JORGE ARTURO CORSO SARMIENTO

PLATAFORMA DE CO-EMULAÇÃO DE FALHAS EM CIRCUITOS
INTEGRADOS

São Paulo

2.010

JORGE ARTURO CORSO SARMIENTO

PLATAFORMA DE CO-EMULAÇÃO DE FALHAS EM CIRCUITOS
INTEGRADOS

Tese apresentada à Escola Politécnica
da Universidade de São Paulo para
obtenção do título de Doutor em
Engenharia Elétrica.

Área de Concentração:

Sistemas Eletrônicos

Orientador:

Professor Titular

Dr. Francisco Javier Ramirez Fernandez

São Paulo

2.010

FICHA CATALOGRÁFICA

Sarmiento, Jorge Arturo Corso

**Plataforma de co-emulação de falhas em circuitos integrados
/ J.A.C. Sarmiento. -- São Paulo, 2010.
161 p.**

**Tese (Doutorado) - Escola Politécnica da Universidade de
São Paulo. Departamento de Engenharia de Sistemas Eletrô-
nicos.**

**1. CAD 2. Circuitos integrados 3. Circuitos FPGA 4.
Arquitetu-
ra reconfigurável I. Universidade de São Paulo. Escola Politéc-
nica. Departamento de Enge-nharia de Sistemas Eletrônicos II. t.**

Dedico este trabalho a minha família
pelo amor e apoio incondicional em
cada momento da minha vida.

Agradecimentos

A Deus por estar sempre comigo, guiando-me e protegendo.

A meu orientador Prof. Dr. Francisco Javier Ramirez Fernandez por sua ajuda valiosa na realização do trabalho, e pela confiança e amizade oferecida, que me fizeram crescer em todos os aspectos da minha vida.

A minha futura esposa Valeria Bahamondes pelo amor incondicional, sua paciência, força moral e de espírito.

Aos meus amigos da Freescale, em especial ao Fabio Silva por sua valiosa contribuição neste trabalho, pelas discussões conceituais e as correções do texto. Também ao meu amigo Fernando Frank quem me introduziu na área da testabilidade e DFT.

Aos Doutores Marcelo Fukui e Edson Lemos Horta, pelas contribuições na qualificação, que ajudaram a definir os rumos do desenvolvimento.

A meus pais Luis Eduardo e Carmen Cecília, meus irmãos Luz Angela, Luis Eduardo e Juan Manuel, e meus sobrinhos Nicolas, Juliana, e Carmencita, aos quais devo o que sou e o que tenho.

À família Ramirez Starikoff, por ter me recebido e tratado como um filho, irmão e amigo.

À família Bahamondes, minha nova família, por todos os momentos compartilhados.

À Doutora Claudia Lemos pelos conselhos e guia.

E a todos os que colaboraram direta ou indiretamente na realização deste trabalho.

Resumo:

Este trabalho apresenta uma plataforma e uma técnica para o melhoramento da eficiência da graduação de falhas “stuck-at” de padrões de teste através do uso de co-emulação de hardware. Os fabricantes de Circuitos Integrados continuamente buscam novas formas de testar seus dispositivos com o intuito de distribuir partes sem defeitos aos seus clientes. “Scan” é uma técnica bem conhecida que consegue alta cobertura de falhas com eficiência. As demandas por novos recursos motivam a criação de sistemas complexos que fazem uso de uma mistura de blocos analógicos e digitais com uma interface de comunicação, difícil de ser coberta pelos padrões de “scan”. Adicionalmente, a lógica que configura o chip para cada um dos diferentes modos de operação, algumas interfaces com circuitos de teste de memória (BIST), divisores ou geradores de “clocks” assíncronos, entre outros, são exemplos de circuitos que se encontram bloqueados em “scan” ou possuem poucos pontos de observação/controle. Este trabalho descreve uma plataforma baseada em FPGA que usa modelos heterogêneos para co-emular blocos digitais, analógicos e de memória para a graduação de padrões em sistemas complexos. Adicionalmente se introduziu quatro tipos de modelos que podem ser usados no FPGA, e os resultados de aplicar a técnica de co-emulação de falhas em alguns circuitos de benchmark incluindo ISCAS89, um conversor análogo digital, portas configuráveis de entrada/saída e um controlador de memória.

