View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositório Científico do Instituto Politécnico do Porto

Injecção de Falhas por Varrimento Periférico em Processadores

Tema: **Computadores e Sistemas Informáticos (principal)**
Electrónica e Instrumentação (alternativo)

André Vaz Fidalgo **Gustavo Ribeiro Alves**
avf@dee.isep.ipp.pt galves@dee.isep.ipp.pt
Instituto Superior de Engenharia do Porto
Dep. Eng. Electrotécnica
Rua Dr. António Bernardino de Almeida 431
4200-072 Porto
Fax: 228321159
Tel: 228340500

Luís Faria dos Santos
lsantos@isec.pt
Instituto Superior de Engenharia de Coimbra
Dep. Eng. Informática e de Sistemas
Rua Pedro Nunes – Quinta da Nora
3030-199 Coimbra
Fax: 239.790.390
Tel: 239.790.200

Mário Zenha Rela
mzrela@dei.uc.pt
Universidade de Coimbra
Dep. Eng. Informática / Pólo II / 298
3030-290 Coimbra
Fax: 239.701.266
Tel. 239.790.000

Resumo

As técnicas de injecção de falhas mais utilizadas podem ser classificadas em técnicas de indução, técnicas de injecão por hardware e técnicas de injecção por software. Dentro das técnicas de injecção por hardware surgem as técnicas de injecção de falhas recorrendo à tecnologia de teste por varrimento periférico. Esta encontra-se fortemente implantada nos processadores mais recentes, fornecendo um método de acesso ao seu interior de forma a permitir operações de teste e depuração. A sua utilização para a injecção de falhas é um passo lógico e tem sido estudada desde os anos 90. Os trabalhos desenvolvidos nesta área são diversos e incluem soluções que não exigem qualquer alteração à infraestrutura normalizada e soluções que modificam as células e a infraestrutura de controlo de forma a suportarem esta funcionalidade. Estas alterações implicam atrasos temporais acrescidos e um aumento da área de silício destinada a funções de teste, tornando-se importante uma preocupação com a optimização para que a inclusão de capacidades de injecção de falhas não afecte significativamente o desempenho e o custo dos componentes onde são implementadas.

1. Introdução

A confiabilidade (*dependability*) representa o grau de operacionalidade de um determinado sistema, em face de condições de utilização, onde se inclui a possibilidade de existência de falhas no interior ou na periferia (i.e. no interface com o exterior) desse sistema. A verificação da confiabilidade recorre geralmente a um método designado de "injecção de falhas" que visa avaliar o comportamento do sistema na presença controlada de uma, ou mais, falhas (i.e. conhecem-se à priori os vários parâmetros associados – tipo, localização, instante de aplicação, e duração da falha).

A confiabilidade assume uma expressão relevante no campo dos sistemas informáticos (ou baseados em processadores) em face da sua crescente utilização em aplicações / sistemas críticos (controlo de navegação em aviões, sistemas espaciais, etc.). Numa outra dimensão, tem-se assistido a uma crescente aderência dos processadores (enquanto circuitos integrados, CI) à norma IEEE 1149.1, desde o seu aparecimento no início da década de 90 [1]. Esta norma especifica uma arquitectura de varrimento periférico e um porto de acesso ao teste que permite: a) o teste de interligações entre CI digitais montados em cartas de circuito impresso; b) o acesso a mecanismos complementares de teste localizados no interior dos próprios CI. Esta última possibilidade tem sido largamente utilizada no caso específico dos processadores, tendo-se assistido ao aparecimento de mecanismos de apoio à emulação / depuração [2], ou ainda, em casos pontuais, de mecanismos de injecção de falhas [3], dado o grau de utilização dos processadores em sistema críticos. Esta última vertente constitui o assunto principal abordado neste artigo onde se começa por caracterizar, na secção 2, a área da injecção de falhas em sistemas computacionais, seguindo-se na secção 3 um estudo sumário da implantação da norma IEEE 1149.1 (ou *Boundary Scan Test*, BST) em processadores. A secção 4 combina as duas anteriores, apresentando o estado-de-arte na injecção de falhas por varrimento periférico, nomeadamente os trabalhos publicados nesta área, e que incluem propostas de alteração daquela infraestrutura de teste para esse efeito específico. Com base na apresentação anterior, e em algumas lacunas identificadas, propõe-se na secção seguinte uma célula de varrimento periférico optimizada para injecção de falhas, explicando-se em detalhe a sua arquitectura, as alterações à restante lógica da infraestrutura de teste, a sequência de operações ao nível do porto de acesso ao teste para injectar cada tipo de falha suportado, seguindo-se a descrição da implementação e experimentação da solução proposta e a análise do seu impacto (em termos de atraso e quantidade de lógica adicionais), tendo presente uma comparação com as soluções já existentes e apresentadas na secção 4.

2. Injecção de falhas em sistemas computacionais

A injecção de falhas é um processo com raízes em vários campos da ciência. Por exemplo, na indústria automóvel os testes de colisão (*crash-tests*) são usados para avaliar os mecanismos de protecção à vida presentes no veículo. Tal fomenta a confiança nesses mecanismos mas também permite validar e aperfeiçoar os modelos usados na fase de projecto do seu desenvolvimento [4].

