Alteração de Circuitos para Tolerância à Falhas

Prof. Maurício Acconcia Dias

Anteriormente discutimos as técnicas de solução de problemas relativos a falhas em projeto de circuitos como, por exemplo, técnicas de replicação de unidades de execução. Porém, esta área de pesquisa possui uma outra abordagem possível.

Ao desenvolver um sistema microcontrolado, podendo ser considerado um co-projeto de hardware e software, seria interessante poder testar as falhas que viriam a ocorrer ao longo do tempo de vida útil do sistema. Por padrão a engenharia de testes trata de formas de encontrar problemas de desenvolvimento tanto no software, quanto no hardware (co-síntese – teste de hardware e software simultaneamente). Para isso são desenvolvidos métodos de listagem de testes necessários para validação de determinada funcionalidade do software/hardware (normalmente descrita no documento de requisitos).

Por exemplo, imagine que você possui 5 parâmetros que podem ser modificados em um algoritmo genético: taxa de crossover, taxa de mutação, número de indivíduos, número de gerações e taxa de predação. Considere também que você pré-define (já que os valores são reais e podem assumir infinitas possibilidades) um conjunto de 4 valores para cada uma das 5 variáveis. Quantos experimentos são necessários para validar seu trabalho?

Montando a matriz de experimentos (errada):

	P11	P12	P13	P14														P54
V1	0.5	0.4	0.2	0.8														
V2																		
V3																		
V4																		
V5																		

Portanto, para validar seu experimento você terá que realizar 5 (variáveis) x 20 (possíveis modificações de parâmetros) certo? Errado, o número de experimentos é uma combinação dos vinte possíveis parâmetros, 5 a 5. Lembrando a combinação:

$$C = \frac{20!}{5!\,(20-5)!} = \frac{2.432902e+18}{120 * (15!)} = 15504\ testes$$

E como o algoritmo genético trabalha com números aleatórios, uma execução não é relevante. Sendo assim cada uma das versões deveria ser executada ao menos 10 vezes com cálculo de média e desvio padrão do resultado. Finalmente temos o número, portanto de 155040 execuções do código.

Imagine então um sistema de uma empresa que possui diversos módulos com milhares de variáveis que podem gerar combinações de testes. O que fazer? Existem neste caso duas alternativas (no

mínimo): a primeira delas é verificar de acordo com os requisitos quais as funcionalidades vitais para o sistema e testá-las do modo mais completo possível; a segunda opção é automatizar ao máximo os testes (incluindo a utilização de algoritmos de IA) para que a cobertura do espaço de busca de possiblidades seja a maior possível.

1. Teste de Co-Projetos de Hardware/Software

Ao se desenvolver um sistema composto por hardware e software a questão fica ainda mais complexa. O número de testes a ser executado aumenta exponencialmente ao se incluir dispositivos de hardware sua estrutura de conexão. Portanto, se só o teste de software já parecia impossível o teste de hardware comprova a teoria. Além de buscar as principais funcionalidades do sistema nos requisitos, há outra maneira de tentar garantir um funcionamento correto do sistema pelo tempo de vida esperado. Este método é chamado de Injeção de Falhas (ou em inglês *Fault Injection*).

Definição: “A injeção de falhas é uma técnica que consiste em como sistemas se comportam em situações extremas de stress considerando diferentes perspectivas”.

O objetivo da injeção de falhas é aumentar a cobertura feita pelos testes no espaço de busca de possibilidades. Este método tem base em resultados experimentais e este fato o torna mais válido (no sentido de proximidade com o sistema real) em comparação com os métodos estatísticos normalmente executados.

Uma falha, como visto anteriormente, tem sua propagação passível de modelagem em um ciclo. Inicialmente a falha ocorre (ou é injetada no sistema), em seguida um erro é causado; este erro pode gerar uma reação em cadeia causando diversos outros erros e estes novos erros podem ser considerados falhas para a análise. Este ciclo é chamado de ciclo falha-erro-falha.

1.1 Técnicas para injeção de falhas em software

Basicamente existem duas maneiras de injetar falhas em software: em tempo de compilação e em tempo de execução. Em tempo de compilação existem técnicas como:

- Teste de mutação: falhas propositais são injetadas no código gerado a partir de um sistema. Exemplo: em uma linha que possui o código b = a * c; é injetada a falha para a operação ser modificada e se tornar b = a / c;

Seria possível, portanto, reprogramar um compilador para inserir falhas no código ao realizar suas análises (léxica, sintática e semântica). Porém, estas técnicas são complexas e demandam conhecimento de ferramentas de compilação. No caso de falhas de tempo de execução é possível inserir funções que alteram valores específicos, a saber:

- Corrupção de espaços de memória – injeção de falhas no mapeamento da memória

- Corrupção de sistema operacional – inserção de chamadas de sistema não esperadas com erros em parâmetros
- Corrupção de comunicação – mudanças em bits de pacotes, reordenar pacotes de forma a interferir na remontagem da informação.
- Corrupção de protocolo – Fuzzing: geração de entradas invalidas para o protocolo em execução

1.2 Técnicas de injeção de falhas em hardware

No início as técnicas utilizadas eram feitas na etapa de prototipação, atualmente os circuitos finais também podem possuir unidades de tolerância à falhas. A ideia principal é a mudança de voltagem em partes do circuito, aumento ou diminuição significativos da temperatura, bombardeamento de radiação na placa, dentre outras técnicas. Devido ao alto número de possibilidades foram criados três parâmetros comuns para fazer isso de forma eficiente:

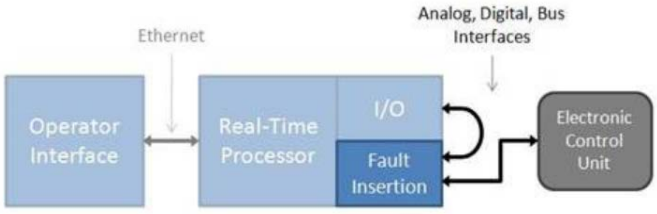
- Qual tipo de falha deve ser inserida e em qual intensidade?
  - Atraso, valores de variáveis fixos, ignorar chamadas de função, ruído nos bits.
- Quando e como a falha deve ser ativada?
  - Por exemplo, definir o momento que a falha será ativada e/ou as condições
- Onde a falha deve ocorrer?
  - Pode ocorrer na comunicação (barramentos/rede), dentro de sistemas e subsistemas

Apesar das definições parecerem simples, a ideia é que um sistema muito complexo terá um número proibitivo de casos de teste considerando os três parâmetros apresentados. Neste caso é possível utilizar algumas técnicas para auxiliar na escolha de quais falhas injetar.

- Análise sensitiva – os sinais elétricos mais importantes do sistema são definidos e as falhas aplicas neles. Neste caso, com foco nestes sinais é possível testar várias partes importantes do sistema em si.
- Reinforcement Learning – algoritmos de aprendizado por reforço podem ser utilizados para explorar o espaço de busca de forma eficiente.

Para uma lista exaustiva de ferramentas comerciais e não comerciais de injeção de falhas consultar (links da wikipedia). Segundo (NI, 2020) existe uma configuração típica para a unidade de injeção de falhas como mostra a Figura 1.

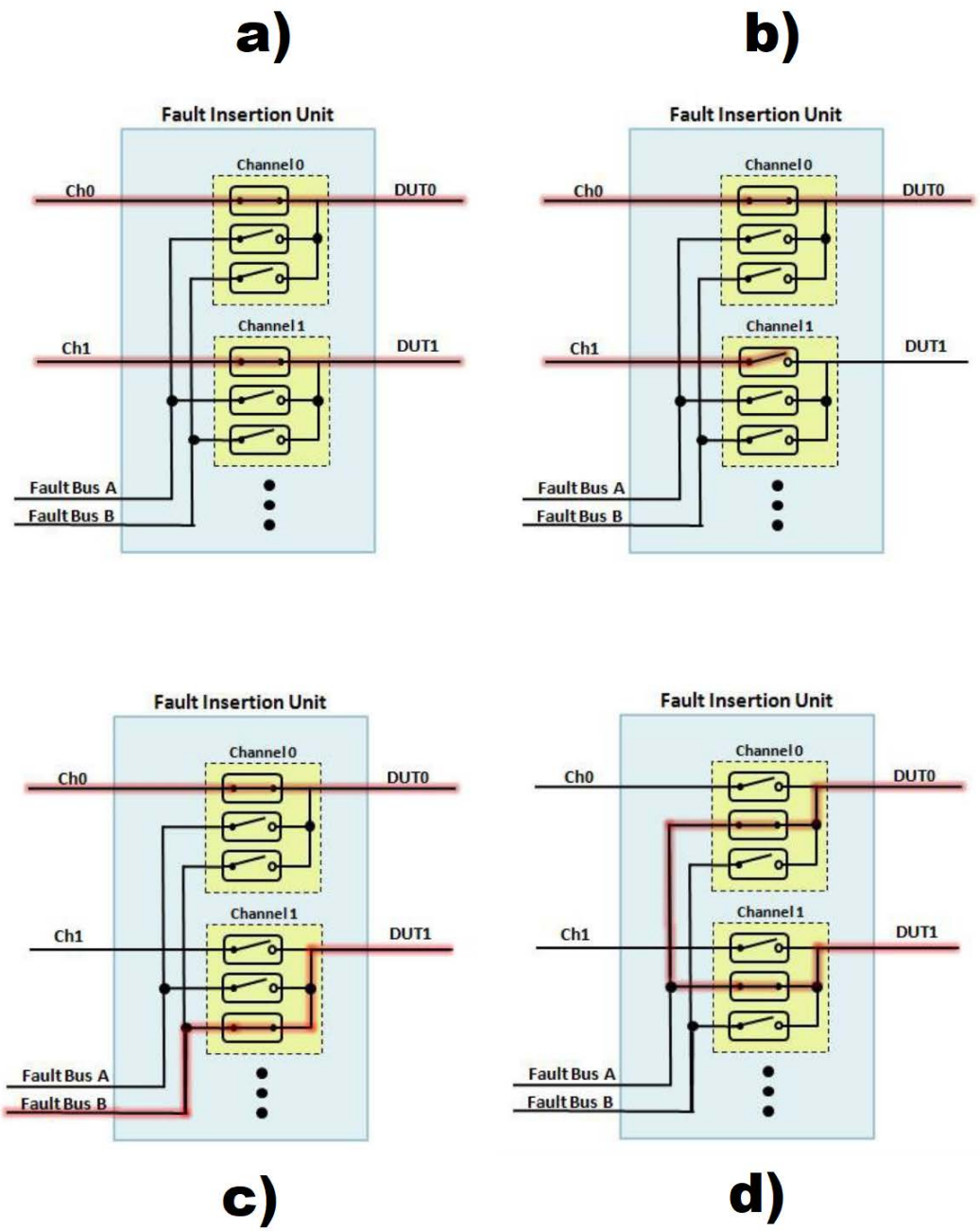
Figura 1. Posicionamento típico da unidade em arquiteturas de hardware



