). As mensagens solicitadas são iniciadas e transmitidas por um controlador externo para o microprocessador, sendo as outras enviadas pelo microprocessador e transmitidas normalmente em ocasiões aleatórias.

Todas as mensagens se iniciam com um código de 6 bits (TCODE), que define de forma única o tipo de mensagem. Cinquenta e seis TCODEs (valores 0 a 55) indicam uma mensagem pública definida na norma, ou reservada para versões futuras. Oito TCODEs indicam mensagens definidas pelo projectista, podendo utilizar-se um código adicional de segundo nível (i.e. enviado posteriormente ao TCODE) para subclassificar a mensagem específica. Os microprocessadores conformes com a norma devem suportar o conjunto mínimo de mensagens para a sua classe de conformidade. As mensagens públicas são apresentadas na Tabela 3.4, ordenadas por TCODE, e referindo a classe de conformidade em que são obrigatórias.

Tabela 3.4 – Mensagens públicas definidas na proposta NEXUS

|                                                                         |                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Debug Status<br/>TCODE 0</b>                                         | Esta mensagem é enviada pela infra-estrutura sempre que ocorrer uma das seguintes mudanças de estado: (1) Entrada ou saída de modo de depuração, (2) modificação no estado do microprocessador ou (3) detecção de um ponto de paragem.                                                                                                                                                        |
| <b>Device ID<br/>TCODE 1, Classe 2</b>                                  | Se o porto AUX estiver a ser utilizado (i.e. se uma ferramenta de depuração estiver fisicamente ligada), esta mensagem deve ser enviada pela infra-estrutura após uma reinicialização ( <i>reset</i> ).                                                                                                                                                                                       |
| <b>Ownership Trace<br/>TCODE 2, Classe 2</b>                            | Esta mensagem é enviada pela infra-estrutura sempre que ocorrer uma modificação do processo ou tarefa em execução tal como identificado pelo sistema operativo (utilizando o registo adequado).                                                                                                                                                                                               |
| <b>Program Trace, Direct Branch<br/>TCODE 3, Classe 2</b>               | Esta mensagem é enviada pela infra-estrutura quando é detectada uma modificação no fluxo do programa em execução, devido a um salto condicional que foi aceite.                                                                                                                                                                                                                               |
| <b>Program Trace, Indirect Branch<br/>TCODE 4, Classe 2</b>             | Esta mensagem é enviada pela infra-estrutura quando é detectada uma modificação no fluxo do programa em execução, devido à chamada a uma subrotina, instrução de retorno ( <i>return</i> ) ou interrupção assíncrona.                                                                                                                                                                         |
| <b>Data Trace, Data Write<br/>TCODE 5, Classe 3</b>                     | Esta mensagem é enviada pela infra-estrutura quando é detectada uma operação de escrita em memória, prevista nos atributos programados da monitorização de dados.                                                                                                                                                                                                                             |
| <b>Data Trace, Data Read<br/>TCODE 6, Classe 3</b>                      | Esta mensagem é enviada pela infra-estrutura quando é detectada uma operação de leitura em memória, prevista nos atributos programados da monitorização de dados.                                                                                                                                                                                                                             |
| <b>Data Acquisition<br/>TCODE 7, Classe 3</b>                           | Esta mensagem é enviada pela infra-estrutura quando é detectada a escrita do valor 0x0 no Data Acquisition Control Register, de forma a iniciar a aquisição de dados.                                                                                                                                                                                                                         |
| <b>Error<br/>TCODE 8, Classe 2</b>                                      | Para cada possibilidade de sobrecarga da monitorização ( <i>Trace Overrun</i> ), uma mensagem de erro, contendo um código de erro, é enviada para informar que ocorrências de monitorização foram perdidas, devido a espaço insuficiente nos respectivos <i>buffers</i> .                                                                                                                     |
| <b>Program Trace Synchronization<br/>TCODE 9, Classe 2</b>              | Esta mensagem é enviada pela infra-estrutura quando ocorre um evento que torne necessário sincronizar a monitorização do programa. O valor actual do registo PC ( <i>Program Counter</i> ) poderá ser incluído.                                                                                                                                                                               |
| <b>Program Trace Correction<br/>TCODE 10, Classe 2</b>                  | Esta mensagem é enviada pela infra-estrutura sempre que esta determina que uma mensagem anterior de monitorização de programa continha um valor errado de instruções executadas. O valor corrigido é enviado na mensagem.                                                                                                                                                                     |
| <b>Program Trace, Direct Branch with Sync.<br/>TCODE 11, Classe 2</b>   | Esta mensagem é enviada pela infra-estrutura quando é detectada uma modificação no fluxo do programa, devido a um salto condicional que foi aceite, sendo simultaneamente necessário sincronizar a monitorização, devido à ocorrência de um evento ou por indicação externa.                                                                                                                  |
| <b>Program Trace, Indirect Branch with Sync.<br/>TCODE 12, Classe 2</b> | Esta mensagem é enviada pela infra-estrutura quando é detectada uma modificação no fluxo do programa, devido à chamada a uma subrotina, instrução de retorno ( <i>return</i> ) ou interrupção assíncrona, sendo simultaneamente necessário sincronizar a monitorização, devido à ocorrência de um evento ou por indicação externa.                                                            |
| <b>Data Trace, Data Write with Sync.<br/>TCODE 13, Classe 3</b>         | Esta mensagem é enviada pela infra-estrutura quando é detectada uma operação de escrita em memória, prevista nos atributos programados da monitorização de dados, sendo simultaneamente necessário sincronizar a monitorização, devido à ocorrência de um evento ou por indicação externa.                                                                                                    |
| <b>Data Trace, Data Read with Sync.<br/>TCODE 14, Classe 2</b>          | Esta mensagem é enviada pela infra-estrutura quando é detectada uma operação de leitura em memória prevista nos atributos programados da monitorização de dados, sendo simultaneamente necessário sincronizar a monitorização, devido à ocorrência de um evento ou por indicação externa.                                                                                                     |
| <b>Watchpoint Hit<br/>TCODE 15, Classe 2</b>                            | Esta mensagem é enviada pela infra-estrutura sempre que ocorrer um ponto de observação. Múltiplos pontos de observação podem ser indicados na mesma mensagem, que não deve nunca ser eliminada devido a limitações nos <i>buffers</i> . Se tal acontecer, deverá ser gerada mensagem de erro, garantindo-se o seu envio antes de outra mensagem sinalizando um ponto de observação posterior. |

|                                                          |                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Target Ready</b><br>TCODE 16, Classe 3                | A infra-estrutura envia esta mensagem quando terminar as acções requeridas numa mensagem proveniente do exterior, ordenando uma operação de leitura ou escrita. Esta mensagem serve para indicar que a infra-estrutura está pronta para receber novo comando.                                                                                                           |
| <b>Read Register</b><br>TCODE 17, Classe 3               | A ferramenta envia esta mensagem quando pretende ler informação de controlo ou estado. A infra-estrutura responde com uma mensagem do tipo <i>Read/Write Response Message</i> , contendo a informação requerida.                                                                                                                                                        |
| <b>Write Register</b><br>TCODE 18, Classe 3              | A ferramenta utiliza esta mensagem para controlar recursos utilizados para depuração e para enviar dados para o interior do microprocessador. Após o processamento da mensagem deverá ser enviada para o exterior uma outra do tipo <i>Target Ready Message</i> .                                                                                                       |
| <b>Read/Write Response</b><br>TCODE 19, Classe 3         | Esta mensagem é enviada pela infra-estrutura em resposta a outra do tipo <i>Read Register</i> ou <i>Write Register</i> .                                                                                                                                                                                                                                                |
| <b>Port Replacement, Output</b><br>TCODE 20, Opcional    | Esta mensagem é enviada pela infra-estrutura para preparar a lógica externa de substituição de porto.                                                                                                                                                                                                                                                                   |
| <b>Port Replacement, Input</b><br>TCODE 21, Opcional     | Esta mensagem é enviada pela ferramenta na ocorrência de uma mudança de estado de um dos pinos do porto em substituição.                                                                                                                                                                                                                                                |
| <b>Read Target/Tool</b><br>TCODE 22, Classe 3            | Esta mensagem é enviada pela ferramenta, quando pretende ler o conteúdo de uma posição de memória mapeada no microprocessador alvo, e pela infra-estrutura, se pretender ler uma posição de memória mapeada na ferramenta.                                                                                                                                              |
| <b>Write Target/Tool</b><br>TCODE 23, Classe 3           | Esta mensagem é enviada pela ferramenta, quando pretende escrever numa posição de memória mapeada no microprocessador alvo, e pela infra-estrutura, se pretender escrever numa posição de memória mapeada na ferramenta.                                                                                                                                                |
| <b>Read Next Target/Tool Data</b><br>TCODE 24, Classe 3  | Este tipo de mensagem é utilizada para leituras consecutivas de posições de memória, através da ferramenta ou da infra-estrutura. É utilizada para leitura em bloco da memória, de forma rápida e mais eficiente.                                                                                                                                                       |
| <b>Write Next Target/Tool Data</b><br>TCODE 25, Classe 3 | Este tipo de mensagem é utilizada escritas consecutivas em posições de memória, através da ferramenta ou da infra-estrutura. É utilizada para escrita em bloco em memória, de forma rápida e mais eficiente.                                                                                                                                                            |
| <b>Target/Tool Response</b><br>TCODE 26, Classe 3        | Esta mensagem é enviada pela infra-estrutura ou pela ferramenta, e o seu conteúdo (i.e. pacotes subsequentes) indica se a mensagem recebida mais recentemente consistiu numa <i>Read Target Message</i> ou numa <i>Write Target Message</i> . No primeiro caso é retornado o valor pedido e no segundo é confirmado se a operação de escrita foi concluída com sucesso. |

Os TCODEs 27 a 55 são classificados como reservados (*reserved*) na versão 2.0 da proposta NEXUS e os TCODEs 56 a 63 podem ser usados pelo projectista da infra-estrutura. O TCODE 63 é um caso especial, pois permite a utilização de um código secundário, enviado no pacote subsequente ao TCODE, para permitir a qualquer infra-estrutura utilizar mais de sete códigos proprietários.

### 3.2.7 Sinais e Protocolos

As infra-estruturas que utilizem o porto AUX para comunicação com o exterior devem utilizar os pinos indicados na Tabela 3.5. Pinos obrigatórios e opcionais são designados com OBR e OPC, respectivamente. Pinos não autorizados são apresentados a sombreado.

Tabela 3.5 – Pinos obrigatórios e opcionais do porto AUX

| Pino | Direcção | Full-Duplex | Full-Duplex<br>(c/IEEE1149.1) | Half-Duplex<br>(c/IEEE1149.1) |
|------|----------|-------------|-------------------------------|-------------------------------|
| MCKI | Entrada  | OBR         |                               |                               |
| MDI  | Entrada  | OBR         |                               |                               |
| MSEI | Entrada  | OBR         |                               |                               |
| MCKO | Saída    | OBR         | OBR                           |                               |
| MDO  | Saída    | OBR         | OBR                           |                               |
| MSEO | Saída    | OBR         | OBR                           |                               |
| EVTI | Entrada  | OBR         | OBR                           | OPC                           |
| RSTI | Entrada  | OBR         |                               |                               |
| EVTO | Saída    | OPC         | OPC                           | OPC                           |

Cada sinal ou barramento executa a função indicada em seguida:

**MCKO** – O *Message Clockout* (MCKO) é um sinal de relógio enviado pela infra-estrutura para permitir a sincronização dos sinais MDO e MSEO. O MCKO pode ser independente do relógio do microprocessador. Em alternativa, pode ser utilizado um sinal de relógio enviado pelo microprocessador.

**MDO[M:0]** – O *Message Data Out* (MDO[M:0]) é um pino (se M=0) ou barramento (se M>0) utilizado para a transmissão de dados por parte da infra-estrutura. É utilizado para OTM, BTM, DTM, substituição de memória, etc. Os sinais são considerados válidos no flanco ascendente do MCKO e o número de pinos utilizados depende dos requisitos de largura de banda, devendo ser sempre múltiplo de 2.

**MSEO[1:0]** – Os *Message Start/End Out* (MSEO [1:0]) são pinos que indicam o início e o final de uma mensagem no MDO, e o fim de um pacote de dimensão variável. Só é necessário um pino, mas dois podem ser utilizados para um protocolo de comunicação mais eficiente. Os sinais são considerados válidos no flanco ascendente do MCKO.

**MCKI** – O *Message Clockin* (MCKI) é um sinal de relógio utilizado por uma ferramenta externa para sincronizar os sinais MDI e MSEI. O MCKI pode ser independente do relógio do microprocessador.

**MDI[M:0]** – O *Message Data In* (MDI[M:0]) é um pino (se M=0) ou barramento (se M>0) utilizado para transmitir mensagens para a infra-estrutura. Os sinais são considerados válidos

no flanco ascendente do MCKI e o número de pinos utilizados depende dos requisitos de largura de banda, devendo ser sempre múltiplo de 2.

**MSEI[1:0]** – Os *Message Start/End In* (MSEI [1:0]) são pinos que indicam o início e final de uma mensagem no MDI, e o fim de um pacote de dimensão variável. Só é necessário um pino, mas podem ser utilizados dois, para um protocolo de comunicação mais eficiente. Os sinais são considerados válidos no flanco ascendente do MCKI.

**EVTI** – Uma transição descendente no pino *Event In* (EVTI) permite suspender a execução da aplicação ou a transmissão de uma mensagem de sincronismo (previamente solicitada).

**RSTI** – O pino *Reset In* (RSTI) é utilizado para reinicializar a infra-estrutura NEXUS e os seus recursos.

**EVTO** – O *Event Out* (EVTO) é um pino opcional para permitir a sinalização precisa de um ponto de paragem ou de observação. A ocorrência de um evento a sinalizar é assinalada pela aplicação de um nível lógico elevado no pino, pelo tempo mínimo de um período do MCKO.

A Tabela 3.6 apresenta alguns exemplos de portos AUX. Resumidamente, a utilização de mais pinos de dados permite um melhor desempenho nas comunicações, um pino MSEO ou MSEI adicional permite um protocolo de comunicação mais eficiente e o uso de um relógio independente permite melhor controlo (e eventualmente melhor desempenho).

Tabela 3.6 – Exemplos de portos AUX

| Número de pinos (exemplos) |      |      |      |       | Observações                                                                                                           |
|----------------------------|------|------|------|-------|-----------------------------------------------------------------------------------------------------------------------|
| MDO                        | MSEO | MCKO | EVTI | Total |                                                                                                                       |
| 1                          | 1    | 0    | 1    | 3     | Versão base                                                                                                           |
| 2                          | 1    |      |      | 4     | 2x mais rápido                                                                                                        |
| 4                          | 1    |      |      | 6     | 4x mais rápido                                                                                                        |
| 4                          | 2    |      |      | 7     | 1 ciclo de relógio mais rápido por mensagem                                                                           |
| 8                          | 2    |      |      | 11    | Mais de 8x mais rápido                                                                                                |
| 1                          | 1    | 1    |      | 4     | Um relógio independente permite uma transferência mais rápida (ou mais lenta) do que utilizando o relógio do sistema. |
| 2                          | 1    |      |      | 5     |                                                                                                                       |
| 4                          | 1    |      |      | 7     |                                                                                                                       |
| 4                          | 2    |      |      | 8     |                                                                                                                       |
| 8                          | 2    |      |      | 12    |                                                                                                                       |

A proposta NEXUS [IEEE03] explica em detalhe outros assuntos, como suporte para múltiplos clientes ou ferramentas, a configuração na reinicialização e a segurança e substituição de portas.

O protocolo para a transmissão de mensagens de e para a infra-estrutura, utilizando o porto AUX, é realizado através dos pinos MSEI e MSEO, respectivamente, sendo o mesmo para ambos os casos. Os diagramas presentes na Figura 3.3 foram retirados da norma e representam as versões do protocolo de comunicação, de acordo com o número de pinos utilizado para o MSEI ou MSEO (referidos indistintamente como MSE).

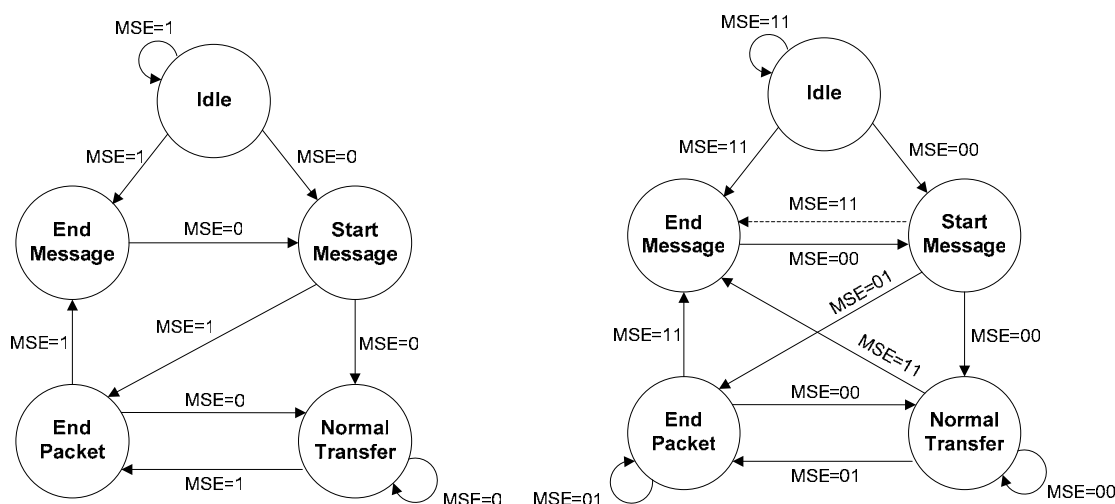


Figura 3.3 – Diagramas explicativos do protocolo de comunicação

Opcionalmente, as infra-estruturas compatíveis com a norma podem implementar o sistema de mensagens através de um porto IEEE 1149.1. Porém, o reduzido desempenho e a dificuldade de lidar com mensagens espontâneas (possível, mas complexo e moroso), desaconselham a sua utilização, para aplicações que exijam mais do que um desempenho mínimo.

Diversas opções para conectores e especificações eléctricas são sugeridas na proposta NEXUS, de acordo com a classe de conformidade e os objectivos de desempenho [Motorola01]. Esta informação foi consideravelmente revista entre as versões 1.0 e 2.0 da proposta e é de esperar que sofra ainda algumas modificações, à medida que mais actualizações surgem no mercado.

### 3.2.8 Aceitação

A demonstração de conceito (*proof-of-concept*) da especificação NEXUS foi apresentada na ESS 2000 [Vetterli00] pela Motorola, através de um equipamento experimental baseado numa infra-estrutura OCD, em conformidade com a classe 2 da norma, incorporada num microprocessador M-CORE M340. Foi utilizado um ambiente de depuração que incluía um analisador lógico Tektronix e um ambiente integrado de desenvolvimento HIWARE, para demonstrar as capacidades de depuração dinâmica. Os resultados apresentados então permitiram comprovar que a adição da infra-estrutura NEXUS acrescentava funcionalidades e desempenho de depuração (estático e dinâmico) consideráveis a processadores do tipo M300. A experiência obtida com esta primeira implementação foi utilizada para finalizar e publicar a primeira versão proposta (*draft*) da norma.

O microprocessador MPC565 “*Spanish Oak*” [Motorola02a] da Motorola foi o primeiro componente comercial a incluir uma infra-estrutura OCD conforme a proposta NEXUS. Este componente integra um CPU RISC de 32 bits capaz de operar a 40 ou 56 MHz, incluindo unidade de vírgula flutuante e até 1 Mbyte de memória Flash. A família de processadores MPC560 utiliza um conjunto de instruções compatível com PowerPC, tendo o MPC565 sido seguido por outras variantes, designadamente pelos MPC561 “*Silver Oak*”, MPC563 “*Green Oak*” e MPC566. Todos os microprocessadores desta família incluem duas infra-estruturas OCD, uma do tipo NEXUS e designada por READI (*Real-time Embedded Application Development Interface*), e outra do tipo BDM. As infra-estruturas foram implementadas com considerável partilha de recursos entre si, sendo apenas possível a utilização de uma delas de cada vez. A depuração dinâmica só é possível utilizando a versão READI. Com os cabos adequados, os depuradores da Ashling, iSystem e Lauterbach, podem funcionar em ambos os modos.

Adicionalmente, estes microprocessadores incluem um porto JTAG, que não pode ser utilizado para depuração, sendo reservado para teste. Duas configurações do porto AUX podem ser utilizadas, nomeadamente uma configuração recomendada com 51 pinos (incluindo 8 pinos para MDO e 2 pinos para MDI) e uma configuração reduzida com 40 pinos (incluindo 2 para MDO e 1 para MDI). Esta família de microprocessadores está em conformidade com a classe 2 e inclui algumas características da classe 3, o que é designado em alguma documentação (da Motorola e não só) como classe 2+.

Em 2005 a Freescale<sup>14</sup> iniciou a comercialização de microprocessadores da família 5500, que incluem uma infra-estrutura OCD em conformidade com a classe 3 da proposta NEXUS. Estes dispositivos são também orientados para aplicações de electrónica automóvel.

A adopção formal da norma NEXUS por parte do IEEE não é previsível num futuro próximo, sendo mais provável uma inclusão progressiva de infra-estruturas compatíveis num número crescente de dispositivos, particularmente microprocessadores Freescale. As razões para este lento crescimento radicam na tradicional tendência para utilizar arquitecturas e soluções proprietárias. À medida que as vantagens de uma abordagem normalizada se acentuem, será possível que se gere uma massa crítica de equipamento e utilizadores, que actue como catalizador da aceitação de uma norma. Com o trabalho já desenvolvido e considerando que algumas das maiores empresas da área fazem parte do consórcio, é realista pensar que a norma que venha a emergir seja baseada na proposta NEXUS.

Como nota final, existem algumas referências que mencionam a possibilidade de infra-estruturas do tipo NEXUS poderem vir a ser utilizadas em microprocessadores da Intel (i.e. *Pentium* ou posterior), o que consistiria num impulso considerável para a sua inclusão e aceitação generalizadas [Wong03].

### **3.3 Injecção de Falhas via infra-estruturas OCD**

Os requisitos comuns da depuração e da injecção de falhas sugerem a reutilização dos recursos e técnicas disponíveis. Em ambos os casos pretende-se um determinado nível de controlabilidade e observabilidade, sobre o sistema alvo, de forma não intrusiva. As infra-estruturas de teste e/ou depuração fornecem um mecanismo base para acesso aos recursos internos dos microprocessadores, sendo depois necessário desenvolver externamente aplicações e equipamentos dedicados à gestão das operações de injecção de falhas e análise dos dados obtidos.

A utilização de infra-estruturas OCD para injecção de falhas tem evoluído com as tecnologias de depuração e com as crescentes funcionalidades disponibilizadas para este fim. As primeiras abordagens limitavam-se a utilizar as capacidades de controlo de execução e acesso a recursos, de forma a suspender a execução do sistema alvo e a modificar os dados presentes em memória ou em registos internos. A maioria das metodologias desenvolvidas

---

<sup>14</sup> Em Outubro de 2003 a divisão de semi-condutores da Motorola separou-se formalmente da empresa mãe, dando origem à Freescale.

tiravam partido de pontos de paragem, ou de sinais externos, para suspender a execução em momentos pré-definidos. Actualmente, existe um maior enfoque no alto desempenho e em baixa intrusividade, tirando partido das capacidades de operação em tempo real, disponibilizadas por infra-estruturas mais recentes.

No contexto geral de análise da confiabilidade, as funcionalidades de monitorização presentes em algumas infra-estruturas OCD permitem melhorar a componente de observabilidade do processo de injeção de falhas, pois podem ser utilizadas para diagnóstico dos efeitos das falhas, de forma não intrusiva. Em certos casos, adopta-se uma abordagem híbrida, com as infra-estruturas OCD a serem utilizadas exclusivamente para a execução de tarefas de controlo ou observação. Existem exemplos de sistemas em que as falhas são inseridas por *software* e os seus efeitos são observados através de infra-estruturas OCD [Prinetto98], e casos em que estas são utilizadas apenas para a inserção das falhas, com o diagnóstico a basear-se nos valores das saídas [Benso98a]. Entre as diferentes abordagens publicadas, que recorrem a infra-estruturas OCD e à injeção de falhas para apoiar processos de avaliação de confiabilidade, podemos salientar os seguintes exemplos:

**FlexFI** [Benso98b][Rebaudengo99a] [Rebaudengo99b] [Prinetto98]

O FlexFI (*Flexible Fault Injection*) é um ambiente de injeção de falhas em sistemas computacionais. Integra diversas metodologias, de forma a permitir a melhor solução para cada cenário específico, incluindo injeção de falhas por *software*, híbrida (com apoio de *hardware* adicional) ou via infra-estruturas OCD do tipo BDM. Esta última foi aplicada em microprocessadores do tipo MC68332, utilizando um depurador dedicado [LA7902]. As falhas são inseridas em memória, ou em registos internos, sendo aceite à partida uma sobrecarga temporal para o processo de inserção. São consideradas diversas alternativas em termos de activação e inserção de falhas, sendo possível a sua execução recorrendo apenas às funcionalidades disponibilizadas pela infra-estrutura OCD (leitura/escrita e pontos de paragem). Os resultados publicados permitem comparar a eficiência das diferentes soluções, em cenários semelhantes, e indicam que a utilização do BDM implica uma intrusividade relativamente baixa, com desempenho adequado (sendo inferior à solução híbrida) e com custos mais baixos (de equipamento e desenvolvimento).

**INERTE** [Yuste03a][Yuste03b][Yuste03c][Pardo06]

O INERTE (*Integrated Nexus-based Real-Time fault injection tool for Embedded systems*) permite a injeção de falhas em sistemas de controlo de motorização automóvel, baseados no

microprocessador MPC565. As falhas são inseridas apenas em memória, em tempo real, e são activadas através de um sinal gerado por um contador externo. Esta abordagem surgiu como forma de superar as limitações encontradas em soluções que utilizavam SWIFI, que se revelaram incompatíveis com os crescentes requisitos de aumento de desempenho do processo de injeção, e de minimização da intrusividade. O equipamento utilizado para interface com o sistema alvo foi o depurador Trace32 da Lauterbach. Os resultados publicados confirmam a validade da solução como forma de injeção de falhas, em tempo real e em elementos de memória, tendo sido identificadas limitações, em termos de cobertura e mecanismo de activação. Visto ser indesejável a suspensão do sistema alvo, as falhas apenas podem ser injectadas em memória. Por outro lado, as limitações técnicas decorrentes do ambiente utilizado restringem as capacidades de sincronização (com eventos internos ao sistema alvo) e impõem atrasos ao tempo necessário para a inserção de cada falha. Estas limitações exigem alguma monitorização adicional e um pós-processamento de dados, de forma a identificar e eliminar experiências cuja execução não decorreu como pretendido.

#### **GOOFI (NEXUS) [Vinter05] [Aidmark03]**

O ambiente de injeção de falhas GOOFI (*Generic Object-Oriented Fault Injection*) foi actualizado de forma a suportar a comunicação via porto NEXUS. Este ambiente foi utilizado para a injeção de falhas num sistema de controlo de uma turbina a jacto, para utilização em veículos não tripulados. O protótipo empregue como alvo, designado por FADEC (*Full Authority Digital Engine Control*) utiliza um microprocessador MPC565 para controlo do motor, incluindo a limitação dos modos de avaria, via mecanismos de *software* e *hardware*. Estes devem permitir valores errados nas saídas do sistema de controlo, desde que dentro de limites aceitáveis, actuando posteriormente sobre os parâmetros do sistema, para promover a recuperação dos referidos erros, activando sistemas de recurso, se necessário. A verificação da confiabilidade do sistema completo é efectuada via injeção de falhas, através de um modelo simples de inversão de bit, em memória de dados efectivamente utilizada e em registos do microprocessador. Não são injectadas falhas na memória de programa, visto esta ser limitada a componentes do tipo ROM. A activação das falhas recorre a pontos de paragem no sistema alvo, sendo portanto executada com o sistema suspenso, mas com sobrecarga temporal mínima. O equipamento utilizado para interface com o sistema alvo é o depurador IC3000 da iSystem. Os resultados publicados indicam que a utilização deste ambiente de injeção de falhas permite um desempenho adequado aos objectivos pretendidos, sendo no entanto impostas limitações temporais, pela necessidade de suspensão

do sistema. No caso específico em análise, estas limitações não afectam significativamente a cobertura de falhas, pois o modelo utilizado para o motor executa a uma frequência significativamente mais baixa do que o microprocessador utilizado para o seu controlo, sendo as falhas injectadas neste último.

Como já foi referido, este documento descreve um trabalho experimental com ambientes de injeção de falhas, visando a verificação de mecanismos de tolerância, presentes em sistemas computacionais. Em todos os casos considerados, o objectivo principal consistiu na avaliação da confiabilidade de um determinado sistema alvo, através da injeção de falhas em memória e subsequente análise das suas consequências. A Tabela 3.7 compara três cenários em que foram utilizadas infra-estruturas OCD para este fim. Cada um dos casos é analisado de forma mais detalhada nos parágrafos seguintes.

Tabela 3.7 – Cenários de injeção de falhas via infra-estruturas OCD, testados neste trabalho

| Cenário              | TSC695                                                                                       | LEON2                                                                            | MPC565                                                                                                                 |
|----------------------|----------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| Sistema alvo         | Microprocessador de 32 bits compatível com SPARC v7                                          | Microprocessador de 32 bits compatível com SPARC v8                              | Microprocessador de 32 bits com arquitectura tipo PowerPC                                                              |
| Aplicações propostas | Sistemas aeroespaciais                                                                       | Sistemas aeroespaciais                                                           | Motorização automóvel ou aeroespacial                                                                                  |
| Infra-estrutura OCD  | Sistema proprietário<br>Acesso via infra-estrutura de varrimento periférico tipo IEEE 1149.1 | Sistema proprietário disponível em modelo VHDL<br>Acesso via porta série (RS232) | Em conformidade com a proposta de norma NEXUS 5001 v1.0<br>Acesso via porto dedicado (AUX) ou via BDM (com limitações) |
| Cobertura de falhas  | Registos internos<br>Memória                                                                 | Registos internos<br>Memória                                                     | Registos internos<br>Registos especiais<br>Memória                                                                     |
| Activação            | Ponto de paragem<br>Sinal externo                                                            | Ponto de paragem<br>Sinal externo                                                | Ponto de paragem<br>Ponto de observação<br>Sinal externo                                                               |
| Leitura / escrita    | Com paragem do sistema alvo                                                                  | Com paragem do sistema alvo                                                      | Com ou sem paragem do sistema alvo                                                                                     |
| Monitorização        | Indisponível                                                                                 | Dados lidos em tempo real, mas disponíveis apenas com sistema alvo suspenso      | Dados lidos e disponibilizados em tempo real                                                                           |

### TSC695

Este microprocessador está presente em alguns sistemas utilizados em aplicações aeroespaciais da ESA (*European Space Agency*) e das agências espaciais brasileira e chinesa. A infra-estrutura de teste e depuração implementada está em conformidade com a norma IEEE 1149.1, permitindo ainda o acesso a registos especiais para depuração

[Gaisler94]. É possível controlar a execução do sistema alvo, inserir pontos de paragem e aceder a registos internos. O acesso a memória externa é também possível, utilizando para tal a cadeia de varrimento que controla as saídas do microprocessador. A activação de falhas pode ser feita através de pontos de paragem ou de um sinal externo. Todas as operações de configuração da infra-estrutura OCD e de injeção de falhas têm de ser executadas com o sistema alvo suspenso. Não é possível a monitorização de execução em tempo real, sendo o diagnóstico de falhas executado com recurso à execução passo-a-passo ou através da leitura das saídas do sistema alvo e do conteúdo da memória. O ambiente de injeção de falhas permite uma cobertura adequada, mas com desempenho limitado e sem capacidades de tempo real [Rela06].

### **LEON2/3**

Este microprocessador foi projectado pela ESA como um possível substituto do TSC695, tendo sido desenvolvido como um modelo VHDL aberto e livremente disponível [Gaisler02]. Como infra-estrutura OCD, encontra-se equipado com uma solução especificamente desenvolvida para o LEON, cujo acesso se faz através de uma porta série do tipo RS-232. As operações de leitura e escrita de registos ou memória, bem como a configuração da própria infra-estrutura, exigem a suspensão da execução da aplicação alvo. A monitorização do programa e o acesso a dados são possíveis, mas são executados utilizando memória interna da infra-estrutura OCD. Esta abordagem limita a quantidade de dados que podem ser registados e faz com que o acesso à informação requeira a suspensão do sistema alvo. Em relação ao exemplo anterior, as vantagens dizem respeito ao desempenho da interface de comunicações e à possibilidade de monitorização, mesmo que limitada, para efeitos de diagnóstico.

### **MPC565**

Este microprocessador foi o primeiro componente comercial equipado com uma infra-estrutura OCD em conformidade com a proposta NEXUS. Este facto, associado ao já considerável número de ferramentas de depuração e à disponibilidade do sistema alvo em placas de desenvolvimento, com documentação relativamente completa, fazem deste microprocessador um componente bastante utilizado para experiências de injeção de falhas. Foram já referidos trabalhos focalizados em sistemas de controlo de electrónica automóvel e operação em tempo real [Yuste03a][Ruiz05], bem como aplicações de controlo de turbinas a jacto [Vinter05]. A partir dos resultados conhecidos e após trabalho experimental com este

tipo de sistemas, foi possível concluir que as capacidades de monitorização e acesso à memória em tempo real, permitem funcionalidades adicionais para operações de injeção de falhas, particularmente quando se pretende realizá-la sobre a memória. As vantagens sobre os cenários anteriores são o melhor desempenho, a monitorização ilimitada no tempo (visto depender apenas de memória externa) e principalmente o acesso em tempo real, com a inerente redução de intrusividade. Este cenário é analisado em maior detalhe no capítulo 4.

### **3.4 *Resumo e Conclusões***

É possível concluir que a depuração de sistemas computacionais é um problema complexo e actual, e que as tecnologias destinadas a este fim estão em permanente evolução, de forma a acompanhar a crescente miniaturização e o ganho de desempenho, dos sistemas electrónicos em geral e dos microprocessadores, em particular. À medida que o encapsulamento de componentes e sistemas impossibilita o uso das técnicas clássicas de depuração, o recurso a infra-estruturas OCD tem a tendência a aumentar. Desta forma, este tipo de infra-estruturas deverá aumentar em utilização e importância, mas também evoluir tecnologicamente. À medida que os mecanismos e ferramentas de depuração se desenvolvem, de forma a aumentar a controlabilidade e a observabilidade disponibilizadas, também aumenta a sua utilidade para a injeção de falhas. As infra-estruturas OCD fornecem os meios para modificar dados em memória ou registos, e para activar essa injeção, de acordo com o estado do sistema alvo ou de um componente externo, e simultaneamente para analisar o comportamento dos diversos componentes envolvidos, antes, durante, e após a injeção da falha. Desenvolvimentos ao nível da cobertura, desempenho (em tempo real e não só) e funcionalidades de depuração, providenciarão benefícios claros para a injeção de falhas.

No que toca à normalização, a situação actual é algo instável, sendo difícil prever a evolução futura. É claro que os fabricantes de sistemas computacionais e de ferramentas de depuração beneficiariam da adopção de uma norma nesta área, sendo o crescente número de esforços uma prova disso mesmo. Será necessário algum ajustamento e definição, de forma a que se evite alguma sobreposição entre as diversas propostas e se consiga uma solução ou soluções equilibradas. Considerando a norma IEEE 1149.1, é possível que surja outra com impacto semelhante, embora tal não seja previsível num futuro próximo. Mesmo assim, as propostas existentes, particularmente a proposta de norma NEXUS 5001, permitem antever alguma uniformização, nem que seja sectorial. No caso específico desta proposta, o objectivo é

permitir alguma flexibilidade, propondo um acesso normalizado, mas uma estrutura de registos aberta. A depuração requer que se tenham em consideração os seguintes pontos:

- Apesar de a proposta permitir a utilização de um porto JTAG para interface, esta solução apresenta limitações em funcionalidade e desempenho, não sendo portanto recomendável para a maioria dos casos. Nenhum dos produtos comerciais existentes utiliza esta solução e é extremamente provável que a esmagadora maioria siga esta mesma doutrina.
- O número de pinos utilizados nos conectores AUX é geralmente elevado, sendo adaptável ao desempenho pretendido. As infra-estruturas emergentes deverão seguir a tendência actual, onde uma mesma implementação pode utilizar portos de acesso com diferente dimensão.
- O protocolo de comunicação é baseado em mensagens e encontra-se bem documentado. No entanto, a estrutura de registos e a interpretação das mensagens não estão definidas, o que torna necessária informação detalhada sobre o microprocessador alvo, para o desenvolvimento de um depurador compatível.
- A proposta refere a necessidade de partilha de informação via API (*Application Protocol Interface*) entre os fabricantes do microprocessador e dos depuradores. Esta necessidade pode impor restrições à implantação da norma, uma vez que a compatibilidade de um depurador com um determinado microprocessador fica dependente da informação disponível.

Em conclusão, as infra-estruturas de depuração fornecem funcionalidades excelentes para a injeção de falhas, particularmente em elementos de memória, de forma eficiente e não intrusiva. O esforço de normalização é um passo adicional que permitirá a reutilização de ferramentas e soluções, não só para depuração, mas também para a verificação e validação de confiabilidade, através de técnicas de injeção de falhas.



*“We all agree that your theory is crazy, but is it crazy enough?”*

Niels Bohr

**Capítulo 1**

**Capítulo 2**

**Capítulo 3**

**Capítulo 4**

**Capítulo 5**

**Capítulo 6**

**Capítulo 7**

**Referências**

**Anexo**

# **Injecção de Falhas em Tempo Real via OCD**



## 4 Injecção de Falhas em Tempo Real via OCD

Este capítulo faz uma apresentação mais detalhada do problema que se pretende abordar, com ênfase nas restrições e dificuldades impostas para a avaliação de características de confiabilidade, através de injecção de falhas em sistemas de tempo real. É feito um enquadramento deste problema específico, com referência às abordagens usualmente empregues, apresentado o ambiente de injecção de falhas desenvolvido e a metodologia proposta, bem como as diferentes soluções projectadas para endereçar as limitações identificadas. O capítulo divide-se em quatro secções, com a primeira a explicar o campo de aplicação da injecção de falhas em tempo real, incluindo uma avaliação comparativa das técnicas mais utilizadas. A segunda secção apresenta a metodologia e o ambiente propostos para a injecção de falhas através de infra-estruturas OCD, em conformidade com a proposta NEXUS. A secção seguinte propõe formas de melhorar o desempenho e acrescentar funcionalidades, com e sem modificação dessa mesma infra-estrutura. A última secção apresenta um resumo do capítulo e algumas considerações relativas à forma de implementar as soluções propostas.

### 4.1 *Campo de Aplicação*

#### 4.1.1 **Sistemas de Tempo Real**

Os sistemas de tempo real são regularmente utilizados em múltiplas e variadas aplicações, incluindo controlo de máquinas e veículos, equipamento aeroespacial, sistemas multimédia e de comunicações [Krishna97][Almeida04]. Como exemplos de utilização específica podem citar-se os sistemas aviónicos, o controlo de motorização, a gestão de transacções bancárias, as redes de comunicações, o controlo de semáforos, etc. A sua utilização no dia-a-dia é já considerável e com tendência para aumentar, à medida que os sistemas de alto desempenho se tornam cada vez mais importantes na sociedade moderna [Stankovic88]. Um sistema de tempo real não é necessariamente rápido, sendo a sua velocidade dependente do meio envolvente. A título de exemplo, um controlo de semáforos apresenta requisitos temporais muito diferentes do piloto automático de um avião.

O meio ambiente apresenta o seu próprio ritmo de evolução, que define a escala de tempo real. A ocorrência de eventos e a sua distribuição no tempo estabelecem os requisitos

temporais que são impostos a um sistema que pretenda interagir dinamicamente com o meio onde se insere, incluindo outros sistemas [Buttazo97], como se ilustra na Figura 4.1.

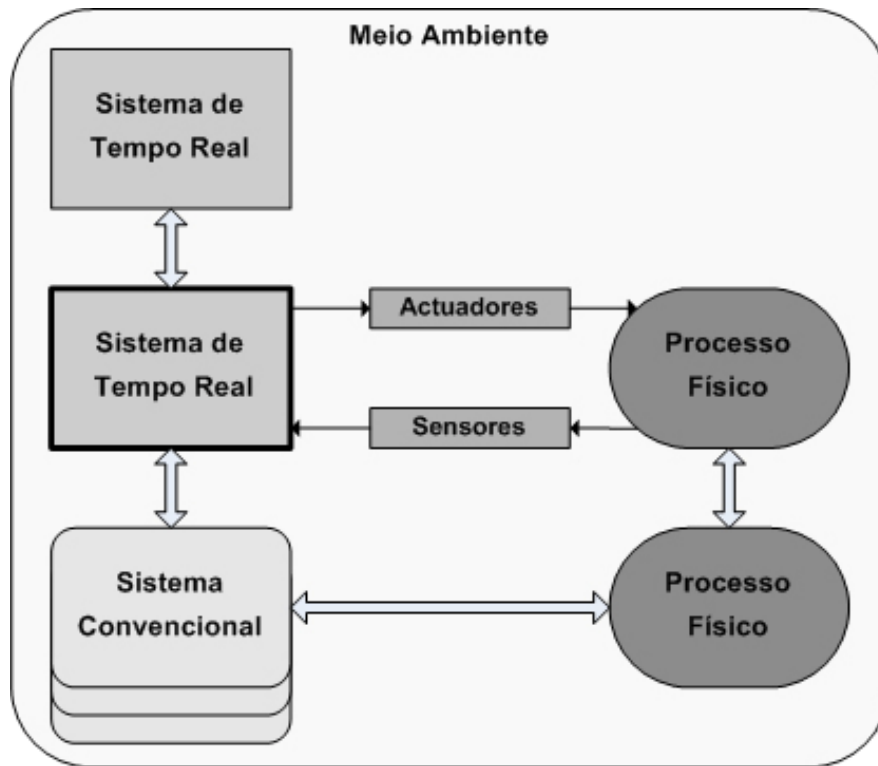


Figura 4.1 – Sistema de Tempo Real e seu Meio Ambiente

As características de tempo real de um determinado sistema definem a sua capacidade de interagir atempadamente com o meio envolvente. Desta forma, podemos afirmar que um sistema de tempo real terá um funcionamento correcto se for capaz de fornecer respostas adequadas, dentro de uma janela temporal bem delimitada, e de apresentar um ritmo de evolução adequado ao processo físico com que interage [Chevochot00][Hansson05]. Se o comportamento temporal do meio envolvente puder ser controlado pelo sistema, ou por um seu utilizador, a classificação de tempo real não será aplicável. Como exemplo destes últimos, podem citar-se os sistemas de reservas utilizados pela aviação comercial, em que é possível atrasar as respostas, em momentos de maior utilização.