2.1 Objectivo

No âmbito dos sistemas lógicos (sistemas computacionais, aplicações de software, etc.) a unidade é igualmente submetida de forma intencional a uma falha (Figura 1). Mesmo depois de "contaminado" o sistema pode continuar a prestar um serviço de qualidade. Porém, se a falha injectada se manifestar de forma visível para o utilizador do sistema, diz-se que este último se encontra avariado por não ter conseguido continuar a cumprir os requisitos de serviço esperados [5].

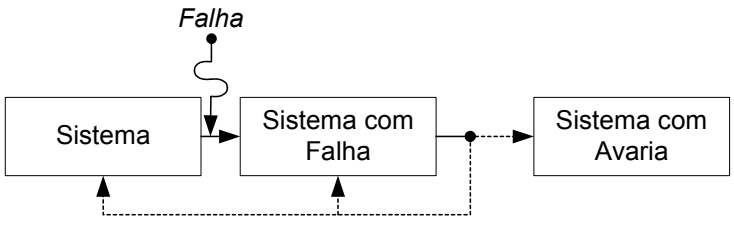


Figura 1: Injecção de falhas

Existem três motivos principais para realizar uma campanha de injecção de falhas: apoiar a actividade de teste clássico, complementar a sua abrangência e estender os seus objectivos. O primeiro motivo decorre normalmente da inviabilidade prática de realizar o teste exaustivo de um sistema lógico. Como solução procura-se o conjunto de testes que ofereça a melhor cobertura, ou seja, aquele que consegue detectar o maior número de falhas injectadas [6]. Uma segunda aplicação da injecção de falhas visa testar mecanismos especificamente introduzidos num sistema para deliberadamente lidar com falhas (detectando-as ou tolerando-as) que se julguem poder vir a afectar o mesmo. De facto a actividade de tais mecanismos só assim pode ser estimulada [7]. Finalmente, a injecção de falhas é cada vez mais usada para submeter o sistema-alvo a cenários tão bizarros que durante o seu projecto nunca foram tidos em consideração. Como se refere em [4] trata-se de uma forma exagerada de *stress test* melhor designada por *torture test*, com a qual se torna possível conhecer melhor o espaço de resposta do sistema.

2.2 Técnicas

Na Figura 2 agrupam-se as principais técnicas de injecção de falhas de acordo com o nível de abstracção a que consideram o sistema-alvo. Esta organização é normalmente simplificada, considerando-se apenas três classes distintas: técnicas de indução, técnicas HWIFI (*Hardware Implemented Fault Injection*) e técnicas SWIFI (*Software Implemented Fault Injection*).

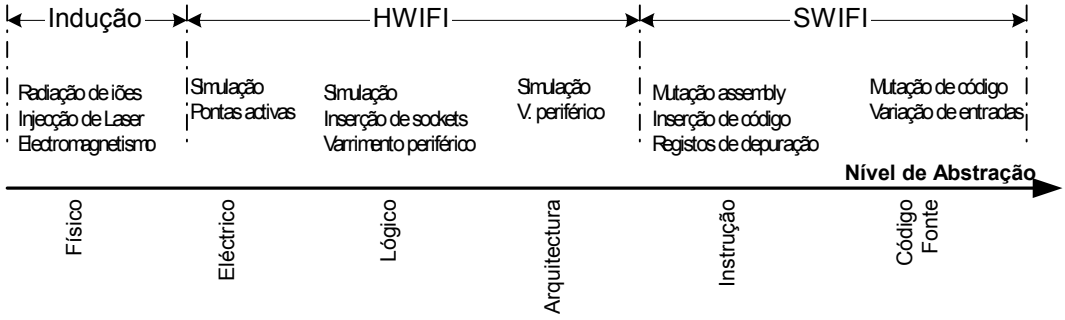


Figura 2: Técnicas de injecção de falhas

As técnicas de indução não envolvem o contacto físico directo com o alvo. Assentam antes em processos de radiação (e.g. [8]) ou criação de campos electromagnéticos destinados a desencadear *Single Event Upsets*. Apontam-se-lhe como vantagens a acessibilidade – podem virtualmente afectar qualquer zona do alvo, e a sua potencial representatividade – reproduzem características semelhantes às presentes no ambiente de utilização previsto para o sistema (e.g. órbitas elevadas).

As técnicas de HWIFI intervêm por instrumentação física junto do sistema-alvo (e.g. forçando valores lógicos nos pinos do CI [9]). Como vantagens encontra-se a sua resolução temporal, a sua baixa intrusividade e sua proximidade às falhas reais. O custo envolvido na adopção desta classe de técnicas, a inflexibilidade associada, a sua decrescente acessibilidade ao espaço funcional do alvo e a sua exequibilidade cada vez mais condicionada pelas elevadas frequências de funcionamento, têm-lhe retirado gradualmente o interesse outrora detido.

Nos níveis de abstracção mais elevados surgem as técnicas SWIFI, completamente suportadas por software. A sua popularidade e notoriedade tem crescido consideravelmente, sendo aplicadas quer à emulação de falhas de níveis de abstracção inferiores (e.g [10]) quer à perturbação directa do código fonte (e.g. [11]). O seu sucesso deve-se essencialmente à sua flexibilidade, ao seu custo e complexidade relativamente reduzidos e à facilidade de adaptação a diferentes alvos. Como principais desvantagens encontra-se a baixa resolução temporal exibida e a intrusividade, em termos de carga, que em geral impõe ao sistema.