Palavras chave: Co-Emulação De Falhas, Simulação De Falhas, Testabilidade, FPGA, Circuitos Integrados, CAD.

Abstract:

A platform and a technique to improve stuck-at fault grading efficiency through the use of hardware co-emulation is presented. IC manufacturers are always seeking for new ways to test their devices in order to deliver parts with zero defects to their customers. Scan is a well known technique that attains high fault coverage results with efficiency. Demands for new features motivate the creation of high complex systems with a mixture of analog and digital blocks with a communication interface that is difficult to cover with scan patterns. In addition, the logic that configures the chip for each of the different test modes, some BIST memory interfaces, asynchronous clock dividers or generators, among others, are examples of circuits that are blocked or have few observation/control points during scan. A FPGA based-platform that uses heterogeneous models to emulate digital, analog and memory blocks for fault grading patterns on complex systems is described. Also introduced in our proposal are four types of models that can be used with FPGAs, and the results of applying our fault co-emulation technique to some benchmark circuits including ISCAS89, ADC, iopads and memory controllers.

Keywords: Fault Co-Emulation, Fault Simulation, Testability, FPGA, Integrated Circuits, CAD.

Lista de ilustrações

Figura 2-1: Exemplo de falha em circuitos integrados ("bridge")	20
Figura 2-2: Arquitetura de Scan com multiplexador	36
Figura 3-1: Processo de simulação de falhas	43
Figura 3-2: Exemplo de simulação de falhas dedutivo	50
Figura 3-3: Exemplo de simulação de falhas concorrente	55
Figura 4-1: Elementos básicos de um FPGA.....	65
Figura 4-2: Diagrama genérico de um bloco lógico reconfigurável	66
Figura 4-3: Técnica de injeção dinâmica de falhas. (a) circuito original, (b) falhas dinâmicas, sinal 'a' SA1 e sinal 'g' SA0 injetadas por G1 e G2.	75
Figura 4-4: (a) Tabela verdade da porta 'E' considerando o valor lógico desconhecido. (b) lógica de trilho dual para a porta 'E'	78
Figura 5-1: Fluxograma da metodologia para co-emulação de falhas	84
Figura 5-2: Hardware da plataforma de co-emulação.....	87
Figura 5-3: Relação de comunicação entre elementos da plataforma de co-emulação	89
Figura 5-4: Diagrama de processos com entradas e saídas	90
Figura 6-1: Exemplo de importância do tipo de sinal 'X'	94
Figura 6-2: Modelos "dual railed logic" para portas simples.....	96
Figura 6-3: Modelo de multiplexador em dual railed logic.....	98
Figura 6-4: Transição de sinal de relógio	99
Figura 6-5: Modelo de "flip-flop" D usando "dual railed logic" desconsiderando 'X' em sinais de controle	100
Figura 6-6: Método de geração de modelo de DRL modificado para lógica combinacional	101
Figura 6-7: Modelo de "dual railed logic" modificado de portas simples	102
Figura 6-8: Modelo de multiplexador usando "dual railed logic" modificado.....	103
Figura 6-9: Modelo de "flip-flop" D usando "dual railed logic" modificado	104
Figura 6-10: Pseudocódigo para a aplicação de modelos à lista de conexão.....	105
Figura 6-11: Modelo de "buffer tri-state" em lógica de quatro níveis.....	109
Figura 6-12: Modelo do concentrador usando lógica de quatro níveis.....	110