Um sistema de tempo real é sujeito a restrições temporais que podem ser classificadas em diversas categorias, sendo as mais utilizadas apresentadas na Figura 4.2.

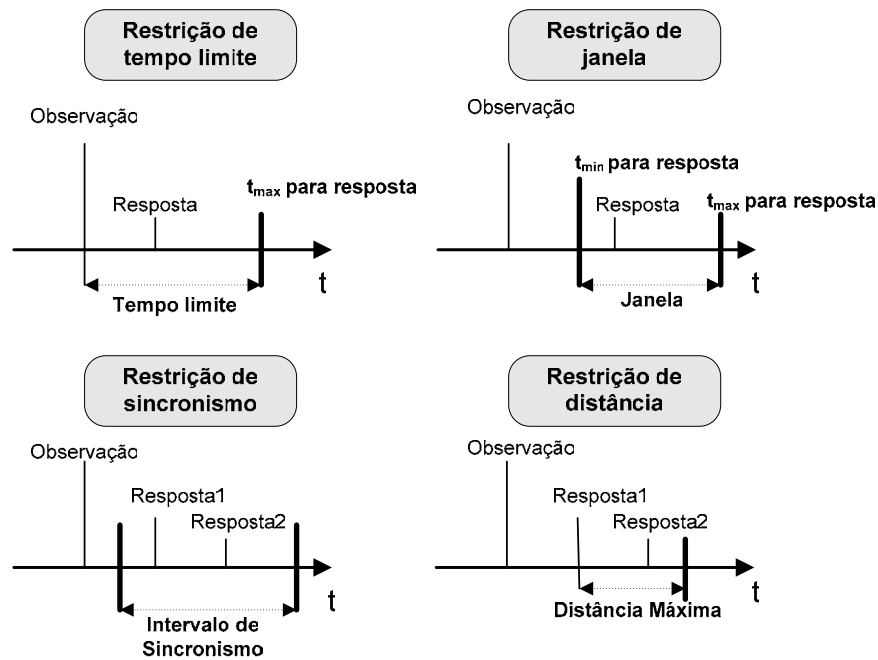


Figura 4.2 – Restrições de tempo real

As restrições mais frequentemente utilizadas são as de tempo limite (*deadline restrictions*), em que a resposta do sistema tem de ser fornecida dentro de um intervalo máximo, que se segue à observação das entradas. Existem também restrições de janela (*window restrictions*), onde são definidos atrasos máximo e mínimo para a resposta, restrições de sincronismo (*synchronism restrictions*), que definem limitações ao intervalo de tempo entre respostas (ou observações) por parte do sistema, e de distância (*distance restrictions*), em que existem limitações ao intervalo de tempo entre respostas consecutivas.

No caso específico dos sistemas computacionais, as restrições impostas pela utilização em tempo real são também classificadas de acordo com as consequências da sua infracção, nomeadamente nas três categorias seguintes:

- **Restrição suave** (*soft*), quando o resultado obtido apresenta alguma utilidade, mesmo se produzido fora de tempo.
- **Restrição firme** (*firm*), quando o resultado não apresenta nenhuma utilidade quando produzido fora de tempo, mas o sistema mantém a operação normal.
- **Restrição rígida** (*hard*), quando o não cumprimento da uma restrição apresenta consequências catastróficas para o sistema.

A partir destas consequências, os sistemas computacionais de tempo real podem ser classificados como:

- **Tempo Real Suave** (*Soft Real Time*), quando apenas apresentam restrições do tipo suave e firme.
- **Tempo Real Rígido** (*Hard Real Time*), quando apresentam pelo menos uma restrição do tipo rígido.

Neste tipo de sistemas, a confiabilidade é simultaneamente mais difícil de assegurar e mais complexa de avaliar, pois não só têm de ser verificados os resultados em termos de exactidão dos dados, como também a sua disponibilidade nos limites temporais estabelecidos. A distinção entre sistemas de tempo real suave ou rígido é também particularmente importante para a confiabilidade. Devido às suas próprias características, um sistema de tempo real rígido apresenta exigências de confiabilidade mais apertadas, sendo portanto frequentemente dotado com equipamento ou aplicações específicas, para garantir essa confiabilidade.

Em termos gerais, os sistemas de tempo real confiáveis podem ser dotados de mecanismos de tolerância a falhas por *hardware* e/ou *software*. Todo o processo de verificação e validação dos referidos mecanismos deve ser executado sem alterar ou suspender o sistema alvo, de forma a não influenciar os resultados. Esta restrição afecta particularmente o processo de injecção de falhas, o qual irá ser penalizado por um maior número de restrições, em termos de cenários exequíveis e possibilidades de actuação.

#### 4.1.2 Injecção de Falhas em Tempo Real

Em sistemas computacionais, a injecção de falhas considera-se em tempo real se for realizada com o sistema alvo a operar à frequência de relógio normal e com o mínimo de intrusividade. As experiências serão particularmente representativas se forem executadas com o sistema inserido no seu ambiente operacional e sujeito aos estímulos externos normais [Maman98]. Por vezes, designam-se também como de tempo real, algumas experiências de injecção de falhas em que é utilizada uma representação de baixo desempenho (i.e. mais lenta) do sistema alvo, seja por recurso a uma frequência de relógio mais baixa ou um modelo de simulação projectado para esse fim. Neste caso, a injecção de falhas pode ser considerada em tempo real, pois trata-se do ajuste controlado do referencial de tempo, mesmo que este não seja idêntico ao utilizado no ambiente operacional. A precisão dos resultados obtidos poderá ser afectada pelo método utilizado, sendo dependente dos parâmetros de confiabilidade que se pretendem avaliar e dos efeitos das modificações ou modelo utilizados.

Em geral, pretende-se que um ambiente de injeção de falhas seja o menos intrusivo possível, pois é importante evitar que alterações ao desempenho ou ao comportamento do sistema alvo possam afectar os resultados das experiências. Em sistemas de tempo real, a probabilidade de ocorrência de perturbações aumenta, pois pequenos atrasos na execução podem introduzir erros ou modificar acções e alterar inadvertidamente o comportamento do sistema alvo. A intrusividade pode manifestar-se de diferentes formas, normalmente através da modificação do *hardware* ou do *software* alvo. O grau de intrusividade será tanto maior quanto mais alterado for o comportamento do sistema, podendo essas alterações ser ou não identificáveis e mensuráveis. Pode ocorrer que um determinado ambiente de injeção de falhas introduza alterações no comportamento do sistema alvo, que falseiem os resultados do teste. Por exemplo, devido a limitações do ambiente de teste, a utilização de frequências de relógio mais baixas do que as utilizadas no ambiente operacional podem mascarar problemas, que deveriam ser detectados por esses testes.

A maioria das técnicas clássicas de injeção de falhas apresenta limitações mais ou menos consideráveis, que afectam a sua adequabilidade a cenários de tempo real, dependendo dos requisitos de confiabilidade a testar. As técnicas baseadas em simulação podem ser adequadas em fases iniciais do desenvolvimento, mas são frequentemente exigentes em termos de tempo de execução e intrinsecamente dependentes da qualidade do modelo disponível para o sistema alvo [Gracia01]. Adicionalmente, pode ser complicado implementar um modelo que represente com precisão todos os atrasos e problemas de temporização que afectarão o sistema físico final, tornando necessária uma técnica diferente assim que aquele esteja disponível, quer em versão protótipo ou de produção.

A injeção de falhas por *software* acrescenta rotinas que realizam esta tarefa por manipulação de dados, o que implica modificações ao código fonte ou o uso de interrupções, dando origem a atrasos e limitando a cobertura de falhas às zonas acessíveis por este método. Apesar de algum trabalho ter mostrado que é possível utilizar esta técnica em diversos sistemas de tempo real [Cunha99][Madeira02], as limitações presentes em termos de intrusividade e cobertura são consideráveis, e mesmo inaceitáveis para sistemas mais exigentes em termos de desempenho.

As limitações em termos de frequência de operação e tempos de resposta tornam difícil aplicar técnicas de injeção de falhas físicas que utilizem contacto com o sistema alvo, dados os seus efeitos no respectivo desempenho [Arlat03]. A maioria das soluções que utilizam injeção de falhas físicas em tempo real utilizam técnicas sem contacto [Martinez99] ou infra-estruturas dedicadas especiais [Valderas06][MacNamee00][Benso98c], sendo ambas

complexas e onerosas. As técnicas sem contacto apresentam também problemas de controlabilidade e replicabilidade, sendo muito difícil, e mesmo frequentemente impossível, o controlo da localização e do instante exacto de injecção de cada falha. As infra-estruturas especiais geralmente impõem grandes sobrecargas em termos de lógica e normalmente requerem uma versão especial do sistema alvo, não sendo comum a sua aplicação nos dispositivos finais. Adicionalmente, o acesso aos elementos internos de um microprocessador, onde as falhas apresentam maior probabilidade de ocorrência, como sejam os elementos de memória e os barramentos de comunicação, é também problemática, de executar sem perturbar a aplicação em execução [Rahbaran04].

Uma possível alternativa para a injecção de falhas em sistemas computacionais, em tempo real, é a utilização das infra-estruturas OCD presentes em muitos dos dispositivos recentes. Apesar de, na maioria dos casos, este tipo de injecção não ser aplicável, devido à sua elevada latência, os avanços recentes nas capacidades das infra-estruturas OCD apresentam soluções possíveis para este problema.

A Tabela 4.1 representa um quadro das diferentes técnicas de injecção de falhas em tempo real, comparando as suas características mais importantes. A classificação é feita utilizando os símbolos “+” e “-”, para resultados considerados “adequados” e “inadequados”, tendo em conta o desempenho possível e as limitações impostas ao processo de injecção de falhas. Os símbolos “++” e “--” são utilizados para casos onde a característica em questão é particularmente beneficiada ou prejudicada. A combinação “+-” indica os casos em que os resultados obtidos variam consideravelmente entre soluções específicas. A análise considera os ambientes de injecção de falhas mais utilizados e documentados.

Tabela 4.1 – Análise comparativa de diferentes técnicas de injecção de falhas

| Característica          | Técnica   |                 |                 |              |                 |
|-------------------------|-----------|-----------------|-----------------|--------------|-----------------|
|                         | Simulação | Software        | Física          |              |                 |
|                         |           |                 | sem contacto    | com contacto | OCD             |
| <b>Controlabilidade</b> | ++        | +               | -               | +            | +               |
| <b>Observabilidade</b>  | ++        | - <sup>1)</sup> | - <sup>1)</sup> | +            | +               |
| <b>Intrusividade</b>    | N/D       | -               | ++              | -            | +               |
| <b>Replicabilidade</b>  | +         | +               | -               | +-           | +               |
| <b>Custo</b>            | +         | ++              | -               | -            | +               |
| <b>Desempenho</b>       | -         | +               | +               | +-           | - <sup>2)</sup> |
| <b>Cobertura</b>        | +         | --              | +               | -            | +-              |
| <b>Exactidão</b>        | --        | +               | +               | +-           | +-              |

1) É frequente utilizar equipamento adicional para melhorar a observabilidade, com alguns custos adicionais em termos de intrusividade.

2) O desempenho pode ser melhorado consideravelmente se forem utilizadas infra-estruturas OCD com capacidade de tempo real (as quais representam ainda uma pequena percentagem).

Para endereçar as necessidades de depuração dos dispositivos de alto desempenho, e em consequência de progressos recentes nas tecnologias de miniaturização, os microprocessadores modernos têm sido dotados de infra-estruturas OCD que permitem capacidades de depuração melhoradas, reutilizáveis para injecção de falhas. Especificamente, a capacidade de escrita em memória fornece as bases para a injecção de falhas em tempo real, enquanto as capacidades de monitorização e de leitura em tempo real permitem uma melhor observabilidade para diagnóstico de falhas.

A proposta de norma NEXUS fornece um suporte, não só para a normalização das operações de depuração, mas também da injecção de falhas em tempo real. Como foi referido na secção 3.2, aquele documento define um conjunto mínimo de funcionalidades, bem como o porto de acesso e o protocolo de comunicações, sendo a conformidade dividida em quatro classes. As funcionalidades disponibilizadas nas diferentes classes proporcionam um conjunto de ferramentas muito útil para a injecção de falhas em geral. As classes de conformidade superiores oferecem recursos particularmente valiosos para a injecção de falhas em tempo real, como se pode ver pela Tabela 4.2.

Tabela 4.2 – Funcionalidades NEXUS requeridas para injecção de falhas (por classe)

| <b>Funcionalidade</b>                     | <b>Classe</b> | <b>Utilização em Injecção de Falhas</b>   |
|-------------------------------------------|---------------|-------------------------------------------|
| Controlo de execução                      | <b>1</b>      | <b>Activação externa</b>                  |
| Pontos de paragem                         | <b>1</b>      | <b>Activação interna</b>                  |
| Pontos de observação                      | <b>1</b>      | <b>Activação em tempo real</b>            |
| Acesso estático a registos e memória      | <b>1</b>      | <b>Injecção estática de falhas</b>        |
| Monitorização de programa                 | <b>2</b>      | <b>Diagnóstico dos efeitos das falhas</b> |
| Acesso em tempo real a registos e memória | <b>3</b>      | <b>Injecção de falhas em tempo real</b>   |
| Monitorização de dados                    | <b>3</b>      | <b>Diagnóstico melhorado</b>              |
| Substituição de memória                   | <b>4</b>      | <b>Controlabilidade melhorada</b>         |

A injecção de falhas em tempo real é impossível em dispositivos limitados à conformidade com a classe 1, pois todas as operações de depuração têm de ser executadas com o microprocessador parado. Os pontos de observação podem também ser utilizados para o diagnóstico e adicionalmente para garantir o sincronismo com um mecanismo externo de activação das falhas. Os dispositivos conformes com a classe 2 permitem monitorizar a execução das aplicações em tempo real, o que pode ser utilizado para um melhor diagnóstico dos efeitos das falhas. No entanto, em dispositivos estritamente conformes com a classe 2 (i.e. sem capacidades adicionais), não é possível a injecção de falhas em tempo real, pois as

operações de escrita requerem a paragem do sistema alvo. A conformidade com a classe 3 acrescenta a capacidade de escrita em memória com o microprocessador em execução, fundamental à utilização de infra-estruturas OCD para a injecção de falhas em tempo real. A classe de conformidade 4 não se encontra presente em componentes actuais, estando prevista para eventual inclusão em microprocessadores mais avançados. As funcionalidades planeadas para esta classe são potencialmente úteis, particularmente a substituição de memória.

Uma vantagem acrescida da injecção de falhas via infra-estruturas OCD, do tipo NEXUS, é a intrusividade mínima, pois não é necessária nenhuma modificação ao sistema alvo, nem alterações às aplicações (ou sistemas operativos) utilizadas, uma vez que toda a infra-estrutura já se encontra disponível para efeitos de depuração.

## 4.2 Injecção de Falhas via Infra-estruturas NEXUS

### 4.2.1 Ambiente e Metodologia

Para uma avaliação das capacidades e limitações da injecção de falhas em tempo real utilizando infra-estruturas de depuração NEXUS, foi feito um estudo baseado num microprocessador Freescale MPC565<sup>15</sup>, que integra uma infra-estrutura OCD em conformidade com a classe 2 e equipamento de depuração compatível. Esta abordagem foi similar a outros trabalhos desenvolvidos nesta área [Yuste03a] [Vinter05] e o ambiente de injecção de falhas utilizado é apresentado na Figura 4.3.

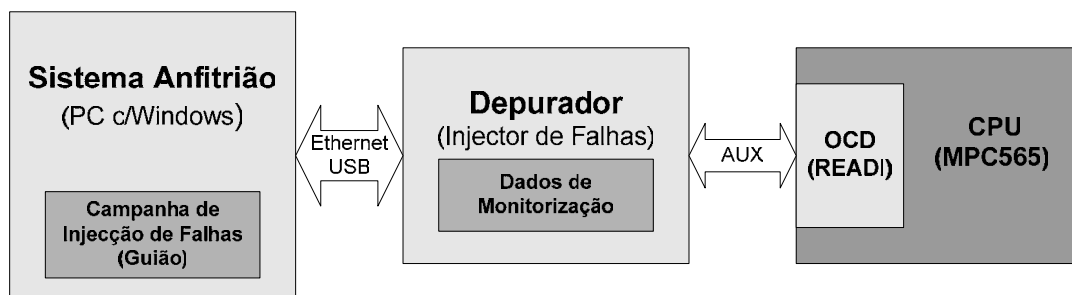


Figura 4.3 – Ambiente de injecção de falhas (MPC565)

O ambiente pode dividir-se em três elementos principais, interligados por dois canais de comunicação. Temos em primeiro lugar o sistema anfitrião, onde é feita toda a interface com

<sup>15</sup> É importante referir que alguma documentação indica este microprocessador como estando equipado com uma infra-estrutura NEXUS de classe 3. Esta informação foi posteriormente corrigida para classe 2 ou classe 2+. A família de microprocessadores MPC5500, que irá substituir a MPC560, já vem equipada com uma infra-estrutura NEXUS de classe 3.

o utilizador e a gestão das experiências, na forma de campanhas de injeção de falhas. Representado à direita na Figura 4.3, o sistema alvo é constituído por uma placa de desenvolvimento, onde se encontra o microprocessador sobre o qual são realizadas as experiências de injeção de falhas. O depurador controla a comunicação entre ambos e armazena o arquivo intermédio dos dados de monitorização. A comunicação entre o sistema anfitrião e o depurador pode ser efectuada de várias formas, sendo o USB e *Ethernet* as mais usuais. A ligação entre o depurador e o sistema alvo faz-se utilizando um cabo normalizado entre os portos AUX de ambos os componentes (por vezes designados por portos NEXUS). A metodologia de realização de experiências baseia-se na abordagem clássica descrita no capítulo 3, com algumas adaptações impostas pelas restrições de tempo real. Cada sequência ou campanha de experiências de injeção de falhas é descrita na forma de um guião, formado por uma sequência de comandos a executar pelo depurador, que é guardado e executado a partir do sistema anfitrião.

O modelo de falhas utilizado consiste na inversão de bits, predominantemente em memória, de forma a reconstituir a ocorrência de SEUs (*Single Event Upsets*) na zona mais vulnerável do microprocessador [Asenek98][Gil00][Velazco05].

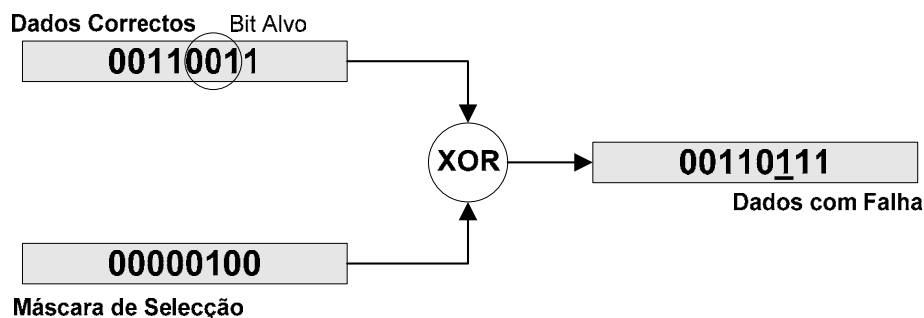


Figura 4.4 – Injecção de falhas num único bit (exemplo)

Para obter uma maior cobertura, realizaram-se algumas experiências que tiveram como alvo os registos internos do microprocessador. Estas experiências apresentam restrições adicionais, já que os registos internos não podem ser acedidos sem parar o microprocessador (ao contrário da memória). Desta forma, a verificação da confiabilidade, através de injeção de falhas, divide-se em campanhas, cada uma delas visando zonas diferentes do espaço de memória do microprocessador.

Para o MPC565, bem como para a maioria dos microprocessadores, o alvo específico de uma experiência de injeção de falhas, do tipo inversão de bit, pode estar numa de três zonas distintas: (1) a memória de dados, (2) a memória de programa ou (3) os registos internos. Dependendo da zona considerada, devem usar-se variantes da metodologia, devido às

limitações impostas. Assume-se que é possível distinguir entre memória de dados e memória de programa a partir do conhecimento da aplicação alvo utilizada. Se esta distinção não for possível, será necessário tratar toda a memória como memória de programa, sendo esta a que apresenta maiores limitações.

Cada campanha de injeção de falhas é constituída por uma série de experiências distintas<sup>16</sup>, cada uma delas constituída por uma fase de inicialização do sistema alvo, a injeção de uma falha do tipo inversão de bit e uma posterior conclusão e classificação de resultados. A estrutura de uma campanha é apresentada na Figura 4.5

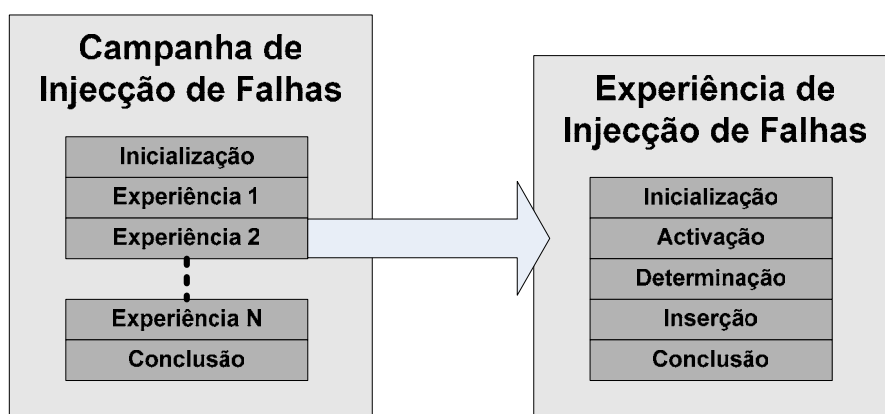


Figura 4.5 – Campanha e experiência de injeção de falhas

Para cada campanha, começa-se pela reinicialização (*reset*) do sistema alvo e por configurar a infra-estrutura OCD. Esta fase de **inicialização** deve incluir o carregamento em memória das aplicações e do sistema operativo, bem como a configuração de eventuais periféricos. Caso seja necessário equipamento externo para o procedimento de injeção de falhas, ele deverá também ser configurado.

A configuração da infra-estrutura OCD inclui a programação de elementos utilizados para controlo e observação, tais como pontos de paragem, pontos de observação ou monitorização de programa / dados. Esta programação é normalmente executada escrevendo em registos específicos a informação pretendida, como por exemplo as condições (endereços, dados) dos pontos de paragem, ou a configuração do mecanismo de monitorização. Nesta fase é também inicializado o sistema que será utilizado como fonte do sinal de activação, para a inserção da falha pretendida, a partir de um conjunto de condições temporais ou espaciais. No caso geral a aplicação alvo é executada desde o início mas, em cenários onde o desempenho seja

<sup>16</sup> Em certos casos particulares, em que se pretende avaliar questões de replicabilidade ou a congruência de resultados, é desejável repetir a mesma experiência.

particularmente importante, toda ou parte da configuração da infra-estrutura OCD pode ser efectuada com o sistema inicializado e em execução.

Na inicialização de cada campanha são efectuadas as configurações que serão comuns a todas as **experiências**, como por exemplo o tipo de monitorização e o carregamento da aplicação em memória, se aplicável. Na fase de **conclusão**, é efectuada a paragem segura de todos os componentes e arquivados os dados relevantes, para posterior análise. Esta última é normalmente efectuada a partir da informação de monitorização, recebida do sistema alvo pelo depurador.

No caso do MPC565 a monitorização pode ser feita de acordo com os três tipos definidos na proposta NEXUS, nomeadamente (1) monitorização de programa (*program trace*), (2) de dados (*data trace*), ou (3) de processo (*ownership trace*). A monitorização de programa fornece informação sobre a execução de instruções pelo microprocessador, o que permite a reconstrução do fluxo do programa. A monitorização de dados permite observar acessos a elementos mapeados em memória pelo microprocessador. Estes podem, e devem, ser limitados a zonas e tipos de operação com interesse para o processo em análise, pois a monitorização generalizada de todas as operações de acesso a memória poderá provocar a perda de dados ou mesmo bloqueios. A monitorização de processo requer um sistema operativo com controlo das tarefas em execução no sistema alvo, e que este utilize um registo para a respectiva identificação. As alterações a este registo são assinaladas, permitindo portanto reconstituir a evolução da aplicação base. No nosso caso, utilizou-se a monitorização de programa em todas as experiências, sendo a de dados usada de forma limitada e apenas quando necessário. A de processo não foi utilizada, visto não acrescentar informação relevante para o pretendido e requerer um sistema operativo específico.

Para simplificar o problema e permitir uma análise comparativa dos resultados, optou-se por realizar cada experiência de injecção de falhas a partir da inicialização do microprocessador e da execução da mesma aplicação desde o início. As campanhas que utilizam como alvo áreas de memória foram executadas em tempo real. Aquelas que endereçam registos internos requereram a suspensão do microprocessador alvo para a inserção da falhas, visto não ser possível aceder a esses registos através da infra-estrutura OCD, com o microprocessador em operação. Cada experiência de injecção de falhas segue os passos indicados na

Figura 4.6 e uma descrição mais pormenorizada de cada passo é apresentada na Tabela 4.3.

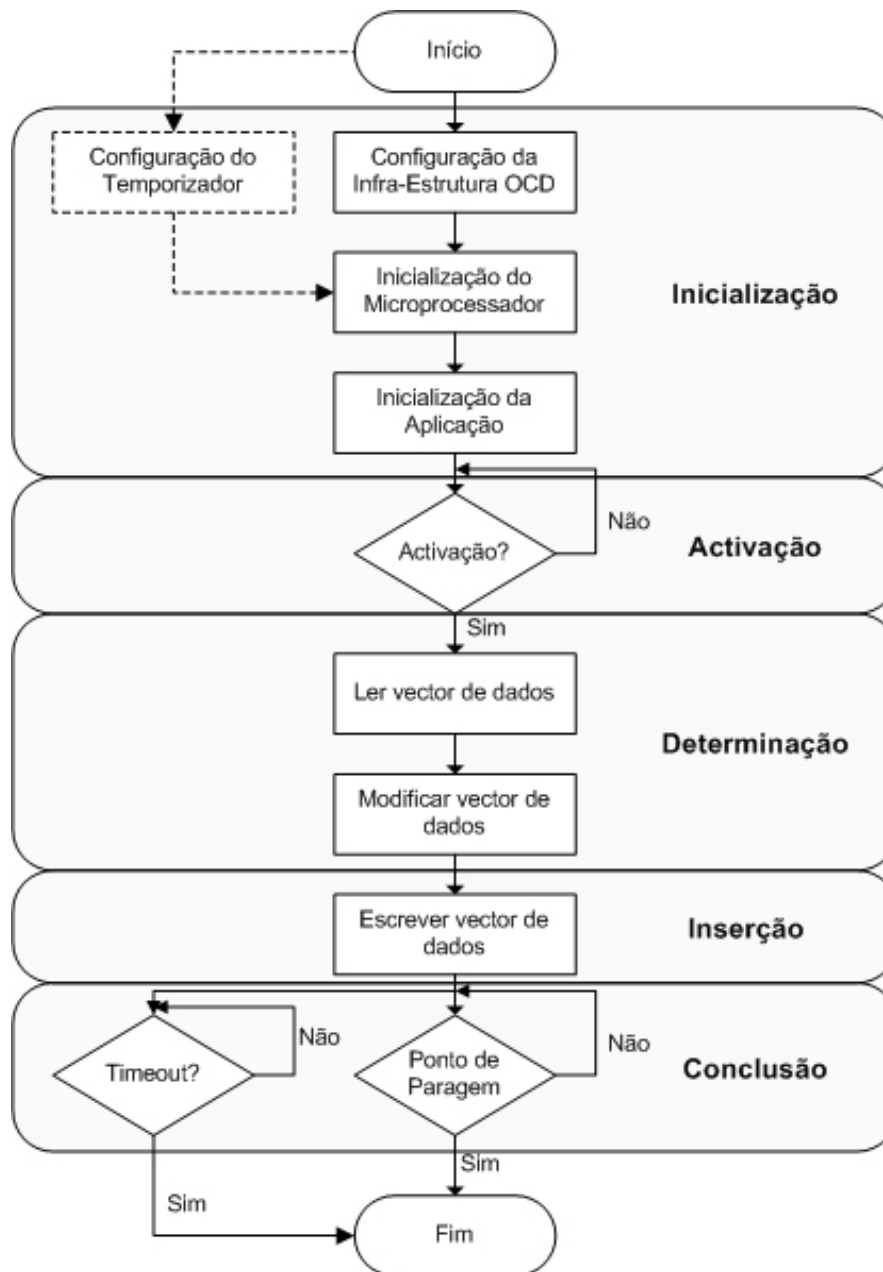


Figura 4.6 – Fluxo de uma experiência de injeção de falhas

Tabela 4.3 – Passos para a Injecção de Falhas no MPC565

| # | Passo                 | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | Inicialização         | A) A infra-estrutura OCD é configurada, incluindo o modo de monitorização e eventuais pontos de paragem ou observação.<br>B) Temporizador é inicializado (se utilizado).<br>C) O microprocessador é inicializado.<br>D) A aplicação alvo é inicializada.                                                                                                                                                                            |
| 2 | Activação da Falha    | A) Um sinal ou mensagem comunica ao sistema anfitrião que foi atingido o instante em que deve ser inserida a falha.<br>A condição de activação pode ser:<br>- Uma condição temporal no sistema anfitrião.<br>- Uma ocorrência específica detectada no sistema alvo e comunicada ao sistema anfitrião (via depurador).<br>- Uma combinação das duas anteriores.                                                                      |
| 3 | Determinação da Falha | A) Na ocorrência da condição de activação o sistema anfitrião transmite ao sistema alvo (via depurador e via OCD):<br>- Um comando de leitura de dados<br>- O endereço da célula de memória<br>B) O sistema alvo comunica ao sistema anfitrião (via depurador e via OCD) o conteúdo da célula de memória.<br>C) O sistema anfitrião determina o novo valor que deverá ser escrito nessa célula de memória (i.e. o valor com falha). |
| 4 | Inserção da Falha     | A) O vector de dados com falha e o endereço de memória são transmitidos ao depurador.<br>B) O depurador envia uma mensagem ao OCD instruindo a escrita do vector de dados no endereço pretendido.                                                                                                                                                                                                                                   |
| 5 | Conclusão             | A) A aplicação é executada até se atingir o ponto de paragem ou se esgotar o tempo máximo permitido (o que ocorrer primeiro).<br>B) A aplicação é suspensa.                                                                                                                                                                                                                                                                         |

A inicialização representa as acções executadas no início de cada experiência, incluindo o carregamento e execução do próprio guião. Nas campanhas em que a zona alvo é a memória de dados ou os registos internos, a aplicação é carregada em memória uma única vez antes da execução de cada campanha. Se a zona alvo for memória de programa, a aplicação em execução terá de ser carregada antes de cada experiência. Caso seja necessário um ponto de paragem, ou ponto de observação, estes deverão ser programados nesta fase. Apesar de ser possível realizar esta tarefa com o sistema alvo em funcionamento, é preferível evitá-lo, para que não surjam conflitos ou resultados errados, devidos à ocorrência do ponto pretendido, antes de terminada a sua programação.

A condição de activação da falha pode basear-se em sincronização temporal ou espacial, ou numa combinação de ambas. A sincronização temporal é baseada no relógio do sistema anfitrião e consiste num contador simples, que activa um sinal após um número determinado de ciclos. A sincronização espacial baseia-se na utilização de um sinal proveniente do sistema alvo, gerado na ocorrência de um evento interno detectável. Apesar de estarem previstas na norma diversas condições em que um sinal é enviado do sistema alvo, a sua

concretização depende da versão da infra-estrutura OCD e da funcionalidade específica implementada. Normalmente, as condições disponíveis baseiam-se na execução de determinadas linhas de código, ou o acesso a recursos em situações pré-especificadas (e.g. escrita de um dado valor numa determinada área de memória). A utilização combinada dos dois tipos de sincronização pode ser feita sequencialmente ou através de uma função lógica. A solução mais frequente consiste em activar uma das condições para iniciar a outra, como sucede ao executar-se a aplicação alvo até que se atingir uma determinada linha de código, esperando-se então  $n$  ciclos de relógio. Se pretendido, poderão utilizar-se condições mais complexas, que combinem requisitos temporais e espaciais [Yuste03c].

A determinação da falha consiste na leitura do valor presente no elemento alvo, seja este uma célula de memória ou um registo interno, e posterior inversão do bit pretendido, de forma a determinar o valor com falha. Para que esta operação possa ser executada, o vector de dados relevante tem de ser lido a partir do sistema alvo e comunicado ao sistema anfitrião, para aí ser alterado.

A última operação necessária é a escrita do vector de dados, com a falha pretendida, no elemento de onde foi obtido o vector original. Esta tarefa requer uma operação de escrita, sobre o elemento relevante do sistema alvo.

Posteriormente à execução de cada campanha de injecção de falhas, é necessário analisar os seus resultados, de forma a diagnosticar o seu efeito no sistema alvo. É necessário estabelecer previamente uma classificação dos resultados das experiências, normalmente baseada no modelo de falhas utilizado. A Figura 4.7 representa a classificação utilizada para as experiências realizadas com o MPC565.

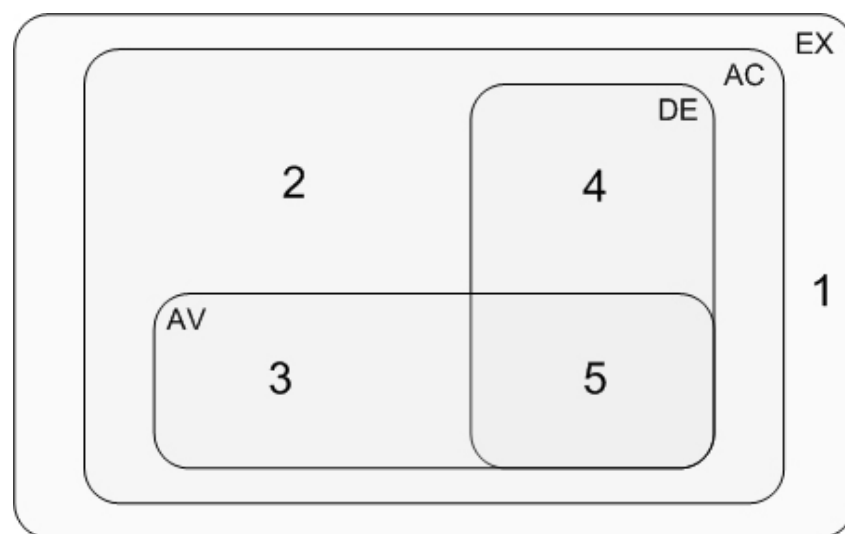


Figura 4.7 – Classificação dos resultados das experiências de injecção de falhas

O conjunto mais abrangente será o conjunto EX (de Experiência), que representa a totalidade das experiências realizadas. O conjunto AC (de Activada) representa as falhas activadas, ou seja, aquelas experiências em que o valor com falha foi efectivamente lido pelo sistema alvo. Dentro do conjunto AC temos dois outros parcialmente sobrepostos, designados por DE (de Detectada) e AV (de Avaria), que representam respectivamente as falhas detectadas, por um mecanismo de detecção e aquelas que provocaram uma avaria no sistema alvo. Entende-se como avaria um desvio do comportamento do sistema, em relação ao que ocorreria se a falha não tivesse sido inserida. A partir dos conjuntos referidos podemos definir cinco diagnósticos possíveis, identificados numericamente na Figura 4.7:

1. **Falha não activada** – corresponde a uma falha cujo valor nunca foi lido (e portanto permanece latente no final da experiência) ou que foi eliminado (escrito por cima) sem nunca ter sido lido.
2. **Falha inconsequente não detectada** – corresponde a uma falha activada, mas que não provoca nenhuma alteração ao funcionamento do sistema, sendo mascarada pela sua redundância natural.
3. **Falha consequente não detectada** – corresponde a uma falha que provoca alterações ao comportamento do sistema (i.e. uma avaria), sem ser detectada pelos mecanismos de detecção existentes.
4. **Falha inconsequente detectada** – corresponde a uma falha que é detectada pelos mecanismos de detecção de falhas, mas que não provoca nenhuma avaria.
5. **Falha consequente detectada** – corresponde a uma falha que, apesar de detectada, altera o comportamento do sistema (para além da detecção da falha).

O tempo de duração de cada experiência é também um factor importante e directamente relacionado com o diagnóstico que se poderá obter. Uma falha classificada como latente, ao fim de um determinado tempo de experiência, poderia ser activada, se a aplicação alvo fosse executada durante mais algum tempo. Adicionalmente, dependendo da forma como é realizada a detecção e eventual correcção de erros, é possível que, dado um suficiente tempo de execução, uma falha detectada venha a provocar uma avaria, ou que o mecanismo de detecção actue só depois da falha provocar uma avaria. Estas possibilidades têm de ser tomadas em consideração, de forma a evitar que o limitar do tempo de experiência, ou a sua suspensão e classificação na primeira ocorrência (seja esta uma detecção ou uma avaria), não causem um diagnóstico errado. A solução mais prática será o recurso a experiências com tempo finito, directamente relacionado com a aplicação em causa, ou o recurso a aplicações

que se concluam num tempo previsível. Terminar qualquer experiência na ocorrência de um evento relacionado com a falha, poderá reduzir artificialmente os resultados de falhas consequentes detectadas, sendo uma possível solução o prolongamento da experiência por um determinado intervalo de tempo, posteriormente à sua detecção ou à manifestação de uma avaria. Por outro lado, algumas implementações de tolerância a falhas baseiam-se na suspensão ou reinicialização do sistema, quando uma falha é detectada. Face a uma destas situações, e existindo um grau de certeza aceitável de que a suspensão ou reinicialização são efectivas, poder-se-á terminar a experiência quando ocorre a detecção. Este procedimento é frequentemente utilizado para acelerar o processo de diagnóstico associado à verificação da confiabilidade por injeção de falhas, particularmente para sistemas e aplicações alvo mais complexos.

A forma pela qual se procede ao diagnóstico depende das funcionalidades disponibilizadas pela infra-estrutura, da aplicação e sistema alvo, do modelo de falhas considerado e do equipamento adicional que se pretenda utilizar. O diagnóstico de falha depende das acções que são executadas pela aplicação alvo, no momento da detecção. Em geral, pode-se monitorizar o registo ou o endereço de memória acedido após a detecção de uma falha, utilizando para isso a monitorização de dados ou um ponto de observação (ou de paragem). O diagnóstico de ocorrência de uma avaria depende também do que se pretende que seja interpretado como tal. Sempre que possível, deverão ser comparadas as saídas do sistema com os seus valores esperados. As infra-estruturas OCD, conforme a proposta NEXUS, podem fornecer algumas ferramentas que facilitem este processo, permitindo o diagnóstico sem utilizar equipamento externo [Yuste03a]. No caso mais geral, em que uma avaria corresponde a qualquer alteração ao funcionamento do circuito, poderá ser comparado o fluxo de programa monitorizado com o fluxo esperado, identificando-se as avarias através das discrepâncias entre ambos. Adicionalmente (ou em alternativa) poderá ser utilizada a monitorização de dados, associada aos endereços das entradas e saídas. Caso a aplicação guarde resultados em memória, estes poderão ser lidos pela infra-estrutura OCD durante ou após a execução da aplicação, de forma a comparar os resultados com os seus valores esperados.

As variáveis mais importantes deste processo serão sempre os requisitos de diagnóstico e as funcionalidades disponibilizadas pela infra-estrutura OCD, ou por outro equipamento adicional. Qualquer solução terá sempre de levar em consideração estes factores, que poderão dar origem a ajustes nos requisitos.

O diagnóstico da activação de uma falha poderá ser feito pela monitorização de dados ou pela leitura da posições de memória relevantes, em momentos importantes da execução. Em certos casos, este processo poderá ser impraticável, não sendo possível a diferenciação entre falhas não activadas e falhas inconsequentes activadas.

#### 4.2.2 Sistema Alvo

Foram utilizados dois sistemas alvo para a injeção de falhas em microprocessadores, com uma infra-estrutura OCD NEXUS, ambos baseados no microprocessador MPC565 [Motorola02a]. Esta opção deve-se ao facto deste ser um produto comercial e de estar disponível na forma de placas de desenvolvimento. O MPC565 é um membro da família MPC500 e suporta um conjunto de instruções compatíveis com a arquitectura PowerPC [Motorola00][Paap93]. É um microprocessador de 32 bits, desenvolvido para equipamento de controlo e utilizado principalmente em electrónica automóvel, mais especificamente no controlo de motores e transmissões (*powertrain*). As principais características do MPC565 são apresentadas na Tabela 4.4.

Tabela 4.4 – Características principais do MPC565

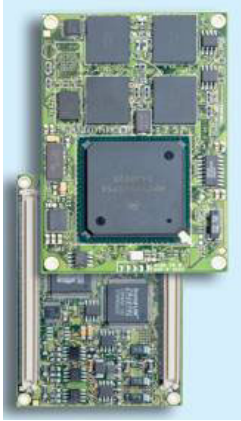
|                                 |                                                                                                                                         |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>Fabricante</b>               | Freescall                                                                                                                               |
| <b>Frequência de relógio</b>    | 40 MHz (56 MHz em configurações específicas)                                                                                            |
| <b>Encapsulamento</b>           | PBGA (388 pinos)<br>Dimensão 27x27mm                                                                                                    |
| <b>Alimentação</b>              | 5 V (tensão interna de 2,6 V)                                                                                                           |
| <b>Dimensão dos barramentos</b> | 24 bits (endereços)<br>32 bits (dados)                                                                                                  |
| <b>Conjunto de Instruções</b>   | Compatível PowerPC                                                                                                                      |
| <b>Memória Interna</b>          | 1 Mbyte Flash (2x512 Kbyte)<br>36 Kbyte RAM                                                                                             |
| <b>Interfaces</b>               | 3xCAN, 2xQSPI, 4xUART                                                                                                                   |
| <b>Teste e Depuração</b>        | Infra-estrutura compatível NEXUS Classe 2 (READI)<br>Infra-estrutura de teste por varrimento periférico<br>Interfaces NEXUS, BDM e JTAG |

O MPC565 é um microprocessador do tipo RISC (*Reduced Intruction Set Computer*), que executa a maioria das instruções num único ciclo de relógio. Integra cinco unidades distintas, nomeadamente uma unidade de inteiros (IU – *Integer Unit*), uma unidade de armazenamento (LSU – *Load/Store Unit*), uma unidade de processamento de saltos (BPU – *Branch Processing Unit*), uma unidade de vírgula flutuante (FPU – *Floating Point Unit*) e uma

unidade de multiplicação/divisão de inteiros (IMD – *Integer Multiplier/Divider*). Este microprocessador é orientado principalmente para a indústria automóvel, sendo utilizado comercialmente em diversas aplicações de controlo.