3. Implantação de tecnologias de teste por varrimento periférico em processadores

Actualmente é possível encontrar um porto de acesso ao teste compatível com a norma IEEE 1149.1 numa grande variedade de processadores, processadores embebidos e processadores de sinal.
A tabela seguinte apresenta de forma não exaustiva, algumas famílias de processadores comerciais, indicando os fabricantes do processador e do equipamento de depuração e/ou emulação para cada caso.

Tabela 1 – Processadores com BST

Família	Fabricantes	Emuladores	Observações
AMD 29K	AMD	Corelis	
AMD 5K86	AMD	Corelis	
ARM7/9/10	ARM Atmel Outros	Lauterbach; Aiji; Ashling; EPI; Abatron; American Arium	CPU Cores
Celeron	Intel	American Arium	
Coldfire	Motorola	Avocet; Abatron	
ERC32	Atmel	Spacebel	
H8S	Hitachi	Lauterbach	CPU Cores
Intel i960	Intel	Corelis	CPU Cores para imagem e *networking*
Itanium	Intel	American Arium	
MIPS	AMD Broadcom IDT MIPS NEC Philips PMC Toshiba Wintegra	Corelis; Lauterbach; Abatron; Ashling; EPI	CPU Cores
Pentium	Intel	American Arium	
Pentium 4	Intel	American Arium	
Pentium II/III	Intel	American Arium	
Pentium Pro	Intel	American Arium	
PowerPC 400	IBM Motorola	IBM; Corelis; Avocet; Abatron	
PowerPC 500 (MPC5XX)	Motorola	Ashling; Avocet	CPU cores
PowerPC 600	IBM Motorola	IBM; Corelis; Lauterbach; Abatron	
PowerPC 700	IBM Motorola	IBM; Corelis; Lauterbach; Abatron	
Xeon	Intel	American Arium	
Xscale	Intel	Lauterbach; Abatron; American Arium; EPI	Destinado a uso em redes e ambientes *wireless*

As funcionalidades disponibilizadas via BST por cada fabricante vão desde a implementação mínima do disposto na norma até à inclusão de circuitos de teste e depuração razoavelmente complexos. É já frequente em processadores recentes que estes apresentem um elemento deste tipo com funcionalidades avançadas, como controlo de execução, inserção de pontos de paragem por condição (*breakpoints*) por hardware e acesso registos internos. Este tipo de tecnologia é também utilizado para aceder aos periféricos dos processadores. Por exemplo é possível aceder à memória externa, podendo ser feita a instalação ou modificação de software e correcção ou inserção de erros.
No caso do processador ERC32 da Atmel, por exemplo, é possível aceder a um OCD (O*n-Chip Debugger*) exclusivamente através de um porto de acesso ao teste (*Test Access Port*, TAP) que permite parar, arrancar e

reinicializar o processador, aceder a praticamente todos os registos internos para operações de leitura e escrita bem como a já referida inserção de *breakpoints*.
Torna-se assim possível desenvolver emuladores completos para processadores com este tipo de estrutura. Este tipo de ferramentas existe já comercialmente e são utilizadas em teste e depuração, desenvolvimento de software e de hardware.

4. Injecção defalhas por varrimento periférico

Cerca de dois anos depois da publicação da norma IEEE 1149.1, Wilcox e Hjartarson apresentam, na forma de patente, a primeira proposta conhecida de exploração da tecnologia de varrimento periférico para apoio à actividade de injecção de falhas [12]. Nos anos seguintes Chau [13], Nadeau-Dostie, Hulvershorn e Adham [14] e Ke [15] publicam trabalhos no mesmo âmbito. Em [14] refere-se mesmo um caso real de aplicação da proposta de Wilcox et. al em comutadores de telecomunicações da Northern Telecom.
Os estudos citados propõem-se ultrapassar uma das limitações mais problemáticas que a infra-estrutura de teste definida pela norma IEEE 1149.1 levanta quando posta ao serviço da injecção de falhas: a incapacidade de endereçar individualmente cada uma das células do registo de varrimento periférico. De facto esta lacuna funcional levanta sérias dificuldades ao controle da localização das falhas injectadas. As formulações realizadas sugerem células de varrimento modificadas, novas instruções de teste, novos sinais de controlo ou abordagens híbridas. Apesar de pretenderem satisfazer as necessidades específicas da injecção de falhas, estas propostas pautam-se por algum esforço de preservação da filosofia subjacente à própria norma, apresentando-se maioritariamente como extensões conformes da mesma. Na Tabela 2 resumem-se de forma comparada as diversas abordagens.