Fonte: -----

Ao se desenvolver um sistema de inserção de falhas é possível criar barramentos específicos para o tratamento da inserção de falhas (Figura 2 – c) e d)). Neste caso, para cada tipo de falha desenvolvida ou necessária um barramento é adicionado na transmissão de informações padrão do sistema. Curtos (Figura 2 – d)) e chaveamentos (Figura 2 – b)) podem também ser incluídos para formar o “banco” de falhas possíveis.

Figura 2. Tipos de inserção de falhas em hardware.



Fonte: NI (2020)

2 Detecção de tratamento de falhas

Para detectar as falhas em hardware é possível adotar técnicas já desenvolvidas e comprovadamente eficientes. Esta etapa é necessária pois não é possível tratar uma falha que não foi identificada corretamente. Dentre os métodos existentes é possível citar (HELIX):

- Tratamento de sanidade – as unidades de hardware mandam mensagens periódicas de que não detectaram problemas. No caso de falha, esta mensagem não irá chegar ao controlador do hardware que automaticamente saberá que existe um problema.
- Tratamento utilizando watchdog – ao invés da mensagem para o monitor, um watchdog é configurado e reinicia o sistema caso não receba a confirmação das unidades de hardware.
- Protocolo de falhas – composto por mensagens específicas que são enviadas às outras unidades de hardware que estão em comunicação com a unidade que falhou.
- Diagnóstico em tempo de execução – as unidades de hardware são projetadas para conter auto checagem em tempo de execução. Em caso de falha o protocolo de falhas pode ser acionado entre o bloco e o monitor.
- Estruturas de dados para controle de falhas – existem falha que podem ocorrer e não prejudicam diretamente o sistema. Sendo assim é possível criar estruturas que de alguma forma contar as falhas e a partir de um determinado número indicam a falha.
    - Problemas de interrupção – objeto não encontrado
    - Falhas “falsas” – no ciclo de vida que será apresentado abaixo, quando uma unidade falha ela é marcada como suspeita. Neste caso cabe uma contagem de unidades suspeitas antes de se considerar as falhas. Aqui também entram os casos de falhas causadas por serviços terminados por motivos externos (exemplo – aumento brusco de tensão (raio)).
- Isolamento de falha – caso uma unidade de hardware tenha “caído”, vários sinais de falha serão emitidos dela e de outras envolvidas em comunicação. É necessário que projete o sistema de modo que seja possível identificar qual a unidade que falhou e isolar o problema. (Ex – protocolo de falha pode conter a informação na mensagem).