Entre as várias placas de desenvolvimento disponíveis, foram utilizadas duas no decorrer do trabalho descrito neste capítulo. As principais características de cada uma delas são apresentadas na Tabela 4.5.

Tabela 4.5 – Características das placas de desenvolvimento utilizadas

|                   |                                                                                    |                                                                                     |
|-------------------|------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
|                   |  |  |
| <b>Fabricante</b> | Phytec                                                                             | Axiom                                                                               |
| <b>Modelo</b>     | phyCORE-MPC565                                                                     | CMD-565                                                                             |
| <b>Memória</b>    | 1 Mbyte a 16 Mbyte                                                                 | 512 Kbyte                                                                           |
| <b>Interfaces</b> | 4xUART<br>3xCAN<br>Ethernet 10 Mbit/s                                              | 2xUART<br>2xCAN                                                                     |
| <b>Depuração</b>  | READI / BDM<br>Conector AUX de 40 pinos                                            | READI / BDM<br>Conector AUX de 40 pinos                                             |
| <b>Outros</b>     | Construção em 2 níveis<br>(ligados por conectores)                                 | Área de desenvolvimento com<br>alimentação a 5V<br>(tipo <i>Breadboard</i> )        |

Apesar de ambas as placas apresentarem uma interface de depuração idêntica, optámos por utilizar um ambiente de depuração diferente para cada um dos sistemas alvo. Da mesma forma, as aplicações utilizadas foram substancialmente diferentes, pretendendo-se assim lidar com restrições técnicas e logísticas e, ao mesmo tempo, avaliar cenários operacionais diferentes.

### 4.2.3 Infra-estrutura OCD

A infra-estrutura OCD incluída na família de microprocessadores MPC560 está em conformidade com a classe 2 da proposta de norma, e dispõe de algumas funcionalidades de

classe 3, o que é por vezes designado por classe de conformidade 2+. Esta infra-estrutura (designada por READI pela Freescale) foi implementada de forma a que a depuração do microprocessador possa ser efectuada através de um porto AUX normalizado ou de uma interface BDM. Neste último caso não estão disponíveis as principais características de depuração, que requerem a interface AUX, nomeadamente as funcionalidades de tempo real. O acesso e controlo das funcionalidades de depuração estática podem ser feitos através das interfaces AUX ou BDM.

O controlo de execução permite as funcionalidades padrão, como sejam parar, arrancar, reinicializar o processador e executar em modo passo-a-passo. É possível aceder a recursos de memória, para leitura e escrita, com o microprocessador em funcionamento, bem como à maioria dos registos utilizados para depuração (e.g. configuração dos pontos de paragem e parâmetros de monitorização), mas o acesso a registos internos do microprocessador apenas é possível com este parado [Motorola02a].

A infra-estrutura READI permite a utilização de um máximo de oito pontos de paragem por *hardware*, divididos de acordo com o tipo de condição que podem detectar. Quatro baseiam-se em endereços de instrução, dois em endereços de memória e dois em dados de memória. Uma estrutura lógica combinacional permite a utilização de conjuntos de condições. Adicionalmente existem dois contadores de 16 bits, de forma a permitir que a activação de um ponto de paragem ou de observação, seja feita após um determinado número de ocorrências da condição pretendida. A sinalização da ocorrência de pontos de observação é feita por mensagens, de acordo com a norma, não sendo utilizado para tal o sinal opcional EVTO. A execução passo-a-passo é realizada utilizando um registo de depuração. Uma vez configurado, cada instrução executada é imediatamente seguida por um ponto de paragem, que implementa a funcionalidade pretendida. Os recursos de depuração podem ser acedidos em tempo real, através da interface AUX. É portanto possível programar pontos de paragem ou observação, e alterar parâmetros de monitorização, com o processador em execução.

#### 4.2.4 Ambientes de Depuração

A injeção de falhas em microprocessadores conforme a proposta NEXUS pode ser efectuada utilizando depuradores comerciais. Entre as diversas opções disponíveis, encontram-se equipamentos da Lauterbach, Ashling e iSystem. Para a avaliação das suas capacidades e limitações, foram utilizados o Trace32 da Lauterbach [Lauterbach02], com a placa de desenvolvimento da Axiom [Axiom07], e o WinIdea da iSystem [iSystem04]

[iSystem06], com a placa de desenvolvimento da Phyttec [Phyttec07]. As experiências com cada conjunto foram desfasadas no espaço e no tempo, tendo o primeiro conjunto sido utilizado operacionalmente durante a já referida visita ao GSTF [GSTF07], a qual teve a duração de uma semana. Ambos os equipamentos requerem a utilização de um ambiente de depuração proprietário, sendo muito semelhantes em termos de capacidades e interface. A comunicação com o sistema anfitrião é feita via USB ou *Ethernet*, estando disponíveis para o MPC565 conectores NEXUS de 40 e 50 pinos, bem como conectores BDM. A implementação de pontos de paragem é também similar, sendo possível a sua utilização ilimitada por *software*, através de interrupções inseridas no código do sistema alvo. Os pontos de paragem por *hardware* estão limitados às capacidades da infra-estrutura READI. Existem algumas diferenças ao nível dos pontos de observação, no ambiente Trace32 é possível inseri-los, sendo disponibilizada informação sobre a sua ocorrência, em conjunto com a informação de monitorização. No caso do ambiente WinIdea não é possível aceder directamente aos pontos de observação. Por outro lado, a informação de monitorização apenas pode ser acedida pelo ambiente Trace32 com o sistema alvo parado, enquanto que no ambiente WinIdea tal já é possível a pedido do utilizador, após ter sido preenchido o *buffer* de monitorização, ou na ocorrência de um evento determinado pelo utilizador. É provável que os pontos de observação disponibilizados pela infra-estrutura READI sejam utilizados internamente para esta operação. A Figura 4.8 e a Figura 4.9 representam as montagens utilizadas em cada ambiente de injeção de falhas descrito.

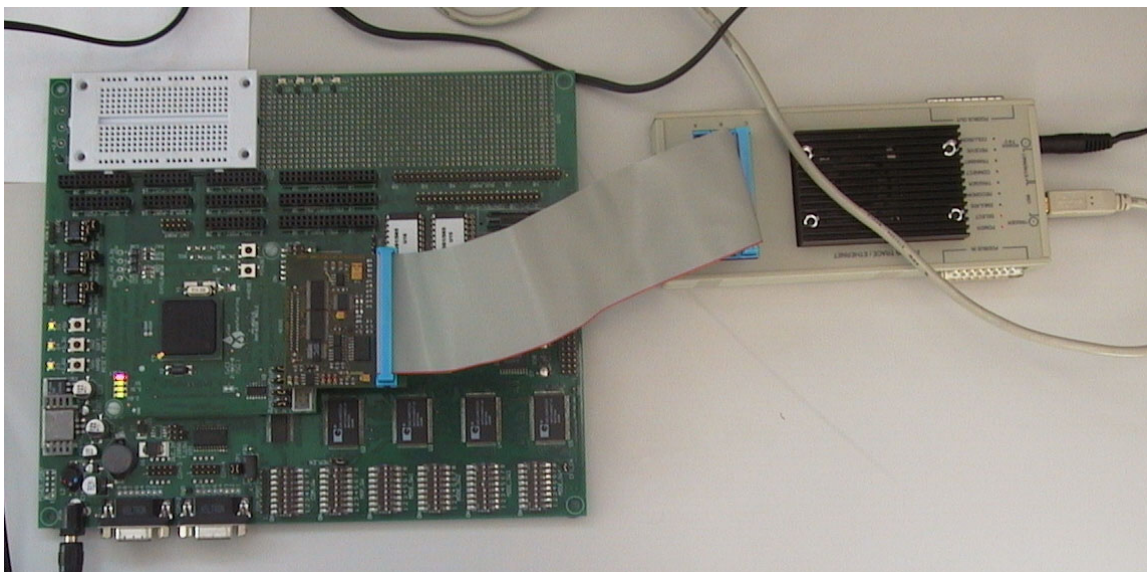


Figura 4.8 – Ambiente de injeção de falhas utilizando a combinação Axiom e Lauterbach

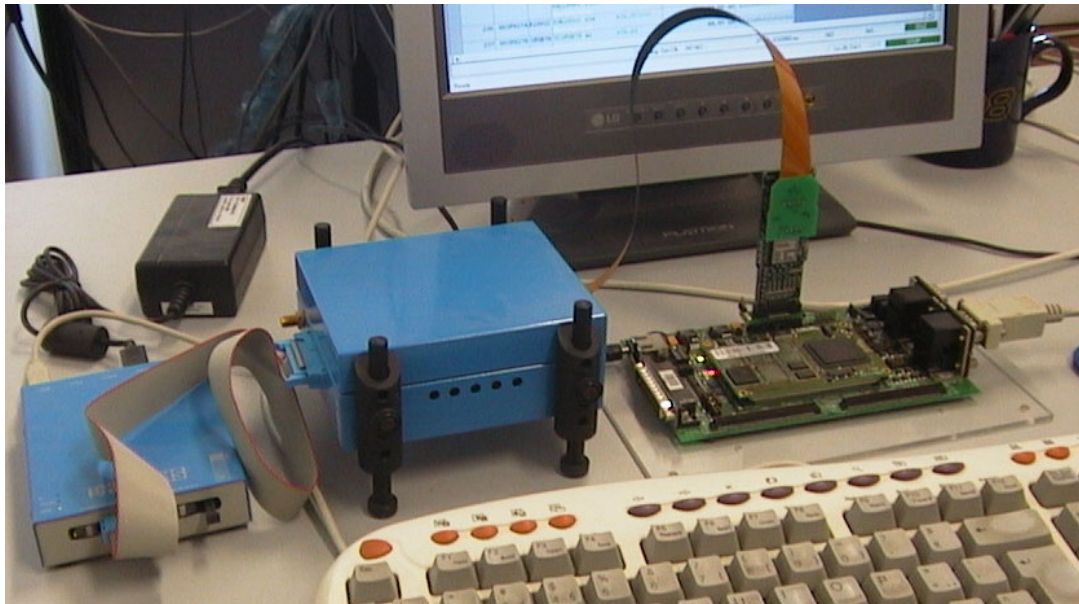


Figura 4.9 – Ambiente de injeção de falhas utilizando a combinação Phytex e iSystem

O Trace32 e o WinIdea disponibilizam também ferramentas de alto nível, permitindo a utilização integrada de compiladores, analisadores lógicos e análise estatística de resultados. Esta última pode utilizar o conhecimento da aplicação em execução para uma análise exaustiva do fluxo de programa.

#### 4.2.5 Realização das Experiências

Em ambos os ambientes de injeção de falhas, cada campanha é descrita por um único guião e composta por uma série de experiências, durante as quais a aplicação alvo é executada e uma falha específica é inserida, na ocorrência de uma determinada condição de activação. O sistema é inicializado antes de cada experiência e a aplicação é executada desde o início. O comportamento do sistema é monitorizado e a informação é registada tão compreensivelmente quanto possível, para permitir o subsequente diagnóstico e avaliação do efeito da falha.

No caso do ambiente experimental baseado na placa Axiom e depurador da Lauterbach, as experiências realizadas seguiram as seguintes directivas:

- A aplicação alvo consiste num algoritmo de controlo que opera em ciclo infinito, utilizando como entrada valores guardados previamente em memória.
- O sistema alvo encontra-se equipado com mecanismos de detecção de erros, baseados na identificação de instruções ilegais e em acessos a zonas de memória não autorizadas. A memória de dados não é protegida.

- A condição de activação baseia-se na utilização de um temporizador externo, que gera o sinal para iniciar o processo de injecção de falhas, após um intervalo de tempo pseudo-aleatório.
- Foram feitas algumas experiências para analisar a exequibilidade de utilizar pontos de observação para activação de falhas.
- As falhas podem ser inseridas em registos internos, memória de dados e memória de programa. A localização de cada falha é escolhida aleatoriamente, dentro do espaço de memória ou conjunto de registos utilizados pela aplicação.
- Quando o elemento alvo é um registo interno, o processo de injecção inicia-se por uma paragem do sistema, utilizando um sinal enviado pelo depurador.
- O diagnóstico é feito utilizando monitorização de programa, monitorização de dados e pontos de observação.
- Verificou-se ser necessário confirmar se a aplicação em execução não escrevia sobre o elemento alvo da injecção de falhas, durante este processo. O procedimento adoptado conta o número de ocorrências de pontos de observação, sendo detectado um possível conflito caso aquele seja incrementado durante a inserção de uma falha (devido a um acesso simultâneo por parte da aplicação).

A execução de cerca de 200 experiências de injecção de falhas, permitiu as seguintes conclusões:

- Verificou-se ser possível injectar falhas em registos internos do microprocessador, com paragens relativamente curtas (i.e. na ordem dos 200 ciclos de relógio) do sistema alvo. No que respeita à memória, foi possível fazê-lo em tempo real.
- Os mecanismos de detecção de erros mostraram-se razoavelmente eficientes a detectar erros em memória de programa, mas a capacidade de detecção de erros em memória de dados é praticamente nula.
- Verificou-se não ser exequível activar a injecção de falhas utilizando um ponto de observação, devido à elevada latência e a atrasos não determinísticos. A diferença entre o instante em que ocorre o ponto de observação e o instante em que é lido o valor do elemento alvo, foi sempre superior a 100 ciclos de relógio, com variações na ordem das dezenas, para as mesmas condições experimentais.
- Verificou-se ser geralmente impossível replicar experiências, quando a execução tem lugar em tempo real. Mesmo no caso de experiências idênticas e sendo o

sistema iniciado da mesma forma, obtiveram-se resultados diferentes, inclusive no valor lido para o mesmo elemento alvo.

- As experiências que utilizaram como alvo registos internos (recorrendo à paragem do sistema) também não se puderam replicar, devido a variações no instante preciso de paragem do sistema (o que provoca alterações consideráveis no conteúdo dos registos).
- A inserção de falhas em registos apresentou uma grande variação de resultados, mostrando que este tipo de alvo requer uma análise mais aprofundada, recorrendo a um elevado número de experiências.
- A inserção de falhas em memória de programa provocou geralmente a detecção de um erro ou uma avaria não detectada (i.e. instrução ilegal).
- A inserção de falhas em memória de dados provocou avarias em cerca de 65% dos casos, consistindo em valores errados nas saídas. Nenhum dos mecanismos de detecção de erros permitiu a sua detecção.
- Utilizando aplicações alvo relativamente complexas (mais de 300 Kbytes de memória), verificou-se que cerca de 15% das experiências tiveram de ser descartadas, devido a resultados inconclusivos, provocados por um acesso a memória durante o processo de injecção de falhas.

No caso do ambiente experimental baseado na placa Phytex e no depurador da iSystem, as experiências executadas seguiram as seguintes directivas:

- As aplicações alvo utilizadas consistem em operações matriciais (adição, multiplicação e comparação) que duram um intervalo de tempo finito, utilizando como entrada valores guardados previamente em memória.
- O sistema encontra-se equipado com mecanismos de detecção de erros, nas operações de dados, incluídos na própria aplicação alvo (detecção de erros por *software*).
- Foram utilizadas duas condições de activação. Uma baseada na utilização de um temporizador análogo ao sistema anterior e outra baseada no início da monitorização, provocada pela ocorrência de um evento no sistema alvo (este início da monitorização é detectado pelo depurador e pode ser utilizado como sinal de activação). Os eventos testados consistiram na execução de linhas de código e em acessos a determinadas posições de memória.

- As falhas podem ser inseridas em registos internos, memória de dados e memória de programa. A localização de cada falha é escolhida aleatoriamente dentro do espaço de memória ou do conjunto de registos utilizado pela aplicação.
- Quando o elemento alvo é um registo interno, o processo de injecção utiliza um ponto de paragem para suspender o sistema.
- O diagnóstico é feito por monitorização de programa e monitorização de dados. A indisponibilidade de pontos de observação requer um maior recurso à monitorização de dados. Em alternativa, o diagnóstico também poderia ser efectuado por comparação dos dados de saída (em memória) com os resultados esperados.
- Não estando disponíveis pontos de observação, a análise das perturbações impostas pelo processo de injecção de falhas, sobre a aplicação alvo, só pode ser efectuada a partir da análise dos resultados. Este processo requer conhecimento do código executado e o recurso à monitorização de dados, sendo consideravelmente demorado.

A execução de cerca de 500 experiências de injecção de falhas<sup>17</sup>, permitiu as seguintes conclusões:

- Também neste caso se verificou a exequibilidade da injecção de falhas em tempo real ou com suspensão curta, de forma análoga e com desempenho semelhante ao caso anterior. No entanto, para aplicações com tempo de execução relativamente curto (como as usadas nestas experiências), a duração do processo de injecção de falhas revelou-se excessiva.
- Utilizando aplicações alvo relativamente simples (50k de memória), em termos de código, mas utilizando uma razoável quantidade de dados (1 Mbyte de memória), verificou-se que cerca de 30% das experiências tiveram de ser descartadas, devido a resultados inconclusivos provocados por um acesso à memória durante o processo de injecção de falhas.
- Quando se utilizaram vectores de dados mais pequenos (menos de 256 Kbytes), verificou-se que na maioria dos casos a aplicação terminava a execução antes do processo de injecção de falhas estar concluído.

---

<sup>17</sup> O número de experiências executado com o equipamento Lauterbach / Axiom foi mais reduzido devido a restrições logísticas na sua utilização.

- A inexistência de mecanismos de detecção de erros em memória de programa, faz com que praticamente todas as injeções (mais de 99% dos casos) de falhas nesta área de memória causem uma avaria não detectada (i.e. instrução ilegal).
- A impossibilidade de utilizar pontos de observação para diagnóstico dificultou esta tarefa, tornando mais difícil a identificação dos casos em que a execução da aplicação alvo interfere com o processo de injeção de falhas.
- A inserção de falhas em memória de dados provocou avarias em cerca de 45% dos casos e a detecção de erros em 35% dos casos. Como a detecção de uma avaria provocava a suspensão da execução, não houve casos de erros detectados e simultaneamente avaria.

Uma análise comparativa dos resultados obtidos experimentalmente, utilizando ambos os ambientes de injeção de falhas, permite concluir que a metodologia utilizada é eficiente, mas apresenta algumas limitações. Em particular, verificou-se que o tipo de aplicações alvo, o eventual sistema de detecção de erros e os recursos de depuração disponíveis, afectam a condução das experiências e a qualidade dos resultados. O trabalho experimental efectuado com os dois ambientes permitiu uma análise das vantagens e limitações da injeção de falhas via infra-estruturas OCD, utilizando componentes COTS (*Commercial-Off-The-Shelf*) em conformidade com a proposta NEXUS.

#### 4.2.6 Análise de Resultados

Os resultados obtidos permitiram confirmar a maioria das potencialidades esperadas e simultaneamente identificar alguns problemas potenciais. Estes últimos incluem as previsíveis lacunas em termos de cobertura, mas também algumas limitações nos métodos de activação de falhas e no desempenho em geral. Verificou-se que é possível utilizar infra-estruturas OCD para inserir falhas em espaço de memória, sem afectar a aplicação em execução no microprocessador. Simultaneamente, confirmou-se que a utilização da informação de monitorização é uma solução eficiente para analisar o fluxo do programa e acessos a dados, antes e depois da activação de uma falha, e assim diagnosticar os seus efeitos.

No entanto, as limitações impostas pelo sistema operativo presente no sistema anfitrião (i.e. *Windows 2000*) e pelo canal de comunicações com o depurador, causam um

estrangulamento na transferência de dados e aumentam consideravelmente o tempo necessário para cada operação de memória. Na prática, a leitura do conteúdo de uma posição de memória e a escrita de um valor modificado, requerem tempos na ordem dos milissegundos. Considerando que o sistema alvo opera a 50 MHz, este atraso permite que a aplicação escreva na mesma célula nesse intervalo, invalidando o modelo de falhas utilizado. Se o problema descrito ocorrer, a experiência não estará a inserir na célula de memória uma falha do tipo inversão de bit simples, mas antes um número indeterminado de inversões de bit. A magnitude do problema depende da aplicação alvo e da área de memória utilizada para a experiência.

O sinal de activação apresenta problemas adicionais. Verificou-se que a utilização de informação de monitorização ou de pontos de observação não é fiável, devido aos atrasos de comunicação e, principalmente, ao facto destes não serem determinísticos. A activação externa, utilizando um temporizador, torna praticamente impossível sincronizar a inserção da falha com os eventos da aplicação alvo. Este facto diminui a controlabilidade, tornando praticamente impossível a replicação de experiências.

Os problemas referidos não são directamente atribuíveis às capacidades da infra-estrutura OCD, mas antes às ferramentas e ambiente de injecção de falhas utilizados. Estes não permitem algumas funcionalidades importantes, por não serem necessárias para a depuração, mas que são fundamentais para a injecção de falhas. Minimizar o intervalo de tempo necessário para a injecção de uma falha, minimiza também a probabilidade da aplicação em execução escrever sobre o elemento utilizado como alvo. Adicionalmente, a transferência da totalidade ou de parte do processo de decisão, que medeia entre a activação de uma falha e a sua inserção, para um elemento fisicamente mais próximo do alvo, permitiria diminuir os atrasos e tornar a activação mais rápida, fazendo com que o depurador reagisse automaticamente a uma ocorrência relevante, sinalizada pela infra-estrutura OCD.

### ***4.3 Propostas de Aperfeiçoamento***

#### **4.3.1 Objectivos e Abordagem**

As propostas de aperfeiçoamento apresentadas nesta secção, visam lidar com as limitações identificadas na injecção de falhas em tempo real, utilizando infra-estruturas OCD, em conformidade com a proposta NEXUS. Para tal, são definidos objectivos particularmente importantes para a verificação e validação de características de confiabilidade de sistemas de

tempo real, que não são completamente atingíveis se utilizarmos apenas as soluções disponíveis comercialmente. Os objectivos que se pretendem alcançar são:

- Controlo preciso da localização da(s) falha(s) e do seu instante de ocorrência com a possibilidade de replicar experiências;
- Observabilidade dos efeitos das falhas, de forma a permitir um diagnóstico completo dos seus efeitos;
- A intrusividade mínima no sistema alvo e nas aplicações em execução;
- Operação em tempo real (i.e. sem parar a aplicação);
- Cobertura total do modelo de falhas do tipo inversão de bit individual (i.e. tornar possível a injecção de falhas em todo o espaço de memória, em qualquer instante).

Tendo em conta a complexidade do problema e a diversidade de requisitos, foram consideradas quatro situações diferentes, de acordo com o equilíbrio entre os requisitos de confiabilidade e as modificações necessárias à infra-estrutura OCD presente no sistema alvo. Estas quatro situações são as seguintes:

**Nenhuma alteração** – Pretende-se que a infra-estrutura OCD não seja alterada de nenhuma forma, para efeitos de injecção de falhas. As propostas nesta área limitam-se a alterações na metodologia e em elementos exteriores ao sistema alvo.

**Alteração de desempenho** – As únicas alterações introduzidas são ao nível do desempenho, mantendo-se todas as funcionalidades originais. As propostas nesta área baseiam-se em modificações à infra-estrutura OCD, e nos elementos que comunicam com ela, apenas em termos de desempenho, não requerendo alterações funcionais e mantendo completa a conformidade com a proposta de norma.

**Alteração de funcionalidade** – As alterações são de carácter funcional mantendo-se a conformidade com a proposta de norma, sem aumentar a intrusividade. Para este efeito são introduzidas funcionalidades adicionais na infra-estrutura OCD e elementos de apoio à injecção de falhas. O microprocessador mantém-se inalterado

**Alteração intrusiva** – Alterações na infra-estrutura OCD, que visam acrescentar funcionalidades necessárias à injecção de falhas, cujo custo/benefício justifica o acréscimo de intrusividade. Permite a inclusão de funcionalidades críticas, com eventual custo em termos de intrusividade ou de não conformidade NEXUS. Poderão ser necessárias alterações ao sistema alvo, particularmente na forma como este interage com a infra-estrutura OCD.

A principal alteração à metodologia baseia-se na pré-determinação do valor que estará presente na célula de memória (ou registo) onde se pretende inserir a falha, de forma a obter o melhor desempenho. Esta pré-determinação só será possível quando se verificarem simultaneamente as seguintes condições:

- Existe um conhecimento completo do fluxo do programa até ao instante de injecção da falha.
- Não são utilizados sinais ou dados externos ao sistema alvo (ou, caso o sejam, é possível observar ou controlar o seu valor).
- O instante em que a falha é inserida pode ser precisamente controlado.

Se esta pré-determinação não puder ser executada, torna-se necessário ler o conteúdo da célula de memória ou do registo alvo, imediatamente antes da injecção da falha, de forma a determinar o valor a inserir, para emular um SEU, tal como na metodologia original. Este processo é muito importante em termos de desempenho e deverá ser tomado em consideração em qualquer cenário de injecção de falhas.

#### 4.3.2 Depuração Aperfeiçoada

A proposta para melhorar o desempenho, sem alterações à infra-estrutura OCD, requer modificações aos seus elementos externos, nomeadamente ao sistema anfitrião e ao depurador. Um depurador que se pretenda utilizar para a injecção de falhas, em tempo real, terá sempre de apresentar um bom desempenho, particularmente no que respeita à velocidade de comunicação. Deverá ser executada por *hardware* o maior número possível de tarefas, sendo de primordial importância a capacidade de reacção automática. No caso de equipamento destinado à depuração de sistemas conforme a proposta NEXUS, a interface está pré-definida, e a frequência de operação é normalmente limitada pelo sistema alvo.

O desempenho de um ambiente de injecção de falhas pode ser consideravelmente melhorado, se for possível acelerar o processo de inserção. O recurso a infra-estruturas OCD implica a existência de dois canais de comunicação (do sistema anfitrião para o depurador e deste para a infra-estrutura OCD), os quais podem impor atrasos.

Este problema pode ser reduzido se eliminarmos a comunicação entre o depurador e sistema anfitrião, uma vez iniciado o processo de injecção de falhas. Esta solução será possível se:

- O depurador operar autonomamente;

- O depurador estiver pré-programado com todos os passos necessários para a injeção de falhas;
- A leitura de dados para diagnóstico não tiver de passar entre o depurador e o sistema anfitrião, durante a operação de injeção de falhas.

A capacidade de operação autónoma é tecnicamente exequível, sendo necessárias capacidades de memória no depurador, de forma a lidar com a informação requerida para a execução de pelo menos uma experiência de injeção de falhas. Os dados para diagnóstico podem ser guardados no depurador, ou enviados directamente para o sistema anfitrião (i.e. via leitura da memória). Nestas condições, o depurador torna-se responsável pela execução da campanha de injeção de falhas, como se representa na Figura 4.10.

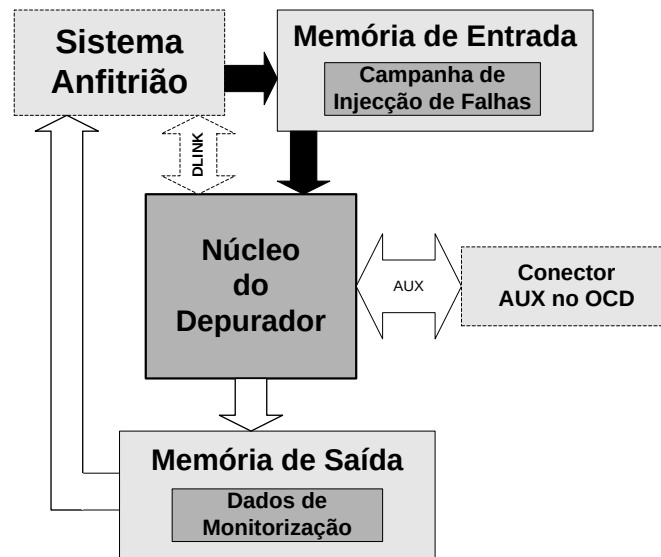


Figura 4.10 – Depurador autónomo otimizado para injeção de falhas

Um depurador autónomo deverá incluir **memória de entrada**, onde será armazenada a informação relativa ao processo de injeção de falhas. Apesar de não ser estritamente necessária, uma **memória de saída** será a opção mais adequada para sistemas de injeção de falhas por OCD, com capacidades de monitorização, como é o caso dos sistemas NEXUS. Nestes, a informação de monitorização é enviada para o depurador, a partir da infra-estrutura OCD, pelo que deverá ser arquivada em memória, para posterior análise. Opcionalmente, o acesso a esta memória de saída poderá ser permitido ao sistema anfitrião (ou outro dispositivo), desde que tal não afecte o desempenho do processo de injeção de falhas. A memória de saída poderá arquivar a informação de monitorização, respostas da infra-estrutura e mensagens de erro. A quantidade de informação gerada pela infra-estrutura varia

de acordo com a aplicação em execução, o nível de detalhe da monitorização e as operações realizadas durante a injecção de falhas. Eventualmente, será necessário algum compromisso, dependendo da capacidade de memória disponível. Em termos gerais, quanto maior a memória de saída, maior o nível de detalhe e/ou tempo de monitorização possível. Os dados assim arquivados permitirão a reconstituição do fluxo de execução do programa. Poderá também ser utilizada a monitorização de acesso a dados ou operações de leitura de dados intermédias, para obter um maior nível de observabilidade.

Desta forma, o **núcleo do depurador** actuará como controlador do processo, lendo comandos da memória de entrada e interpretando as respostas recebidas a partir do sistema alvo. Estas poderão estar relacionadas com o comando em execução e activar uma reacção do depurador (e.g. um comando que indique ao depurador para esperar por uma determinada ocorrência), serem guardadas na memória de saída ou simplesmente ignoradas, se não forem relevantes. Esta capacidade reactiva é extremamente importante para a injecção de falhas em tempo real e é uma funcionalidade que não está disponível na maioria dos depuradores comerciais actuais. Um controlador de comunicações, incluído no núcleo, fará a tradução dos comandos lidos da memória para mensagens normalizadas, a enviar para a infra-estrutura OCD, bem como a gestão do porto de comunicações (AUX) e o arquivamento das mensagens recebidas. A dimensão do barramento de comunicações, entre o depurador e a infra-estrutura OCD, define o tempo necessário para a transmissão de cada mensagem e determina o desempenho possível com esta solução.

O **sistema anfitrião** é utilizado apenas para carregar os guiões experimentais (campanhas de injecção de falhas) na memória de entrada do depurador e para mais tarde retirar os dados de monitorização, não tomando parte no processo de injecção de falhas propriamente dito. O núcleo do depurador é um componente com capacidade de processamento, que lê instruções da memória de entrada, controla a execução das mesmas e gere as comunicações com a infra-estrutura OCD, podendo ser necessária alguma capacidade de gestão de erros. O controlo directo não é fundamental para realizar a injecção de falhas, mas poderá ser incluído, pois permite uma maior flexibilidade de depuração e de execução de experiências sem recurso a guiões.

Adicionalmente, poderão incluir-se capacidades de manuseamento lógico de informação no depurador, para automatizar operações típicas. Para a injecção de falhas do tipo inversão de bit, será importante proporcionar operações do tipo XOR com uma máscara, que proporcionarão o valor a escrever na memória. Esta funcionalidade permitirá a inserção de falhas em tempo real, de forma autónoma e sem conhecimento prévio do valor a inserir.

Um depurador com as funcionalidades descritas está optimizado para a injeção de falhas e deverá permitir um melhor desempenho, sem alterações ao sistema alvo ou à infra-estrutura OCD. Como foi referido, o barramento de comunicações entre o depurador e a infra-estrutura OCD poderá constituir uma barreira ao aumento do desempenho. Este barramento poderá ser ampliado, mantendo a conformidade com a norma. É importante referir que os componentes comerciais existentes fornecem normalmente a possibilidade de utilizar um número mais elevado de sinais, para situações em que os requisitos de desempenho o justifiquem. Utilizando o mesmo princípio, é possível que os requisitos de verificação e validação da confiabilidade de um sistema, justifiquem a utilização de uma infra-estrutura OCD com maior capacidade de transmissão de dados.

Desta forma, o depurador externo, que terá de ser compatível com o porto de acesso de maior capacidade, poderá utilizar a maior largura de banda, para que o processo de injeção de falhas seja mais rápido e eficiente. Dependendo da dimensão e tipo do aumento de desempenho, serão afectadas diferentes partes do processo de injeção de falhas. Um aumento da largura de banda disponível, à entrada da infra-estrutura OCD, permitirá uma escrita de dados mais rápida, reduzindo o tempo de configuração e o tempo de escrita dos valores com falha. O aumento da largura de banda disponível, à saída da infra-estrutura OCD, terá efeitos principalmente ao nível da monitorização, o que explica que se recorra a esta solução, para cenários de depuração mais exigentes. A monitorização de programa, e particularmente a monitorização de dados, são por vezes limitadas pelas restrições da interface entre a infra-estrutura OCD e o exterior. Uma maior capacidade permitirá aumentar o volume de dados disponibilizado, e melhorando por essa via as capacidades de monitorização. A título de exemplo, a área de memória cujos acessos podem ser seguidos por monitorização de dados, está directamente relacionada com a taxa máxima de transferência de dados para o exterior e, caso esta não seja suficiente, serão inevitáveis perdas de informação, com prejuízo do processo de depuração. Adicionalmente, a leitura de valores de memória, ou de registos internos do microprocessador, também é acelerada, com a subsequente melhoria do processo de leitura dos valores utilizados na injeção de falhas, se não for possível utilizar a pré-determinação do valor com falha.

De uma forma geral, o aumento da largura de banda entre o depurador e a infra-estrutura OCD terá sempre vantagens, do ponto de vista da injeção de falhas. Tal como no caso da depuração, a solução efectivamente implementada será sempre um compromisso entre os requisitos da avaliação da confiabilidade e os limites aceitáveis da sobrecarga imposta ao sistema alvo.

### 4.3.3 OCD-FI

A melhoria do desempenho do processo de injeção de falhas também pode ser obtida transpondo algumas funcionalidade para o interior da infra-estrutura OCD, recebendo esta solução a designação de depuração e injeção embutida (OCD-FI – *On Chip Debug and Fault Injection*) [Fidalgo06b]. O objectivo desta abordagem será o aumento do desempenho, pela redução dos tempos de comunicação e eventualmente pelo acrescento de funcionalidades. Em termos de intrusividade, será possível manter o nível original da infra-estrutura, ou optar pela inclusão de funcionalidades mais intrusivas, se tal for considerado útil. A utilização de lógica adicional deve ser sempre encarada com prudência, particularmente em elementos dedicados a teste, depuração ou funções afins. O aumento da sua área deverá ser sempre mantido dentro de níveis aceitáveis (dependente da função do sistema) e avaliado em comparação com os benefícios fornecidos.

É razoável supor que qualquer infra-estrutura OCD NEXUS, será realizada de forma modular, para facilitar o seu desenvolvimento e posterior aperfeiçoamento. Desta forma, as modificações à infra-estrutura poderão ser efectuadas através da adição de um módulo dedicado a operações de injeção de falhas. Este módulo poderá interagir com outros elementos da infra-estrutura, reutilizando funcionalidades de depuração. Estando já disponíveis as capacidades de detecção de eventos e de leitura e escrita em memória e registos, a injeção de falhas apenas deverá necessitar de um módulo que possa aproveitar os sinais gerados dentro da infra-estrutura OCD, para determinar o instante em que a injeção de falhas deve ser efectuada, podendo então controlar os recursos necessários à sua execução. Um módulo deste tipo, designado como módulo FI (*Fault Injection*), deverá ser embutido na infra-estrutura OCD, tal como apresentado na Figura 4.11, e reutilizar parte das suas funcionalidades e das suas interligações com o exterior.

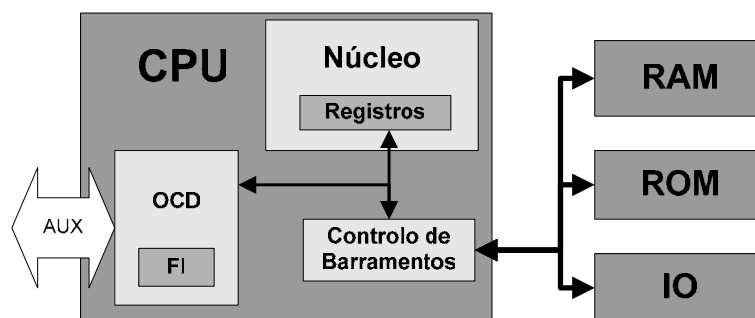


Figura 4.11 – Infra-estrutura OCD e módulo FI

O módulo de injeção de falhas deverá automatizar a inserção de uma falha, no instante em que tal é pretendido. Esta operação requer uma boa observabilidade, que permita determinar o instante correcto, e controlabilidade suficiente para permitir a inserção da falha. Atendendo a que esta inserção requer, no mínimo, uma operação de escrita, o módulo de injeção de falhas deverá proporcionar esta funcionalidade. Estes comentários evidenciam a importância da metodologia a utilizar para a injeção das falhas. Se for possível a pré-determinação do valor a inserir, bastará que o módulo de injeção de falhas possa controlar a inserção. Se a pré-determinação não for possível, o módulo de injeção de falhas só poderá operar autonomamente se for capaz de executar a determinação do valor com falha. Esta capacidade requer a pré-definição do modelo de falhas, existindo necessariamente um compromisso entre a versatilidade do modelo e a lógica adicional necessária. No caso geral, o processo de monitorização não deverá ser afectado e as mensagens com ele relacionadas não deverão ser alteradas.

A forma mais simples de executar a operação pretendida consiste em efectuar a pré-programação prévia da infra-estrutura OCD-FI, de forma a que informação relativa à injeção de falhas não interfira com o processo de inserção propriamente dito. Dependendo das funcionalidades necessárias, terão de estar disponíveis várias operações na infra-estrutura de base. A capacidade de acesso aos elementos alvo da injeção de falhas em tempo real é fundamental, limitando a cobertura que é possível atingir. Adicionalmente, deverá ser possível carregar em registos o valor e o endereço da operação de escrita, bem como controlar a execução desta operação. A informação poderá assim ser pré-carregada na infra-estrutura OCD-FI e executada apenas quando for necessária a injeção da falha. O modo normal de funcionamento utilizará uma ocorrência da aplicação em execução, detectada pela infra-estrutura OCD, sendo necessário utilizar pontos de observação ou mecanismos de detecção similares. Esta possibilidade encontra-se prevista na proposta NEXUS, não sendo no entanto obrigatória. Em alternativa, poderá ser utilizado um sinal exterior, para activar a inserção da falha, não sendo necessário o ponto de observação.

Se as funcionalidades indicadas estiverem disponíveis na infra-estrutura OCD, a implementação do módulo FI não deverá implicar alterações substanciais. A infra-estrutura OCD-FI proposta e a metodologia que lhe está associada foram projectadas para a injeção de falhas em elementos de memória, com intrusividade mínima e desempenho máximo. Nos casos em que os requisitos de confiabilidade sejam particularmente elevados, pode ser aceitável e mesmo desejável, a inclusão de capacidades acrescidas, que justifiquem uma maior intrusividade ou considerável lógica adicional. Estas abordagens implicarão soluções

mais especializadas para cada caso, ajustadas ao sistema alvo específico. É de supor que algumas dessas situações sejam recorrentes, como por exemplo a necessidade de uma maior cobertura de falhas, ou de lidar com mecanismos de tolerância a falhas incluídos no *hardware*. A utilização de uma infra-estrutura OCD-FI, otimizada para a injecção de falhas, pode ser estendida para os casos em que existem requisitos especiais de verificação de confiabilidade. Nestes casos poderão ser implementadas variantes da versão de base, que permitam lidar com problemas específicos, ao nível do modelo de falhas ou da arquitectura do sistema alvo. A utilização de extensões será particularmente importante quando é imprescindível uma cobertura de falhas que inclua os registos internos do microprocessador, ou quando seja necessário injectar falhas em elementos de memória protegidos por mecanismos de tolerância a falhas por *hardware*.

#### **4.4 Resumo e Conclusões**

Uma solução simples será sempre preferível para qualquer problema, desde que efectivamente funcione. No caso da injecção de falhas isto é particularmente verdade, pois os sistemas mais complexos e dispendiosos nem sempre fornecem os melhores resultados. No caso dos sistemas computacionais, se for possível obter resultados adequados com recurso a alguma intrusividade ou redução de desempenho, e se tal for admissível para os objectivos pretendidos, então essa será provavelmente a melhor solução. No entanto, se um determinado sistema alvo executar em tempo real, não sendo aceitável modificar o código em execução (reduzindo a aplicabilidade do SWIFI), ou mesmo suspender o seu funcionamento, então a injecção de falhas exige uma metodologia mais complexa. Neste caso, poderá ser necessário recorrer a técnicas físicas complexas e onerosas (e.g. radiação, laser) ou a infra-estruturas OCD. É neste cenário operacional que a utilização de infra-estruturas NEXUS se torna particularmente interessante, pois permite algumas vantagens importantes, nomeadamente:

- Operação em tempo real, incluindo leitura e escrita de dados e monitorização.
- Controlabilidade elevada, permitindo introduzir falhas no instante e local pretendido com elevada precisão, e permitindo a replicação de experiências.

- Reutilização do “equipamento” de injeção de falhas para diagnóstico dos seus efeitos, visto ambas as operações serem executáveis através da infra-estrutura OCD.

O ambiente e metodologias de injeção de falhas propostos, foram projectados tendo por base o trabalho anterior com este tipo de infra-estruturas e tendo em vista a escalabilidade das soluções propostas. Mais uma vez, a solução adequada para um determinado problema poderá não ser necessariamente a mais complexa. Todas as soluções visam reduzir a sobrecarga temporal das comunicações de dados e eventualmente colocar o processo de tomada de decisões e manipulação de dados mais próximo do seu alvo operacional. Este tipo de solução implica sempre alguma sobrecarga, mas poderá representar a melhor escolha para determinados tipos de sistema alvo ou cenários de injeção de falhas.

As propostas apresentadas baseiam-se numa metodologia comum, escalável em quatro níveis de acordo com o impacto sobre o sistema alvo, que poderá até não existir (depurador optimizado), mas que poderá também levar a soluções de confiabilidade personalizadas, requerendo estas modificações importantes no sistema alvo. A solução indicada para cada situação dependerá do sistema, da sua função e da importância e requisitos da injeção de falhas. No limite, a solução mais simples poderá ser a preferível, se for perfeitamente aceitável suspender o sistema alvo para injectar falhas, não sendo necessária nenhuma modificação ou equipamento especial.