Tabela 2: Características das células modificadas

Abordagem	Falha			Implementação	
	Local	Tipo	Duração (TCKs)[1]	Lógica adicional	Conformidade com a norma
IEEE 1149.1 – 1990	Componente	*stuck-at* 0/1/Z	[2.5, ∞[	[referência]	[referência]
Wilcox et. al – 1992	Célula	*stuck-at* 0/1	[2.5, ∞[	Elevada	Média
Chau –1994	Célula	*stuck-at* 0/1	[2.5, ∞[	Baixa	Média
Dostie et. al – 1995	Célula	*stuck-at* 0/1	[2.5, ∞[	Elevada	Média
Ke – 1996	Célula	*stuck-at* 0/1/Z	[1.5, ∞[	Média	Elevada

Nenhuma das propostas prevê ou discute qualquer processo de sincronização entre a operação da lógica de teste e a operação da lógica de sistema. Esta lacuna verifica-se tanto ao nível do momento de injecção, como ao nível da duração da falha. A ausência de mecanismos de sincronização pode ajudar a explicar uma omissão mais flagrante: a inexistência de qualquer suporte para falhas do tipo *bit-flip*. Como a literatura sobejamente refere (e.g. [16]), trata-se de uma das falhas mais representativas no domínio dos sistemas digitais.
Em 1998 Folkesson, Svensson e Karlsson desenvolveram um injector que explora a infra-estrutura IEEE 1149.1 inalterada para testar e validar os mecanismos de detecção de erros do processador Thor da Saab Ericsson Space AB [17]. Os princípios de operação explorados encontram-se actualmente integrados no GOOFI, uma ferramenta que pretende oferecer acesso uniforme à exploração de um leque de técnicas de injecção distintas [3].
Ao contrário dos trabalhos citados anteriormente, que focavam a injecção de falhas ao nível dos pinos, este considera-a ao nível dos elementos de estado internos do alvo. A técnica descrita, denominada pelos autores de SCIFI (*Scan Chain Implemented Fault Injection*), explora o facto de no processador considerado os mecanismos de depuração (OCD) se encontrarem integrados na própria infra-estrutura de teste IEEE 1149.1. Através do TAP começa-se por definir o momento de injecção da falha programando os registos de paragem condicional. De seguida o processador é posto em modo de

[1] Relógio de teste (*Test Clock*; TCK)

execução igualmente através do TAP. Quando o alvo alcança o ponto de paragem definido congela a sua execução e notifica o injector através de um pino dedicado ao efeito. Inicia-se então, através do TAP, a consulta do estado interno do processador. Este estado é perturbado exteriormente de acordo com o modelo de falhas definido e conduzido ao interior do alvo, consumando-se a injecção pretendida. Através do TAP retoma-se a execução, em modo passo-a-passo quando se pretende estudar detalhadamente a propagação da falha injectada, ou de forma livre quando está em causa a observação do impacto final da falha.
Esta abordagem, embora impondo uma elevada intrusividade temporal, não requer que as células de varrimento sejam alteradas. A sua aplicação porém encontra-se limitada a sistemas digitais dotados de mecanismos de depuração evoluídos integrados na própria infraestrutura IEEE 1149.1. Relativamente à utilização de um pino marginal ao próprio porto de teste, e apesar desta constituir uma clara violação de um acesso dito "normalizado", não deixa de ser uma solução possível em grande parte dos processadores modernos.

5. Proposta de uma célula optimizada para injecção de falhas

Da secção anterior, nomeadamente da Tabela 2, resultam os requisitos para a definição de uma célula optimizada para a injecção de falhas, nomeadamente:

- o suporte dos tipos de falha mais representativos (*bit-flip* e *stuck-at* 0/1) ao nível individual de cada pino;
- o controlo da duração da injecção (desde 1 ciclo de TCK até infinito);
- o diminuto impacto ao nível da lógica adicional (mantendo-se ao nível da proposta de [13] ou se possível menor);
- e a conformidade com a norma (situando-se ao nível da proposta de [15]).

Dada a complexidade associada e a sua não implementação opta-se por não endereçar a questão do sincronismo com a lógica funcional do sistema-alvo (i.e. do processador).

5.1 Análise de requisitos

A implementação de falhas do tipo *bit-flip*, *stuck-at* 0 (s@0), ou *stuck*-at 1 (s@1), em qualquer pino (de entrada ou de saída), mantendo a transparência de funcionamento para os restantes implica a necessidade de reter para cada célula a seguinte informação:

- Pino em modo de funcionamento normal
- Pino seleccionado para injecção de falha
    - Tipos: bit-flip; s@0; e s@1

Dado que uma célula genérica de varrimento periférico (designada a partir daqui pela forma abreviada de célula BS) possui dois registos (Figura 3: andar de deslocamento - A_D - e andar de retenção - A_R) que codificam no máximo quatro estados ($2^2 = 4$), não é possível reter toda a informação listada anteriormente (cinco situações) no interior da célula. Note-se que esta possibilidade permitiria activar o modo de funcionamento proposto (injecção de falhas) com apenas uma instrução adicional, diminuindo assim a quantidade de lógica suplementar. Não sendo possível, definem-se então duas instruções opcionais: uma designada de *BIT-FLIP* e outra designada de *STUCK@*. Em termos de distinção do modo de funcionamento da célula (normal / injecção de falhas) opta-se por guardar essa informação no A_R, ficando disponível o A_D para reter a informação acerca do estado lógico pretendido ('0' ou '1'), no caso de uma falha do tipo *s@*. Os requisitos identificados determinam assim as alterações a efectuar na arquitectura da célula BS.