Figura 6-13: Modelo de concentrador que considera a força dos sinais de entrada	111
Figura 6-14: Interface entre lógica de quatro níveis e a de três	111
Figura 6-15: Modelo de PADS configuráveis de entrada/saída	112
Figura 6-16: Modelo estrutural do PAD	112
Figura 6-17: Diagrama de blocos de conversor analógico digital	114
Figura 7-1: Elementos para inserção de falhas “stuck-at” em modelo “dual railed logic” modificado	116
Figura 7-2: Pseudocódigo do “script” para inserção de falhas	117
Figura 7-3: Fluxo de geração de padrões de teste	118
Figura 7-4: Diagrama de blocos do “testbench” de geração de vetores de teste	120
Figura 7-5: Fluxograma do programa do processador de co-emulação	123
Figura 8-1: Diagrama de blocos do conversor análogo digital	126
Figura 8-2: ULA com pinos de entrada/saída configuráveis	127
Figura 8-3: Diagrama de blocos de circuito com memória	127

Lista de tabelas

Tabela 3-1: Lista de propagação em simulador de falhas dedutivo	49
Tabela 4-1: Vetores de reconfiguração de falhas para uma LUT que implementa $F=a.b+c.d$	81
Tabela 6-1: Resultado da combinação de sinais em um barramento	92
Tabela 6-2: Codificação de tipos de sinal “dual railed logic”	95
Tabela 6-3: Exemplo de risco fundamental	99
Tabela 6-4: Codificação “Dual railed logic” modificado.....	101
Tabela 6-5: Codificação de lógica de quatro níveis	109
Tabela 7-1: Codificação dos vetores de teste	119
Tabela 8-1: Características dos circuitos de “benchmark” utilizados do ISCAS89..	125
Tabela 8-2: Algumas características do segundo conjunto de circuitos usados para “benchmark”.....	128
Tabela 8-3: Resultados de sínteses e ATPG	129
Tabela 8-4: Resultados de compilação de circuitos para o FPGA usando SRL e DRLM.....	129
Tabela 8-5: Utilização de recursos do FPGA nos modelos SRL e DRLM, com controlador de falhas e circuitos de ativação.....	130
Tabela 8-6: Tamanho de vetores de teste.....	131
Tabela 8-7: Resultados da simulação e co-emulação de falhas	132
Tabela 8-8: Resultados de sínteses e ATPG	134
Tabela 8-9: Resultados de compilação de circuitos para o FPGA usando SRL e DRLM.....	135
Tabela 8-10: Utilização de recursos do FPGA nos modelos SRL e DRLM, com controlador de falhas e circuitos de ativação.....	135
Tabela 8-11: Tamanho de vetores de teste.....	136
Tabela 8-12: Resultados da simulação e co-emulação de falhas	136
Tabela 8-13: Resultado de padrões que exercitam blocos não digitais	137

Lista de abreviaturas e siglas

AC: Alternating Current;

ADC: Conversor Análogo Digital;

ATE: Automated Test Equipement;

ATPG: Automatic Test Pattern Generation

BIST: Built-In Self-Test;

CI: Circuito Integrado

CLK: Clock;

CMOS: Complementary Metal-Oxide-Semiconductor

CPU: Central Processing Unit

CSA: Combination Stuck-At;

D: Dado

DATPG: Deterministic Automatic Tpg;

DC: Direct Current;

DFF: Flip-Flop D

DFT: Design For Testability;

DRL: Dual Railed Logic;

DRLM: Dual Railed Logic Modificado;

DSIM: Differential Fault Simulation;

DUT: Device Under Test, Dispositivo Sob Teste;

FAN: Fanout-Oriented Tpg;

FIFO: First In First Out;

FMOSSIM: Fault Mos Simulation

FPGA: Field Programmable Gate Array;

IDDQ: Supply Current (I_{dd}) In The Quiescent State;

IP: Intellectual Property;

ITRS: International Technology Roadmap For Semiconductors

JTAG: Joint Test Action Group;

LBIST: Logic Bist;

LUTS: Lookup Tables;

MARS: Microprogrammable Accelerator For Rapid Simulations;

MBIST: Memory Bist;

MEMS: Microelectromechanical Systems;

MIF: Memory Initialization File;

MLT: Multi-List Traversal;

MOTIS: Mos Timing Simulator;

MSAF: Multiple Stuck-At Fault;

MTPG: Manual Tpg;

NRZ: Non Return To Zero;

PODEM: Path Oriented Decision Making;

PPSFP: Parallel Pattern Single Fault Propagation;

RATPG: Random Automatic Tpg;

ROM: Read Only Memory

RTL: Register Transfer Level

SA0: Stuck-At-0;

SA1: Stuck-At-1;

SAF: Stuck-At Faults

SDI: Scan Data Input;

SE: Scan Enable;

SIA: Associação Da Indústria De Semicondutores, Semiconductor Industry Association;

SIP: System-In-Package;

SOC: System-On-Chip;

SOCRATES: Structured-Oriented Cost-Reducing Automatic Test Pattern Generation System).