*“The solution of every  
problem is another problem.”*

Goethe

**Capítulo 1**

**Capítulo 2**

**Capítulo 3**

**Capítulo 4**

**Capítulo 5**

**Capítulo 6**

**Capítulo 7**

**Referências**

**Anexo**

# Solução Proposta



## 5 Solução Proposta

Este capítulo apresenta o trabalho desenvolvido para validar as propostas do capítulo anterior. Numa primeira parte é descrita a metodologia e as adaptações necessárias para as variantes do ambiente de injeção de falhas. Subsequentemente é descrito o sistema alvo utilizado e a infra-estrutura OCD projectada. Seguem-se as descrições dos componentes desenvolvidos para injeção de falhas e das modificações aos componentes existentes. São apresentados o depurador otimizado para injeção de falhas, as modificações necessárias para aumentar a velocidade de comunicação entre este depurador e a infra-estrutura OCD, e finalmente a infra-estrutura OCD-FI. O capítulo termina com a análise da expansibilidade desta solução a problemas de injeção de falhas mais complexos, sendo apresentados dois casos de estudo, desenvolvidos para o efeito. As configurações específicas utilizadas, os respectivos parâmetros numéricos e os resultados obtidos, são apresentados no capítulo seguinte. Os diagramas completos dos modelos desenvolvidos e utilizados neste trabalho são disponibilizados no CD anexo.

### 5.1 Metodologia

O objectivo deste trabalho consiste em desenvolver um ambiente de injeção de falhas capaz de operar em tempo real, de forma eficiente e não intrusiva. Consideram-se variantes da mesma metodologia de base, de forma a permitir vários pontos de equilíbrio entre as alterações necessárias e os ganhos possíveis.

A metodologia proposta pressupõe um modelo composto por falhas individuais, do tipo inversão de bit, em memória e em registos internos do microprocessador. As campanhas de injeção de falhas consistirão num conjunto de operações, como explicado no capítulo anterior, cujo número será definido à partida e limitado apenas pelos recursos disponíveis, particularmente a memória do depurador.

Apesar de focada na operação em tempo real, a metodologia proposta permite também a injeção de falhas com o sistema alvo suspenso. Neste caso tenta-se tirar partido da elevada velocidade da interface, de forma a que a injeção seja executada o mais rapidamente possível, permitindo assim encurtar a duração deste processo.

Qualquer operação de injeção de falhas, em sistemas de tempo real, terá de lidar com dois requisitos possíveis e/ou necessários. Deverá ser avaliado se é aceitável parar a execução do

sistema alvo para injectar falhas e se é possível determinar à partida a falha que se pretende inserir, ou se esta terá de ser determinada em execução. Estes dois requisitos poderão ser simultâneos e apresentam considerável impacto no desempenho e potencialidades do ambiente de injeção de falhas.

Caso seja aceitável suspender a execução do sistema alvo, poderá haver um limite para o tempo máximo de suspensão, como sucede quando uma falha deve ser injectada num intervalo de tempo definido por um componente externo, que não é suspenso em conjunto com o sistema alvo. No caso em que a suspensão não é limitada no tempo, as soluções propostas apresentam ainda algumas vantagens, nomeadamente na activação da falha e na respectiva sincronização, e em termos de desempenho, quando forem importantes parâmetros como o número de falhas por segundo. Nos casos em que a injeção de falhas pode ser executada com o sistema alvo suspenso, sem limites temporais apertados, e quando o desempenho não é particularmente importante, a utilização de pontos de paragem para activação e inserção de falhas (com o sistema alvo parado) é perfeitamente adequada, podendo não se justificar a utilização de soluções mais complexas.

A pré-determinação<sup>18</sup> do valor com falha é o segundo requisito que poderá ser exequível, sendo particularmente importante para a injeção de falhas em tempo real. Permite eliminar as operações de leitura do elemento onde se pretende injectar a falha e o subsequente manuseamento de dados (i.e. inversão de bit), com considerável ganho em termos de desempenho e redução do risco de a operação ser mal executada. No entanto, esta pré-determinação exige que:

- Seja conhecido, ou possível de determinar, o código executado pela aplicação alvo, até ao instante da injeção da falha.
- Seja possível controlar com precisão o instante em que a falha é inserida.
- Todas as operações de leitura e escrita do microprocessador alvo possam ser monitorizadas e/ou controladas.

A pré-determinação é particularmente recomendada para os casos em que a sua utilização é bastante facilitada ou mesmo garantida, como por exemplo a injeção de falhas em zonas de memória onde a aplicação alvo não escreve, sendo a injeção de falhas em memória de programa um caso particular desta situação.

---

<sup>18</sup> A pré-determinação do valor com falha será designada de forma abreviada por Predet em algumas partes deste documento (tabelas).

Caso seja possível a injeção de falhas com o sistema alvo parado, a pré-determinação apenas será necessária caso existam limitações no tempo em que essa paragem se efectua. A metodologia proposta subdivide-se portanto de acordo com as variantes apresentadas na Tabela 5.1.

Tabela 5.1 – Variantes da metodologia proposta e seus efeitos

| Variante | Pré-requisito |        | Característica   |            |                             |
|----------|---------------|--------|------------------|------------|-----------------------------|
|          | Tempo Real    | Predet | Controlabilidade | Desempenho | Intrusividade <sup>19</sup> |
| #1       | Não           | Não    | Excelente        | Baixo      | Elevada                     |
| #2       | Não           | Sim    | Excelente        | Médio      | Elevada                     |
| #3       | Sim           | Não    | Baixa            | Bom        | Baixa                       |
| #4       | Sim           | Sim    | Média            | Excelente  | Baixa                       |

As soluções propostas terão de ser adaptadas a cada um dos casos referidos anteriormente, garantindo-se que a metodologia resultante da combinação dos requisitos é flexível e escalável. A metodologia de base foi apresentada no capítulo anterior e é adaptável às diferentes situações, residindo as principais diferenças nos elementos onde são executadas as operações. São considerados três procedimentos de injeção de falhas para as quatro configurações apresentadas no capítulo anterior. Temos assim um procedimento para injeção de falhas com suspensão do sistema alvo e dois procedimentos para a injeção de falhas em tempo real, um via infra-estrutura OCD e o outro via infra-estrutura OCD-FI. O procedimento que utiliza a infra-estrutura OCD é idêntico para a versão base ou para as versões que permitem um desempenho adicional, através do aumento da velocidade de comunicação dos portos AUX. Estes três procedimentos apresentam duas variantes, dependendo da possibilidade de pré-determinar o valor com falha. A Tabela 5.2 apresenta o procedimento utilizado para a injeção de falhas com suspensão do microprocessador. A Tabela 5.3 apresenta o procedimento para a injeção de falhas em tempo real através da infra-estrutura OCD, sendo o procedimento independente da dimensão do barramento de comunicações entre o depurador e a infra-estrutura OCD. Finalmente, a Tabela 5.4 apresenta o procedimento para a injeção de falhas utilizando uma infra-estrutura OCD-FI.

<sup>19</sup> Neste caso, uma intrusividade elevada representa um efeito negativo.

Tabela 5.2 – Procedimento para injeção de falhas com suspensão do sistema alvo

| #  | Passo                      | Descrição                                                                                                                                                                                                                                                                   |
|----|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1  | Inicialização              | O programa de injeção de falhas é carregado na memória de entrada do depurador<br>Um ponto de paragem é programado no sistema alvo (via OCD)<br>O microprocessador é reinicializado<br>A aplicação é executada desde o início<br>O depurador espera por um ponto de paragem |
| 2  | Activação                  | Um ponto de paragem é atingido e o microprocessador é suspenso<br>A suspensão é comunicada ao depurador                                                                                                                                                                     |
| 3  | Determinação (Predet.)     | O depurador obtém o vector com falha da memória de entrada                                                                                                                                                                                                                  |
| 3+ | Determinação (sem Predet.) | O depurador ordena a leitura do conteúdo do elemento alvo e modifica o valor de um ou mais bits, gerando o vector com falha                                                                                                                                                 |
| 4  | Inserção                   | O depurador ordena a escrita do vector com falha no elemento alvo                                                                                                                                                                                                           |
| 5  | Conclusão                  | O depurador ordena o retomar da execução pelo sistema alvo                                                                                                                                                                                                                  |

Tabela 5.3 – Procedimento para injeção de falhas em tempo real via OCD

| #  | Passo                      | Descrição                                                                                                                                                                                                                                                                         |
|----|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1  | Inicialização              | O programa de injeção de falhas é carregado na memória de entrada do depurador<br>Um ponto de observação é programado no sistema alvo (via OCD)<br>O microprocessador é reinicializado<br>A aplicação é executada desde o início<br>O depurador espera por um ponto de observação |
| 2  | Activação                  | Um ponto de observação é atingido e sinalizado ao depurador                                                                                                                                                                                                                       |
| 3  | Determinação (Predet.)     | O depurador obtém o vector com falha da memória de entrada                                                                                                                                                                                                                        |
| 3+ | Determinação (sem Predet.) | O depurador ordena a leitura do conteúdo do elemento alvo e modifica o valor de um ou mais bits, gerando o vector com falha                                                                                                                                                       |
| 4  | Inserção                   | O depurador ordena a escrita do vector com falha no elemento alvo                                                                                                                                                                                                                 |
| 5  | Conclusão                  | Sem operação                                                                                                                                                                                                                                                                      |

Tabela 5.4 – Procedimento para injeção de falhas em tempo real via OCD-FI

| #  | Passo                      | Descrição                                                                                                                                                                                                                                                                                                                                             |
|----|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1  | Inicialização              | O programa de injeção de falhas é carregado na memória de entrada do depurador<br>Um ponto de observação é programado no sistema alvo (via OCD)<br>O módulo FI é activado e configurado com a informação necessária<br>O microprocessador é reinicializado<br>A aplicação é executada desde o início<br>O depurador espera por um ponto de observação |
| 2  | Activação                  | Um ponto de observação é atingido e sinalizado ao depurador                                                                                                                                                                                                                                                                                           |
| 3  | Determinação (Predet.)     | O vector com falha já foi pré-carregado na infra-estrutura OCD-FI (ver ponto 1)                                                                                                                                                                                                                                                                       |
| 3+ | Determinação (sem Predet.) | A infra-estrutura OCD-FI lê o conteúdo do elemento alvo e modifica internamente o número de bits pretendido, gerando o vector com falha                                                                                                                                                                                                               |
| 4  | Inserção                   | A infra-estrutura OCD-FI ordena a escrita do vector com falha no elemento alvo                                                                                                                                                                                                                                                                        |
| 5  | Conclusão                  | Sem operação                                                                                                                                                                                                                                                                                                                                          |

Em todos os procedimentos, os pontos 3 e 3+ são utilizados alternativamente, consoante seja ou não feita a pré-determinação do vector com falha.

Como é possível verificar, os procedimentos apresentados aplicam a metodologia proposta a um sistema alvo específico. As principais diferenças entre eles residem na forma como são determinados os dados a escrever no elemento alvo, o componente onde esta operação é executada e principalmente a forma como se activa a injeção de falhas.

## 5.2 Sistema Alvo

### 5.2.1 Microprocessador e Periféricos

O sistema alvo utilizado consiste num núcleo (*core*) de microprocessador parametrizável, uma memória ROM onde é guardado o programa a executar, uma memória RAM onde são guardados os dados das aplicações e opcionalmente um porto de interface com o exterior (I/O – *Input/Output*). O microprocessador é dotado de uma infra-estrutura OCD NEXUS, equipada com um porto AUX, a qual foi desenvolvida separadamente e requer algumas modificações ao núcleo base. A arquitectura do sistema alvo completo é apresentada na Figura 5.1.

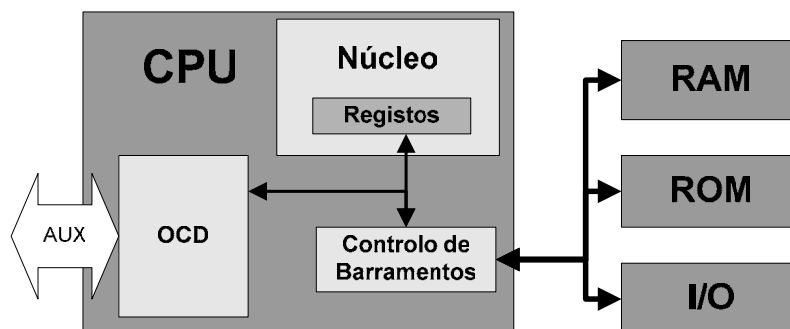


Figura 5.1 – Arquitectura do sistema alvo

Os núcleos de microprocessador utilizados foram criados utilizando a ferramenta *cpugenerator*, versão 2.0, que se encontra disponível livremente via internet [OpenCores07][Ferrante03]. Esta ferramenta cria núcleos tipo RISC configuráveis em termos de dimensão e tipo de barramentos, registos internos, profundidade da pilha (*stack*) e suporte para interrupções adicionais e endereçamento indirecto. A ferramenta cria automaticamente o modelo do microprocessador, a partir dos parâmetros escolhidos, na

forma de ficheiros VHDL. Um conjunto base de instruções é definido automaticamente com o modelo, sendo possível modificar ou acrescentar instruções de forma relativamente simples. É também disponibilizado um compilador e ferramentas para simulação e síntese de memórias. A interface com o utilizador, com as principais opções disponíveis para o microprocessador, é apresentada na Figura 5.2.

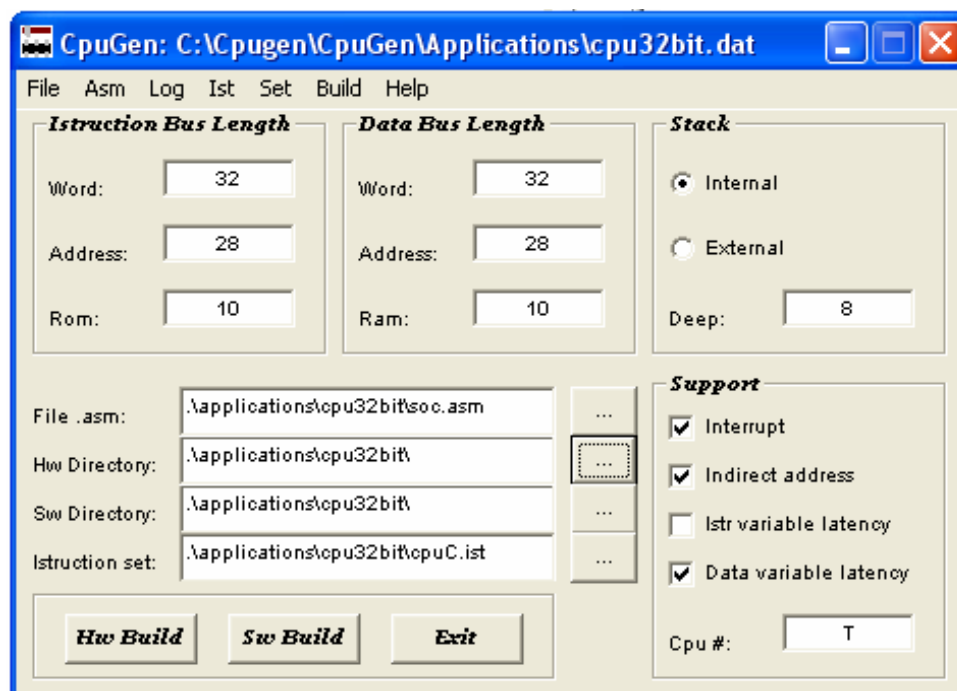


Figura 5.2 – Interface do *CpuGenerator 2.0*

Os modelos gerados por esta ferramenta apresentam uma arquitectura base semelhante, que se apresenta na Figura 5.3.

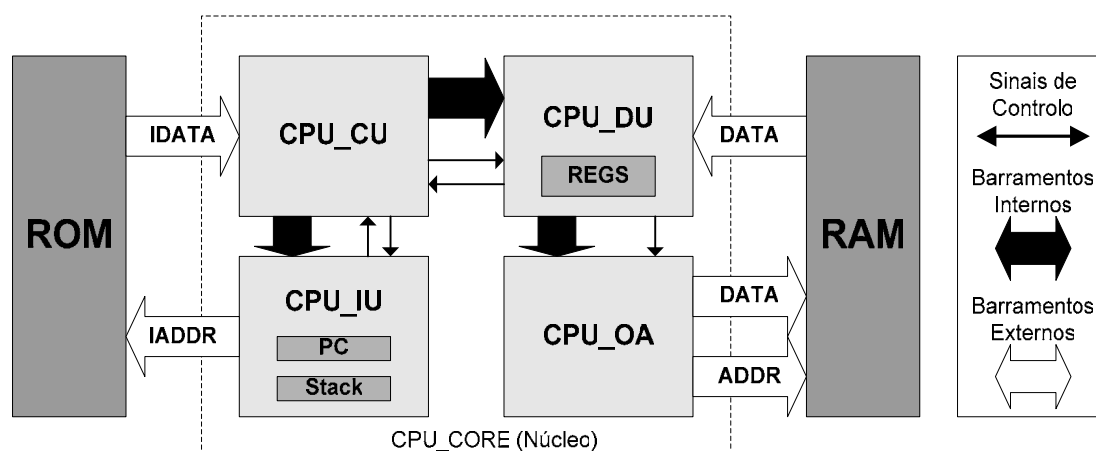


Figura 5.3 – Arquitectura Interna do Microprocessador

O núcleo é constituído por quatro elementos base interligados, com as seguintes funções:

- CPU\_IU (*CPU Instruction Unit*) – representa a unidade de instruções do microprocessador, responsável pela geração dos endereços de instrução (IADDR), pela gestão do contador de programa (PC – *Program Counter*) em operação normal e pelo arquivo de informação em *stack* quando necessário (sub rotinas ou interrupções).
- CPU\_CU (*CPU Control Unit*) – representa a unidade de controlo, responsável pela descodificação das instruções (IDATA) e pela gestão de acesso a memória, interrupções e condições de erro.
- CPU\_DU (*CPU Data Unit*) – representa a unidade de dados, responsável por operações aritméticas e análise de condições lógicas. Utiliza os dados provenientes de memória (DATA), um acumulador e outros registos (REGS) internos. Os dados para escrita são enviados para memória através da unidade CPU\_OA.
- CPU\_OA (*Cpu Over Addressing Unit*) – representa a unidade de endereçamento indirecto, necessária para a utilização de memória externa de grande dimensão (não endereçável directamente) e para permitir a extensão dos barramentos. Esta unidade é responsável pelos acessos à memória, sendo configurável de acordo com a sua dimensão.

A Figura 5.4 exemplifica a designação completa de um modelo do microprocessador e da respectiva infra-estrutura OCD. A designação pode ser simplificada, removendo elementos não relevantes para a análise em questão.

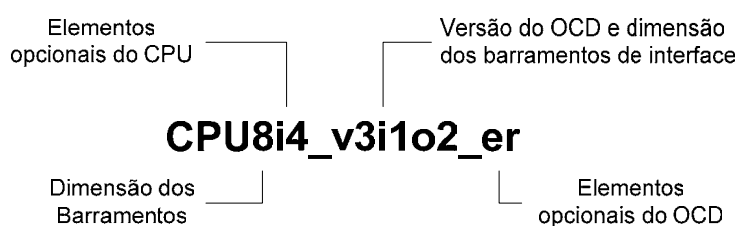


Figura 5.4 – Designação do modelo do microprocessador e infra-estrutura OCD

Neste caso o microprocessador consiste numa versão de 8 bits com suporte para interrupções, *stack* com uma profundidade de quatro *bytes*, a versão 3 do OCD<sup>20</sup> com uma interface com 1 bit de entrada e 2 bits de saída, e incluindo suporte para EDAC e RTREG (estes elementos serão explicados na subsecção 5.2.3). Os números relativos à interface do

<sup>20</sup> A versão do OCD permite identificar diferentes soluções, a correcção de erros e a introdução de funcionalidades adicionais.

OCD indicam o número de bits dos barramentos MDI e MDO, de acordo com a proposta NEXUS.

### 5.2.2 Aplicações

Foram utilizadas quatro aplicações de base para a avaliação experimental. Todas são adaptáveis às diversas implementações do microprocessador, sendo designadas de acordo com a sua função, dimensão dos barramentos e inclusão (ou não) de capacidades de tolerância a falhas, tal como apresentado no exemplo da Figura 5.5.

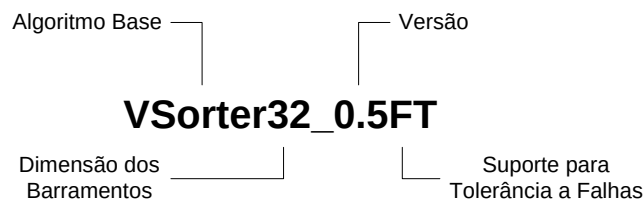


Figura 5.5 – Designação das Aplicações

As quatro aplicações de base utilizadas estão apresentadas na Tabela 5.5.

Tabela 5.5 – Aplicações de Base Utilizadas

| Designação             | Descrição                                                                                                                                                                                        |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b><i>ITest</i></b>    | Executa sequencialmente todo o conjunto de instruções definido para o microprocessador.                                                                                                          |
| <b><i>MAdder</i></b>   | Executa a adição de duas matrizes guardadas em memória e escreve o resultado num espaço de memória adicional. A dimensão das matrizes é configurável.                                            |
| <b><i>VSorter</i></b>  | Ordena um vector de números inteiros guardados em memória, utilizando o método <i>bubble sort</i> . A dimensão do vector e a sua posição em memória são configuráveis.                           |
| <b><i>XControl</i></b> | Executa o controlo de uma variável de saída a partir de uma variável de entrada, um sinal de controlo e um conjunto de tabelas de conversão ( <i>LUT – Look Up Table</i> ) guardadas em memória. |

Todas as aplicações são baseadas em algoritmos que fazem utilização intensiva da memória e são adaptáveis a diferentes dimensões de barramentos e volume de dados, sendo também relativamente simples de depurar. A aplicação *XControl* requer dois valores de entrada, nomeadamente um vector de dados e um sinal de controlo. Para poder ser utilizada, o

sistema alvo terá de suportar uma interface com o exterior e um eventual modelo de simulação terá de incluir a geração externa dos valores de entrada.

Cada uma das aplicações foi desenvolvida em duas versões, uma versão básica e outra com tolerância a falhas por *software*. A tolerância a falhas foi implementada recorrendo à duplicação de dados em memória e à execução repetida de cada operação aritmética [Velazco95][Velazco02][Velazco05], consistindo portanto na aplicação de redundância espacial e temporal, tal como representado na Figura 5.6.

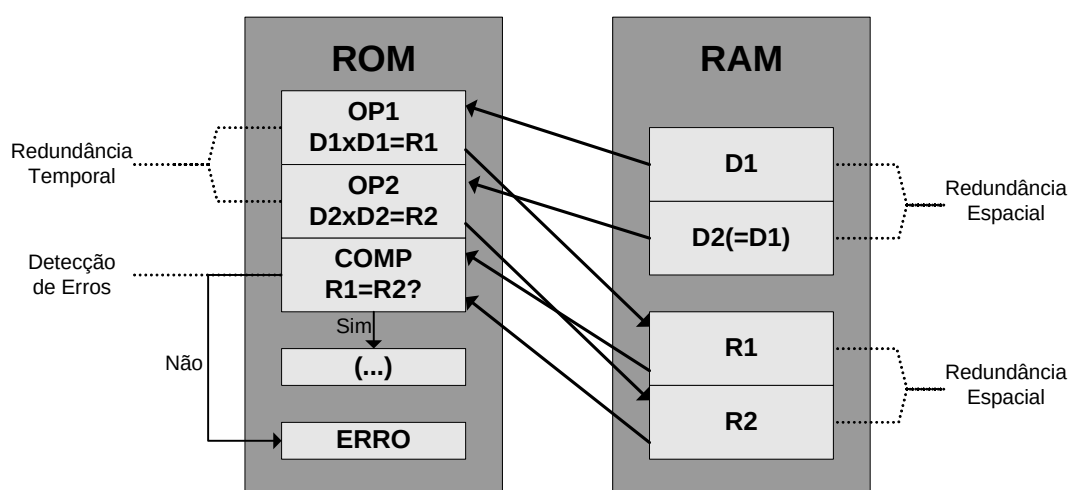


Figura 5.6 – Tolerância a falhas por *software*

Desta forma, cada operação aritmética ou lógica é executada duas vezes, sequencialmente, utilizando áreas de memória diferentes. Esta abordagem permite um nível limitado de detecção de falhas, incorrendo numa sobrecarga em termos de memória necessária e do tempo de execução, mas sem necessitar de alterações ao *hardware* utilizado. A Tabela 5.6 compara algumas das aplicações utilizadas, no que respeita às dimensões e ao espaço de memória de dados.

Tabela 5.6 – Sobrecarga imposta pela tolerância a falhas (exemplos)

| Aplicação               | Dimensão (bytes) | Memória de Dados (bytes) |
|-------------------------|------------------|--------------------------|
| <i>MAdder8_1.0</i>      | 37               | 104                      |
| <i>MAdder8_1.0FT</i>    | 74 (+100%)       | 204 (+96%)               |
| <i>VSorter32_1.2</i>    | 51               | 2060                     |
| <i>VSorter32_1.2FT</i>  | 108 (+111%)      | 4112 (+99%)              |
| <i>XControl32_1.0</i>   | 42               | 1540                     |
| <i>XControl32_1.0FT</i> | 78 (+85%)        | 3092 (+100%)             |

Esta abordagem foi utilizada na maioria dos cenários experimentais, nomeadamente por ser de implementação simples nos componentes COTS e portanto representativa do problema que se pretende abordar. A utilização de mecanismos de tolerância a falhas por *hardware* mais complexos é abordada na secção 5.6.

### 5.2.3 Infra-Estrutura OCD

A infra-estrutura OCD foi projectada em conformidade com a proposta NEXUS 5001 [Fidalgo06c]. Como não existe nenhuma implementação obrigatória, optou-se por desenvolver uma solução semelhante à utilizada no microprocessador MPC565, que se encontra relativamente bem documentada [Motorola02]. A versão de base implementada está em conformidade com a classe 2 da proposta NEXUS e inclui funcionalidades de acesso à memória em tempo real, sendo semelhante às infra-estruturas classificadas como estando conformes com a classe 2+. A infra-estrutura OCD foi projectada de forma modular e configurável, sendo facilmente adaptável às características de cada arquitectura concreta.

Não foram considerados no projecto da infra-estrutura OCD nenhuns requisitos de confiabilidade ou de injeção de falhas, sendo aquela definida apenas pelos requisitos de depuração. Pretendeu-se desta forma que a solução desenvolvida fosse análoga a uma solução comercial, de forma a replicar as capacidades disponibilizadas para injeção de falhas, bem como os problemas envolvidos na sua adaptação e modificação para melhor desempenho.

A interface da infra-estrutura OCD com o exterior usa o porto AUX, sendo configurável a dimensão dos barramentos de dados (MDI, MDO) e a profundidade dos *buffers*, bem como os sinais de controlo obrigatórios (MCKO, MCKI, MSEI e MSEO, EVTI). O sinal EVTI é utilizado para alteração do estado do microprocessador, permitindo parar e arrancar a execução. Os sinais opcionais EVTO e RSTI estão também disponíveis para sinalizar a ocorrência de pontos de observação e reinicialização da infra-estrutura OCD.

O protocolo de comunicações adoptado é o mais simples previsto na norma, utilizando apenas um sinal de controlo. Foi considerado que o ganho mínimo de desempenho e as necessidades de depuração do sistema alvo em causa, não justificavam a complexidade adicional.

A infra-estrutura OCD encontra-se dividida em três módulos principais e dois módulos de interface com barramentos do microprocessador, tal como apresentado na Figura 5.7.

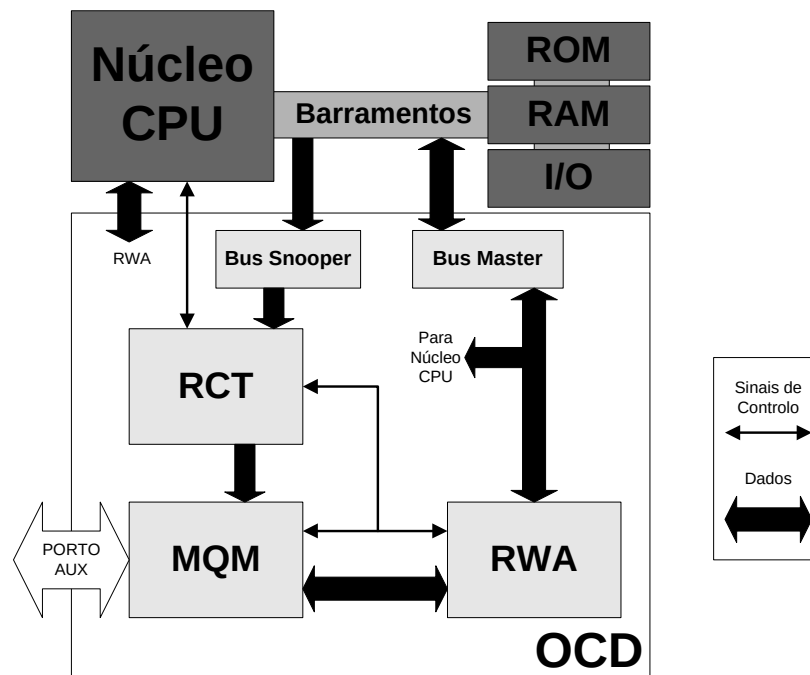


Figura 5.7 – Infra-estrutura de depuração embutida (OCD)

Os módulos de interface são responsáveis pela ligação de dados entre a infra-estrutura OCD e o microprocessador, com o módulo *Bus Snooper* a executar leitura de dados de forma não intrusiva e o módulo *Bus Master* a ser utilizado para operações que requerem controlo sobre um ou mais barramentos (i.e. operações de escrita). Estes módulos deverão ser adaptados à arquitectura do sistema alvo, nomeadamente às dimensões do barramento, sinais de controlo associados e mecanismos de gestão de colisões e partilha de recursos. Adicionalmente a estes módulos, são utilizados alguns sinais de controlo directos entre o microprocessador e a infra-estrutura OCD, para efeitos de sincronismo e controlo da execução (i.e. paragem, arranque, reinicialização), bem como um conjunto de registos que definem a configuração da infra-estrutura OCD. Os três módulos principais desta infra-estrutura são os seguintes:

- O módulo de controlo e gestão de mensagens, designado por **MQM** (*Message Queuing and Management*), responsável pelo controlo global da infra-estrutura e comunicação com o exterior.
- O módulo de controlo de execução e monitorização, designado por **RCT** (*Run Control and Trace*), responsável pelas funcionalidades de controlo do microprocessador e pela recolha de informação de monitorização.
- O módulo de acesso para leitura e escrita de dados, designado por **RWA** (*Read and Write Access*), responsável pelas operações de escrita e leitura de dados em memória, nos registos do microprocessador e nos registos da infra-estrutura OCD.

A infra-estrutura OCD inclui também um conjunto de registos, os quais podem ser associados num banco de memória ou fisicamente divididos entre os módulos, apresentando esta segunda opção efeitos mínimos no modelo, afectando principalmente a implementação física da infra-estrutura. Os registos utilizados são os seguintes:

- Registo de identificação do dispositivo (DIR – *Device Identification Register*), que é obrigatório pela proposta NEXUS, apenas pode ser acedido para leitura e inclui informação relativa ao microprocessador e à infra-estrutura OCD.
- Registo de configuração (CR – *Configuration Register*), que permite alterar os parâmetros de funcionamento da infra-estrutura OCD.
- Registo de estado (SR – *State Register*), que permite consultar o estado da infra-estrutura OCD.
- Registos do endereço de ponto de paragem (BPn – *Breakpoint n*): registos múltiplos (um para cada ponto de paragem) que contêm o endereço onde deverá ocorrer a paragem.
- Registos do endereço de ponto de observação (WPn – *Watchpoint n*): registos múltiplos (um para cada ponto de observação) que contêm o endereço que deverá ser monitorizado.
- Registo do número de ocorrências de condição de paragem ou observação (Ncounter): este registo permite definir o número de ocorrências de uma determinada condição. Pode ser associado a um ponto de paragem ou a um ponto de observação, dependendo do registo de configuração da infra-estrutura OCD. Para efeitos de depuração, é normalmente suficiente um único registo partilhado, mas poderão ser implementados mais, se necessário.

Na infra-estrutura OCD utilizada são implementados dois pontos de paragem por instrução, um ponto de paragem por acesso a dados e um ponto de observação por instrução, com um único registo de contagem de ocorrências partilhado entre o primeiro ponto de paragem e o ponto de observação.

O modulo MQM engloba as funções de controlo e comunicação, devido aos elevados requisitos de sincronismo entre estas duas operações. Um diagrama simplificado deste módulo é apresentado na Figura 5.8.

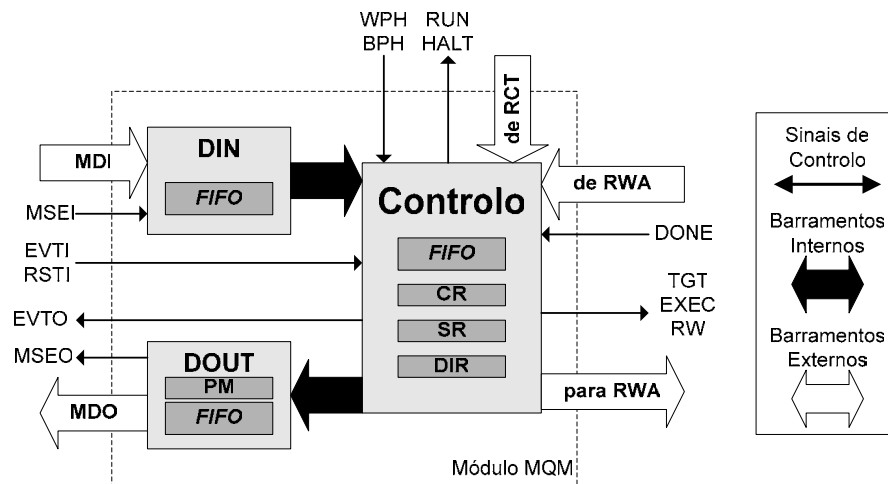


Figura 5.8 – Arquitectura do módulo MQM (simplificada)

As mensagens de entrada e saída são geridas separadamente, cada qual contando com um *buffer* tipo FIFO (*First-In First-Out*) dedicado, com suporte para mensagens prioritárias (PM – *Priority Message*) na saída. A dimensão destes *buffers* é configurável e depende da largura de banda utilizada nas comunicações com o exterior, do tipo de mensagens suportadas e da frequência de utilização prevista. A carga lógica imposta por estes elementos é um factor importante na dimensão do módulo, pelo que a sua profundidade deve ser seleccionada de forma a evitar perda de informação, sem sobrecarregar demasiado a infra-estrutura.

As mensagens de entrada são descodificadas após terem sido completamente recebidas e, dependendo da sua natureza, podem requerer acção imediata ou prosseguirem para uma FIFO de acções a executar. Como exemplos de acções imediatas, podem citar-se as instruções de mudança de estado (i.e. paragem/arranque do processador), sendo as acções de leitura e escrita enviadas para a FIFO. Desta forma, as operações de acesso a recursos do microprocessador são executadas pela ordem em que são recebidas, enquanto as operações de controlo urgentes são executadas de acordo com as capacidades da infra-estrutura OCD. Em funcionamento normal, a descodificação de cada mensagem deverá ser mais rápida do que a transmissão da mensagem seguinte, sendo portanto executada imediatamente. As excepções são as mensagens com operações de leitura de memória em tempo real, que apenas se podem considerar executadas após o envio da resposta. Neste caso, uma sequência demasiado rápida e/ou problemas de acesso, devido a conflitos na utilização de barramentos, poderão levar a atrasos e fazer com que uma ou mais mensagens tenham de esperar pelo processamento completo da mensagem anterior. A largura de banda para escrita (no OCD) do porto AUX, o funcionamento do *Bus Master* e a profundidade das FIFOs, são factores que influenciam estes casos.

As mensagens de saída são processadas em paralelo com as de entrada, sendo o processamento e a comunicação completamente independentes. São utilizados três tipos principais de mensagens de saída, nomeadamente (1) a sinalização de eventos do microprocessador, (2) as respostas a operações de leitura/escrita e (3) a informação de monitorização. A implementação é feita através de uma FIFO independente, equipada com o referido mecanismo que permite o envio de mensagens prioritárias (PM), sendo as de baixa prioridade, com informação de monitorização e respostas a operações de leitura / escrita, processadas por ordem de chegada ao MQM. Os eventos prioritários, que indicam a mudança de estado do microprocessador, e comunicados por mensagem são enviados imediatamente após a conclusão da mensagem anterior (i.e. a mensagem em transmissão não é interrompida). Os eventos cuja ocorrência é comunicada por sinal, são processados paralelamente e não influenciam a transmissão de mensagens.

Os sinais de controlo entre o MQM, os restantes módulos da infra-estrutura OCD e o microprocessador, são gerados ou lidos na execução dos diferentes comandos. Os sinais RW e TGT permitem escolher o tipo de alvo (i.e. memória, registos) de uma operação de leitura ou escrita, sendo o sinal EXEC utilizado para sincronismo, permitindo o controlo do instante em que a operação é executada.

O módulo RCT apresenta duas funções principais, nomeadamente o controlo de execução e a recolha de informação de monitorização. Um diagrama simplificado deste módulo é apresentado na Figura 5.9.

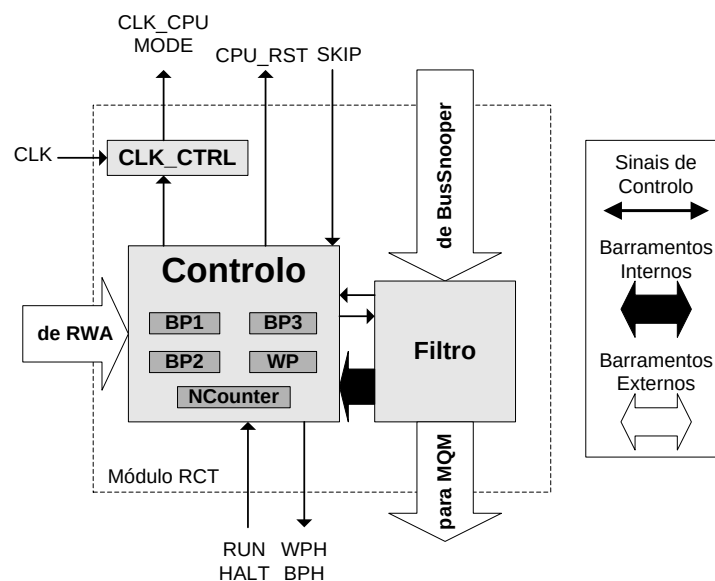


Figura 5.9 – Arquitectura do módulo RCT (simplificada)

O controlo de execução é feito através do controlo do relógio do microprocessador (CLK\_CTRL) e do sinal de reinicialização (CPU\_RST), sendo possíveis os seguintes modos de operação:

- **Modo de execução (RUN):** Neste modo o microprocessador encontra-se em execução normal, sendo as instruções executadas de forma sequencial e sem paragens.
- **Modo de depuração (DEBUG):** Neste modo o microprocessador encontra-se suspenso. No momento da suspensão (por comando da infra-estrutura OCD), os registos internos do microprocessador não são alterados.
- **Modo de execução passo-a-passo (SSTEP)** este modo é um caso particular do modo DEBUG, pois apenas permite a execução de uma instrução de cada vez, suspendendo a execução após cada uma.

A passagem entre modos depende dos sinais recebidos do módulo MQM e do conteúdo do registo de configuração (CR). A selecção entre os modos DEBUG e SSTEP é feita através de um bit deste registo, de forma a garantir um comportamento consistente, requerendo uma alteração do referido registo quando for necessário comutar entre ambos.

Optámos por implementar o módulo RCT utilizando também um sinal (SKIP<sup>21</sup>) proveniente do núcleo do microprocessador, que permite identificar a ocorrência de instruções de salto. Esta opção foi tomada por permitir a identificação de forma mais simples e rápida, e por este sinal estar disponível no barramento interno do microprocessador, podendo ser lido sem afectar o seu funcionamento. A monitorização da informação enviada e recebida pelo microprocessador, através dos barramentos de dados e endereços, permite a monitorização da sua execução de forma não intrusiva. Opcionalmente, poder-se-ia introduzir no módulo RCT a capacidade de identificar este tipo de instruções e verificar a sua ocorrência. No caso específico do microprocessador alvo utilizado, esta solução não acrescentaria nenhum ganho em termos de desempenho e teria vantagens mínimas em termos de intrusividade, uma vez que a interface entre o microprocessador e o exterior terá forçosamente de ser alterada, de modo a permitir o acesso aos registos internos para depuração, obrigando a uma sobrecarga maior em termos de lógica. O módulo RCT pode utilizar informação disponível nos barramentos externos ao microprocessador (via *Bus Snooper*), para determinar a instrução em execução e os acessos a dados. Esta monitorização pode ser efectuada com maior ou

---

<sup>21</sup> A designação deste sinal é definida pela aplicação *cpugenerator* e advém da forma como são realizadas as condições de salto, sinalizadas pela não execução (*skipping*) de uma instrução sequencial.

menor detalhe, tendo o módulo RCT de estar dimensionado para o volume de informação a processar. É por vezes necessário filtrar o que é monitorizado e efectivamente enviado para processamento pelo módulo MQM, de forma a garantir que é possível executar a monitorização pedida. Deste modo, um módulo RCT poderá limitar a monitorização de dados a operações de um determinado tipo, ou num número limitado de endereços, de forma a que a informação possa ser processada pela infra-estrutura, mantendo uma dimensão aceitável em termos de portas lógicas e de blocos de memória.

Os pontos de paragem e de observação são também controlados por este módulo, que é responsável por monitorizar as condições e o respectivo número de ocorrências. As condições possíveis dizem respeito à execução de instruções e ao acesso a dados. Esta última pode ser restringida em termos da gama de endereços, tipo de acesso (escrita, leitura ou ambos) e, opcionalmente, em termos de conteúdo. Caso se verifiquem as condições necessárias, o próprio módulo executa a paragem do microprocessador ou sinaliza a ocorrência de um ponto de observação. A contagem do número de ocorrências de uma determinada condição pode ser feita, de forma a que esta só seja considerada na enésima ocorrência. Um registo (Ncounter) está disponível para esta contagem e pode ser utilizado pelo primeiro ponto de paragem por instrução, pelo ponto de paragem por dados ou pelo ponto de observação (por instrução). A opção utilizada é definida no registo de configuração (CR). Na infra-estrutura implementada, a sinalização da activação de um ponto de observação é feita por um sinal dedicado (WPH), que é activado imediatamente na ocorrência da respectiva condição.

O módulo RWA interage com o microprocessador, com a infra-estrutura OCD e com o *Bus Master*, de forma a permitir operações de leitura e escrita nos diversos recursos disponíveis. Um diagrama simplificado deste módulo é apresentado na Figura 5.10.