5.2 Arquitectura da célula modificada

A modificação principal introduzida na arquitectura da célula BS normal (Figura 3) reside no multiplexador que alimenta a saída paralela (*Parallel Output*, PO) e que permite seleccionar entre a saída do A_R e a entrada paralela (*Parallel Input*, PI) da célula em questão. Os requisitos identificados

anteriormente implicam um aumento do número de entradas do multiplexador de dois para quatro, para permitir a selecção de duas entradas adicionais: uma correspondente à negação da PI (emulando assim uma falha do tipo *bit-flip*), e outra correspondente à saída do A_D (emulando assim uma falha do tipo *s@0*, ou *s@1*, consoante o valor armazenado naquele registo). O acréscimo de duas entradas de dados num multiplexador 2:1 (que passa a 4:1) implica uma entrada de controlo adicional que no nosso caso servirá para distinguir o modo de funcionamento da célula. De acordo com o esquema da célula BS modificada ilustrada na Figura 4, onde se indica a ordem de ligação das entradas de dados do multiplexador 4:1, apresenta-se na Tabela 3 a codificação dos valores a atribuir às correspondentes entradas de controlo, consoante o valor lógico de um conjunto de sinais associados à instrução actual, carregada no registo de instrução (*Instruction Register*, IR), e ao estado actual do controlador do porto TAP.

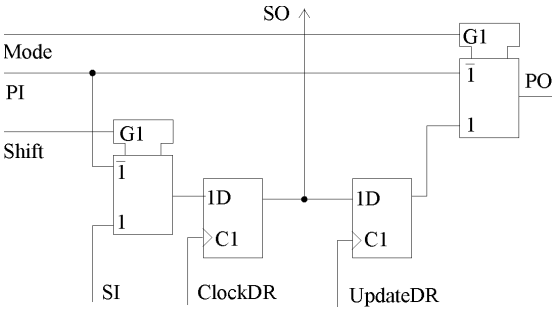


Figura 3: Arquitectura de uma célula BS normal.

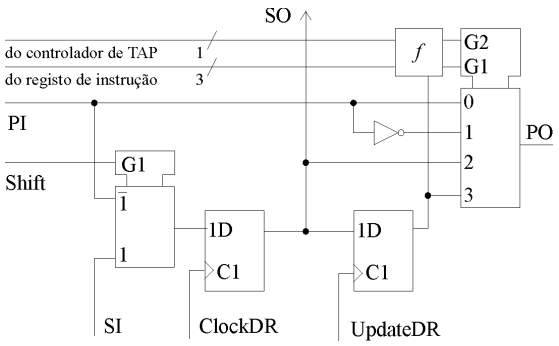


Figura 4: Arquitectura da célula BS modificada

Tabela 3: Tabela de verdade para os sinais de controlo do multiplexador 4:1 de saída

Sinal *Mode* (activo quando a instrução *EXTEST* corresponde à instrução actual)	Sinal *BF* (activo quando a instrução *BIT-FLIP* corresponde à instrução actual)	Sinal *S@* (activo quando a instrução *STUCK@* corresponde à instrução actual)	Sinal *RTI* (activo quando o controlador de TAP se encontra no estado *Run-Test/Idle*)	G1	G2	Observações
0	0	0	X	0	0	PO=PI (normal)
0	0	1	1	0	A_R	PO=!PI (*bit-flip*)
0	1	0	1	A_R	0	PO=A_D (*S@*)
1	X	X	X	1	1	PO=A_R (*extest*)

No nosso caso optou-se por activar o modo de injecção de falhas apenas no estado *Run-Test/Idle*, por duas razões: 1) evitar que durante um processo de deslocamento a saída da célula pudesse comutar de valor lógico (no caso da instrução *STUCK@* estar activa, uma vez que a saída depende do valor existente no A_D); 2) permitir um maior controlo da duração da falha injectada, i.e. a duração mínima da falha corresponde a um único período de TCK, equivalente ao período mínimo de tempo em que se pode manter o controlador de TAP naquele estado.

5.3 Alteração à restante lógica de teste

Para além das alterações introduzidas na célula BS, é necessário criar um novo sinal (*RTI*) activo quando o controlador de TAP se encontra no estado *Run-Test/Idle*, bem como aumentar a lógica de descodificação do IR, em função do suporte das duas novas instruções opcionais (*BIT-FLIP* e *STUCK@*), nomeadamente através dos sinais *BF* e *S@*, activos quando aquelas instruções estiverem carregadas no IR, respectivamente. Não sendo relevante o impacto em termos de lógica adicional, uma vez que se resume a um pequeno número de portas lógicas por componente, deve-se realçar o facto desta alteração aumentar significativamente o número de interligações globais a cada célula BS (de quatro para sete), o que implica uma área acrescida. As alterações introduzidas na geração dos sinais ClockDR e UpdateDR podem ser desprezadas em função do argumento anterior, uma vez que já ligavam a todos os ciclos BS, sendo a lógica adicional mínima.