Uma das maneiras de se evitar as falhas é a redundância das unidades do sistema. Segundo Helix (2019) existe um ciclo de vida de redundância que pode ser analisado. O caso apresentado abaixo é de um sistema que possui uma cópia dele no hardware, sendo assim o ciclo de vida se comporta da seguinte maneira:

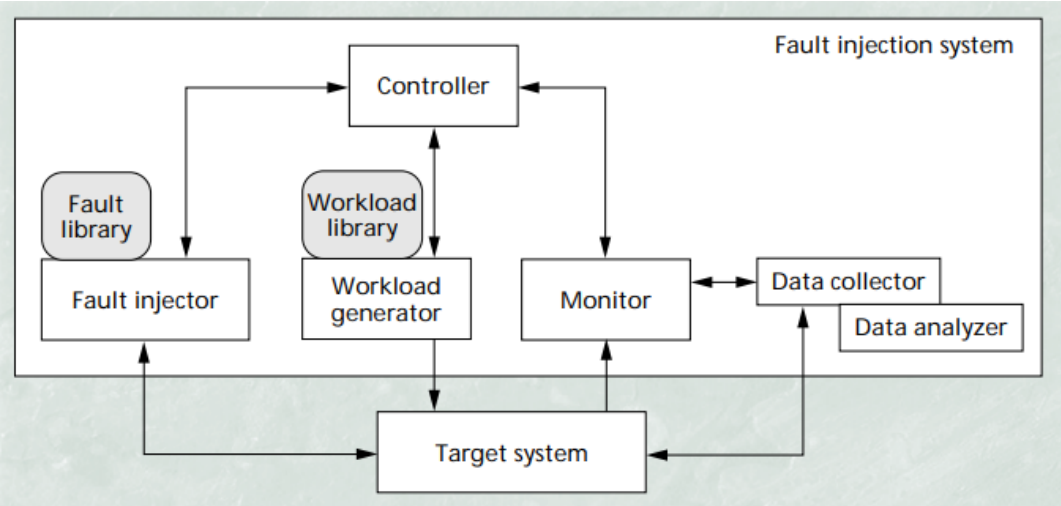
1. Assume-se que é executada a cópia 0 e a cópia 1 fica em standby. Retomada da configuração inicial dos avisos de falha.
2. Quando a cópia 0 falha, a cópia 1 detecta a falha utilizando métodos pré-definidos.
3. Neste momento, a cópia 1 toma imediatamente o lugar da cópia 0 que é então classificada como suspeita.

4 Alguma forma de alerta é emitida no sistema para avisar que a redundância não está mais funcional (protocolo de falhas).
5 São realizados diagnósticos na cópia 0 pelo sistema.
6 Se a cópia passar nos testes ela entra em standby e volta-se ao passo 1, caso contrário é marcada como permanentemente com falha e outra notificação é emitida.
7 Um operador pode substituir a cópia 0 por outra para retomar a redundância.
8 Caso seja feita a substituição, a nova cópia colocada na posição da cópia 0 é analisada.
9 Volta ao passo 6.

3 Elementos básicos de um sistema de injeção de falhas.

Segundo Hsueh, Tsai e Iyer (1997) um sistema de injeção de falhas possui elementos básicos utilizados para a operação da inserção em si, além da coleta de dados. Estes elementos são apresentados na Figura 3.

Figura 3. Componentes básicos de um sistema de injeção de falhas.



Fonte: Hsueh, Tsai e Iyer (1997).

Como é possível visualizar na Figura 3, os elementos básicos incluem:

- Controlador do sistema de injeção de falhas que irá executar corretamente a inserção, execução e obtenção de dados.
- Unidade injetora de falhas que possui um “banco” de falhas possíveis.
- Gerador de cargas de trabalho que irá simular uma execução do sistema para o teste.
- Monitor que é responsável por receber informações do sistema alvo da injeção de falhas. As informações do sistema são coletadas para futura análise de dados ou então modificar algo no controle

- A unidade de coleta de dados irá analisar o comportamento do sistema perante o cenário de teste abordado.

Referências:

Moradi, Mehrdad; Van Acker, Bert; Vanherpen, Ken; Denil, Joachim (2019). Chamberlain, Roger; Taha, Walid; Törngren, Martin (eds.). "Model-Implemented Hybrid Fault Injection for Simulink (Tool Demonstrations)". Cyber Physical Systems. Model-Based Design. Lecture Notes in Computer Science. Springer International Publishing. 11615: 71–90. doi:10.1007/978-3-030-23703-5_4. ISBN 9783030237035.

J. Voas, "Fault Injection for the Masses," Computer, vol. 30, pp. 129–130, 1997.

J. V. Carreira, D. Costa, and S. J. G, "Fault Injection Spot-Checks Computer System Dependability," IEEE Spectrum, pp. 50–55, 1999.

Benso, Alfredo; Prinetto, Paolo, eds. (2003). Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation. Frontiers in Electronic Testing. Springer US. ISBN 978-1-4020-7589-6.

NI. https://www.ni.com/pt-br/innovations/white-papers/09/using-fault-insertion-units--fius--for-electronic-testing.html. 2020.

HSUEH, TSAI e IYER. Fault Injection Techniques and Tools. https://course.ece.cmu.edu/~ece846/docs/faultInjectionSurvey.pdf. 1997

HELIX. Fault Handling Techniques. https://www.eventhelix.com/RealtimeMantra/FaultHandlingTechniques.htm. 2019