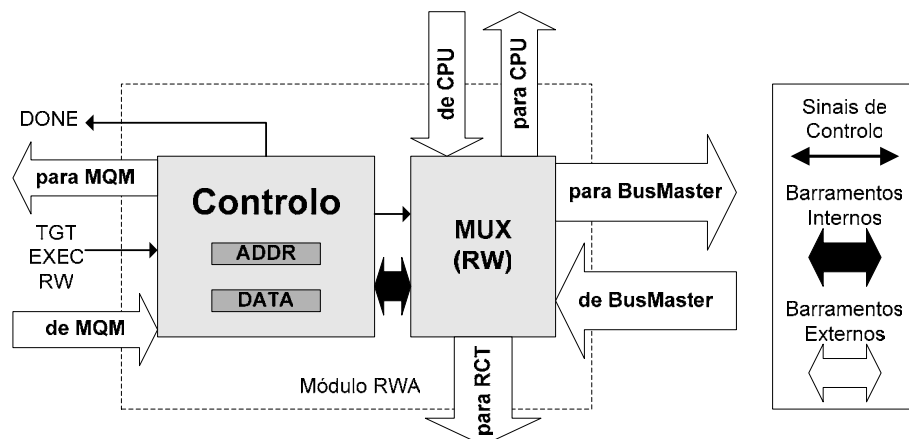


Figura 5.10 – Arquitectura do módulo RWA (simplificada)

Como componentes alvo, podemos considerar os recursos internos do microprocessador (e.g. registos ou memória interna), os recursos internos da própria infra-estrutura OCD (e.g. registos) e os recursos externos via *Bus Master* (e.g. memória ou I/O). Qualquer operação deverá ser precedida pela actualização do endereço (ADDR) sobre o qual será executada, incluindo bits adicionais (TGT, RW) que indiquem o componente alvo e tipo de acesso, bem como dos dados (DATA) para as operações de escrita. Subsequentemente, um sinal externo (EXEC) ao módulo será utilizado para a execução da operação, sendo os valores resultantes de uma operação de leitura colocados no registo de dados. A conclusão de cada operação é transmitida por sinal próprio (DONE) e acompanhada dos dados lidos, no caso de operações de leitura.

O acesso a recursos internos do microprocessador exige acesso directo aos seus barramentos internos e/ou aos registos. Normalmente, deve salvaguardar-se que este acesso seja possível apenas quando o microprocessador não se encontra em execução, de forma a não perturbar o seu funcionamento, e garantindo que as modificações necessárias para este tipo de acesso não afectam o desempenho do sistema alvo.

O acesso a recursos internos da própria infra-estrutura OCD visa permitir a sua configuração e é feito através da leitura do registo de estado e escrita do registo de configuração e registos associados aos pontos de paragem e observação. Adicionalmente, é possível aceder para leitura a um registo de identificação do dispositivo, tal como previsto na proposta NEXUS.

Os módulos *Bus Snooper* e *Bus Master* poderiam fazer parte dos módulos RCT e RWA, mas são implementados separadamente, para melhor adaptabilidade a diferentes arquitecturas do sistema alvo. Idealmente, estes módulos seriam os únicos que exigiriam alterações, de forma a adaptar a infra-estrutura a diferentes arquitecturas alvo, mas nem sempre será possível uma solução tão simples.

O módulo *Bus Snooper* faz a leitura não intrusiva dos dados que circulam nos barramentos externos do microprocessador, de forma a identificar e classificar as operações relevantes executadas. Estas operações incluem a leitura de instruções, a leitura e escrita de dados em memória, e o acesso a componentes externos. Tipicamente, os elementos que deverão ser lidos, são os dados e endereços de cada barramento, podendo ser utilizado um barramento único ou uma divisão em barramentos de instruções e de dados, dependendo da arquitectura utilizada no microprocessador alvo. No caso específico do *cpugenerator*, ambas as opções são possíveis, pelo que são necessárias duas versões do módulo *Bus Snooper*. Adicionalmente, terão de ser monitorizados os sinais de controlo presentes, de forma a classificar os acessos entre leitura e escrita, e a distinguir os acessos à memória e as operações sobre periféricos. O módulo *Bus Snooper* terá de ser adaptado ao microprocessador alvo, podendo eventualmente ser reutilizado dentro da mesma arquitectura.

O *Bus Master* é um módulo de construção mais complexa, se pretendermos evitar a intrusividade. Em qualquer barramento, a inclusão de um controlador adicional poderá acarretar custos de desempenho, se for necessário partilhar recursos com os demais controladores. Em implementações clássicas de infra-estruturas OCD, o problema resolve-se impondo a paragem do microprocessador alvo durante os acessos realizados por essas infra-estruturas. Para o acesso à memória em tempo real, com o sistema alvo em execução, esta abordagem não é possível, sendo necessário um equilíbrio entre intrusividade e complexidade acrescida da infra-estrutura OCD, e sua interface com o sistema alvo.

A abordagem mais simples será a partilha dos barramentos, com prioridade a um dos elementos. Dependendo de serem prioritários o microprocessador ou a infra-estrutura OCD, teremos efeitos no respectivo desempenho. Se a infra-estrutura tiver prioridade, existirá o risco de acessos atrasados do microprocessador à memória, com inerente perda de desempenho; caso seja dada prioridade ao microprocessador, serão impostos atrasos às operações de depuração. Ambas as soluções poderão ser aceitáveis, dependendo da severidade das suas consequências e do tipo de aplicações consideradas. A alternativa será a utilização de barramentos de acesso e/ou memória mais complexos, de forma a permitir o acesso simultâneo. Neste caso, será apenas preciso acautelar operações de escrita simultâneas no mesmo endereço, que por terem uma probabilidade de ocorrência relativamente baixa, têm um efeito mínimo no desempenho.

### 5.3 Depurador Optimizado

O depurador desenvolvido foi projectado para otimizar a execução das operações de injeção de falhas, visando suportar as seguintes funcionalidades:

- Conformidade com a proposta NEXUS 5001, classe 2, e utilizando exclusivamente o porto de comunicações AUX.
- Adaptabilidade da interface de comunicações com a infra-estrutura OCD (AUX), a diferentes implementações do protocolo de comunicações e a diferentes dimensões do barramento.
- Controlo directo das funcionalidades de depuração básicas
- Capacidade de reacção automática a eventos sem necessidade de intervenção pelo sistema anfitrião

As funcionalidades descritas visam fazer com que o depurador seja genérico, para maximizar a sua adaptabilidade, manter uma dimensão lógica razoável e simultaneamente fornecer o melhor desempenho possível, em termos de injeção de falhas. A inclusão de comportamento reactivo é particularmente importante para este efeito e endereça uma limitação identificada nos ambientes de depuração e injeção de falhas existentes. O depurador encontra-se assim adaptado à metodologia proposta, mas apresentando flexibilidade suficiente para utilizações diversas, inclusive em operações de depuração convencionais (i.e. arranque paragem, configuração, leitura de registos e memória, controlo de monitorização). O diagrama simplificado do depurador é apresentado na Figura 5.11.

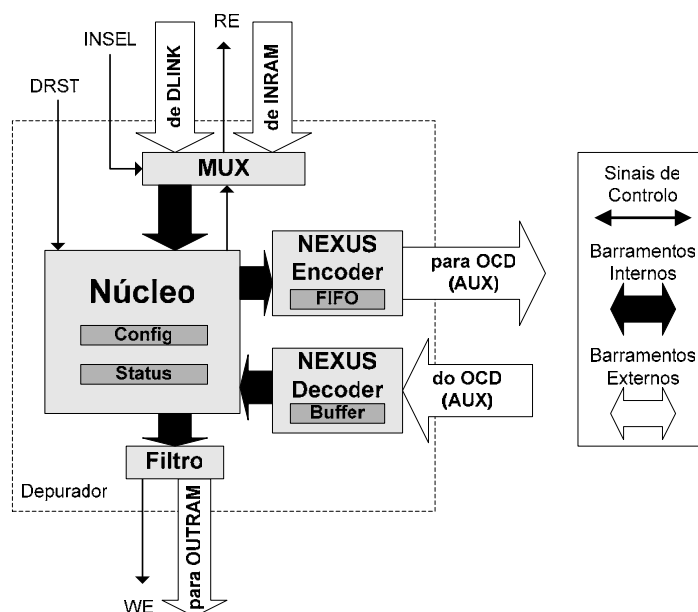


Figura 5.11 – Arquitectura do Depurador (simplificada)

O núcleo do controlador opera como um processador simples, lendo instruções de uma memória externa ou de um barramento de controlo directo, de acordo com um sinal externo (INSEL). Cada instrução pode ser executada imediatamente ou demorar um tempo não determinístico, pelo que é utilizado um sinal específico (RE) que indica a disponibilidade para receber nova instrução. Um sinal de reinicialização (DRST) pode ser utilizado para esse fim ou para recuperar de situações de bloqueio.

O controlo do porto de comunicações é feito recorrendo a dois blocos codificadores/descodificadores. O bloco codificador (NEXUS *Encoder*) transforma o comando em execução numa mensagem, ou sinal, em conformidade com a proposta NEXUS, e envia esta mensagem através do porto AUX. O bloco descodificador (NEXUS *Decoder*) faz a operação inversa e interpreta as mensagens recebidas através do porto AUX. Estas últimas são filtradas e a informação relevante é enviada para a memória externa. Dois registos de configuração (*Config*) e de estado (*Status*), definem o modo de funcionamento do depurador.

A utilização de duas memórias separadas é opcional, devendo no entanto ser garantida a separação virtual dos espaços de memória, no caso de utilização de uma memória física única.

Cada comando pode ser seguido por um ou mais parâmetros, só podendo ser executado após a recepção de todos eles. O depurador apresenta dois modos de funcionamento principais, o modo NORMAL e o modo REACTIVO. Este último é activado pela execução de um comando com do mesmo tipo e mantém-se até que seja atingido um determinado tempo de espera, que é opcional e configurável, sendo possível bloquear indefinidamente o depurador. Dois comandos são especialmente destinados à manipulação de dados, de forma a inserir falhas do tipo inversão de bits. A execução destes comandos permite a leitura, modificação e reescrita de posições de memória ou registos, de forma automática, requerendo apenas uma máscara com a informação relativa ao(s) bit(s) a alterar. A listagem dos comandos executáveis pelo depurador, os seus parâmetros, o modo em que são utilizados e a respectiva descrição, são apresentados na Tabela 5.7.

Tabela 5.7 – Comandos do Depurador

| Comando  | Parâmetros       | Tipo     | Descrição                                                                                                                                                                                                                                                                                                              |
|----------|------------------|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HALT     | Nenhum           | ACTIVO   | Ordena a paragem do sistema alvo e sua passagem a modo de depuração (DEBUG).                                                                                                                                                                                                                                           |
| RUN      | Nenhum           | ACTIVO   | Ordena ao sistema alvo que inicie ou retome a execução.                                                                                                                                                                                                                                                                |
| RESET    | Nenhum           | ACTIVO   | Ordena a reinicialização do sistema alvo.                                                                                                                                                                                                                                                                              |
| DRESET   | Nenhum           | ACTIVO   | Reinicia o depurador (este comando é equivalente à activação do sinal DRST).                                                                                                                                                                                                                                           |
| DCONFIG  | <code>           | ACTIVO   | Configura o depurador de acordo com o parâmetro <code>.                                                                                                                                                                                                                                                                |
| WAIT     | <time>           | REACTIVO | Espera um intervalo de tempo definido pelo parâmetro <time>. O depurador não aceita novos comandos durante esse intervalo de tempo.                                                                                                                                                                                    |
| WAITFOR  | <code><br><time> | REACTIVO | Espera por uma mensagem ou sinal específico do sistema alvo. Estes são definidos pelo parâmetro <code> e podem corresponder a uma mensagem de monitorização, uma mensagem de resposta, ou um sinal de ponto de observação atingido. A espera pode ser limitada por um intervalo de tempo definido no parâmetro <time>. |
| XORRAM   | <addr><br><mask> | ACTIVO   | Ordena a leitura de um endereço de memória do sistema alvo indicado pelo parâmetro <addr>. O valor lido é modificado de acordo com uma operação de XOR com o vector contido em <mask> e reescrito na mesma posição de memória.                                                                                         |
| XORREG   | <addr><br><mask> | ACTIVO   | Ordena a leitura de um registo do sistema alvo indicado pelo parâmetro <addr>. O valor lido é modificado de acordo com uma operação de XOR com o vector contido em <mask> e reescrito no mesmo registo.                                                                                                                |
| READRAM  | <addr>           | ACTIVO   | Ordena a leitura de um endereço de memória do sistema alvo indicado pelo parâmetro <addr>.                                                                                                                                                                                                                             |
| WRITERAM | <data><br><addr> | ACTIVO   | Ordena a escrita do valor indicado por <data> no endereço de memória do sistema alvo, indicado pelo parâmetro <addr>.                                                                                                                                                                                                  |
| READREG  | <addr>           | ACTIVO   | Ordena a leitura de um registo do sistema alvo indicado pelo parâmetro <addr>, contido num registo interno do microprocessador ou na infra-estrutura OCD (dependendo do valor de <addr>).                                                                                                                              |
| WRITEREG | <data><br><addr> | ACTIVO   | Ordena a escrita do valor indicado por <data> num registo do sistema alvo, indicado pelo parâmetro <addr>, contido um registo interno do microprocessador ou na infra-estrutura OCD (dependendo do valor de <addr>).                                                                                                   |

A utilização de comandos reactivos permite a espera por períodos prolongados, que é particularmente importante em certos cenários de injeção de falhas. Uma vez em modo

REACTIVO, o depurador não aceita mais comandos, até que se verifique a condição esperada, ou que seja feita a reinicialização do depurador, sendo activado um sinal que indica especificamente o seu estado presente.

As condições de utilização de um comando reactivo são as seguintes:

- Espera por um determinado intervalo de tempo
- Espera por uma determinada mensagem da infra-estrutura OCD
- Espera por um sinal da infra-estrutura OCD

Podem utilizar-se mensagens de monitorização ou mudança de estado, permitindo portanto a identificação de pontos de paragem, de instantes na execução do programa e de pontos de observação (caso sejam sinalizados por mensagem).

O porto de acesso permite a execução directa de comandos, inclusive comandos que colocam o depurador em modo REACTIVO. A utilização do depurador em modo directo para efeitos de injeção de falhas é possível, sendo no entanto de evitar, por não trazer vantagens evidentes e por poder apresentar problemas de sincronização e degradação de desempenho, devido aos atrasos e falta de determinismo, impostos pelo sistema anfitrião.

Tal como explicado no capítulo anterior, a memória de entrada é utilizada para guardar a lista de comandos (programa) a executar, necessários à operação de depuração ou injeção de falhas pretendida. O acesso a esta memória é controlado pelo próprio depurador de forma sequencial. É configurável a utilização simultânea de comandos directos e de memória de entrada, estando no entanto o sinal de reinicialização sempre disponível.

A memória de saída é utilizada para registar mensagens recebidas da infra-estrutura OCD, sendo possível filtrar as mensagens que são efectivamente guardadas. Esta filtragem é feita de acordo com o tipo de mensagem e visa permitir uma utilização racional da memória. São consideradas as seguintes categorias de mensagem:

- Monitorização de programa
- Monitorização de dados
- Mudança de estado
- Erro

É perfeitamente possível utilizar um comando reactivo que responda a uma determinada mensagem e ao mesmo tempo filtrar esse tipo de mensagens, de forma a que não sejam guardadas em memória.

O desempenho do porto de comunicações AUX é dependente da dimensão dos seus barramentos de dados. Para efeitos de depuração, é normal que a amplitude do barramento de entrada da infra-estrutura OCD (barramento de saída do depurador), seja menor que a do barramento de saída. Esta diferença baseia-se na maior expectativa de utilização deste último, para transmitir mensagens de monitorização. É possível que uma mesma versão da infra-estrutura OCD utilize barramentos de diferentes dimensões, mantendo ou não o mesmo porto de acesso. Estas variações são dependentes dos requisitos de depuração de cada sistema alvo e permitem um aumento de desempenho, sem alterações ao funcionamento da infra-estrutura OCD ou do depurador. Poderá ser necessário utilizar um canal de comunicação (i.e. cabo ou ligação virtual) diferente e o ambiente de depuração deverá ser capaz de suportar o desempenho acrescido. As desvantagens são o aumento do número de sinais de saída na infra-estrutura OCD e a necessidade de maior capacidade de armazenamento temporário (i.e. *buffers*), que implicará um aumento da sobrecarga lógica imposta pela infra-estrutura OCD. Este procedimento poderá ser usado quando os requisitos de depuração exigem desempenho melhorado e também para aumentar a eficiência do processo de injeção de falhas.

#### **5.4 Alterações à infra-estrutura OCD**

A proposta de norma NEXUS prevê a alteração dos barramentos de comunicação com o exterior de uma infra-estrutura OCD, como forma de variar o respectivo desempenho. Estas variações requerem que a capacidade esteja disponível na infra-estrutura, nomeadamente ao nível de registos, barramentos internos e *buffers*, e também que seja possível utilizar diferentes portos de acesso, ou modificar o número de pinos. Apesar de serem recomendados na norma diferentes conectores, não existem configurações obrigatórias, sendo o número de sinais de dados expansível. O MPC565 apresenta pelo menos duas configurações possíveis, com diferentes taxas de transferência de dados, e as placas de desenvolvimento utilizadas incluíam também dois conectores diferentes.

No ambiente de injeção de falhas desenvolvido, foram utilizados sistemas alvo com barramentos de dimensões variadas (de 4 a 32 bits), pelo que foi necessário desenvolver diferentes portos de acesso. Foi considerado para a versão inicial de cada infra-estrutura OCD o número de sinais sugerido na proposta NEXUS. Como já foi referido, uma forma de aumentar o desempenho do processo de acesso a memória (ou registos), via OCD, é

aumentar a taxa de transferência de dados, aumentando a amplitude dos barramentos entre a infra-estrutura OCD e o depurador. Obviamente, o porto de saída do depurador terá de apresentar a mesma configuração do utilizado na infra-estrutura OCD. As modificações no depurador e infra-estrutura requerem alterações apenas nos portos de acesso e respectivos controladores, sendo necessário acrescentar sinais de entrada/saída, e eventualmente redimensionar os *buffers* internos. Estes últimos devem ser ajustados à taxa de transferência de dados, levando em consideração o desempenho máximo da infra-estrutura OCD.

Foi também desenvolvida uma solução alternativa para efeitos comparativos, a qual consiste na eliminação dos portos AUX e na comunicação directa da informação em paralelo, tal como apresentado na Figura 5.12.

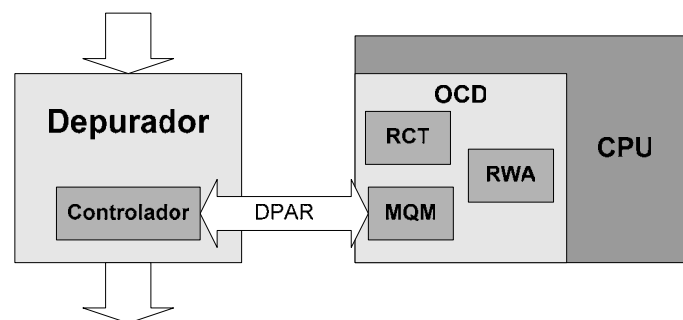


Figura 5.12 – Ligação directa entre depurador e infra-estrutura OCD

Neste caso, são eliminadas as componentes de codificação/descodificação presentes no depurador e módulo MQM, e feita a transmissão directa dos comandos através de uma ligação em paralelo (DPAR). É necessária alguma lógica adicional para a tradução e interpretação dos comandos, mas os requisitos em termos de elementos de memória são bastante mais reduzidos e torna-se possível executar comandos num único ciclo de relógio. As limitações e desempenho passam a ser definidos apenas pela infra-estrutura OCD e pela sua interface com o microprocessador.

Esta abordagem tem interesse prático limitado, pois perde-se a conformidade com a norma e aumenta-se consideravelmente a dimensão do barramento de comunicações. A sua utilização seria limitada a protótipos ou a componentes utilizados exclusivamente para depuração ou injeção de falhas [Fidalgo06e].

5.5 Injecção de Falhas Embutida (OCD-FI)

A injecção de falhas embutida baseia-se na expansão da infra-estrutura OCD, de forma a incluir funcionalidades específicas para este fim. Estas modificações deverão manter as funcionalidades de depuração já implementadas e as interfaces entre o exterior e a infra-estrutura modificada, que neste caso será designada como OCD-FI (*On Chip Debug and Fault Injection*).

A inclusão de funcionalidades adicionais foi efectuada através da adição de um módulo dedicado à injecção de falhas (módulo FI), que interage apenas com elementos da infra-estrutura OCD. Foi inicialmente desenvolvida uma versão de base, com o objectivo de permitir a sua execução em tempo real, com bom desempenho e o mínimo de sobrecarga lógica. De forma a minimizar essa sobrecarga, o módulo FI básico apenas pode ser utilizado quando é possível pré-determinar o vector que contém a falha que se pretende injectar.

Posteriormente, foi desenvolvida uma versão com capacidades adicionais, de forma a permitir um funcionamento mais autónomo, realizando grande parte da operação de injecção de falhas dentro da infra-estrutura OCD-FI. Esta última versão apresenta uma maior complexidade e sobrecarga lógica, mas não é tão restritiva, podendo operar autonomamente e sem pré-determinação do vector de dados com falha a injectar. As características e funcionalidades de cada versão são apresentadas na Tabela 5.8.

Tabela 5.8 – Características das duas versões propostas para a infra-estrutura OCD-FI

| Versão   | Activação                                                            | Predet      | Funcionalidades                                          | Memória                                           |
|----------|----------------------------------------------------------------------|-------------|----------------------------------------------------------|---------------------------------------------------|
| Base     | Por ponto de observação<br>Por ponto de paragem                      | Obrigatório | Controlo de activação                                    | 1 bit                                             |
| Avançada | Por ponto de observação<br>Por ponto de paragem<br>Por sinal externo | Opcional    | Controlo de activação<br>Determinação de valor com falha | N+1 bits<br>(N – dimensão do barramento de dados) |

A versão de base tem como objectivo incluir comportamento reactivo na infra-estrutura OCD da forma mais simples possível, estando o respectivo módulo FI representado na Figura 5.13.

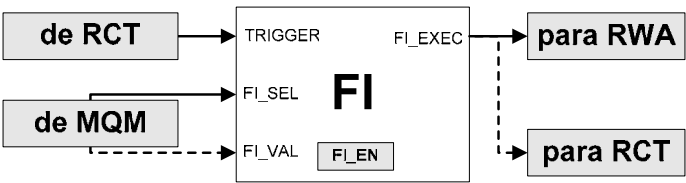


Figura 5.13 – Versão Base do Módulo FI

O módulo apenas necessita das três entradas e uma saída apresentadas, sendo eventualmente necessárias modificações mínimas (multiplexagem) na lógica de controlo do módulo RWA. Os sinais de entrada são utilizados para activar e configurar o módulo FI (via MQM), e para activar a injeção de falhas (via RCT). O sinal de saída é utilizado para activar a injeção de falhas no módulo RWA. Os sinais opcionais visam lidar com a possibilidade de desactivar a injeção de falhas, uma vez iniciado o processo, e de reiniciar a execução automaticamente, caso aquela obrigue à paragem do microprocessador.

Esta abordagem apresenta limitações em termos de flexibilidade de utilização, mas pode ser implementada com sobrecarga lógica muito reduzida. Deverá ser utilizada para sistemas relativamente simples, onde a pré-determinação dos valores com falha possa ser efectuada sem grandes problemas e onde a infra-estrutura OCD seja pequena, ou existam limites rígidos para a sobrecarga imposta sobre o sistema alvo.

A versão avançada é indicada para a maioria das situações, pois pode ser utilizada para a injeção de falhas automática, mesmo quando não é conhecida a aplicação em execução. A Figura 5.14 apresenta a arquitectura da infra-estrutura OCD resultante da inclusão do módulo FI, bem como o fluxo de dados resultante para aquelas operações.

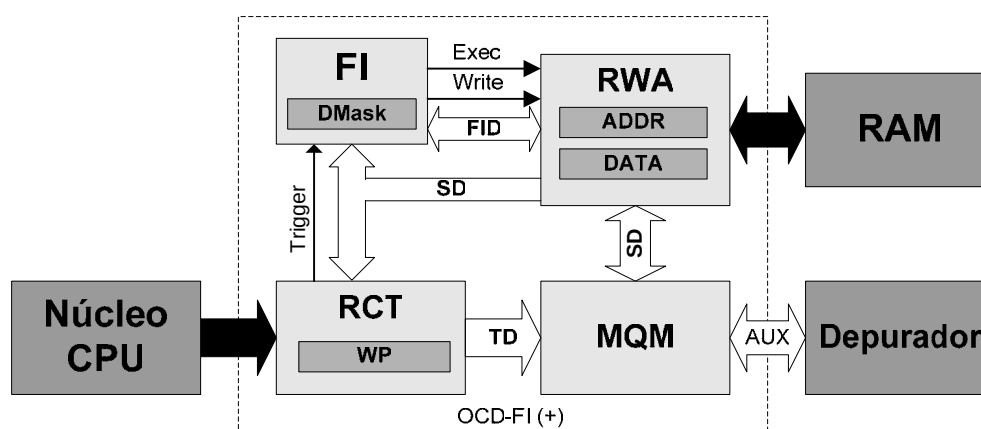


Figura 5.14 – Infra-Estrutura OCD-FI (simplificada)

Esta versão do módulo FI é consideravelmente mais complexa, pois terá de efectuar modificações num vector de dados. Os barramentos são identificados de acordo com o tipo de informação transmitida, durante cada experiência de injeção de falhas, designando TD os dados de activação (*Triggering Data*), SD os dados de configuração (*Set-Up Data*) e FID os dados de injeção de falhas (*Fault Injection Data*). Os principais sinais de controlo são também apresentados e consistem num sinal de activação (*Trigger*), que inicia o processo de

injecção de falhas, e em sinais de execução e escrita (*Exec* e *Write*), que são utilizados para controlar as operações de leitura e escrita, realizadas sobre o elemento alvo pelo módulo RWA, que também é responsável pela configuração do módulo FI (por comando externo do depurador). Como a injecção de falhas é limitada a um modelo do tipo inversão de bit, a operação executada é do tipo XOR, entre um vector de dados contido no módulo RWA (posterior a uma operação de leitura) e uma máscara contida num registo interno do módulo FI (*Dmask*), que define os bits a inverter. É portanto necessária alguma capacidade de memória e processamento. Em termos de interface, é necessário um barramento para escrita na *Dmask* e uma interface bidirecional com o módulo RWA, para as operações executadas durante a injecção de falhas. Devido ao interesse no elevado desempenho destas operações, este barramento deverá ser dedicado, de forma a evitar conflitos e os consequentes atrasos, existindo um outro barramento, utilizado para configuração do módulo FI e da respectiva máscara (via RWA). Adicionalmente, podem ser implementadas versões modificadas para situações específicas, as quais são abordadas na secção seguinte.

## 5.6 Extensões

Os mecanismos de injecção de falhas e metodologias descritos nas secções anteriores foram desenvolvidos para lidar com requisitos de injecção de falhas em tempo real, em dispositivos do tipo COTS. Foi também feito um esforço para manter a sobrecarga lógica e funcional num valor baixo e para minimizar efeitos no desempenho do sistema alvo. Para maximizar a aplicabilidade das soluções propostas, foi mantida em todas elas a conformidade com a proposta NEXUS. Esta abordagem de baixa intrusividade impõe algumas limitações à cobertura de falhas, a qual se pode revelar inadequada para aplicações ou componentes utilizados em funções críticas. Nestes casos é dada grande importância a características de confiabilidade, sendo normalmente aceitáveis algumas limitações em termos de desempenho, e a inclusão de infra-estruturas de teste e depuração, com alguma intrusividade ou sobrecarga.

Este tipo de conceito também pode ser utilizado para a injecção de falhas, podendo nestes casos ser útil acrescentar funcionalidades ou melhorar o desempenho da infra-estrutura OCD-FI. Em geral, estas modificações visam lidar com um problema específico, sendo concebidas à sua medida, pelo que a melhor forma de as implementar é recorrendo a

extensões da infra-estrutura, que deverão ser ajustadas para cada caso, de forma modular e com efeito mínimo sobre os componentes já implementados.

Para exemplificar este conceito, foram considerados dois problemas específicos cuja solução pode passar pela utilização deste tipo de extensões, e que são frequentes em cenários de injeção de falhas em microprocessadores tolerantes a falhas:

- **Injecção de falhas em memória protegida** por mecanismos de EDAC<sup>22</sup> (*Error Detection And Correction*): Este cenário visa a injeção de falhas em memória de sistemas com barramentos tolerantes a falhas, sem desactivar a protecção existente. O objectivo consiste em replicar a ocorrência de falhas o mais realisticamente possível, de forma a verificar a eficácia dos mecanismos de tolerância existentes. Neste caso específico, lida-se com soluções do tipo EDAC, que são muito utilizadas em sistemas críticos propensos a SEUs em memória.
- **Injecção de falhas nos registos internos** do microprocessador em tempo real: Este cenário visa estender as capacidades da infra-estrutura OCD, mantendo a conformidade com a proposta NEXUS, de forma a permitir operações de leitura e escrita nos registos internos do microprocessador (e não apenas no espaço de memória) durante a execução normal.

As extensões desenvolvidas para lidar com os cenários propostos designam-se por OCD-FI (EDAC) e OCD-FI (RTREG).

### 5.6.1 Injecção em mecanismos de detecção e correcção de erros

Os mecanismos de memória são a parte dos sistemas computacionais mais sensível aos efeitos dos SEUs, por representarem uma área considerável dos CIs modernos. É portanto frequente que os componentes de memória utilizados em sistemas críticos apresentem algum tipo de protecção contra este fenómeno.

A protecção mais frequente baseia-se na utilização de redundância espacial, na forma de bits adicionais guardados em conjunto com cada vector de dados. A versão mais simples consiste na utilização de um bit de paridade, que permite a detecção de erros do tipo inversão de bit, com o mínimo de sobrecarga lógica. A utilização de um número adicional de bits acrescenta capacidades de detecção de inversões múltiplas e correcção de inversões simples (ou mesmo

---

<sup>22</sup> Por vezes utiliza-se o acrónimo EDC.

múltiplas). Mais uma vez, o grau de protecção depende da sobrecarga lógica que se está disposto a tolerar. Nas versões mais frequentemente utilizadas em sistemas computacionais, utilizam-se códigos de Hamming [Lint82] [Kumar07], de forma a incluir a capacidade de correcção de inversões de bit simples, o que acarreta sobrecargas da ordem dos 3 bits, para sistemas de 8 bits, e de 6 a 8 bits para sistemas de 32 bits. Estes mecanismos são normalmente implementados nos barramentos de acesso à memória, sendo portanto externos ao núcleo do microprocessador.

No caso do *cpugenerator*, os sistemas alvo utilizados não se encontram equipados de origem com nenhum tipo de tolerância a falhas, pelo que se optou por introduzir um mecanismo do tipo EDAC no barramento de comunicação com a memória. Este mecanismo foi desenvolvido apenas para a versão de 32 bits e utiliza um conjunto de 6 bits adicionais. A Figura 5.15 e a Figura 5.16 apresentam mais detalhadamente o funcionamento da solução utilizada.

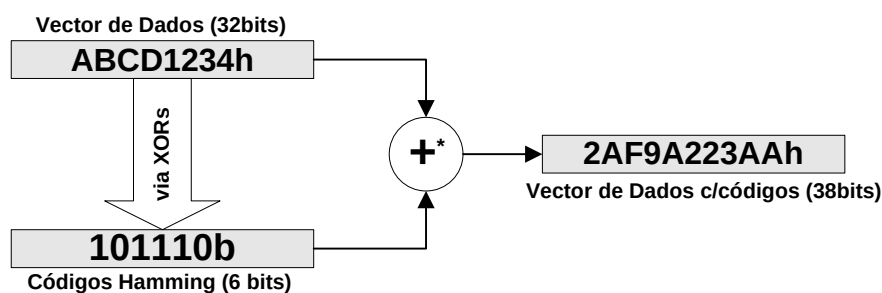


Figura 5.15 – Exemplo de codificação com EDAC

Neste exemplo o código é obtido através da execução de operações do tipo XOR a partir de elementos seleccionados do vector de dados (de acordo com o código). A operação indicada por  $+^*$  é uma concatenação em que os bits correspondentes ao código são colocados em posições pré-determinadas de forma a gerar o vector final.

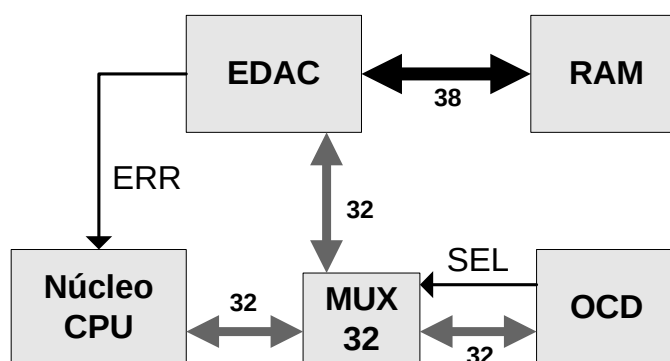


Figura 5.16 – Arquitectura EDAC (exemplo para 32 bits)

O sinal ERR sinaliza a detecção de um erro pelo mecanismo de EDAC e o sinal SEL faz o controlo do multiplexador (MUX32), de forma a permitir a partilha do EDAC pelo microprocessador e pela infra-estrutura OCD. Os seis bits adicionais permitem garantir a detecção de até dois erros do tipo inversão de bit, em cada vector de dados, bem como a detecção e correcção de erros individuais. Fisicamente, o mecanismo EDAC consiste na utilização de células de memória de maior dimensão, para guardar a informação adicional, e na inclusão de um módulo dedicado no barramento de comunicação entre microprocessador e memória. A determinação e escrita dos códigos adicionais é feita na escrita de dados em memória, com a detecção e correcção de erros a ser realizada no acesso para leitura. Sendo estas operações completamente combinatórias, o seu efeito no desempenho do microprocessador deverá ser mínimo, pois não necessitam de ciclos de relógio adicionais, impondo apenas alguns atrasos, derivados do maior número de componentes lógicos utilizados e das operações realizadas em cada acesso a memória.

O principal problema da introdução de falhas em memória protegida por EDAC, consiste em contornar os mecanismos de protecção existentes. As soluções utilizadas mais frequentemente são a injeção de falhas físicas sem contacto (directamente na memória), ou a desactivação dos mecanismos EDAC, pelo menos durante a operação de escrita. No primeiro caso, o processo apresenta as limitações de controlabilidade e custo identificadas anteriormente; no segundo, o principal problema é a perda de representatividade assim introduzida. Desligar o principal elemento de tolerância a falhas durante a operação de injeção só faz sentido se esta for executada com o microprocessador suspenso, pois apenas desta forma será possível garantir que o mecanismo opera adequadamente, uma vez que esteve parado simultaneamente com o microprocessador alvo.

A alternativa proposta é permitir à infra-estrutura OCD-FI o acesso directo à memória, sem desligar o mecanismo EDAC, que deverá ser contornado. Para poder efectuar este acesso, aquela infra-estrutura deverá seguir os mecanismos de controlo de colisões utilizados anteriormente, mas também terá de calcular os bits adicionais de controlo de erros. Adicionalmente, ao ser utilizada para injeção de falhas, aquela infra-estrutura deverá permitir a introdução de todos os 38 bits. Se o valor com falha for determinado exteriormente, esta operação terá de ser realizada em conjunto com os sinais de controlo, pois se apenas for alterado um dos 32 bits de dados e a infra-estrutura OCD calcular os novos bits de controlo antes da sua escrita em memória, o valor será interpretado como correcto pelo mecanismo de EDAC e a falha do tipo inversão de bit não será introduzida. Uma infra-estrutura do tipo OCD-FI (EDAC) deverá ser capaz de efectuar a inversão de um

bit na palavra que irá ser escrita em memória (38 bits), fornecendo portanto uma solução mais eficiente. A Figura 5.17 apresenta a configuração modificada para o acesso a memória por parte da infra-estrutura OCD-FI (EDAC), em paralelo com o mecanismo EDAC.

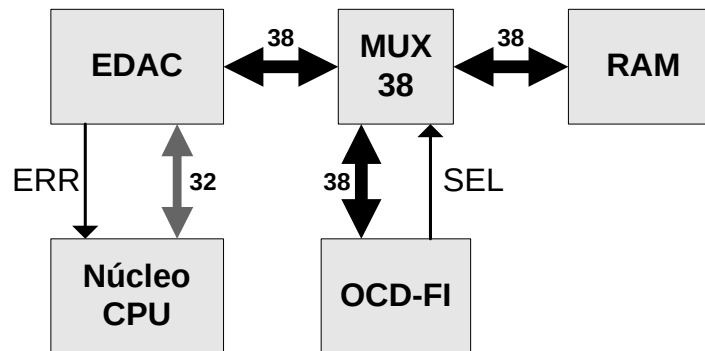


Figura 5.17 – Arquitectura EDAC modificada com OCD-FI

Esta extensão requer que a infra-estrutura OCD-FI possa operar como OCD normal em operações de escrita, adicionando os valores de controlo correctos, ou ignorando-os, quando utilizada em áreas de memória não protegidas por EDAC (e.g. é possível ter blocos de memória separados e apenas proteger parte deles). Quando se pretender uma operação de injeção de falhas, a infra-estrutura poderá utilizar a pré-determinação do valor com falha. Caso esta seja possível, o valor a introduzir terá de ser determinado com os bits de controlo. Caso a determinação seja executada em tempo real pela infra-estrutura, esta deverá alterar apenas um bit, e escrever o valor resultante na posição de memória alvo.

### 5.6.2 Injeção em registos em tempo real

O acesso, por parte de infra-estruturas de depuração e teste, a recursos do microprocessador em tempo real, é limitado pelas restrições temporais inerentes ao seu núcleo e seus registos internos. Os elementos de memória externos poderão ser acedidos do exterior, de forma relativamente simples, embora o mesmo já não seja verdade para os elementos internos, devido a limitações espaciais e temporais. Em termos espaciais, o problema é a impossibilidade de partilhar os meios de acesso utilizados pelo microprocessador, sendo necessário utilizar vias paralelas, com aumento da lógica de controlo e portos de acesso. Em operações de escrita a complexidade aumenta, pois torna-se necessário (1) o controlo de colisões, de modo a evitar escritas simultâneas, e (2) a transmissão bidireccional de dados. Em termos temporais, as limitações prendem-se com a influência deste tipo de acesso (e

depuração) no desempenho do microprocessador alvo. Uma possibilidade de resolver estes problemas seria o acesso a frequência mais elevada, sincronizado com momentos de inacção por parte do microprocessador. Considerando que os registos internos operam à sua frequência de relógio, que normalmente se encontra perto dos limites físicos do componente, torna-se pouco provável que a infra-estrutura OCD possa aceder eficientemente a esses registos a uma frequência substancialmente mais elevada. Esta solução não poderá assim ser considerada para a generalidade dos casos. A alternativa passa por utilizar um controlo de colisões equilibrado, que permita o acesso aos registos para efeitos de depuração, quando não estejam a ser usados pelo microprocessador. Desta forma, será possível manter o desempenho dentro de limites aceitáveis e permitir o acesso parcial aos seus registos internos.

Para lidar com a injeção de falhas nos registos internos do microprocessador foi desenvolvida a extensão OCD-FI (RTREG), que requer alterações ao sistema alvo e à infra-estrutura OCD-FI, mas sem obrigar a perda de conformidade com a proposta NEXUS. O problema de acesso foi resolvido através de modificações ao ficheiro de registos (*Register file*) do componente, de forma a permitir acesso simultâneo por parte do microprocessador e da infra-estrutura de depuração, como se apresenta na Figura 5.18.

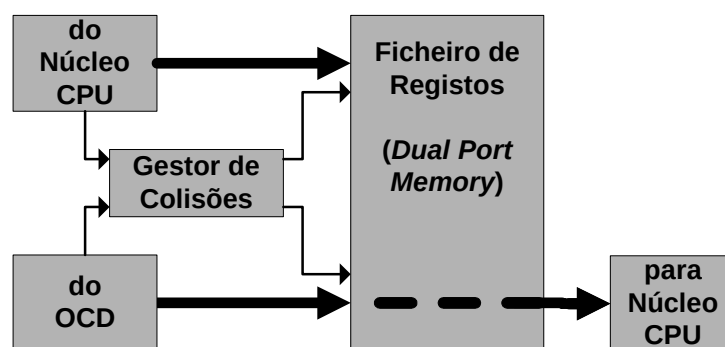


Figura 5.18 – Infra-estrutura OCD-FI (RTREG)

A leitura pode ser executada por ambas entidades, sendo possível o acesso simultâneo ao mesmo endereço ou a endereços diferentes. Esta solução implica alguma lógica de controlo adicional e eventuais perdas de desempenho, dependendo da configuração física dos blocos de memória. De uma forma geral, o acesso para leitura apenas acarretará uma sobrecarga espacial, enquanto que o acesso para escrita será limitado, quanto aos instantes de tempo em que é possível aceder a um determinado registo. A dimensão do problema dependerá da utilização de cada registo, tendo os mais utilizados um acesso mais complexo. É importante notar que, em tempo real, é sempre difícil garantir a actualidade da informação obtida por

esta forma. Considerando que os registos do microprocessador são acedidos e modificados com bastante frequência, é elevada a probabilidade da informação recolhida pela infra-estrutura de depuração estar desactualizada, quando chega ao utilizador (seja este um mecanismo de injeção de falhas, um periférico ou um utilizador humano).

Para as operações de escrita, é necessário definir uma política de prioridades, com efeitos no equilíbrio entre a cobertura de falhas (do mecanismo de injeção), a intrusividade e o desempenho. Este último factor é decisivo e torna lógica a opção pela prioridade no acesso ao microprocessador, sendo sempre possível dar prioridade à infra-estrutura, ou uma solução mista, nos casos em que a cobertura de falhas seja particularmente crítica. É importante relembrar que é sempre possível recorrer à injeção de falhas em registos internos, com paragem do microprocessador alvo, mesmo com as infra-estruturas de depuração mais simples, sendo esta uma outra solução, que permite obter maior cobertura, mas também com penalizações em termos de intrusividade e desempenho.