5.4 Sequência de operações para injecção de falhas

Apresenta-se em seguida a sequência de operações, a efectuar ao nível do controlo externo do TAP, para injectar uma falha do tipo *bit-flip* num ou vários pinos de saída de um componente com uma infraestrutura que suporte a instrução opcional *BIT-FLIP*:

- Colocar o controlador de TAP no estado *Shift-IR*
- Deslocar a instrução *PRELOAD*[2]
- Colocar o controlador de TAP no estado *Shift-DR*
- Colocar um valor lógico alto ('1') nas células BS dos pinos onde se pretendem injectar as falhas.
- Colocar o controlador de TAP no estado *Shift-IR*
- Deslocar a instrução *BIT-FLIP*
- Colocar o controlador de TAP no estado *Run-Test/Idle*. Enquanto o controlador se mantiver neste estado [bastando para tal que se mantenha a linha de selecção de modo de teste (*Test Mode Select*, TMS) ao nível lógico baixo '0'] os valores presentes nos pinos de saída associados às células onde se colocou um '1' no A_R serão o inverso dos valores ditos "normais"

A lista seguinte descreve a sequência de operações para injectar uma falha do tipo *stuck@* num ou vários pinos de saída do mesmo componente:

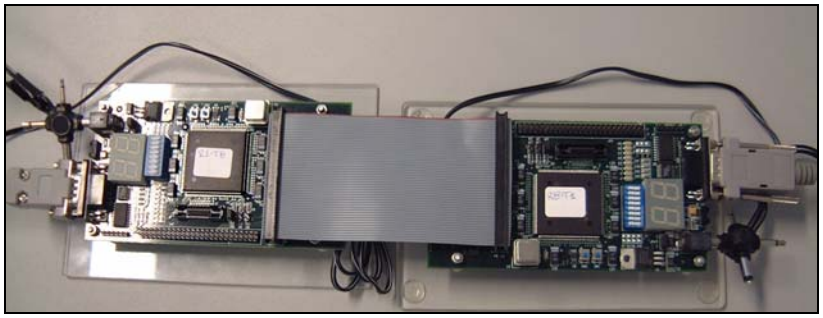
- Colocar o controlador de TAP no estado *Shift-IR*
- Deslocar a instrução *PRELOAD*
- Colocar o controlador de TAP no estado *Shift-DR*
- Colocar um valor lógico alto ('1') nas células BS dos pinos onde se pretendem injectar as falhas.
- Colocar o controlador de TAP no estado *Shift-IR*
- Deslocar a instrução *SAMPLE*
- Colocar o controlador de TAP no estado *Shift-DR*
- Colocar um valor lógico alto ('1') nas células BS dos pinos onde se pretendem injectar falhas do tipo *s@1*, e um valor lógico baixo ('0') nas células dos pinos onde se pretendem injectar falhas do tipo *s@0*. Note-se que apenas as células que possuírem um valor lógico alto no A_R é que estarão aptas a injectar falhas (conforme definido através da acção de *PRELOAD*)
- Colocar o controlador de TAP no estado *Shift-IR* (ao passar no estado *Update-DR*, não serão destruídos os valores presentes no A_R, uma vez que a definição da instrução *SAMPLE* não contempla esta acção)
- Deslocar a instrução *STUCK@*
- Colocar o controlador de TAP no estado *Run-Test/Idle*. Enquanto o controlador se mantiver neste estado, os valores presentes nos pinos de saída associados às células onde se colocou um '1' no A_R serão iguais aos valores existentes no A_D associado

5.5 Implementação e experimentação da infraestrutura modificada

A solução proposta foi implementada numa FPGA (*Field Programmable Gate Array*), da família Virtex™-E da Xilinx, existente numa carta de demonstração de baixo custo [18]. Para o projecto do circuito utilizou-se o sistema de desenvolvimento WebPACK™, disponível gratuitamente através do portal da Xilinx [19]. O projecto seguiu o fluxo clássico (descrição em VHDL + esquemático de topo, simulação funcional, síntese, *place&route*, simulação temporal, e finalmente programação), tendo-se no final efectuado alguns procedimentos básicos de verificação, utilizando uma ferramenta computacional que permite controlar cadeias de varrimento periférico, através da Porta Paralela de um qualquer computador pessoal [20]. Em face dos resultados obtidos (concordância com a simulação), optou-se por definir e implementar um sistema de demonstração que evidenciasse o processo de injecção de falhas. A Figura 5 representa esse sistema, baseado em duas cartas onde se implementaram duas unidades de comunicação via RS-232, uma em cada carta, rodeadas de uma infraestrutura

[2] A revisão da norma efectuada em 2001 [1] permite a separação da antiga instrução *SAMPLE/PRELOAD*] em duas: uma corresponde à função de amostragem (*SAMPLE*) e outra à função de pré-carregamento (*PRELOAD*). Em termos de implementação, esta possibilidade implica apenas uma ligeira alteração na geração do sinal UpdateDR.

modificada. Numa das cartas implementou-se a unidade de recepção de uma UART (*Universal Asynchronous Receiver Transmitter*) e na outra implementou-se a unidade de envio, ligando-se ambas através de um barramento de dados com oito bits (mais uma linha de controlo / sincronismo), onde são colocados os códigos ASCII dos caracteres enviados pela porta série de um PC ligado à primeira carta. A segunda carta lê esses mesmos códigos, enviando-os, em formato série para um segundo PC. Uma aplicação a correr no primeiro PC envia caracteres, ciclicamente pela porta série, enquanto outra aplicação a correr no segundo PC, recebe esses caracteres. A injecção de uma falha é efectuada através da aplicação JTAGer, verificando-se no final a corrupção do(s) caracter(es) recebido(s), enquanto a falta esteve activa, por simples comparação entre o código ASCII do carácter enviado e o código ASCII do carácter recebido. Através de um esquema de encapsulamento do carácter enviado (envio prévio do código '00000000' e envio posterior do código '11111111') é possível determinar, no destino, qual a falha injectada, validando-se assim a solução implementada.