É portanto necessário considerar dois casos, que requerem algum tipo de controlo de colisões. O primeiro corresponde à escrita por um dos componentes, em simultâneo com a leitura por parte do outro. Esta solução deverá ser possível, sendo apenas necessário algum controlo, no caso de as operações serem executadas sobre o mesmo endereço. Deverá ser utilizado o valor em escrita como fonte para a operação de leitura, visto que os dados desta operação são disponibilizados um ciclo de relógio após a leitura do endereço. O segundo caso será o de uma operação de escrita simultânea por ambos os componentes, que apenas exigirá controlo de colisões. Se o endereço for o mesmo é importante dar a prioridade ao microprocessador. O valor efectivamente escrito no registo será o pretendido pelo microprocessador. O valor a escrever pela infra-estrutura pode ser processado de duas formas diferentes (configuráveis). No caso padrão, o valor é simplesmente ignorado, considerando-se para efeitos de depuração e injeção de falhas, que o valor que se pretendia escrever foi imediatamente sobrescrito pelo microprocessador. Esta abordagem é aceitável para a maioria das aplicações, não sendo no entanto completamente representativa. Uma alternativa será a utilização de um mecanismo de espera, em que a infra-estrutura de depuração aguarda que seja possível executar a operação pretendida. Como esta operação é primordialmente utilizada para a injeção de falhas, a infra-estrutura OCD-FI não deverá executar outras operações em simultâneo, pelo que estes tempos de espera não são muito relevantes, mesmo que longos. É apenas importante evitar situações de bloqueio, em que a operação não seja exequível em tempo útil. O problema do não determinismo é resolvido pelo planeamento da experiência. O processo de pré-determinação do valor com falha deverá

ser ajustado de forma a incluir a escolha do momento correcto para a activação da falha, de modo a que esta seja inserida no local e no instante pretendidos. Haverá uma redução da cobertura, pois não será possível inserir falhas em registos, nos instantes em que o microprocessador também escreve neles. Este problema é portanto uma simplificação, que assume que se ocorrer um SEU num registo interno do microprocessador, no preciso momento em que este é acedido para escrita, os seus efeitos serão eliminados pelos dados em escrita.

## **5.7 Resumo e Conclusões**

Este capítulo fez a apresentação das propostas de solução para as limitações identificadas nos capítulos anteriores. A metodologia proposta baseia-se no aumento de desempenho e migração de algumas funcionalidades para o interior da infra-estrutura OCD. Simultaneamente é proposta a utilização de um depurador especificamente desenvolvido para suportar a injeção de falhas via infra-estruturas em conformidade com a proposta NEXUS e tirando partido das funcionalidades por ela disponibilizadas.

Na sua abordagem mais simples, propõe-se a utilização do referido depurador com uma infra-estrutura normalizada, sendo nele executada a maioria das operações envolvidas na injeção de cada falha. O aumento da largura de banda da interface entre o depurador e a infra-estrutura OCD, através da utilização de sinais adicionais, é considerado como forma de aumentar o desempenho, permitindo uma inserção de falhas mais rápida. Uma solução alternativa baseada em modificações ao funcionamento da própria infra-estrutura OCD é proposta, como forma de fornecer um desempenho igual ou superior e capacidade de reacção melhorada. Esta última característica baseia-se na inclusão de capacidades básicas de manuseamento de dados num módulo a adicionar à infra-estrutura OCD de modo a que o processo de injeção de falhas possa ser efectuado de forma automática com a mínima intervenção por parte de elementos externos. A solução proposta é assim escalável, podendo ser adaptada ao sistema alvo específico, permitindo diferentes equilíbrios entre capacidades fornecidas e modificações necessárias.

*“Under the most rigorously controlled conditions of pressure, temperature, humidity and other variables, the experiment will do as it damn well pleases.”*

Corolário da Lei de Murphy

**Capítulo 1**

**Capítulo 2**

**Capítulo 3**

**Capítulo 4**

**Capítulo 5**

**Capítulo 6**

**Capítulo 7**

**Referências**

**Anexo**

# Trabalho Experimental



## 6 Trabalho Experimental

Dividido em cinco secções, este capítulo apresenta uma descrição do trabalho experimental, incluindo a verificação das diferentes soluções propostas e uma análise comparativa dos resultados obtidos. A primeira secção apresenta uma descrição das condicionantes do trabalho experimental, a segunda secção aborda cenários básicos em que se pretendeu avaliar a eficiência das diferentes técnicas de injeção de falhas em microprocessadores de diferente dimensão, a terceira secção visa uma análise mais aprofundada das diferentes técnicas, focada num único microprocessador relativamente complexo, e a quarta tem como objectivo avaliar os resultados obtidos com as extensões à infra-estrutura OCD-FI. A última secção apresenta um resumo e as conclusões gerais. Em cada um dos casos experimentais são utilizados diferentes cenários, cada qual definido pelo sistema alvo (microprocessador, aplicação e infra-estrutura OCD) e pelas características das campanhas de injeção de falhas. Para cada caso são descritos os objectivos pretendidos, os procedimentos experimentais mais importantes, os principais resultados obtidos e os eventuais problemas detectados.

### 6.1 Condicionantes do Trabalho Experimental

O trabalho experimental apresentado foi executado numa máquina tipo PC, utilizando o ambiente de simulação *Modelsim* [MentorG04][MentorG07]. Todos os componentes foram implementados como módulos VHDL [IEEE00] e sintetizados utilizando o ambiente de desenvolvimento Xilinx ISE versão 7 [Xilinx05][Xilinx07]. A síntese foi executada de forma idêntica para todos os componentes, utilizando parâmetros padrão, visando um equilíbrio entre área e desempenho.

Apesar de não utilizar a implementação física dos componentes, esta abordagem permite o melhor equilíbrio entre a exactidão dos resultados, o tempo necessário para as experiências e as restrições logísticas. Os modelos e as ferramentas utilizados permitem considerar diferentes configurações, analisar detalhadamente o desempenho e o comportamento dinâmico e obter resultados fiáveis em tempo útil. Considera-se que a inclusão em dispositivos programáveis (e.g. FPGAs) forneceria resultados idênticos, salvo erros de síntese ou utilização muito próxima dos limites do equipamento. Para minimizar esta possibilidade e obter resultados representativos, a simulação recorreu a frequências de relógio de aproximadamente 90% do valor máximo possível evitando-se também opções de

síntese demasiado agressivas. Utilizou-se como alvo para as operações de síntese uma FPGA do tipo Virtex-2, por ser um dispositivo tecnologicamente estável, disponível em diferentes configurações e dimensões. O ambiente de injeção de falhas utilizado foi descrito em detalhe nos capítulos anteriores e é novamente ilustrado na Figura 6.1.

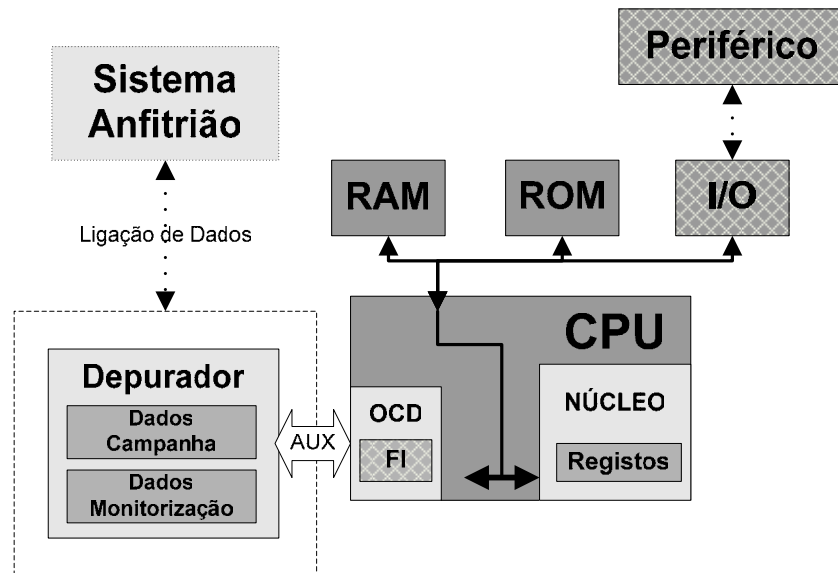


Figura 6.1 – Ambiente de Injecção de Falhas

Alguns elementos apenas são utilizados em certas configurações específicas, como é o caso do módulo FI, utilizado apenas com a infra-estrutura OCD-FI, e dos módulos I/O e periféricos associados, utilizados apenas com a aplicação *XControl*. O sistema anfitrião representa uma máquina externa, destinada à gestão das campanhas de injeção de falhas e à posterior análise dos dados de monitorização. Este tipo de tarefa exige capacidades consideráveis de processamento e de armazenamento de informação. A sua actuação foi emulada, de forma abstracta, através da escrita de dados nas memórias do sistema alvo e do depurador, antes do início de cada simulação, e sua posterior leitura, após a execução de cada campanha de injeção de falhas.

O depurador utilizado é estruturalmente o mesmo para todos os cenários, embora, dependendo do sistema alvo, sejam alterados alguns parâmetros do modelo, particularmente os relacionados com a interface e dimensão dos barramentos. Da mesma forma, as memórias incluídas no depurador podem ser alteradas em largura e em profundidade. Para maior eficiência, a largura deverá estar sempre de acordo com a dimensão dos barramentos do sistema alvo, enquanto que a profundidade é limitada apenas pelas capacidades do componente (i.e. FPGA) onde se pretende sintetizar o depurador. Cada campanha de

injecção de falhas é descrita por um único programa, dividido em experiências, tal como descrito no capítulo anterior. Todas as campanhas foram geradas utilizando guiões (*templates*), onde foram substituídos os valores relevantes, como endereços e vectores de dados, utilizados para activação e inserção de falhas. A análise dos resultados de monitorização foi efectuada através de comparação com os resultados de uma execução sem falhas (*gold run*), de onde resultaram os valores esperados para as saídas e memória, sendo o mecanismo de diagnóstico adaptado a cada aplicação específica. Os sistemas alvo foram gerados recorrendo à ferramenta *cpugenerator*, sendo acrescentados os elementos adicionais e as infra-estruturas adequadas a cada cenário. A entrada de controlo requerida pela aplicação *XControl* foi implementada através de um módulo adicional, para simular o comportamento de um periférico exterior, responsável pelo controlo do microprocessador, a cujos comandos responde. As secções seguintes discutem os três objectivos operacionais, sintetizados na Tabela 6.1.

Tabela 6.1 – Cenários operacionais e suas características

|                             | <b>Análise de Exequibilidade</b>                                       | <b>Utilização Operacional</b>                                              | <b>Extensões</b>                                                           |
|-----------------------------|------------------------------------------------------------------------|----------------------------------------------------------------------------|----------------------------------------------------------------------------|
| <b>Objectivo principal</b>  | Análise comparativa das diferentes soluções propostas                  | Estudo de um caso operacional realista                                     | Avaliação de possibilidades de extensão da infra-estrutura OCD-FI          |
| <b>Sistema alvo</b>         | 8, 16 e 32 bits                                                        | 32 bits                                                                    | 32 bits<br>Com EDAC / RTREG                                                |
| <b>Aplicações alvo</b>      | Com tolerância a falhas por <i>software</i><br>Sem periféricos         | Com e sem tolerância a falhas por <i>software</i><br>Com e sem periféricos | Com e sem tolerância a falhas por <i>software</i><br>Com e sem periféricos |
| <b>Infra-estruturas OCD</b> | OCD base<br>OCD com interface melhorado<br>OCD paralelo<br>OCD-FI base | OCD base<br>OCD com interface melhorado<br>OCD-FI base<br>OCD-FI autónoma  | OCD-FI (EDAC)<br>OCD-FI (RTREG)                                            |
| <b>Injecção de Falhas</b>   | Em tempo real                                                          | Com suspensão do sistema alvo<br>Em tempo real                             | Em tempo real                                                              |

Cada um dos cenários operacionais apresenta os seus objectivos específicos e condicionantes, subdividindo-se em cenários experimentais, tal como exposto nas secções seguintes.

## 6.2 *Análise de Exequibilidade*

Numa primeira parte analisou-se a exequibilidade da metodologia e soluções propostas, testando-as em vários cenários, onde se utilizaram diversas variantes do microprocessador. Os objectivos propostos para esta fase foram os seguintes:

- Verificar a exequibilidade da injeção de falhas em tempo real, utilizando infra-estruturas OCD normalizadas, em diversos tipos de microprocessador.
- Testar um depurador optimizado para a injeção de falhas e verificar a sua adaptabilidade nos cenários considerados.
- Avaliar a influência do desempenho da infra-estrutura e da sua interface no processo de injeção de falhas. Comparar diferentes alternativas, em termos de largura de banda do porto de comunicações.
- Analisar a exequibilidade e desempenho do acesso em paralelo entre depurador e infra-estrutura OCD.
- Validar a infra-estrutura OCD-FI básica, como uma solução viável para a injeção de falhas em tempo real e sem intrusividade. Comparar o seu desempenho com soluções alternativas em diferentes cenários.

### 6.2.1 *Execução das Experiências*

Nesta fase foram consideradas três configurações de microprocessador, diferindo apenas em termos de dimensão dos barramentos. O suporte para interrupções externas não foi utilizado e em todos os casos a profundidade da pilha é de quatro *bytes*. Todas as configurações utilizam memória ROM e RAM separadas, destinando-se a primeira a armazenar a aplicação alvo e a segunda a armazenar os dados e os resultados. Foram consideradas duas aplicações alvo, nomeadamente a *MAdder* e a *VSorter*, sendo as dimensões dos dados variáveis consoante o microprocessador. Ambas as aplicações foram utilizadas na versão base e numa versão com tolerância a falhas por *software*, incluída no respectivo código.

As campanhas de injeção de falhas utilizadas foram definidas da seguinte forma:

- As falhas são exclusivamente inseridas em memória RAM, que é a única acessível para escrita, neste microprocessador e infra-estrutura OCD.
- As campanhas compreendem 10 experiências individuais, cada uma consistindo na injeção de uma falha do tipo inversão de bit, numa célula de memória RAM, num determinado instante da execução do programa.

- A instrução utilizada para a activação da injeção de falhas é seleccionada aleatoriamente, para cada experiência, a partir do conjunto de instruções executadas (em ROM). Apesar de ser possível, não faz sentido utilizar partes da memória ROM, que não são efectivamente lidas, pois isso levaria à não activação da injeção de falhas.
- O conhecimento prévio do programa a executar permite a escolha do instante de injeção de falha, a partir do conjunto de instruções executadas, utilizando para tal um ponto de observação, associado à enésima execução de uma determinada instrução.
- No caso da aplicação *MAdder*, podemos determinar à partida quantas vezes cada instrução é executada, sendo portanto possível escolher o número de execuções a esperar antes da activação da falha.
- No caso da aplicação *VSorter*, o fluxo de programa e o número de execuções de cada instrução depende dos dados a processar, o que obriga a testar a activação de cada falha individualmente.
- Considera-se o instante de activação como o momento em que é gerado um sinal na infra-estrutura OCD, que indica a ocorrência da respectiva condição. A partir deste instante, segue-se um intervalo de tempo, até ocorrer a inserção da falha propriamente dita, que é inserida em memória após um atraso, dependente da metodologia utilizada.
- A célula e o bit onde a falha é inserida são também seleccionados aleatoriamente, a partir da totalidade do espaço de memória. Seria possível considerar apenas a memória efectivamente utilizada, mas a opção pelo espaço total permite replicar de forma mais realista os fenómenos do tipo SEU e quantificar o efeito da redundância natural da aplicação.
- O sistema alvo e a respectiva aplicação são carregados antes de cada campanha. As experiências são executadas sequencialmente, iniciando-se cada uma pela reinicialização do sistema alvo e pela configuração da infra-estrutura OCD.
- Todas as operações de configuração da infra-estrutura OCD são executadas antes de se iniciar a execução do sistema alvo.
- Os dados retornados pela infra-estrutura OCD são lidos após a conclusão da campanha de injeção de falhas. A análise dos resultados é efectuada externamente. Em condições normais (i.e. por omissão), apenas são registados os dados de monitorização.

Os cenários utilizados visam avaliar as seguintes situações:

- Aplicabilidade, funcionalidade e desempenho das propostas de injeção de falhas em tempo real, para os diversos tipos de microprocessador.
- Variação de desempenho da injeção de falhas, utilizando infra-estruturas OCD com diferentes dimensões de barramentos de comunicação. Esta avaliação centrou-se no processador de 8 bits.
- Efeito dos barramentos internos no desempenho da injeção de falhas. Esta avaliação é feita comparando os resultados com os diferentes microprocessadores (8, 16 e 32 bits), utilizando as infra-estruturas OCD e OCD-FI básicas.
- Dimensão lógica e complexidade acrescida dos diversos microprocessadores e dos componentes da infra-estrutura de depuração e injeção de falhas.

Como foi referido, a análise dos resultados é efectuada externamente (i.e. no sistema anfitrião), consistindo numa inspecção dos resultados dos dados de monitorização, das respostas do sistema e da sua comparação com os valores esperados.

### 6.2.2 Implementações Normalizadas

Foram utilizados três grupos de cenários adaptados a objectivos específicos. Numa primeira fase utilizaram-se apenas implementações normalizadas da infra-estrutura OCD, variando na dimensão dos barramentos responsáveis pela interface com o exterior, que fazem parte do porto AUX e recebem as designações MDI e MDO. A Tabela 6.2 apresenta os cenários utilizados, que podem ser comparados com os correspondentes ao MPC565.

Tabela 6.2 – Cenários experimentais com infra-estrutura OCD simples

| #   | Sistema Alvo | BUS (bits) | CLK (MHz) | MDI (bits) | MDO (bits) |
|-----|--------------|------------|-----------|------------|------------|
| A1  | CPU8_i1o1    | 8          | 100       | 1          | 1          |
| A2  | CPU8_i1o2    | 8          | 100       | 1          | 2          |
| A3  | CPU8_i2o4    | 8          | 100       | 2          | 4          |
| A4  | CPU8_i2o8    | 8          | 100       | 2          | 8          |
| A5  | CPU8_i4o8    | 8          | 100       | 4          | 8          |
| A6  | CPU16_i2o4   | 16         | 45        | 2          | 4          |
| A7  | CPU16_i2o8   | 16         | 45        | 2          | 8          |
| A8  | CPU32_i2o4   | 32         | 25        | 2          | 4          |
| A9  | CPU32_i2o8   | 32         | 25        | 2          | 8          |
| A10 | CPU32_i4o16  | 32         | 25        | 4          | 16         |
| N/A | MPC565       | 32         | 40        | 2          | 8          |

As frequências de operação foram seleccionadas ligeiramente abaixo de 90% da frequência máxima da configuração mais complexa, para cada tipo de microprocessador. Por exemplo, no caso do microprocessador de 8 bits, a frequência máxima possível seria cerca de 113 MHz (calculada pela ferramenta de síntese).

Cada cenário foi submetido a vinte campanhas de injeção de falhas, compreendendo cada uma dez experiências diferentes, geradas aleatoriamente. As campanhas são divididas em doze grupos, da forma descrita na Tabela 6.3.

Tabela 6.3 – Campanhas executadas por Aplicação (I)

|                         | Sem Predet | Com Predet |
|-------------------------|------------|------------|
| <i>MAdder8_v1.2FT</i>   | 25 (5x5)   | 25 (5x5)   |
| <i>MAdder16_v1.0FT</i>  | 10 (5x2)   | 10 (5x2)   |
| <i>MAdder32_v1.1FT</i>  | 15 (5x3)   | 15 (5x3)   |
| <i>VSorter8_v1.1FT</i>  | 25 (5x5)   | 25 (5x5)   |
| <i>VSorter16_v1.0FT</i> | 10 (5x2)   | 10 (5x2)   |
| <i>VSorter32_v1.0FT</i> | 15 (5x3)   | 15 (5x3)   |
| TOTAL                   | 100        | 100        |

Metade das campanhas recorre à determinação do vector de dados com falha durante a execução e a outra metade recorre à pré-determinação (Predet), tal como descrito no capítulo 4. Esta abordagem permite obter 100 experiências diferentes, em termos de instante de activação e localização da falha, para cada cenário, repetidas duas vezes, perfazendo um total de 2000 experiências. Como o código das aplicações tem a mesma dimensão para diferentes arquitecturas, é possível manter os instantes de activação para todas as campanhas. As variações em termos de dimensão dos dados, entre as várias arquitecturas, impossibilitam que os pontos onde as falhas são injectadas sejam os mesmos. Metade das campanhas utilizou a aplicação *MAdder* e a outra metade a aplicação *VSorter*, sendo utilizadas apenas as versões com tolerância a falhas por *software*.

Uma análise detalhada do correspondente código VHDL, e das formas de onda geradas na simulação, permitiu determinar com exactidão os intervalos de tempo necessários para as principais operações de injeção de falhas, expressos em ciclos de relógio (Clk) na Tabela 6.4.

Tabela 6.4 – Intervalos de tempo para as operações de inicialização e injeção de falhas (I)

| #   | Sistema Alvo | Com Predet          |                | Sem Predet          |                |
|-----|--------------|---------------------|----------------|---------------------|----------------|
|     |              | Inicialização (Clk) | Inserção (Clk) | Inicialização (Clk) | Inserção (Clk) |
| A1  | CPU8_i1o1    | 20                  | 16             | 20                  | 37             |
| A2  | CPU8_i1o2    | 20                  | 16             | 20                  | 30             |
| A3  | CPU8_i2o4    | 10                  | 9              | 10                  | 17             |
| A4  | CPU8_i2o8    | 10                  | 9              | 10                  | 15             |
| A5  | CPU8_i4o8    | 6                   | 6              | 6                   | 11             |
| A6  | CPU16_i2o4   | 14                  | 13             | 14                  | 23             |
| A7  | CPU16_i2o8   | 14                  | 13             | 14                  | 20             |
| A8  | CPU32_i2o4   | 22                  | 21             | 22                  | 35             |
| A9  | CPU32_i2o8   | 22                  | 21             | 22                  | 30             |
| A10 | CPU32_i4o16  | 12                  | 12             | 12                  | 18             |

Os intervalos de tempo referidos representam valores mínimos e pressupõem que durante a inserção da falha não é executada nenhuma operação de depuração, nem ocorre nenhum evento no sistema alvo, que origine uma mensagem de depuração. O tempo de inicialização indicado é necessário para carregar a infra-estrutura OCD com os dados necessários à execução de cada experiência de injeção de falhas. O tempo de inserção decorre entre o sinal de activação da falha e a sua inserção efectiva no elemento alvo.

A Tabela 6.5 apresenta os resultados obtidos com a execução das campanhas de injeção de falhas, classificados de acordo com a utilização ou não da pré-determinação e divididos nas seguintes categorias:

- ERR (*OCD Error*) – Erro de execução, assinalado pela infra-estrutura OCD, normalmente consistindo em perda de informação em consequência da saturação (*overflow*) de um dos *buffers* internos.
- INC (*Inconclusive Result*) – Resultado inconclusivo, representando uma inserção de falha incorrecta, em termos do instante ou modelo de falha, detectada por monitorização do acesso a dados.
- OK (*No Error*) – Sem erro, representando uma inserção correcta da falha, de acordo com o modelo pretendido.

Tabela 6.5 – Resultados experimentais com infra-estrutura OCD simples

| #   | Sistema Alvo | Com Predet |     |     | Sem Predet |     |     |
|-----|--------------|------------|-----|-----|------------|-----|-----|
|     |              | ERR        | INC | OK  | ERR        | INC | OK  |
| A1  | CPU8_i1o1    | 88%        | 7%  | 5%  | 100%       | 0%  | 0%  |
| A2  | CPU8_i1o2    | 12%        | 16% | 72% | 48%        | 18% | 34% |
| A3  | CPU8_i2o4    | 0%         | 3%  | 97% | 0%         | 5%  | 95% |
| A4  | CPU8_i2o8    | 0%         | 3%  | 97% | 0%         | 4%  | 96% |
| A5  | CPU8_i4o8    | 0%         | 2%  | 98% | 0%         | 3%  | 97% |
| A6  | CPU16_i2o4   | 0%         | 4%  | 96% | 0%         | 7%  | 93% |
| A7  | CPU16_i2o8   | 0%         | 4%  | 96% | 0%         | 6%  | 94% |
| A8  | CPU32_i2o4   | 9%         | 5%  | 86% | 10%        | 8%  | 82% |
| A9  | CPU32_i2o8   | 0%         | 5%  | 95% | 0%         | 6%  | 94% |
| A10 | CPU32_i4o16  | 0%         | 2%  | 97% | 0%         | 3%  | 97% |

Nesta fase, apenas foi analisada a efectiva injeção de falhas, sem que o diagnóstico dos seus efeitos seja executado em detalhe. É importante notar que os resultados classificados como INC requerem uma monitorização de dados, específica ao processo de injeção de falhas, sem a qual seria mais complexo identificar a sua ocorrência. Esta monitorização é não intrusiva, mas acrescenta alguma sobrecarga à infra-estrutura OCD, visto requerer um número adicional de mensagens de monitorização. Caso não seja efectuada esta monitorização adicional, a classificação dos resultados do tipo INC requer uma análise das formas de onda ou uma análise detalhada dos resultados finais da execução da aplicação. Ambas as opções seriam de utilização difícil, num ambiente de injeção de falhas operacional.

Dos resultados obtidos podem ser extraídas as seguintes conclusões, relativas à utilização de infra-estruturas NEXUS:

- A inicialização de cada experiência de injeção de falhas é possível como o sistema alvo em execução, desde que seja verificado *a priori* se este processo pode ser concluído, antes do instante de activação pretendido para a falha. Esta alternativa permitiria eliminar a sobrecarga temporal imposta pela fase de inicialização da maioria das experiências de injeção de falhas. Neste caso específico optou-se por fazer a inicialização antes de se iniciar a execução da aplicação alvo, pois a experiência demoraria um tempo consideravelmente mais elevado, sem vantagens apreciáveis.

- A análise detalhada dos casos classificados como ERR permitiu verificar que estes são causados pela incapacidade de processamento do volume de dados de monitorização, em tempo útil, por parte da interface da infra-estrutura OCD.
- Experiências adicionais permitiram concluir que o aumento da profundidade dos *buffers* internos da infra-estrutura OCD permite atenuar ligeiramente (entre 0 e 10%) a ocorrência de resultados do tipo ERR. No entanto, na maioria dos casos estes advêm de um défice de capacidade de transferência média da interface, sendo necessária a utilização de uma maior largura de banda (i.e. outra configuração).
- Uma análise detalhada dos resultados classificados como INC permitiu verificar que estes são causados pelo acesso para escrita, por parte do microprocessador, no intervalo de tempo que decorre entre a activação da falha e a sua inserção.
- A ocorrência de resultados do tipo INC encontra-se fortemente ligada ao tempo necessário para a inserção de falhas, visto ser durante esta fase que um acesso por parte do microprocessador pode afectar o respectivo processo.
- A pré-determinação reduz o tempo necessário para a injeção da falha, reduzindo também a probabilidade de ocorrência de resultados de tipo INC.
- Algumas das campanhas de injeção de falhas não seriam exequíveis (em tempo real) num sistema alvo utilizando o MPC565 e depuradores comerciais, pois a aplicação alvo é executada num intervalo de tempo relativamente curto. Em certos casos a totalidade da aplicação alvo executa em menos tempo do que seria necessário para a injeção de uma falha num MPC565, utilizando equipamento COTS.

Relativamente à influência interface da infra-estrutura OCD, são possíveis as seguintes conclusões adicionais:

- Cada arquitectura exige um número mínimo de ciclos de relógio para inicializar as campanhas de injeção de falhas. O correspondente intervalo de tempo afecta a duração geral da experiência, se a inicialização for executada com o sistema alvo parado, sendo a diferença menos relevante, se o tempo de execução for elevado.
- A largura do canal de comunicações entre o depurador e a infra-estrutura OCD afecta o desempenho, no que respeita ao número de ciclos de relógio necessários para as operações de injeção de falhas, e consequentemente a percentagem de experiências incorrectas.

- Os efeitos referidos fazem com que o processo de injeção de falhas seja mais eficiente para barramentos de dados com maior dimensão. A utilização da pré-determinação beneficia particularmente do aumento do barramento MDI, enquanto que, no caso geral, o aumento do barramento MDO também aumenta o desempenho.

### 6.2.3 Infra-estruturas modificadas

Numa segunda fase, optou-se por introduzir infra-estruturas do tipo OCD-FI nos três microprocessadores utilizados, para comparar os resultados face ao uso de infra-estruturas OCD sem módulo FI. Para este efeito, foram consideradas as versões que apresentam a interface base e aquelas que apresentaram o melhor desempenho. A versão do OCD-FI utilizada foi a versão mais simples, que requer a pré-determinação dos valores da falha. As campanhas utilizadas são idênticas para os microprocessadores do mesmo tipo, com pequenas variações de inicialização, para as versões que utilizam OCD-FI. Os cenários considerados são apresentados na Tabela 6.6.

Tabela 6.6 – Cenários experimentais com infra-estruturas OCD e OCD-FI

| #   | Sistema Alvo  | Tipo (bits) | CLK (MHz) | MDI (bits) | MDO (bits) | Tipo de infra-estrutura |
|-----|---------------|-------------|-----------|------------|------------|-------------------------|
| B1  | CPU8_i2o4     | 8           | 100       | 2          | 4          | OCD                     |
| B2  | CPU8_i2o4_FI  | 8           | 100       | 2          | 4          | OCD-FI                  |
| B3  | CPU8_i4o8     | 8           | 100       | 4          | 8          | OCD                     |
| B4  | CPU16_i2o4    | 16          | 45        | 2          | 4          | OCD                     |
| B5  | CPU16_i2o4_FI | 16          | 45        | 2          | 4          | OCD-FI                  |
| B6  | CPU32_i2o8    | 32          | 25        | 2          | 8          | OCD                     |
| B7  | CPU32_i2o8_FI | 32          | 25        | 2          | 8          | OCD-FI                  |
| B8  | CPU32_i4o16   | 32          | 25        | 4          | 16         | OCD                     |
| N/A | MPC565        | 32          | 40        | 2          | 8          | OCD                     |

Os cenários que incluem uma infra-estrutura OCD-FI foram submetidos às mesmas dez campanhas de injeção de falhas utilizadas com a versão sem módulo FI. As campanhas dos restantes cenários são as mesmas utilizadas na secção anterior e consequentemente retornaram os mesmos resultados. A Tabela 6.7 apresenta os intervalos de tempo necessários para as principais operações de injeção de falhas.

Tabela 6.7 – Intervalos de tempo para as operações de inicialização e injeção de falhas (II)

| #  | Sistema Alvo  | Inicialização (Clk) | Inserção (Clk) |
|----|---------------|---------------------|----------------|
| B1 | CPU8_i2o4     | 10                  | 9              |
| B2 | CPU8_i2o4_FI  | 13                  | 3              |
| B3 | CPU8_i4o8     | 6                   | 6              |
| B4 | CPU16_i2o4    | 14                  | 13             |
| B5 | CPU16_i2o4_FI | 17                  | 3              |
| B6 | CPU32_i2o8    | 22                  | 21             |
| B7 | CPU32_i2o8_FI | 25                  | 3              |
| B8 | CPU32_i4o16   | 12                  | 12             |

A Tabela 6.8 apresenta os resultados obtidos com a execução das campanhas de injeção de falhas, mais uma vez classificados em termos da execução das experiências.

Tabela 6.8 – Resultados experimentais com infra-estrutura OCD e OCD-FI

| #  | Sistema Alvo  | ERR | INC | OK   |
|----|---------------|-----|-----|------|
| B1 | CPU8_i2o4     | 0%  | 3%  | 97%  |
| B2 | CPU8_i2o4_FI  |     | 0%  | 100% |
| B3 | CPU8_i4o8     |     | 2%  | 98%  |
| B4 | CPU16_i2o4    |     | 4%  | 96%  |
| B5 | CPU16_i2o4_FI |     | 0%  | 100% |
| B6 | CPU32_i2o8    |     | 5%  | 95%  |
| B7 | CPU32_i2o8_FI |     | 0%  | 100% |
| B8 | CPU32_i4o16   |     | 3%  | 97%  |

Pode-se constatar que, tendo sido apenas utilizadas interfaces com largura de banda adequada, não se verificaram resultados do tipo ERR. A Tabela 6.8 permite tirar as seguintes conclusões, relativas aos méritos relativos das infra-estruturas:

- A utilização de infra-estruturas OCD-FI permite eliminar a ocorrência de resultados do tipo INC, nos casos experimentais testados, devido à redução substancial no tempo de inserção de falha.
- Experiências adicionais provaram ser possível a ocorrência de resultados tipo INC, utilizando a infra-estrutura OCD-FI, quando a inserção da falha coincidissem com uma operação de escrita por parte do microprocessador. No entanto, este facto apresenta uma probabilidade reduzida e não ocorreu em nenhuma das experiências geradas aleatoriamente.
- A utilização da infra-estrutura OCD-FI não afecta a máxima frequência de relógio do sistema alvo, quando comparada com a infra-estrutura OCD normal. É possível

utilizar a mesma frequência de relógio para todos os sistemas baseados na mesma arquitectura.

Para efeitos de comparação, repetiram-se 5 campanhas, para um cenário baseado no microprocessador de 8 bits, utilizando a ligação paralela directa entre a infra-estrutura OCD e o depurador. Como explicado no capítulo 5, esta abordagem elimina a codificação/descodificação de acordo com a proposta NEXUS, sendo as mensagens transmitidas num único ciclo de relógio. A Tabela 6.9 apresenta uma comparação entre as principais características e resultados obtidos com estas experiências, relativamente aos cenários B1, B2 e B3. A coluna “interface” indica o número total de sinais presentes no porto de comunicação entre o depurador e a infra-estrutura OCD, sendo o cenário relativo à ligação paralela identificado por CPU8\_PAR.

Tabela 6.9 – Resultados experimentais com ligação paralela Depurador-OCD

| #  | Sistema Alvo | Interface (Total) | com Predet          |                |     | Sem Predet          |                |     |
|----|--------------|-------------------|---------------------|----------------|-----|---------------------|----------------|-----|
|    |              |                   | Inicialização (clk) | Inserção (clk) | INC | Inicialização (clk) | Inserção (clk) | INC |
| B1 | CPU8_i2o4    | 14                | 10                  | 9              | 3%  | 10                  | 17             | 5%  |
| B2 | CPU8_i2o4_FI | 14                | 13                  | 3              | 0%  | Não testado         |                |     |
| B3 | CPU8_i4o8    | 20                | 6                   | 6              | 2%  | 6                   | 11             | 3%  |
| C1 | CPU8_PAR     | 75                | 4                   | 4              | 0%  | 4                   | 8              | 0%  |

A Tabela 6.9 mostra que é possível eliminar a ocorrência de resultados INC, para as campanhas seleccionadas. Tal como a abordagem que utiliza a infra-estrutura OCD-FI, também é possível que estes resultados ocorram se as experiências forem personalizadas para esse objectivo, sendo a probabilidade de erros em experiências geradas aleatoriamente muito baixa (menos de 0,05% para as aplicações consideradas). Esta abordagem tem as desvantagens importantes de implicar a perda de conformidade com a norma e de requerer um elevado número de sinais de interface, os quais aumentam consideravelmente para os microprocessadores de 16 ou 32 bits. Esta solução é portanto indicada apenas nas situações em que estas desvantagens não sejam importantes, como sucede no caso de modelos de simulação ou protótipos dedicados.

#### 6.2.4 Sobrecarga Lógica

A Tabela 6.10, a Tabela 6.11 e a Tabela 6.12, apresentam a sobrecarga lógica dos componentes que constituem as infra-estruturas OCD e OCD-FI, em comparação com a

dimensão dos núcleos de microprocessador utilizados. Para simplificar a leitura, a contagem do *Bus Snooper* e *Bus Master* está incluída nos módulos RCT e RWA, respectivamente. Os valores apresentados foram obtidos a partir da ferramenta de síntese e representam a quantidade equivalente de portas lógicas.

Tabela 6.10 – Sobrecarga lógica (microprocessadores 8 bits)

| CPU        | CPU8_i1o1 |       |       | CPU8_i1o2 |       |       | CPU8_i2o4 |       |       | CPU8_i2o8 |       |       | CPU8_i4o8 |       |       |
|------------|-----------|-------|-------|-----------|-------|-------|-----------|-------|-------|-----------|-------|-------|-----------|-------|-------|
| Componente | # Gates   | % OCD | % TOT | # Gates   | % OCD | % TOT | # Gates   | % OCD | % TOT | # Gates   | % OCD | % TOT | # Gates   | % OCD | % TOT |
| Núcleo     | 9166      |       | 60,5  | 9166      |       | 59,2  | 9166      |       | 56,8  | 9166      |       | 55,1  | 9166      |       | 53,2  |
| RWA        | 369       | 6,2   | 2,4   | 369       | 5,8   | 2,4   | 369       | 5,3   | 2,3   | 369       | 4,9   | 2,2   | 369       | 4,6   | 2,1   |
| RCT        | 2391      | 39,9  | 15,8  | 2391      | 37,8  | 15,4  | 2391      | 34,2  | 14,8  | 2391      | 32,0  | 14,4  | 2391      | 29,6  | 13,9  |
| MQM        | 3225      | 53,9  | 21,3  | 3570      | 56,4  | 23,0  | 4225      | 60,5  | 26,2  | 4702      | 63,0  | 28,3  | 5313      | 65,8  | 30,8  |
| OCD        | 6185      | 100   | 39,5  | 6485      | 100   | 40,8  | 6985      | 100   | 43,2  | 7227      | 100   | 44,9  | 7562      | 100   | 46,8  |

Tabela 6.11 – Sobrecarga lógica (microprocessadores 16 e 32 bits)

| CPU        | CPU16_i2o4 |       |       | CPU16_i2o8 |       |       | CPU32_i2o4 |       |       | CPU32_i2o8 |       |       | CPU32_i4o16 |       |       |
|------------|------------|-------|-------|------------|-------|-------|------------|-------|-------|------------|-------|-------|-------------|-------|-------|
| Componente | # Gates    | % OCD | % TOT | # Gates    | % OCD | % TOT | # Gates    | % OCD | % TOT | # Gates    | % OCD | % TOT | # Gates     | % OCD | % TOT |
| Núcleo     | 20212      |       | 62,8  | 20212      |       | 60,7  | 53717      |       | 74,1  | 53717      |       | 72,9  | 53717       |       | 72,1  |
| RWA        | 516        | 4,3   | 1,6   | 516        | 3,9   | 1,5   | 643        | 3,4   | 0,9   | 643        | 3,2   | 0,9   | 643         | 3,1   | 0,9   |
| RCT        | 3730       | 31,2  | 11,6  | 3730       | 28,5  | 11,2  | 5113       | 27,2  | 7,1   | 5113       | 25,7  | 6,9   | 5113        | 24,6  | 6,9   |
| MQM        | 7715       | 64,5  | 24,0  | 8842       | 67,6  | 26,6  | 13045      | 69,4  | 18,0  | 14166      | 71,1  | 19,2  | 15033       | 72,3  | 20,2  |
| OCD        | 11961      | 100,0 | 37,2  | 13088      | 100,0 | 39,3  | 18801      | 100,0 | 25,9  | 19922      | 100,0 | 27,1  | 20789       | 100,0 | 27,9  |

A área lógica da infra-estrutura OCD é relativamente elevada, dado os núcleos de microprocessador serem relativamente simples, em arquitectura e em conjunto de instruções, sendo a área total principalmente dependente da dimensão dos barramentos e dos *buffers* internos.

É também possível verificar que a dimensão dos barramentos de interface com o exterior têm um efeito não desprezável na infra-estrutura OCD, sendo igualmente de considerar que o número de pinos necessários seria também superior.

Tabela 6.12 – Sobrecarga lógica (infra-estrutura OCD-FI)

| CPU        | CPU8_i2o4_FI |       |       | CPU16_i2o4_FI |       |       | CPU32_i2o8_FI |       |       |
|------------|--------------|-------|-------|---------------|-------|-------|---------------|-------|-------|
| Componente | # Gates      | % OCD | % TOT | # Gates       | % OCD | % TOT | # Gates       | % OCD | % TOT |
| Núcleo     | 9166         |       | 56,5  | 20212         |       | 62,7  | 53717         |       | 72,9  |
| RWA        | 369          | 5,2   | 2,3   | 516           | 4,3   | 1,6   | 643           | 3,2   | 0,9   |
| RCT        | 2391         | 33,9  | 14,7  | 3730          | 31,0  | 11,6  | 5113          | 25,6  | 6,9   |
| MQM        | 4225         | 59,8  | 26,0  | 7715          | 64,1  | 23,9  | 14166         | 70,8  | 19,2  |
| FI         | 75           | 1,1   | 0,5   | 75            | 0,6   | 0,2   | 75            | 0,4   | 0,1   |
| OCD        | 7060         | 100,0 | 43,5  | 12036         | 100,0 | 37,3  | 19997         | 100,0 | 27,1  |

Os resultados da síntese confirmam que a sobrecarga lógica imposta pelo módulo FI básico é mínima, sendo tanto menor quanto mais complexo for o sistema alvo.

Relativamente ao depurador, a Tabela 6.13 apresenta a área lógica das diferentes utilizadas em número equivalente de portas lógicas. Este cálculo exclui as memórias, pois estas representam componentes externos ao depurador, com dimensão independente deste, sendo limitada apenas pelos componentes (ou recursos de uma FPGA) disponíveis.

Tabela 6.13 – Área lógica dos depuradores utilizados

| Cenários | Designação  | MDI | MDO | Barramentos | Área Lógica |
|----------|-------------|-----|-----|-------------|-------------|
| A1       | DBG8_i1o1   | 1   | 1   | 8 bits      | 562         |
| A2       | DBG8_i2o1   | 1   | 2   | 8 bits      | 655         |
| A3,B1,B2 | DBG8_i4o2   | 2   | 4   | 8 bits      | 766         |
| A4       | DBG8_i8o2   | 2   | 8   | 8 bits      | 842         |
| A5,B3    | DBG8_i8o4   | 4   | 8   | 8 bits      | 936         |
| A6,B4    | DBG16_i4o2  | 2   | 4   | 16 bits     | 817         |
| A7,B5    | DBG16_i8o2  | 2   | 8   | 16 bits     | 910         |
| A8       | DBG32_i4o2  | 2   | 4   | 32 bits     | 1079        |
| A9,B6,B7 | DBG32_i8o2  | 2   | 8   | 32 bits     | 1150        |
| A10,B8   | DBG32_i16o4 | 4   | 16  | 32 bits     | 1402        |

É possível verificar que um depurador NEXUS elementar, necessita de relativamente poucos elementos lógicos, e permite a execução de campanhas de injeção de falhas, bem como o arquivo dos seus resultados, sendo o requisito mais complexo, a capacidade de memória. Mais uma vez, a utilização de barramentos maiores implica alguma complexidade adicional, aqui mais notória, devido ao peso da componente de comunicações (principalmente os *buffers*) na área lógica do depurador.

### 6.3 Utilização Operacional

Nesta secção analisa-se a aplicação de variantes da metodologia proposta, a uma situação experimental relativamente complexa. Considerou-se uma utilização operacional normal, em que se dispõe de um sistema alvo e de diferentes aplicações, para as quais se pretende verificar os efeitos de falhas em memória. Os objectivos propostos foram os seguintes:

- Executar uma análise de confiabilidade num sistema alvo, através de injeção de falhas, (1) com paragem do sistema e (2) em tempo real, e comparar os resultados obtidos com ambas as abordagens.
- Quantificar a influência de erros no processo de injeção de falhas nos resultados obtidos.
- Reavaliar a influência do desempenho da infra-estrutura e da sua interface, no processo de injeção de falhas. Comparar diferentes alternativas, em termos de largura de banda do porto de comunicações.
- Validar e testar a infra-estrutura OCD-FI com capacidade de operação autónoma, como uma solução viável para a injeção de falhas em tempo real e sem intrusividade.
- Quantificar a taxa máxima de falhas, do tipo inversão de bit, que podem ser injectadas utilizando cada variante da metodologia proposta.

### 6.3.1 Execução das Experiências

Esta subsecção apresenta as experiências realizadas utilizando apenas a versão mais completa do microprocessador de 32 bits, incluindo o suporte para interrupções e periféricos (via entradas/saídas). Foram utilizadas diferentes versões da infra-estrutura OCD e a versão mais completa da infra-estrutura OCD-FI, tendo sido considerados cenários de injeção de falhas com o sistema alvo suspenso e cenários onde não seria possível a pré-determinação do valor com falha.

Foram utilizadas três aplicações, nomeadamente versões ligeiramente modificadas das aplicações *MAdder* e *VSorter*, e a aplicação *XControl*, sendo desta vez utilizadas versões com e sem tolerância a falhas por *software*.

As campanhas de injeção de falhas foram definidas de forma semelhante às da secção anterior, com as seguintes diferenças:

- No caso da aplicação *XControl*, o fluxo de programa e o número de execuções de cada instrução depende dos dados a processar e dos estímulos exteriores. No entanto, o tipo de aplicação permite estimar o número de execuções de cada instrução, para um dado período de tempo. É assim possível escolher a execução da instrução alvo pretendida.
- A análise do próprio processo de injeção de falhas não é efectuada de origem, sendo portanto impossível detectar a ocorrência de resultados inconclusivos.

- O diagnóstico utiliza dados de monitorização de programa e de dados, e leitura do conteúdo de memória, após a execução da aplicação. Em caso de necessidade, é utilizada a leitura intermédia de valores de memória.

A opção pela não detecção de resultados inconclusivos visa replicar as condições experimentais que estariam presentes numa situação operacional, em que o objectivo fosse apenas a verificação da tolerância a falhas do sistema alvo. Desta forma, pode-se avaliar os efeitos desses resultados no diagnóstico. Sabendo que o processo pode dar origem a falhas diferentes das pretendidas, seria sempre possível incluir mecanismos para a detecção da sua ocorrência, tal como explicado no capítulo 4.

A Tabela 6.14 apresenta os cenários experimentais utilizados, incluindo uma designação simplificada, que será utilizada nas tabelas subsequentes. O sinal “+” indica a não utilização da pré-determinação (i.e. um cenário “mais” complexo) e pressupõe uma solução mais flexível, mas eventualmente com pior desempenho.

Apenas é utilizado um microprocessador alvo, com duas variantes de infra-estrutura OCD, as quais representam uma abordagem padrão (BRT – *Basic Real Time*), considerando exclusivamente os requisitos de depuração de um microprocessador de 32 bits, e uma abordagem melhorada (ERT – *Enhanced Real Time*), para uma injeção de falhas mais eficiente. A configuração utilizada para esta abordagem baseou-se nos resultados anteriores, tendo-se considerado que, com um maior equilíbrio entre barramentos de entrada e saída (i808), seria possível um desempenho semelhante ao da opção anterior (i4016), para um microprocessador de 32 bits, com sobrecarga lógica semelhante.

Tabela 6.14 – Cenários experimentais para microprocessadores de 32 bits

| #  | Cenário | Sistema Alvo  | MDI | MDO | Injecção de Falhas | Predet |
|----|---------|---------------|-----|-----|--------------------|--------|
| 1  | BOF     | CPU32_i2o8    | 2   | 8   | Offline            | x      |
| 2  | BOF+    | CPU32_i2o8    | 2   | 8   | Offline            |        |
| 3  | EOF     | CPU32_i8o8    | 8   | 8   | Offline            | x      |
| 4  | EOF+    | CPU32_i8o8    | 8   | 8   | Offline            |        |
| 5  | BRT     | CPU32_i2o8    | 2   | 8   | Tempo Real         | x      |
| 6  | BRT+    | CPU32_i2o8    | 2   | 8   | Tempo Real         |        |
| 7  | ERT     | CPU32_i8o8    | 8   | 8   | Tempo Real         | x      |
| 8  | ERT+    | CPU32_i8o8    | 8   | 8   | Tempo Real         |        |
| 9  | OCD-FI  | CPU32_i2o8_FI | 2   | 8   | Tempo Real         | x      |
| 10 | OCD-FI+ | CPU32_i2o8_FI | 2   | 8   | Tempo Real         |        |

Os dois cenários que utilizam a infra-estrutura OCD-FI implementam a versão mais completa (sem extensões), sendo as funcionalidades utilizadas dependentes do cenário específico. Para ambos os casos é utilizada uma interface análoga à versão mais simples da infra-estrutura OCD.

Nos cenários utilizados, varia a metodologia de injeção de falhas, no que respeita à suspensão do sistema alvo e à pré-determinação do vector com falha. Estas variações visam avaliar as limitações das várias propostas, em diferentes condições experimentais. A injeção de falhas com o microprocessador alvo suspenso é descrita como *offline*, sendo aplicada a designação de tempo real, caso seja efectuada com o microprocessador em operação normal. “Predet” indica a utilização de pré-determinação do vector com falha. Quando utilizada, esta capacidade baseia-se no conhecimento do código executado e dos dados utilizados, de forma a prever o valor do vector de dados presente no elemento alvo, no instante da injeção da falha, tal como explicado no capítulo 4. Desta forma é possível omitir a sua leitura, com o consequente ganho em desempenho. A Tabela 6.15 apresenta os atrasos de cada operação, de acordo com o cenário escolhido.

Tabela 6.15 – Intervalos de tempo para as operações de inicialização e injeção de falhas (III)

| #  | Cenário        | Inicialização (Clk) | Inserção (Clk) |
|----|----------------|---------------------|----------------|
| 1  | <b>BOF</b>     | 22                  | 22             |
| 2  | <b>BOF+</b>    | 22                  | 31             |
| 3  | <b>EOF</b>     | 6                   | 10             |
| 4  | <b>EOF+</b>    | 6                   | 19             |
| 5  | <b>BRT</b>     | 22                  | 21             |
| 6  | <b>BRT+</b>    | 22                  | 30             |
| 7  | <b>ERT</b>     | 6                   | 9              |
| 8  | <b>ERT+</b>    | 6                   | 20             |
| 9  | <b>OCD-FI</b>  | 57                  | 3              |
| 10 | <b>OCD-FI+</b> | 57                  | 5              |

Devido às limitações de memória do depurador, cada campanha compreendeu 10 experiências, cada uma delas injectando uma única falha, do tipo inversão de bit, emulando um SEU. As campanhas executadas para cada cenário são indicadas na Tabela 6.16.

Tabela 6.16 – Campanhas executadas por Aplicação (II)

| Aplicação                | Campanhas por Cenário |
|--------------------------|-----------------------|
| <i>MAdder32_v1.4</i>     | 10                    |
| <i>MAdder32_v1.4FT</i>   | 10                    |
| <i>VSorter32_v1.3</i>    | 10                    |
| <i>VSorter32_v1.3FT</i>  | 10                    |
| <i>XControl32_v1.2</i>   | 10                    |
| <i>XControl32_v1.3FT</i> | 10                    |

Foram assim executadas um total de 60 campanhas, para cada cenário, utilizando as três aplicações alvo, na sua versão normal e tolerante a falhas, num total de 6.000 experiências. Cada campanha exigiu 2 Kbytes de memória de entrada e 256 Kbytes de memória de saída no depurador.

### 6.3.2 Resultados Experimentais

A Tabela 6.17 apresenta os resultados obtidos com a execução das campanhas referidas, classificados por cenário e aplicação alvo. É importante notar que todos os cenários que utilizam injeção de falhas com suspensão do sistema alvo, retornaram exactamente os mesmos resultados, tal como esperado. Assim, e para simplificar, os resultados obtidos desta forma são agrupados numa única linha da tabela, designada como OFF (de *Offline*).

O diagnóstico de falhas foi classificado nas seguintes categorias:

- $U_{ERR}$  (*Undetected Error*) – Erro não detectado pela rotina de tolerância a falhas (se esta for utilizada).
- $D_{ERR}$  (*Detected Error*) – Erro detectado pela rotina de tolerância a falhas. A aplicação terminou antes do esperado, com um sinal de detecção de erros.
- $N_{ERR}$  (*No Error*) – Sem erros detectados no resultado ou durante a execução. Este diagnóstico engloba os erros eliminados pela própria aplicação (escritos por cima) e outros ainda presentes em memória, no final da execução, não sendo distinguidas estas duas situações.

O diagnóstico de falhas foi efectuado após a execução de cada campanha, a partir dos dados presentes na memória de saída do depurador, referentes à monitorização e às leituras do conteúdo da memória. Os valores lidos foram comparados com os valores esperados, de forma a identificar a ocorrência e detecção de erros.

Tabela 6.17 – Resultados da Injecção de Falhas no Microprocessador de 32 bits (em %)

| Cenário | MAdder           |                  |                  |                  |                  | VSorter          |                  |                  |                  |                  | XControl            |                  |                  |                  |                  |
|---------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|---------------------|------------------|------------------|------------------|------------------|
|         | sem FT           |                  | SW-FT            |                  |                  | sem FT           |                  | SW-FT            |                  |                  | sem FT              |                  | SW-FT            |                  |                  |
|         | U <sub>ERR</sub> | N <sub>ERR</sub> | D <sub>ERR</sub> | U <sub>ERR</sub> | N <sub>ERR</sub> | U <sub>ERR</sub> | N <sub>ERR</sub> | D <sub>ERR</sub> | U <sub>ERR</sub> | N <sub>ERR</sub> | U <sub>ERR</sub>    | N <sub>ERR</sub> | D <sub>ERR</sub> | U <sub>ERR</sub> | N <sub>ERR</sub> |
| OFF     | 19               | 81               | 28               | 13,9             | 58,1             | 98               | 2                | 97               | 2                | 1                | Execução Impossível |                  |                  |                  |                  |
| BRT     | 19,4             | 80,6             | 28,3             | 13,8             | 57,9             | 98,1             | 1,9              | 96,8             | 2                | 1,2              |                     |                  |                  |                  |                  |
| ERT     | 19,2             | 80,8             | 28,1             | 13,9             | 58               | 98               | 2                | 96,9             | 2                | 1,1              |                     |                  |                  |                  |                  |
| OCD-FI  | 19               | 81               | 28               | 13,9             | 58,1             | 98               | 2                | 97               | 2                | 1                |                     |                  |                  |                  |                  |
| BRT+    | 19,5             | 80,5             | 28,4             | 13,8             | 57,8             | 98,2             | 1,8              | 96,7             | 1,9              | 1,4              | 29,3                | 70,7             | 29,1             | 1,5              | 69,4             |
| ERT+    | 19,3             | 80,7             | 28,2             | 13,8             | 58               | 98,1             | 1,9              | 96,8             | 1,9              | 1,3              | 29,6                | 70,4             | 28,9             | 1,2              | 69,9             |
| OCD-FI+ | 19,1             | 80,9             | 28,1             | 13,9             | 58               | 98               | 2                | 96,9             | 1,9              | 1,2              | 29,8                | 70,2             | 28,8             | 1,1              | 70,1             |

A execução das campanhas e os resultados obtidos permitem as seguintes conclusões, relativas à controlabilidade e observabilidade da metodologia proposta:

- Todas as configurações permitem controlo preciso da célula de memória para inserção da falha e respectivo bit.
- O instante em que a falha é inserida depende do atraso entre o momento em que se verifica a condição de activação e a efectiva inserção da falha em memória. Como este atraso é constante para cada configuração e pode ser precisamente determinado, é possível inserir falhas em instantes precisos.
- Todas as experiências são replicáveis, podendo ser repetidas em cenários similares (i.e. utilizando a mesma aplicação alvo e dimensão de dados), nas mesmas condições e quantas vezes for necessário.
- O ambiente e as operações de injeção de falhas não produzem nenhum efeito sobre a execução do sistema alvo, antes da inserção da falha em memória.
- É possível utilizar informação de monitorização, gerada pela infra-estrutura OCD, para reconstituir o fluxo de programa. O diagnóstico dos efeitos das falhas pode ser executado desta forma, com eventual apoio de leituras de memória.
- A infra-estrutura OCD pode ser utilizada para análise dos resultados da aplicação alvo, posteriormente à sua execução, permitindo a automatização desta tarefa, nas campanhas de injeção de falhas.

Em geral, a metodologia proposta permite um elevado grau de controlabilidade da injeção de falhas, e observabilidade adequada para o diagnóstico dos efeitos das falhas, mesmo operando em tempo real.

Apesar de não ser um objectivo principal, a análise dos efeitos das falhas permite algumas conclusões interessantes. É inevitável que a maioria dos resultados apresentados estejam relacionados com o sistema alvo específico e respectivas

aplicações, podendo ser bastante diferentes noutros casos. Mesmo assim, os resultados disponíveis permitem analisar as capacidades e limitações da técnica de tolerância a falhas utilizada. São possíveis as seguintes conclusões, relativas ao mecanismo de tolerância a falhas por *software* e à opção pela injeção de falhas em tempo real:

- Os resultados do diagnóstico de falhas são idênticos para todos os cenários que utilizam injeção de falhas com o microprocessador alvo suspenso.
- Ao utilizar injeção de falhas em tempo real, aparecem algumas diferenças nos resultados do diagnóstico, originadas pelas variações impostas pelo método de injeção.
- Os efeitos das falhas injectadas são consideravelmente influenciados pela aplicação alvo, com erros não detectados a serem muito mais frequentes para a aplicação *VSorter*, provavelmente devido ao uso mais intensivo da memória disponível.
- As versões sem tolerância a falhas das aplicações *MAdder* e *XControl* apresentam alguma tolerância a falhas intrínseca, a qual é mínima para a aplicação *VSorter*. Este tipo de tolerância a falhas advém da existência de áreas de memória com utilização reduzida, o que diminui a probabilidade de activação de falhas injectadas.
- A utilização de tolerância a falhas por *software* reduz substancialmente a ocorrência de erros não detectados, principalmente para as aplicações *VSorter* (redução de 98%) e *XControl* (redução de 96%). A aplicação *MAdder* apresenta uma redução bastante menos notória (redução de 28%), devido à menor taxa de refrescamento dos resultados.
- A percentagem de resultados correctos diminui com a utilização de tolerância a falhas por *software*, devido à maior área de memória necessária para implementar a redundância e consequente maior vulnerabilidade a falhas.

Em conclusão, o mecanismo de tolerância a falhas por *software* fornece capacidades adequadas de detecção de erros, mas reduz a percentagem de serviço correcto, se utilizado individualmente. A sua eficiência beneficiaria da utilização adicional de capacidades de remoção de falhas, possivelmente na forma de reinicialização da aplicação, aquando da detecção de erros. Desta forma, a aplicação poderia ser executada repetidamente, até que nenhum erro fosse detectado, o que melhoraria consideravelmente as taxas de serviço correcto.

### 6.3.3 Análise de Resultados Inconclusivos

Para melhor avaliar as discrepâncias entre os cenários que utilizam injeção de falhas em tempo real, foram efectuadas experiências adicionais. Especificamente, repetiram-se aquelas em que diferentes metodologias retornaram um diagnóstico de falhas diferente, tendo-se utilizado opções de depuração adicionais, para uma análise mais completa. Estas operações consistem na monitorização de dados e, se necessário, em pontos de paragem imediatamente após a inserção da falha. Apesar desta abordagem exigir considerável tempo de execução e análise, o que a tornaria menos interessante para diagnóstico dos efeitos das falhas, permite uma avaliação mais detalhada da metodologia de injeção usada. As inserções de falha erróneas foram novamente classificadas como resultado inconclusivo (INC) e representam os casos em que o processo de injeção em tempo real foi prejudicado pela escrita de um vector de dados na célula de memória alvo, por parte do microprocessador, antes da conclusão do procedimento de inserção. A Tabela 6.18 apresenta a percentagem de ocorrência de resultados inconclusivos para cada cenário.

Tabela 6.18 – Ocorrência de resultados inconclusivos (em %)

| Cenário        | sem FT        |                |                 | FT por software |                |                 |
|----------------|---------------|----------------|-----------------|-----------------|----------------|-----------------|
|                | <i>MAdder</i> | <i>VSorter</i> | <i>XControl</i> | <i>MAdder</i>   | <i>VSorter</i> | <i>XControl</i> |
| <b>OFF</b>     | 0             |                |                 | 0               |                |                 |
| <b>BRT</b>     | 3,1           | 0,9            | Impossível      | 4               | 2,2            | Impossível      |
| <b>ERT</b>     | 1,4           | 0,6            |                 | 2,3             | 1,1            |                 |
| <b>OCD-FI</b>  | 0,2           | 0,1            |                 | 0,2             | 0,2            |                 |
| <b>BRT+</b>    | 3             | 1,2            | 2,1             | 4,8             | 2,8            | 3,2             |
| <b>ERT+</b>    | 2             | 0,8            | 1,5             | 3,7             | 2,1            | 2,4             |
| <b>OCD-FI+</b> | 0,4           | 0,2            | 0,3             | 1,7             | 1,2            | 1,3             |

Uma análise mais detalhada dos resultados acima apresentados, permite uma melhor compreensão das limitações das metodologias de injeção de falhas em tempo real propostas, nomeadamente:

- As configurações em que as falhas são inseridas com o sistema alvo suspenso apresentam resultados mais fiáveis.
- Uma análise pormenorizada das ocorrências dos resultados INC demonstrou que estes representam principalmente a injeção de múltiplas inversões de bit numa única célula de memória.
- O efeito anterior depende do atraso entre a activação da falha e a sua inserção e, como tal, varia entre cenários e configurações. A utilização da infra-estrutura

OCD-FI e a pré-determinação do vector com falha reduzem consideravelmente as ocorrências de resultados INC, particularmente se utilizadas em conjunto.

Para melhor comparação dos efeitos da ocorrência de resultados INC, a Figura 6.2 e a Figura 6.3 apresentam graficamente a sua ocorrência, para cada 1.000 experiências, classificadas por configuração e aplicação.

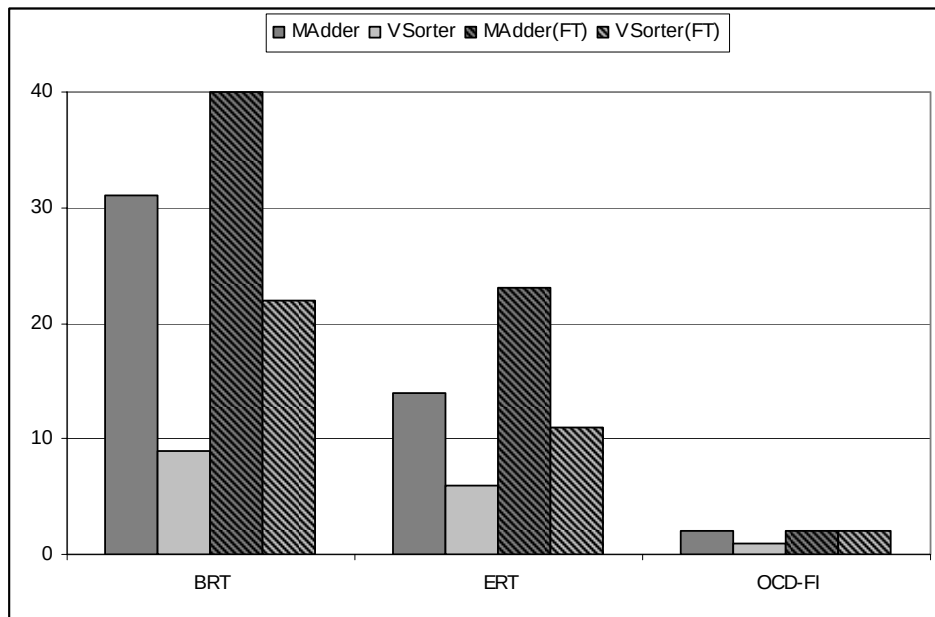


Figura 6.2 – Ocorrência de resultados do tipo INC por metodologia (com Predet.)

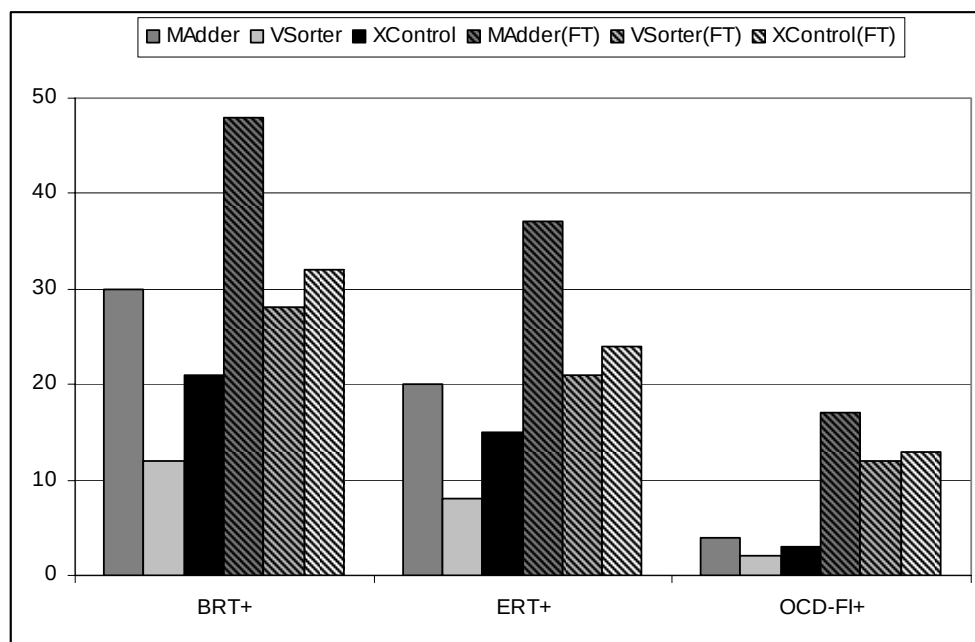


Figura 6.3 – Ocorrência de resultados do tipo INC por metodologia (sem Predet.)

Os resultados obtidos confirmaram que a metodologia proposta é uma alternativa eficiente para a injeção de falhas em memória em tempo real, ou com o sistema alvo suspenso. Neste caso específico, a escolha da configuração adequada depende das características e da arquitectura do sistema e também dos requisitos de confiabilidade. A injeção de falhas com o sistema alvo suspenso é preferível para cenários mais simples (e.g. *MAdder*), com as capacidades de tempo real a serem necessárias em cenários onde o desempenho seja particularmente importante, ou tenha de se incluir uma interface com o exterior no procedimento de injeção de falhas (e.g. *XControl*).

Adicionalmente, dependendo da aplicação alvo e requisitos de precisão do diagnóstico de falhas, deverá ser tomado algum cuidado na utilização de injeção de falhas em tempo real. Em geral, a infra-estrutura OCD-FI apresenta um desempenho consideravelmente melhor e a pré-determinação do valor com falha deverá também ser utilizada, sempre que possível.

### 6.3.4 Falhas Múltiplas

É também importante analisar a metodologia proposta em termos de número de falhas que podem ser injectadas em série. Uma taxa elevada permite simular de forma adequada situações em que os efeitos de SEUs são mais concentrados no tempo. A Tabela 6.19 apresenta as taxas máximas de falhas por segundo, que podem ser atingidas utilizando as configurações indicadas.

Tabela 6.19 – Taxas máximas de falhas por segundo

| Cenário | Tempo Real | Acesso Offline |
|---------|------------|----------------|
| BOF+    | Impossível | 400k           |
| EOF+    |            | 1150k          |
| BRT+    | 454k       | 400k           |
| ERT+    | 1250k      | 1150k          |
| OCD-FI+ | 491k       | 483k           |

Apenas foram consideradas as configurações onde não é necessário conhecimento prévio do conteúdo da célula de memória alvo, pois após a injeção da primeira falha, torna-se inviável a previsão do subsequente fluxo do programa. Os valores indicados assumem que cada falha é injectada individualmente (não são consideradas inversões múltiplas de bit na mesma célula de memória) e a frequência de relógio é fixada em 30 MHz (valor típico), sendo idêntica para todas as configurações. É importante notar que a infra-estrutura OCD partilha o

sinal de relógio com o microprocessador, portanto implementações que operem a frequências mais elevadas (e.g. ASICs ou FPGAs mais rápidas), permitiriam taxas mais elevadas.

Os resultados obtidos confirmam um ligeiro aumento do desempenho, derivado da injeção em tempo real. Este factor só por si não poderá ser considerado decisivo, sendo apenas relevante em situações em que o tempo de execução seja de primordial importância.

As sobrecargas lógicas e temporais impostas pelas diferentes configurações são analisadas no final da secção seguinte, de modo a incluir a comparação com as extensões à infra-estrutura OCD-FI.

## **6.4 OCD-FI com Extensões**

Esta secção divide as experiências em dois cenários distintos, consoante a extensão utilizada. Em ambos os casos recorreu-se ao mesmo sistema alvo considerado na secção anterior, com as modificações necessárias a cada extensão específica.

### **6.4.1 Injeção em mecanismos de detecção e correcção de erros**

O sistema alvo utilizado neste cenário inclui um mecanismo de EDAC entre o microprocessador e a memória. A extensão OCD-FI (EDAC) foi implementada através de um conjunto de modificações à própria infra-estrutura e à sua interface com o microprocessador e memória. O mecanismo de EDAC propriamente dito também requer lógica adicional e elementos de memória, diminuindo ligeiramente o desempenho máximo do microprocessador alvo.

As experiências de injeção de falhas foram geradas de forma análoga aos cenários apresentados na secção anterior, com a diferença de ser necessário de considerar os bits de controlo adicionais, na geração do vector de dados com falhas. Foram reutilizadas as mesmas aplicações da secção anterior e adaptadas ao novo cenário dez das campanhas utilizadas. A adaptação consistiu na inclusão dos dados adicionais, requeridos pelo sistema EDAC, sendo inalterados os alvos e instantes de inserção de cada falha.

A Tabela 6.20 apresenta os resultados obtidos utilizando a extensão OCD-FI (EDAC), considerando apenas as aplicações sem tolerância a falhas por *software*. Os resultados

classificados como  $D_{ERR}$  indicam a ocorrência de erros detectados e corrigidos pelo mecanismo de EDAC.

Tabela 6.20 – Resultados da injeção de falhas num sistema com EDAC (em %)

|                 | sem Predet |      |      |     | com Predet |      |      |     |
|-----------------|------------|------|------|-----|------------|------|------|-----|
|                 | Derr       | Uerr | Nerr | INC | Derr       | Uerr | Nerr | INC |
| <i>MAdder</i>   | 40         | 0    | 58   | 2   | 40         | 0    | 59   | 1   |
| <i>VSorter</i>  | 98         | 0    | 1    | 1   | 99         | 0    | 1    | 0   |
| <i>XControl</i> | 30         | 0    | 69   | 1   | 30         | 0    | 69   | 1   |

A execução das campanhas de injeção de falhas utilizando a extensão EDAC permitem as seguintes conclusões:

- A extensão OCD-FI (EDAC) pode ser utilizada para a injeção automática de falhas, em memória protegida por mecanismos de tolerância a falhas por *hardware* (compatíveis).
- A utilização de um mecanismo de injeção de falhas do tipo EDAC elimina os efeitos de falhas do tipo inversão simples de bit no sistema alvo, pois todos os erros provocados são detectados e corrigidos.
- A extensão OCD-FI (EDAC) permite a injeção de falhas em mecanismos do tipo EDAC, de forma não intrusiva e em tempo real.
- Experiências adicionais permitem verificar que a metodologia proposta possibilita a injeção de falhas do tipo inversão múltipla de bit, sem perda de desempenho e com efeitos evidentes no sistema alvo, pois o mecanismo EDAC não consegue corrigir erros múltiplos na mesma célula de memória.
- A ocorrência de resultados inconclusivos (INC) tem a mesma origem dos casos encontrados em experiências anteriores.

A utilização de mecanismos de tolerância a falhas por *hardware*, como o EDAC, encontra-se em crescimento e deve ser adequadamente testada e validada. A capacidade de injectar falhas directamente em memória, sem desactivar as referidas protecções, é fundamental, e a boa controlabilidade é uma característica importante. A metodologia proposta permite o acesso à memória através do mecanismo de EDAC, ou contornando-o, o que é muito útil para a injeção de falhas e também para o teste e a depuração.

### 6.4.2 Injecção em registos em tempo real

Apesar das implementações de infra-estruturas OCD actuais não permitirem a injecção de falhas nos registos internos do microprocessador, em tempo real, é possível minimizar o intervalo de tempo durante o qual este é suspenso. A Tabela 6.21 apresenta os intervalos durante os quais o microprocessador tem de estar suspenso, para que seja injectada uma falha num registo interno, relativamente aos cenários apresentados na secção anterior.

Tabela 6.21 – Tempo de suspensão para a injecção de falhas em registos

| Cenário | Tempo de Paragem (Clk) |
|---------|------------------------|
| BOF     | 65                     |
| BOF+    | 75                     |
| EOF     | 35                     |
| EOF+    | 42                     |
| BRT     | 41                     |
| BRT+    | 60                     |
| ERT     | 12                     |
| ERT+    | 22                     |
| OCD-FI  | 4                      |
| OCD-FI+ | 6                      |

A inicialização da infra-estrutura é efectuada da mesma forma que para a injecção de falhas em memória e implica o mesmo intervalo de tempo. Os pequenos atrasos no acesso a registos internos do microprocessador permitem um bom desempenho do processo de injecção de falhas e, em certas condições, pode permitir a injecção de falhas com o sistema alvo suspenso.

Para injecção de falhas em tempo real, em registos internos do microprocessador, foi implementada a extensão OCD-FI (RTREG). Foram elaboradas 100 experiências de injecção de falhas personalizadas para esta infra-estrutura específica, utilizando as aplicações *MAdder* e *VSorter*, nas suas versões com e sem tolerância a falhas (25 experiências com cada). Todas as falhas utilizam como alvo o registo ACC (*accumulator*), já que este permite a previsão do seu conteúdo com relativa facilidade, a partir do conhecimento do programa executado, e representa o caso testável potencialmente mais grave. A injecção de falhas no registo PC (*Program Counter*) não é exequível utilizando a extensão OCD-FI (RTREG), por este registo ser modificado em praticamente todos os ciclos de relógio. Todas as experiências foram elaboradas manualmente, a partir de instantes de injecção gerados aleatoriamente, de forma a assegurar uma sincronização adequada. Para cada experiência, é necessário verificar se a instrução seleccionada para activação da falha pode ser utilizada e se permite um

intervalo de tempo adequado para a sua inserção. A Tabela 6.22 apresenta os resultados obtidos a partir da execução das experiências referidas.

Tabela 6.22 – Resultados da injeção de falhas com extensão OCD-FI (RTREG) (em %)

|                | sem FT |      |     | FT por software |      |      |     |
|----------------|--------|------|-----|-----------------|------|------|-----|
|                | Uerr   | Nerr | INC | Derr            | Uerr | Nerr | INC |
| <i>MAdder</i>  | 45     | 6    | 49  | 31              | 11   | 8    | 50  |
| <i>VSorter</i> | 29     | 18   | 53  | 20              | 7    | 15   | 58  |

As conclusões que se podem tirar deste cenário são as seguintes:

- A injeção de falhas nos registos internos, em tempo real, exige a programação manual da activação, de forma a garantir que as falhas são inseridas nos instantes em que o sistema alvo não está a tentar escrever nesse mesmo registo. As instruções que podem ser utilizadas dependem do microprocessador, da aplicação em execução e do registo alvo.
- A escolha das instruções que podem ser utilizadas para activação exige o conhecimento preciso do código em execução e dos atrasos inerentes ao processo de injeção de falhas. No caso específico do registo acumulador, e para as aplicações utilizadas, uma média de 45% do código executado permite a activação<sup>23</sup>.
- Os resultados indicam que as falhas inseridas no acumulador provocam efeitos muito diferentes das falhas inseridas em memória. Por exemplo, a percentagem de erros não detectados é maior para a aplicação *MAdder* e menor para a operação *VSorter*.
- Similarmente, a eficiência da tolerância a falhas por *software* é também diferente e dependente da aplicação alvo.

A utilização da extensão RTREG demonstra que a injeção de falhas nos registos internos de um microprocessador é um problema complexo mas também importante. Os registos são muito sensíveis a erros e, em sistemas críticos, pode ser necessário utilizar componentes específicos, para assegurar a tolerância a falhas, testar a sua implementação e subsequente sensibilidade a falhas. Em alguns sistemas, a adição de suporte para a injeção de falhas em tempo real, nos registos internos do

<sup>23</sup> Este valor foi calculado a partir da análise detalhada do código das aplicações.

microprocessador, pode justificar a intrusividade adicional e a eventual degradação de desempenho.

A sobrecarga lógica imposta por cada elemento e a frequência de operação máxima para cada sistema são apresentadas na Tabela 6.23. Ambos os valores foram obtidos utilizando o ISE 7.1 da Xilinx, considerando uma FPGA Virtex-2 como alvo e utilizando a configuração padrão, visando obter um equilíbrio entre área e desempenho. A referência considerada é a configuração base, ou seja, o microprocessador completo e a infra-estrutura OCD dimensionada apenas para depuração, visto esta representar uma situação COTS típica.

Tabela 6.23 – Sobrecarga Lógica e efeitos no desempenho dinâmico

| Núcleo CPU | OCD | OCD-FI      | EDAC | RTREG | Área Lógica | Sobrecarga | f Max |
|------------|-----|-------------|------|-------|-------------|------------|-------|
|            |     |             |      |       |             | %          | MHz   |
| x          |     |             |      |       | 53926       | 75,4%      | 37    |
| x          |     |             | x    |       | 55018       | 76,9%      | 32    |
| x          | BRT |             |      |       | 71527       | 100,0%     | 36    |
| x          | BRT |             | x    |       | 72619       | 101,5%     | 32    |
| x          | ERT |             |      |       | 76127       | 106,4%     | 36    |
| x          |     | x           |      |       | 71842       | 100,4%     | 36    |
| x          |     | +EDAC       | x    |       | 73184       | 102,3%     | 32    |
| x          |     | +RTREG      |      | x     | 76392       | 106,8%     | 27    |
| x          |     | +EDAC/RTREG | x    | x     | 77484       | 108,3%     | 25    |

A sobrecarga lógica provocada pela infra-estrutura OCD é algo elevada, devido ao sistema alvo e à arquitectura utilizados. O microprocessador baseia-se numa arquitectura RISC relativamente simples, e não inclui *cache*, nem unidade de vírgula flutuante, ou outros elementos opcionais, que poderiam aumentar consideravelmente a área lógica necessária. A infra-estrutura OCD foi desenvolvida para ser genérica, pelo que a sua dimensão não deverá aumentar muito em sistemas mais complexos, desde que os barramentos sejam semelhantes. Em termos comparativos, a área lógica necessária, para as configurações com maior largura de banda, pode ser algo elevada, com a solução baseada numa infra-estrutura OCD-FI a apresentar uma sobrecarga muito menor. Desta forma, esta última solução é preferível, se se considerarem apenas os requisitos de injeção de falhas, uma vez que a maior largura de banda é também um requisito para uma depuração mais rápida, particularmente se for necessária uma monitorização mais exhaustiva.

Como seria previsível, a inclusão de um mecanismo de EDAC causa um pequeno aumento na área lógica do microprocessador e reduz ligeiramente a frequência máxima de operação. A implementação das infra-estruturas OCD-FI (EDAC) e OCD-FI (RTREG) implica uma sobrecarga adicional em ambas as áreas, mesmo quando não utilizadas para a injeção de

falhas, sendo esta sobrecarga bastante maior para a OCD-FI (RTREG). No entanto, convém lembrar que as estas infra-estruturas foram projectadas para utilização em sistemas críticos e, neste tipo de aplicações, as sobrecargas consideradas aceitáveis são normalmente maiores.

## 6.5 *Resumo e Conclusões*

Este capítulo apresentou os resultados obtidos experimentalmente com a execução de diversas campanhas de injeção de falhas em cenários projectados para avaliar as soluções propostas. A análise focalizou-se na validação da metodologia de injeção de falhas e das suas variantes, sendo com ênfase na sua utilização operacional. Como qualquer trabalho desta natureza, é impossível fazer uma análise exaustiva, permitindo os resultados obtidos tirar diversas conclusões, tanto quantitativas como qualitativas. A avaliação mais detalhada foi efectuada ao longo do capítulo, sendo importante reter os seguintes pontos fundamentais:

- A injeção de falhas em tempo real apresenta algumas dificuldades específicas e os problemas de desempenho podem revelar-se de particular importância.
- A utilização de infra-estruturas com melhor desempenho permite resultados de injeção de falhas mais satisfatórios, lidando eficientemente com as limitações presentes nas versões dimensionadas exclusivamente para depuração.
- A utilização da infra-estrutura OCD-FI apresenta benefícios óbvios para o processo de injeção de falhas, permitindo a sua execução em tempo real, sem perda evidente de representatividade, reduzindo drasticamente os problemas identificados.
- Os resultados obtidos com a utilização de extensões desenvolvidas para endereçar problemas de confiabilidade específicos, permitem verificar que esta será uma alternativa possível este fim. No entanto, os efeitos no desempenho do sistema alvo não são desprezáveis, tendo de ser considerados sempre que seja planeada uma solução deste género.

Para o modelo de falhas e requisitos de tempo real considerados, as metodologias de injeção de falhas alternativas mais frequentemente utilizadas baseiam-se em *software* e no uso de métodos sem contacto, como radiação ou laser. Para o sistema alvo específico utilizado (i.e. disponível em modelo VHDL), também seriam possíveis técnicas baseadas apenas em

simulação. Uma comparação da metodologia proposta com as várias abordagens alternativas permite as seguintes conclusões:

- A metodologia proposta permite grande flexibilidade em termos de sistema alvo, pois pode ser utilizada em simulação, num dispositivo programável (i.e. FPGA) ou num circuito integrado (i.e. ASIC).
- Relativamente à injeção de falhas utilizando técnicas físicas sem contacto, a metodologia proposta apresenta vantagens significativas, em termos de controlabilidade e replicabilidade das experiências.
- As técnicas baseadas em *software* são geralmente mais intrusivas, introduzem atrasos semelhantes e apresentam limitações mais consideráveis, em termos de cobertura de falhas.
- A necessidade de lidar com diagnóstico de falhas errados ou impossíveis é um problema comum a todas as técnicas de injeção de falhas, particularmente se efectuadas em tempo real. Como noutras abordagens, os problemas podem ser minimizados, utilizando técnicas estatísticas ou operações de diagnóstico adicionais, aplicadas quando necessário (e possível).

Se comparada com técnicas de injeção de falhas em tempo real, baseadas na utilização de infra-estruturas NEXUS [Yuste03a][Vinter05], a principal vantagem da metodologia proposta consiste no melhor desempenho e na subsequente menor ocorrência de experiências erróneas. As campanhas de injeção de falhas não poderiam ser executadas num ambiente baseado no microprocessador MPC565, devido ao reduzido tempo de execução das aplicações alvo utilizadas, quando comparadas com os atrasos impostos pela injeção de falhas, no referido sistema. De facto, em certos casos (i.e. pequena quantidade de dados) a aplicação é executada em menos tempo do que seria necessário para o processo normal de injeção de falhas num MPC565, utilizando um depurador comercial.



*“The most exciting phrase to hear in science, the one that heralds the most discoveries, is not “Eureka!” but “That’s funny...”*

Isaac Asimov

**Capítulo 1**

**Capítulo 2**

**Capítulo 3**

**Capítulo 4**

**Capítulo 5**

**Capítulo 6**

**Capítulo 7**

**Referências**

**Anexo**

**Conclusão**



## 7 Conclusão

Este capítulo faz uma apreciação crítica do trabalho descrito ao longo do documento. São discutidas as condicionantes que foram encontradas, os méritos e deméritos da proposta apresentada e uma análise qualitativa dos resultados obtidos. São mencionados os principais contributos científicos e sugeridas linhas de investigação futura.

### 7.1 *Apreciação Crítica*

Como base do trabalho apresentado, foi realizada uma análise detalhada do seu enquadramento nos capítulos iniciais. Constatou-se que a confiabilidade de sistemas computacionais é um problema antigo com muitas vertentes, condicionantes e abordagens. O interesse por este domínio tem crescido e o campo da tolerância a falhas, em particular, tem sido alvo de consideráveis evoluções, de forma a lidar com o progresso tecnológico. Da mesma forma, as técnicas utilizadas para verificar e validar a sua implementação têm igualmente evoluído.

A injeção de falhas é hoje em dia muito utilizada para apoiar esta tarefa, sendo crescente a sua implantação em sistemas críticos, e com tendência para se alargar a áreas mais diversificadas. Recentemente, os problemas de acesso aos recursos internos de microprocessador e SoCs, e os requisitos crescentes de controlabilidade, originaram a necessidade de métodos alternativos para este fim. As infra-estruturas OCD oferecem uma solução possível, que permite minimizar algumas das limitações de outras técnicas. Por outro lado, foi possível verificar que mesmo esta abordagem apresenta restrições, particularmente em situações em que se pretende um elevado desempenho.

A solução proposta neste documento foi elaborada tendo em vista as referidas limitações, apresentando uma alternativa eficiente e adaptável a diferentes cenários. Desta forma, a metodologia e o equipamento de suporte descritos foram projectados de acordo com requisitos de modularidade, expansibilidade e compatibilidade com diferentes arquitecturas. Apresentam-se variantes com desempenho e funcionalidades adaptadas a vários requisitos, desde o simples recurso a um depurador optimizado para a injeção de falhas, com intrusividade nula, até modificações no próprio sistema alvo, com eventual redução do seu desempenho. Pelo meio apresentam-se soluções mais equilibradas, baseadas na modificação das infra-estruturas OCD presentes num microprocessador, para apoiar a injeção de falhas. A inclusão nos microprocessadores modernos de infra-estruturas destinadas à depuração é já

hoje vulgarmente aceite. Como tal, o aumento da sobrecarga lógica, em sistemas considerados críticos, deverá ser também aceitável, se acompanhado por melhorias de desempenho e/ou funcionalidades relevantes. As propostas apresentadas nesta tese seguem neste sentido, tendo sido analisada a exequibilidade e interesse de melhorar uma infra-estrutura OCD normalizada NEXUS, já adoptada em diversos componentes comerciais.