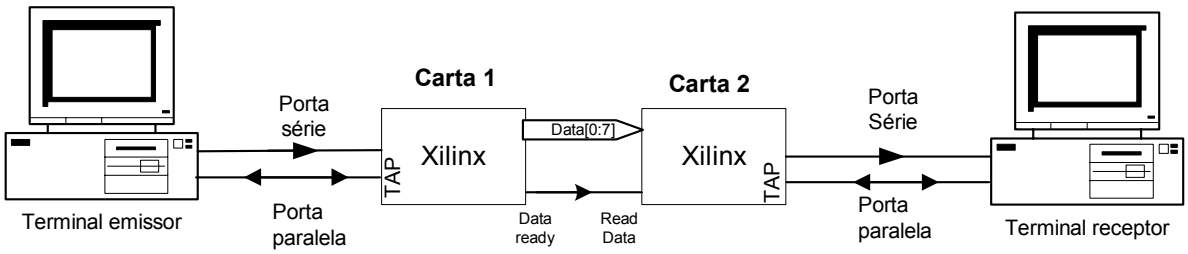


Figura 5: Arquitectura e imagem do sistema de demonstração

5.6 Análise do impacto (atraso e lógica adicional)

O impacto associado à implementação das duas novas instruções opcionais é analisado segundo dois eixos e dois níveis: os eixos correspondem à localização das alterações – célula BS e restante lógica de teste – e os níveis correspondem ao atraso introduzido e quantidade de lógica / área de ligação adicional. Na célula BS modificada efectuaram-se duas alterações: a passagem do multiplexador 2:1 para um multiplexador 4:1 e a consequente geração dos sinais de controlo para este último. Em termos de portas lógicas, e supondo que uma porta tipo AND, OR, NAND, ou outra qualquer que implemente uma função lógica de duas entradas, corresponde a uma unidade, a quantidade de lógica existente numa célula BS normal corresponde a 18 unidades, conforme descriminado na 4. A passagem de um multiplexador 2:1 para um 4:1 corresponde à multiplicação da quantidade de lógica existente naquele, por um factor de 3 (um mux4:1 pode ser implementado à custa de três mux2:1). Em relação à lógica de controlo discrimina-se em seguida a sua composição de forma a verificar qual o número de portas lógicas adicionais (em comparação com a simples ligação do sinal *Mode* na célula "normal" ilustrada na Figura 3).

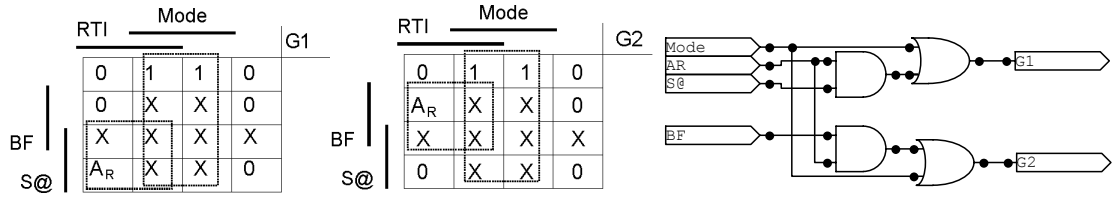


Figura 6: Funções lógicas dos sinais de controlo do multiplexador 4:1 da célula BS modificada

A Figura 6 indica um total de quatro portas para implementar a lógica de controlo. Reunindo estes dados na Tabela 4, chega-se à conclusão que ao nível da quantidade de lógica adicional se obtém um acréscimo de cerca de 55%, por célula. Relativamente ao nível temporal, existe uma duplicação do tempo de atraso, em consequência da passagem do multiplexador de saída de tipo 2:1 para 4:1, independentemente do modo de funcionamento da célula.

Tabela 4: Quantidade de lógica existente em cada tipo de célula BS

Célula	Função / bloco	Nº portas lógicas	Total	Observações
BS normal	Mux2:1 (entrada)	3	18	Existe um acréscimo de 55%, tendo como base de comparação o nº de portas lógicas existentes numa célula BS normal
	A_D	6		
	A_R	6		
	Mux2:1 (saída)	3		
BS modificada	Mux2:1 (entrada)	3	28	
	A_D	6		
	A_R	6		
	Mux4:1 (saída)	9		
	Lógica de controlo	4		

A restante lógica de teste sofre um ligeiro acréscimo em função da lógica de descodificação de duas novas instruções opcionais e de um sinal activo no estado *Run-Test/Idle* do controlador de TAP. O impacto principal reside contudo na área ocupada pelas interligações, dado que estes sinais têm que alimentar todas as células BS modificadas. Refira-se ainda que a solução proposta não implica qualquer violação à norma, uma vez que o modo de injecção de falhas apenas é activado através das novas instruções opcionais, mantendo-se inalterado o modo de funcionamento descrito pelas instruções obrigatórias definidas na norma. Pode-se assim afirmar que a conformidade é elevada.