O ambiente de injeção de falhas desenvolvido representa um caso de estudo, sendo a metodologia base proposta potencialmente aplicável a quaisquer microprocessadores que incluam uma infra-estrutura deste tipo, sendo a sua utilização dependente do desenvolvimento de um depurador compatível, ou da adaptação de equipamento existente. Foi verificado experimentalmente que a injeção de falhas em tempo real apresenta requisitos temporais muito mais exigentes, tornando-se muito mais importantes os tempos de leitura e escrita de dados, e particularmente relevante a forma como se faz a activação das operações de inserção de falhas. É nesta área que são visíveis as vantagens maiores da metodologia proposta, que advêm da inclusão de funcionalidades acrescidas.

Em comparação com outras técnicas de injeção de falhas em tempo real, as vantagens mais importantes são as seguintes:

- A intrusividade é geralmente mais reduzida do que em soluções baseadas em SWIFI e a cobertura é potencialmente mais elevada.
- A controlabilidade é total, sendo muito superior ao actualmente possível com técnicas físicas, não intrusivas.
- A intrusividade é mínima ou nula, sem modificações de código (como no SWIFI) ou equipamento adicional (como em algumas técnicas físicas).
- Na versão base, é possível inserir falhas em tempo real, com elevada precisão, em instantes de tempo bem determinados e sem intrusividade no sistema alvo.
- Nas versões mais complexas, torna-se viável a injeção de falhas em elementos não vulgarmente acessíveis por outras técnicas, se bem que com algum custo em termos de intrusividade.
- A observabilidade é dependente da mesma infra-estrutura utilizada para a injeção, com vantagens técnicas e logísticas.
- Relativamente a abordagens baseadas apenas em simulação, a solução proposta pode ser utilizada de forma análoga em diferentes fases da evolução do sistema alvo (projecto, protótipo, produção ou uso operacional) e é mais facilmente reutilizável, uma vez incluída a infra-estrutura necessária.

- É possível estender a solução proposta, de forma a lidar com requisitos de confiabilidade mais complexos.

Simultaneamente, devem também ser consideradas as limitações que foram identificadas:

- A cobertura é limitada aos recursos acessíveis pelas infra-estruturas utilizadas, podendo ser suficiente para os requisitos operacionais, ou requerer expansões, com eventual perda de conformidade e aumento de intrusividade.
- Existe uma probabilidade de erros no processo de injeção de falhas, quando executado em tempo real, devido ao acesso em paralelo a recursos do sistema alvo. Este problema pode ser minimizado, aumentando as capacidades da infra-estrutura utilizada.
- A solução proposta não é universal e as variantes mais eficientes requerem modificações ao sistema alvo. Mesmo que a sobrecarga seja reduzida, o projecto terá de sofrer alterações.

A evolução das normas de depuração poderá ajudar a reduzir o impacto das limitações referidas. A adopção de infra-estruturas OCD normalizadas apoiará o desenvolvimento de depuradores para a injeção de falhas e facilitará o projecto de alterações para o mesmo efeito. No limite, a inclusão de infra-estruturas do tipo OCD-FI, com elevado desempenho e cobertura adequada, poderão ser a melhor solução para a injeção de falhas em tempo real.

## **7.2 Trabalho Futuro**

As propostas e os resultados apresentados neste documento permitem antever algumas áreas onde se justifica investigação futura, nomeadamente as seguintes:

- Realização de experiências adicionais, recorrendo a outras técnicas de injeção de falhas, de forma a efectuar uma análise comparativa dos resultados, para um mesmo sistema alvo.
- Estudo mais aprofundado da possibilidade de estender a infra-estrutura OCD-FI, de forma a lidar eficientemente com o problema da injeção de falhas em registos e em tempo real, visto ser esta uma vertente importante.
- Aplicação da metodologia proposta a cenários operacionais mais complexos. Seria de particular interesse adaptar as infra-estruturas desenvolvidas a

microprocessadores com diferentes arquitecturas e com implantação em sistemas críticos.

Adicionalmente, deverá ser seguida com interesse a evolução dos esforços de normalização de recursos de depuração de microprocessadores. A inclusão de módulos, eventualmente opcionais, explicitamente destinados à injeção de falhas forneceria um mecanismo importante para uma eficiente verificação e validação da confiabilidade de sistemas críticos, particularmente em componentes com requisitos elevados de desempenho.

*“If everyone is thinking alike,  
someone isn’t thinking.”*

Gen. George S. Patton

**Capítulo 1**

**Capítulo 2**

**Capítulo 3**

**Capítulo 4**

**Capítulo 5**

**Capítulo 6**

**Capítulo 7**

**Referências**

**Anexo**

# Referências



## 8 Referências

- [Aidemark01] J. Aidemark, J. Vinter, P. Folkesson e J. Karlsson, “GOOFI: Generic Object-Oriented Fault Injection Tool”, International Conference on Dependable Systems and Networks (DSN’01), pp. 83-88, Gotemborg, Suécia, Julho 2001
- [Aidemark03] J. Aidemark, J. Vinter, P. Folkesson e J. Karlsson, “GOOFI: Generic Object-Oriented Fault Injection tool”, International Conference on Dependable Systems and Networks (DSN’03), São Francisco, EUA, Junho 2003
- [Almeida04] L. Almeida e P. Pedreiras, “Secção Especial: Sistemas de Tempo-Real: uma abordagem prática”, Revista do DETUA, Volume 4, Número 2, Janeiro 2004
- [Alves99] G. Alves, “Projecto para o Teste e Depuração com base nas Arquitecturas 1149.1 e P1149.4”, Tese de Doutoramento, Faculdade de Engenharia da Universidade do Porto, Portugal, 1999
- [AnalogD04] “ADSP-2106x SHARC® Processor User’s Manual”, Analog Devices, 2004
- [Anghel05] L. Anghel, R. Leveugle e P. Vanhauwaert, “Evaluation of SET and SEU Effects at Multiple Abstraction Levels”, 11<sup>th</sup> IEEE International On-Line Testing Symposium (IOLTS’05), pp. 309-312, Saint Raphael, França, Julho 2005
- [Arlat03] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs e G. Leber, “Comparison of physical and software-implemented fault injection techniques”, IEEE Transactions on Computers, Volume 52, Número 9, Setembro 2003.
- [Arlat90] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. Fabre, J.C. Laprie, E. Martins e D. Powell, “Fault injection for dependability validation: a methodology and some applications”, IEEE Transactions on Software Engineering, Volume 16, Número 2, Fevereiro 1990
- [Asenek98] V. Asenek, C. Underwood, R. Velazco, S. Rezgui, M. Oldfield, P. Cheynet e R. Ecoffet, “SEU induced errors observed in microprocessor systems”, IEEE Transactions on Nuclear Science, Dezembro 1998
- [Ashling03] “Vitra Emulator”, DS224 v3, Ashling, 2003
- [Ashling07] [www.ashling.com](http://www.ashling.com)

- [Avizienis04] A. Avizienis, J. Laprie, B. Randell e C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing”, IEEE Transactions on Dependable and Secure Computing, Volume 1, Número 1, Janeiro 2004
- [Avizienis67] A. Avizienis, “Design of Fault-Tolerant Computers”, Fall Joint Computer Conference, AFIPS, pp.733-43, Washington DC, EUA, 1967
- [Axiom07] [www.axman.com](http://www.axman.com)
- [Baraza00] J. Baraza, J. Garcia, D. Gil e P. Gil, “A prototype of a VHDL-based fault injection tool”, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT’00), Monte Fuji, Japão, Outubro 2000
- [Barton90] J. Barton, E. Czeck, Z. Segall e D. Siewiorek, “Fault injection experiments using FIAT”, IEEE Transactions on Computers, Volume 39, Número 4, Abril 1990
- [Benso98a] A. Benso, P. Civera, M. Rebaudengo, A. Ferro e M. Sonza Reorda, “A Hybrid Fault Injection Methodology for Real Time Systems”, The 28<sup>th</sup> Annual International Symposium on Fault-Tolerant Computing (FTCS-28), pp. 74-75, Munique, Alemanha, Junho 1998
- [Benso98b] A. Benso, M. Rebaudengo e M. Sonza Reorda, “Fault Injection for Embedded Microprocessor-based Systems”, Journal of Universal Computer Science (Special Issue on Dependability Evaluation and Validation), Volume 5, Número 5, pp. 693-711, 1998
- [Benso98c] A. Benso, M. Rebaudengo, M. Sonza Reorda e P. Civera, “An Integrated HW and SW Fault Injection Environment for Real-Time Systems”, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT’98), pp. 117-122, Austin, EUA, 1998
- [Berger03] A. Berger e M. Barr, “Introduction to On-Chip Debug”, Embedded Systems Programming, pp. 47-48, Março 2003
- [Berrojo02] L. Berrojo, I. Gónzalez, F. Corno, M. Reorda, G. Squillero, L. Entrena e C. López, “New Techniques for Speeding-Up Fault-Injection Campaigns”, Design Automation and Test in Europe (DATE’02), pp. 847-853, Paris, França, Março 2002
- [Boue98] J. Boue, P. Petillon e Y. Crouzet, “MEFISTO-L: A VHDL-Based Fault Injection Tool for the Experimental Assessment of Fault Tolerance”, The 28<sup>th</sup> Annual International Symposium on Fault-Tolerant Computing (FTCS-28), p. 168, Munique, Alemanha, Junho 1998

- [Burks97] J. Burks, "Electronic Computing Circuits of the ENIAC", Proceedings of the IEEE, Volume 85-N7, pp 1169-1173, 1997
- [Buttazo97] G. Buttazzo, "Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications", Kluwer Academic Publishers, 1997.
- [Carreira98] J. Carreira, H. Madeira e J.G. Silva, "Xception: a technique for the experimental evaluation of dependability in modern computers", IEEE Transactions on Software Engineering, Volume 24, Número 2, Fevereiro 1998
- [Chevochot00] P. Chevochot, A. Colin, D. Decotigny e I. Puaut, "Are COTS Suitable for Building Distributed Fault-Tolerant Hard Real-Time Systems?", IEEE International Parallel & Distributed Processing Symposium (IPDPS'00), pp. 699-705, Cancun, México, Maio 2000
- [Clark57] W. Clark, "The Lincoln TX-2 Computer Development", Western Joint Computer Conference, Los Angeles, EUA, Fevereiro 1957
- [Cunha99] J. Cunha, M. Rela e J. Silva, "Can Software Implemented Fault-Injection Be Used on Real-Time Systems?", Third European Dependable Computing Conference (EDCC'99), pp. 209-228, Praga, República Checa, Setembro 1999
- [Fidalgo06a] A. Fidalgo, G. Alves e J. Ferreira, "A Modified Debugging Infrastructure to Assist Real Time Fault Injection Campaigns", 9<sup>th</sup> IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS'06), pp. 174-179, Praga, República Checa, Abril 2006
- [Fidalgo06b] A. Fidalgo, G. Alves e J. Ferreira, "OCD-FI: On-Chip Debug and Fault Injection", International Conference on Dependable Systems and Networks (DSN'06), Filadélfia, EUA, Junho 2006
- [Fidalgo06c] A. Fidalgo, G. Alves e J. Ferreira, "Real Time Fault Injection Using a Modified Debugging Infrastructure", International On-Line Symposium (IOLTS'06), pp. 242-250, Lago Como, Itália, Julho 2006
- [Fidalgo06d] A. Fidalgo, M. Gericota, G. Alves e J. Ferreira, "Using NEXUS compliant debuggers for real time fault injection on microprocessors", Sociedade Brasileira de Ciências da Computação e Informática – Chip on the Mountains (SBCCI'06), pp. 214-219, Ouro Preto, Brasil, Agosto 2006
- [Fidalgo06e] A. Fidalgo, M. Gericota, G. Alves e J. Ferreira, "Real Time Fault Injection Using On Chip Debug Infrastructures – A Case Study", 21<sup>st</sup> Conference on Design of Circuits and Integrated Systems (DCIS'06), Barcelona, Espanha, Novembro de 2006

- [Fidalgo06f] A. Fidalgo, G. Alves e J. Ferreira, "Real Time Fault Injection Using Enhanced OCD – A Performance Analysis", 21<sup>st</sup> IEEE International Symposium on Defect. And Fault Tolerance in VLSI Systems (DFT'06), pp. 254-264, Arlington, EUA, Outubro 2006
- [Fidalgo06g] A. Fidalgo, G. Alves e J. Ferreira, "An Enhanced Debugger for Real-Time Fault Injection on Microprocessor Systems", 6<sup>th</sup> European Dependable Computing Conference (EDCC'06), Coimbra, Portugal, Outubro 2006
- [Folkesson98] P. Folkesson, S. Svensson e J. Karlsson, "A Comparison of Simulation Based and Scan Chain Implemented Fault Injection", The 28<sup>th</sup> Annual International Symposium on Fault-Tolerant Computing (FTCS-28), p. 284, Munique, Alemanha, Junho 1998
- [Freescale05] "Programming the DSP56300 OnCE and JTAG Ports", Freescale, 2005
- [Freescale06] "Version 3 Coldfire Core User's Manual", Freescale, 2006
- [Freescale07] [www.freescale.com](http://www.freescale.com)
- [Gaisler02] J. Gaisler, "A Portable and Fault-Tolerant Microprocessor Based on the SPARC V8 Architecture", International Conference on Dependable Systems and Networks (DSN'02), Bethesda, EUA, Junho 2002
- [Gaisler94] J. Gaisler, "Concurrent error-detection and modular fault-tolerance in a 32-bit processing core for embedded space flight applications", Fault-Tolerant Computing, IEEE, 1994.
- [Gericota01] M. Gericota, G. Alves, M. Silva e J. Ferreira, "DRAFT: An On-line Fault Detection Method for Dynamic and Partially Reconfigurable FPGAs", 7th International On-Line Testing Workshop (IOLTW'01), pp. 34-36, Taormina, Itália, Julho de 2001
- [Gil00] D. Gil, J. Gracia, J. Baraza e P. Gil, "A Study of the Effects of Transient Fault Injection into the VHDL Model of a Fault-Tolerant Microcomputer System", 6<sup>th</sup> IEEE International On-Line Testing Workshop (IOLTW'00), Palma de Maiorca, Espanha, Julho 2000
- [Gil02] P. Gil, I. Gracia, D. Gil e I. Baraza, "Using VHDL-Based Fault Injection to exercise Error Detection Mechanisms in the Time-Triggered Architecture", Pacific Rim International Symposium on Dependable Computing (PRDC'02), p. 316, Tsukuba, Japão, Dezembro 2002
- [Gracia01] J. Gracia, J. Baraza, D. Gil e P. Gil, "Comparison and Application of Different VHDL-based Fault Injection Techniques", IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'01), São Francisco, EUA, Outubro 2001

- [GSTF07] [www.disca.upv.es/gstf](http://www.disca.upv.es/gstf)
- [Haller96] C. Haller, “The ZEN of BDM”, MacGraigor Systems, 1996
- [Han95] S. Han, K. Shin e H. Rosenberg, “DOCTOR: An Integrated Software Fault InjeCTiOn EnviRonment for Distributed Real-time Systems,” International Computer Performance and Dependability Symposium (IPDS’95), Erlangen; Alemanha, 1995.
- [Hansson05] H. Hansson, M. Nolin e T. Nolte, “Real-Time Systems”, The Industrial Information Technology Handbook, CRC Press, 2005
- [Hsueh97] M. Hsueh, T. Tsai e R. Iyer, “Fault Injection Techniques and Tools”, Computer, IEEE, Volume 30, Número 4, p.75-82, 1997
- [IBM98] “The PowerPC 405<sup>TM</sup> Core”, IBM Microelectronics, 1998
- [IEEE00] “Draft IEEE Standard VHDL Language Reference Manual”, IEEE, 2000
- [IEEE01] “1364-2001 IEEE standard Verilog hardware description language”, IEEE, 2001
- [IEEE03] “The Nexus 5001 Forum<sup>TM</sup> Standard for a Global Embedded Processor Debug Interface v2.0”, Industry Standards and Technology Organization (IEEE-ISTO), Dezembro 2003
- [IEEE05] “IEEE Standard SystemC® Language Reference Manual – IEEE Std 1666<sup>TM</sup>-2005”, IEEE Computer Society, 2005
- [IEEE07a] [grouper.ieee.org/groups/1687/](http://grouper.ieee.org/groups/1687/)
- [IEEE07b] [standards.ieee.org](http://standards.ieee.org)
- [IEEE90] “IEEE Standard Test Access Port And Boundary Scan Architecture”; IEEE 1149.1; 1990
- [IEEE99] “The Nexus 5001 Forum<sup>TM</sup> Standard for a Global Embedded Processor Debug Interface v1.0”, Industry Standards and Technology Organization (IEEE-ISTO), Dezembro 1999
- [Intel05] “Hot-Debug for Intel Xscale® Core Debug – White Paper”, Intel, Maio 2005
- [iSystem04] “User’s Manual winIDEA V. 9.4”, iSystem A.G., 2004
- [iSystem06] “Freescale PowerPC 5xx/8xx Family On-Chip Emulation”, iSystem A.G., 2006
- [iSystem07] [www.isystem.com](http://www.isystem.com)

- [Iyer95] R. Iyer, K. Trivedi e J. Goldberg, "Introduction to Special Issue on F-T Computing", IEEE Transactions on Computers, Volume 44, Número 2, pp. 165-167, 1995
- [Jenn94] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson e J. Karlsson, "Fault injection into VHDL models: the MEFISTO tool", 24<sup>th</sup> International Symposium on Fault-Tolerant Computing (FTCS-24), Austin, Texas, Junho 1994
- [Kanawati92] G. Kanawati, N. Kanawati e J. Abraham, "FERRARI: a tool for the validation of system dependability properties", 22<sup>nd</sup> International Symposium on Fault-Tolerant Computing (FTCS-22), Boston, EUA, Julho 1992
- [Kanawati95] G. Kanawati, N. Kanawati e J. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System", IEEE Transactions on Computers, Volume 44, Número 2, pp. 248 – 260, Fevereiro 1995
- [Karlsson89] J. Karlsson e U. Gunneflo, "Use of Heavy-ion Radiation from Californium-252 for Fault Injection Experiments", 1<sup>st</sup> IFIP International Working Conference on Dependable Computing for Critical Applications, Santa Barbara, EUA, Agosto 1989
- [Karlsson95] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber e J. Reisinger, "Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture", 5<sup>th</sup> IFIP Working Conference on Dependable Computing for Critical Applications, Urbana-Champaign, EUA, Setembro 1995
- [Kastensmidt05] F. Kastensmidt, L. Sterpone, L. Carro e M. Reorda, "On the Optimal Design of Triple Modular Redundancy Logic for SRAM-based FPGAs", Design Automation and Test in Europe (DATE'05), pp. 1290-1295, Munique, Alemanha, Março 2005
- [Krishna97] C. Krishna e K. Shin, "Real-Time Systems", McGraw-Hill, 1997
- [Kumar07] U. Kumar e B. Umashankar, "Improved Hamming Code for Error Detection and Correction", 2<sup>nd</sup> International Symposium on Wireless Pervasive Computing (ISWPC'07), San Juan, EUA, Fevereiro 2007
- [Lala85] P. Lala, "Fault Tolerant and Fault Testable Hardware Design", Prentice Hall International, Nova Iorque, 1985
- [Laprie85] J. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology", 15<sup>th</sup> International Symposium on Fault Tolerant Computing (FCTS-15), Ann Arbor, EUA, Junho 1985
- .

- [Lardner34] D. Lardner, “Babbage’s Calculating Engines”, Edinburgh Review, Edimburgo, Escócia, 1834
- [Lauterbach02] “Trace32 In-Circuit Debugger User Manual”, Lauterbach, 2002
- [Lauterbach07] [www.lauterbach.com](http://www.lauterbach.com)
- [Lehman03] M. Lehman e J. Ramil, “Software evolution – Background, theory, practice”, Information Processing Letters, Volume 88, pp. 33-44, 2003
- [Leveugle02] R. Leveugle, K. Hadjiat, “Multi-Level Fault Injection Experiments Based on VHDL Descriptions: A Case Study”, 8<sup>th</sup> IEEE International On-Line Testing Workshop (IOLTW’02), pp. 107-111, Ilha de Bendor, França, Julho 2002
- [Lint82] J. Lint, “Introduction to coding theory”, Springer, 1982
- [Lions96] J. Lions (Chairman of the Board), “ARIANE 5 Flight 501 Failure Report by the Inquiry Board”, Julho 1996
- [MacNamee00] C. MacNamee e B. Heffernan “Emerging on-chip debugging techniques for real-time embedded systems”, Computing and Control Engineering Journal, Dezembro 2000
- [Madeira02] H. Madeira, R.R. Some, F. Moreira, D. Costa e D. Rennels, “Experimental Evaluation of a COTS System for Space Applications”, International Conference on Dependable Systems and Networks (DSN’02), Bethesda, EUA, Junho 2002
- [Maia02] R. Maia, L. Henriques, D. Costa e H. Madeira, “Xception™ - Enhanced Automated Fault-Injection Environment”, International Conference on Dependable Systems and Networks (DSN’02), Bethesda, EUA, Junho 2002.
- [Maiden07] V. Maiden, “The Transistor and the Dawn of the Information Age”, Associated Content Online, disponível online em [www.associatedcontent.com](http://www.associatedcontent.com), Julho 2007
- [Maman98] R. Maman e E. Barke, “Real Time Fault Injection Using Logic Emulators”, 3<sup>rd</sup> Asia and South Pacific Design Automation Conference (ASP-DAC’98), pp. 475-479, Yokohama, Japão, Fevereiro 1998
- [Martinez99] R. Martínez, P. Gil, G. Martín, C. Pérez e J. Serrano, “Experimental Validation of High-Speed Fault-Tolerant Systems Using Physical Fault Injection”, 7<sup>th</sup> IFIP Working Conf. Dependable Computing for Critical Applications, San Jose, EUA, Janeiro 1999

- [MentorG04] “ModelSim® Advanced Verification and Debugging – Xilinx – User’s Manual – Version 6.0a”, Mentor Graphics, Setembro 2004
- [MentorG07] [www.model.com](http://www.model.com)
- [MIPI07] [www.mipi.org](http://www.mipi.org)
- [MIPS06] “EJTAG Specification v3.10”, MIPS Technologies, 2006
- [Moore56] E. Moore e C. Shannon, “Reliable Circuits Using Less Reliable Relays”, Journal Franklin Institute, Volume 262, pp. 191-208 e 281-297, Setembro 1956
- [Moreno99] W. Moreno, J. Samson e F. Falquez, “Laser Injection of Soft Faults for the Validation of Dependability Design”, Journal of Universal Computer Science, Volume 5, Número 10, 1999
- [Motorola00] “Programming Environments Manual For 32-Bit Implementations of the PowerPC Architecture Rev 2.0”, Motorola, 2000
- [Motorola01] “Nexus Interface Connector Options for MPC56x Devices Rev0.1”, Motorola, Fevereiro 2001
- [Motorola02a] “MPC565/MPC566 User’s Manual”, Motorola, Setembro 2002
- [Motorola02b] “MPC5xx Family Specification Sheet”, Motorola, 2002
- [Motorola06] “Introduction to HCS08 Background Debug Mode”, Motorola, 2006
- [Motorola96] “CPU32 Reference Manual”, Motorola, 1996
- [Motorola98] “M68HC05 Family – Understanding Small Microcontrollers – Rev 2.0”, Motorola, 1998
- [Multicore07] [www.multicore-association.org](http://www.multicore-association.org)
- [Neumann52] J. Neumann, “Probabilistic Logics and the Synthesis of Reliable Organisms from Un-reliable Components”, Automata Studies, Princeton University Press, pp. 43–98, Janeiro 1952
- [OCP-IP07] [www.ocpip.org](http://www.ocpip.org)
- [Opencores07] [www.opencores.org](http://www.opencores.org)
- [Oshana02] R. Oshana, “Introduction to JTAG”, Embedded Systems Programming, CMP Media, Outubro 2002
- [Oshana06] R. Oshana e G. Swaboda “Compact JTAG (IEEE P1149.7) Overview of Cjtag, a Reduced Pin Debug Access Protocol”, 5<sup>th</sup> IEEE International Board Test Workshop (BTW’06), Denver, EUA, Setembro 2006

- [Paap93] G. Paap e E. Silha, "PowerPC: A Performance Architecture", IEEE Computer Society International Computer Conference, 1993
- [Pardo06] J. Pardo, J. Campelo, J. Ruiz e P. Gil, "Temporal Characterization of Embedded Systems Using Nexus", 6<sup>th</sup> European Dependable Computing Conference (EDCC'06), Coimbra Portugal, Outubro 2006
- [Phytec07] [www.phytec.com](http://www.phytec.com)
- [Pierce65] W. Pierce, "Failure-Tolerant Computer Design", Academic Press, 1965
- [Prinetto98] P. Prinetto, M. Rebaudengo e M. Sonza Reorda, "Exploiting the Background Debugging Mode in a Fault Injection system", The 3rd Annual IEEE International Computer Performance & Dependability Symposium (IPDS'98), p. 277, Durham, EUA, Setembro 1998
- [QLF07] [tima.imag.fr/qlf/](http://tima.imag.fr/qlf/)
- [Rahbaran04] B. Rahbaran, A. Steininger e T. Handl, "Built-in fault injection in hardware – the FIDYCO example", Second IEEE International Workshop on Electronic Design, Test and Applications (Delta'04), pp. 327-332, Perth, Australia, Janeiro 2004
- [Rearick05] J. Rearick, B. Eklow, K. Posse, A. Crouch e B. Bennetts, "IJTAG (Internal JTAG): A Step Toward a DFT Standard", International Test Conference (ITC'05), Austin, EUA, Novembro 2005
- [Rebaudengo02] M. Rebaudengo, M. Reorda, M. Violante, B. Nicolescu e R. Velazco, "Coping with SEUs/SETs in microprocessors by means of low-cost solutions: A comparison study", IEEE Transactions on Nuclear Science, Volume 49, Número 3, Junho 2002.
- [Rebaudengo99a] M. Rebaudengo, A. Benso e M. Sonza Reorda, "FlexFi: a flexible Fault Injection environment for microprocessor-based systems", 18<sup>th</sup> International Conference on Computer Safety, Reliability and Security (SAFECOMP'99) pp. 323-335, Toulouse, França, Setembro 1999
- [Rebaudengo99b] M. Rebaudengo e M. Sonza Reorda, "Evaluating the Fault Tolerance Capabilities of Embedded Systems via BDM", 17th IEEE VLSI Test Symposium (VTS'99). pp. 452-457, Dana Point, EUA, Abril 1999
- [Rela06] M. Rela, J. Cunha, L. Santos, M. Gameiro, P. Gonçalves, G. Alves, A. Fidalgo, P. Fortuna, R. Maia, L. Henriques e D. Costa, "Exploiting the IEEE 1149.1 Standard for Software Reliability Evaluation in Space Applications", European Safety and Reliability Conference, Estoril, Portugal, Outubro 2006.

- [Rodriguez02] M. Rodriguez, A. Albinet e J. Arlat, “MAFALDA-RT: a tool for dependability assessment of real-time systems”, International Conference on Dependable Systems and Networks (DSN’02), pp. 267-272, Bethesda, EUA, Junho 2002
- [Ruiz05] J. Ruiz, J. Pardo, J. Campelo e P. Gil, “On-chip debugging-based fault emulation for robustness evaluation of embedded software components”, 11<sup>th</sup> Pacific Rim International Symposium on Dependable Computing (PRDC’05), Changsha, China, Dezembro 2005
- [Sampson98] J. Samson, W. Moreno e F. Falquez, “A Technique for Automated Validation of Fault Tolerant Designs Using Laser Fault Injection (LFI)”, 28<sup>th</sup> Annual International Symposium on Fault-Tolerant Computing (FTCS-28), Munique, Alemanha, Junho 1998
- [Santos98] J. Santos e J. Ferreira, “Fault Tolerance: New Trends for Digital Circuits”, 5<sup>th</sup> IEEE International Conference on Electronics, Circuits and Systems (ICECS’98), p. 237, Lisboa, Setembro 1998
- [Sieh96] V. Sieh, O. Tschäche e F. Balbach, “VHDL-based Fault Injection with VERIFY”, Internal Report 5/96, Friedrich-Alexander University of Erlangen-Nuremberg, Maio 1996.
- [Sieh97] V. Sieh, O. Tschache e F. Balbach, “VERIFY: evaluation of reliability using VHDL-models with embedded fault descriptions”, 27<sup>th</sup> Annual International Symposium on Fault-Tolerant Computing (FTCS-27), Seattle, EUA, Junho 1997
- [Some01] R. Some, W. Kim, G. Khanoyan, L. Callum, A. Agrawal e J. Beahan, “A software-implemented fault injection methodology for design and validation of system fault tolerance”, The International Conference on Dependable Systems and Networks (DSN’01), pp. 501-506, Gotemborg, Suécia, Julho 2001
- [SPIRIT07a] [www.spiritconsortium.org](http://www.spiritconsortium.org)
- [SPIRIT07b] [www.spiritconsortium.org/tech/p1685](http://www.spiritconsortium.org/tech/p1685)
- [SPIRIT07c] [www.eda.org/spirit-p1685/](http://www.eda.org/spirit-p1685/)
- [SPRINT07] [www.sprint-project.net](http://www.sprint-project.net)
- [Stankovic88] Stankovic, J.A. (ed.), “Real-Time Computing Systems: The Next Generation”, IEEE Tutorial on Hard Real-Time Systems, IEEE Press, Nova Iorque, 1988.
- [Steininger03] A. Steininger, B. Rahbaran e T. Handl, “Built-In Fault Injectors – The Logical Continuation of BIST?”, First Workshop on Intelligent Solutions in Embedded Systems (WISES’03), pp. 187-196, Viena, Austria, Junho 2003

- [UML07] [www.uml.org](http://www.uml.org)
- [Valderas06] M. Valderas, M. García, C. Ongil e L. Entrena, “Emulation-based Fault Injection in Circuits with Embedded Memories”, International On-Line Symposium (IOLTS’06), pp. 183-184, Lago Como, Itália, Julho 2006
- [Velazco02] R. Velazco, A. Corominas e P. Ferreyra, “Injecting Bit Flip Faults by Means of a Purely Software Approach: A Case Studied”, 17<sup>th</sup> IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT’02) pp. 108-116, Vancouver, Canadá, Setembro 2002
- [Velazco05] R. Velazco, R. Ecoffet e F. Faure, “How to Characterize the Problem of SEU in Processors & Representative Errors Observed on Flight”, 11<sup>th</sup> IEEE International On-Line Testing Symposium (IOLTS’05), Saint Raphael, França, Julho 2005
- [Velazco90] R. Velazco, C. Bellon e B. Martinet, “Failure coverage of functional test methods: a comparative experimental evaluation”, International Test Conference (ITC’90), Washington DC, EUA, Setembro 1990
- [Velazco95] R. Velazco, A. Assoum, N.E. Radi, R. Ecoffet e X. Botey, “SEU fault tolerance in artificial neural networks”, IEEE Transactions on Nuclear Science, Volume 42, Número 6, Parte 1, pp. 1856 – 1862, Dezembro 1995
- [Vetterli00] C. Vetterli, “Nexus: Proof of Concept and its First Implementation”, The Embedded Systems Show (ESS’00), Londres, Maio 2000
- [Vinter05] J. Vinter, O. Hannius, T. Norlander, P. Folkesson e J. Karlsson, “Experimental dependability evaluation of a fail-bounded jet engine control system for unmanned aerial vehicles”, International Conference on Dependable Systems and Networks (DSN’05), Yokohama, Japão, Junho 2005.
- [Wong03] W. Wong, “Next Nexus Delivers Debug Data”, Electronic Design Online, Issue 5271, Julho 2003
- [Xilinx05] “Xilinx ISE 7 Software Manuals”, Xilinx Inc., 2005
- [Xilinx07] [www.xilinx.com](http://www.xilinx.com)
- [Yuste03a] P. Yuste, D. Andrés, L. Lemus, J. Serrano e P. Gil, “INERTE: Integrated Nexus-Based Real-Time Fault Injection Tool for Embedded Systems”, International Conference on Dependable Systems and Networks (DSN’03), São Francisco, EUA, Junho 2003

- [Yuste03b] P. Yuste, L. Lemus, J. Serrano e P. Gil, “A Methodology for Software Implemented Fault Injection Using Nexus”, International Conference on Dependable Systems and Networks (DSN’03), São Francisco, EUA, Junho 2003
- [Yuste03c] P. Yuste, “Contribución a la Validación de la Confiabilidad en los Sistemas Empotrados Tolerantes a Fallos”, Tese de Doutorado, Universidad Politécnica de Valencia, 2003
- [Zilmer99] M. Zilmer, “MIPS tailors debug with on-chip unit”, EE Times, Abril 1999

*“640K ought to be enough  
for anybody.”*

Bill Gates

**Capítulo 1**

**Capítulo 2**

**Capítulo 3**

**Capítulo 4**

**Capítulo 5**

**Capítulo 6**

**Capítulo 7**

**Referências**

**Anexo**

**Anexo**



## 9 Anexo

O anexo desta dissertação é constituído por duas partes:

- Uma em formato impresso, que ocupa as páginas seguintes.
- Outra em formato electrónico, armazenada no CD-ROM, que se encontra na bolsa fixada no verso da contracapa.

A seguinte lista descreve os elementos que fazem parte do anexo em formato impresso, todos eles ilustrados na forma de esquemático:

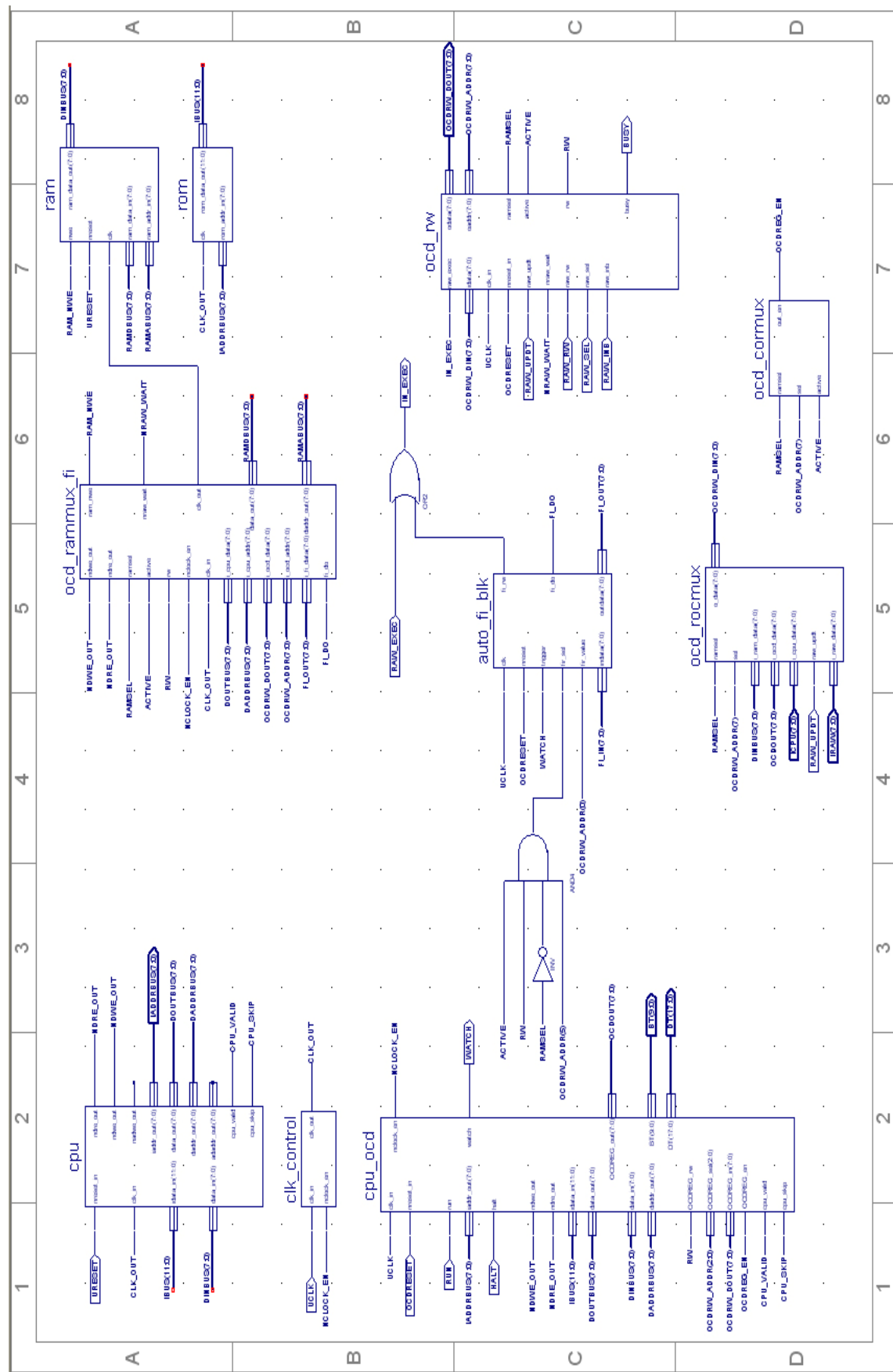
- Arquitectura de topo do microprocessador e infra-estrutura OCD básica
- Arquitectura de topo do microprocessador e infra-estrutura OCD-FI
- Arquitectura de topo do depurador.
- Modelo do sistema utilizado para a simulação das experiências.

O anexo em formato electrónico inclui:

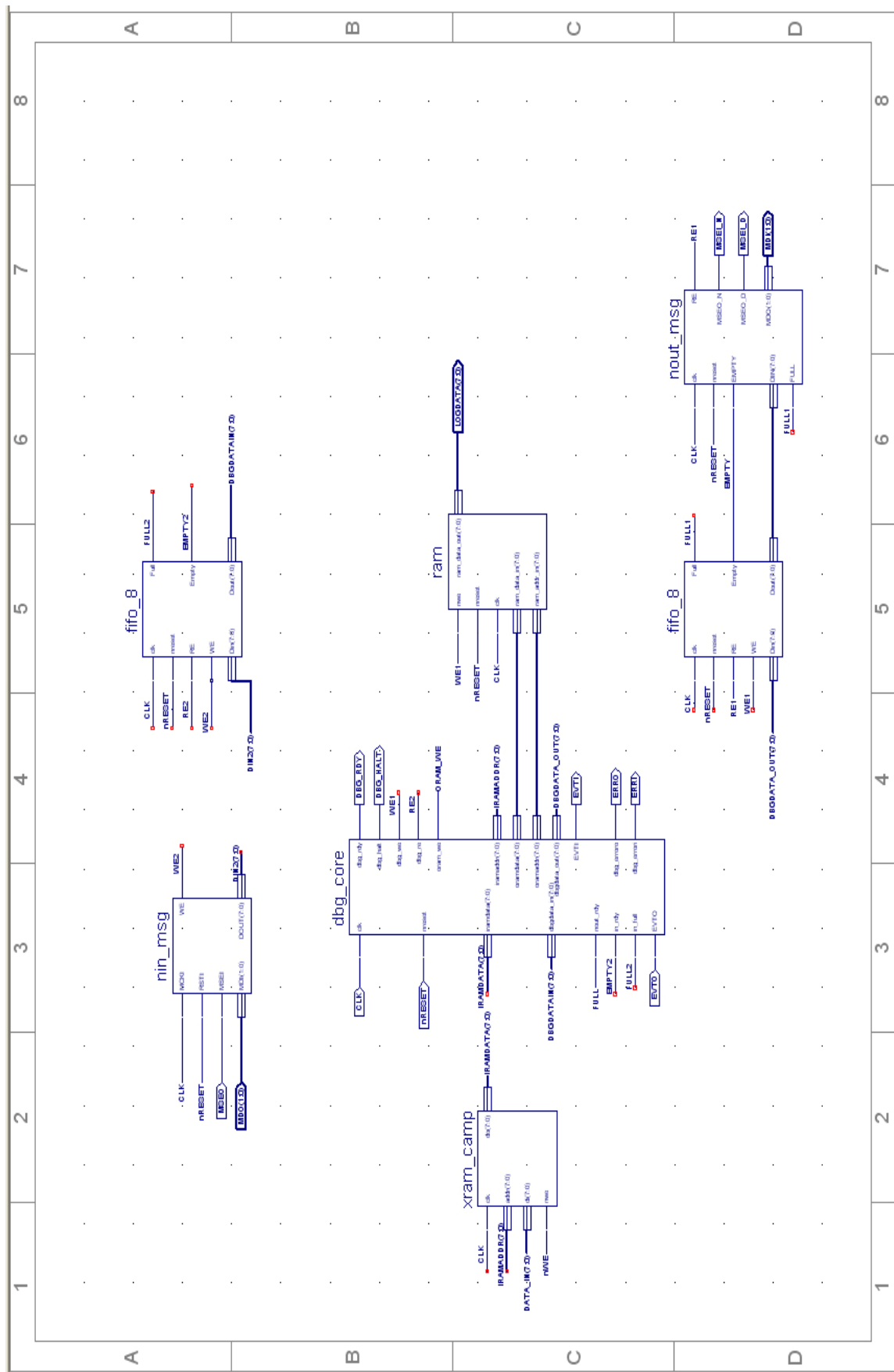
- A aplicação *CpuGenerator* versão 2.0.
- Ligações e instruções sobre a forma de obter o *Webpack* da Xilinx, que permite analisar, modificar e sintetizar os ficheiros VHDL.
- Ligações e instruções sobre a forma de obter o *Modelsim* da Mentor Graphics, que permite simular os componentes descritos em VHDL, bem como o ambiente de injeção de falhas completo.
- Ficheiros com a descrição dos principais microprocessadores utilizados (VHDL).
- Ficheiros com a descrição dos blocos de memória (VHDL).
- Ficheiros com a descrição das infra-estruturas OCD (VHDL).
- Ficheiros com a descrição do depurador optimizado (VHDL).
- Ficheiros com as arquitecturas dos diversos componentes em formato de esquemático, incluído as diversas versões dos elementos ilustrados em seguida e diversos níveis intermédios (Esquemáticos com extensão .SCH).
- Ficheiros com as aplicações utilizadas, em versões de 8, 16 e 32 bits, com e sem tolerância a falhas por *software* (*Assembly*).
- Ficheiros de inicialização dos modelos das memórias, utilizados no ambiente de simulação (Binário).
- Esta dissertação em formato Adobe Acrobat.



## Microprocessador e infra-estrutura OCD-FI



## Depurador



## Ambiente Experimental para Simulação