6. Conclusão

A utilização da infraestrutura BST ara apoio à injecção de falhas surge como uma extensão natural aos exemplos já existentes de reutilização daquela infraestrutura de teste para apoio ao projecto de sistemas baseados em processadores (e.g. emulação / depuração). Da análise efectuada às soluções existentes é possível apontar algumas lacunas bem como a fala de uma metodologia estruturada que defina com clareza a forma de utilização da infraestrutura BST para efeitos de injecção de falhas. A proposta de uma célula optimizada, apresentada neste artigo, visa colmatar algumas das lacunas identificadas. A metodologia estruturada de utilização da infraestrutura BST para injecção de falhas é objecto de um projecto de investigação [21], actualmente em curso, que reúne parceiros académicos e industriais, e que inclui uma componente de demonstração baseada no processador ERC32 da Atmel, presentemente utilizado pela ESA (*European Space Agency*) nas suas missões no espaço.

7. Agradecimentos

Parte do trabalho foi financiando pela AdI (Agência de Inovação) através do contrato ADI-POSI-BSCAN4FI. A célula modificada foi implementada no âmbito da disciplina de Projecto/Estágio do Curso de Engenharia Electrotécnica – Electrónica e Computadores do ISEP, pelos alunos Pedro Moreira e Hugo Gonçalves.

8. Referências

[1] IEEE Std 1149.1-1990 IEEE Standard Test Access Port and Boundary Scan Architecture. IEEE Stds Board, 1990. Revised 1993 (1149.1-1993). Revised 2001 (1149.1-2001)

[2] Lauterbach, http://www.lauterbach.com

[3] Aidemark, J., Vinter, J., Folkesson, P. e Karlsson, J., GOOFI : Generic object-oriented fault injection tool, in "DSN-2001", Goteborg, Sweden, 2001, pp. 83–88.

[4] Voas, J. M. e McGraw, G., Software Fault Injection: inoculating programs against errors, 1998, John Wiley and Sons, Inc.

[5] Arlat, J., Crouzet, Y. e Laprie, J. C., Fault injection for dependability validation of fault-tolerant computing systems, in "FTCS-19", IEEE Computer Society, Chicago IL, 1989, pp. 348–355.

[6] Abramovici, M., Breuer, M. A. e Friedman, A. D., Digital System Testing and Testable Design, 1990, Computer Science Press, New York.

[7] Avižienis, A, "Toward systematic design of fault-tolerant systems", IEEE Computer 18, 51–58, 1997.

[8] O. Gunneflo, J. Karlsson, J. Torin, "Evaluation of Error Detection Schemes Using Fault-Injection by Heavy-ion Radiation", Proc.19th Int'l Symposium on Fault - Tolerant Computing, IEEE CS Press, Los Alamitos, Calif. 1989, pp. 340-347.

[9] Madeira, H., Rela, M., Moreira, F. e Silva, J., RIFLE: A general purpose pin-level fault injector, in "EDCC-1", Vol. 852 of Lecture Notes on Computer Science, Springer-Verlag, Berlin-Germany, pp. 199–216, 1994.

[10] Carreira, J., Madeira, H. e Silva, J. G., "Xception: A technique for the experimental evaluation of dependability in modern computers", IEEE Transactions on Software Engineering 24(2), 125–136, 1998.

[11] Koopman, P., Siewiorek, D. e DeVale, K., Automated robustness testing of off-the-shelf software components, in "FTCS-28", Munich, Germany, 1998, pp. 230–239.

[12] Wilcox, P., Hjartarson, J. e Hum, R., "Software verification by fault insertion", U.S patent no. 5,130,988, 1992.

[13] Chau, S., Fault injection boundary scan design for verification of fault tolerant systems, in "ITC-94", Washington D.C., 1994, pp. 677–681.

[14] Nadeau-Dostie, B., Hulvershorn, H. e Adham, S. M. I., A new hardware fault insertion scheme for system diagnostics verification, in "ITC-95", Washington D.C., 1995, pp. 994–1002.

[15] Ke, W., Hybrid pin control using boundary-scan and its applications, in "ATS-96", Hsinchu, Taiwan, 1996, pp. 44–49.

[16] Lala, P. K., Fault Tolerant and Fault Testable Hardware design, Prentice-Hall, London, 1985.

[17] Folkesson, P. M., Svensson, S. e Karlsson, J., A comparison of simulation based and scan chain implemented fault injection, in "FTCS-28", Munich, Germany, 1998, pp. 284–293.

[18] Avnet, http://www.avnet.com

[19] Xilinx, http://www.xilinx.com

[20] Tiago Gasiba, JTAGER, Trabalho desenvolvido na cadeira de PTSE, FEUP, 2002

[21] BSCAN4FI, http://www.bscan4fi.com/.