SOF: Stuck-Open Fault;

SPMFP: Single Pattern Multiple Fault Propagation;

SRAM: Static Random Access Memory;

SRL: Single Railed Logic;

SSAF: Single Stuck-At Faults

TPG: Test Pattern Generation;

TRS: International Technology Roadmap For Semiconductors;

UART: Universal Asynchronous Receiver/Transmitter;

ULA: Unidade Lógica E Aritmética;

USB: Universal Serial Bus

WGL: Waveform Generation Language;

Sumário

1	Introdução.....	16
2	Conceitos básicos sobre testabilidade	20
2.1	O paradigma de teste digital	22
2.1.1	Seleção de modelo de falha	23
2.1.2	Geração da lista de falhas	29
2.1.3	Geração de padrões de teste	30
2.1.4	Classificação de falhas	37
2.1.5	Cálculo de cobertura.....	39
3	Simulação de falhas em circuitos integrados	42
3.1	Conceitos de simulação de falhas	42
3.1.1	Simulação de falhas serial	44
3.1.2	Simulação de falhas em paralelo	46
3.1.3	Simulação de falhas dedutiva	47
3.1.4	Simulação de falhas concorrentes	51
3.1.5	Simulação de falhas diferencial:	56
3.1.6	Simulação de falhas hierárquica:	60
4	Emulação de falhas	63
4.1	Emulação de falhas baseada em FPGAs.....	64
4.2	Emulação de falhas baseada em múltiplos-processadores	70
4.3	Estado da arte da emulação de falhas em circuitos integrados.....	71
4.3.1	Emulação de falhas serial.....	73
4.3.2	Método de Cheng-Huang&Dai.....	74
4.3.3	Método para manipulação de lógica de três valores	77
4.3.4	Emulação de falhas usando reconfiguração parcial	79
5	Plataforma de co-emulação de falhas	83

5.1	Metodologia	83
6	Modelamento	91
6.1	Lógica digital	91
6.1.1	“Single railed logic”	91
6.1.2	“Dual railed logic”	94
6.1.3	“Dual railed logic” modificado:.....	101
6.2	Memórias	105
6.3	“Hardblocks”	108
6.3.1	Modelamento estrutural	108
6.3.2	Modelamento comportamental	112
7	Detecção de falhas	115
7.1	Síntese do circuito de injeção de falhas	115
7.2	Geração de vetores de teste	117
7.3	Administração do teste.....	120
8	Análises de resultados da co-emulação de falhas.....	124
8.1	Circuitos de “benchmark”	124
8.2	Metodologia: “Benchmark” ISCAS89.....	128
8.3	Metodologia: “Benchmark” circuitos com “hardbloks”	133
9	Conclusões	138
10	Trabalhos futuros	143
	Bibliografia	144

1 Introdução

O incremento na demanda por qualidade e a redução de custos de manufatura fazem com que os fabricantes de circuitos integrados estejam na busca constante por melhorar seus métodos e processos de teste, visando detectar o maior número de defeitos de fabricação, no menor tempo possível. A demanda por qualidade é conduzida principalmente pela necessidade de sistemas com alta complexidade e sem defeitos, tais como aqueles usados para funções de segurança ou de alto desempenho com baixo consumo de potência. As organizações governamentais, juntamente com a indústria estão constantemente revisando os padrões e as métricas de qualidade, para endurecê-las, com o objetivo de garantir o perfeito funcionamento dos dispositivos oferecidos no mercado [AEC2008].

O fluxo mais comum de desenvolvimento de circuitos integrados segue uma metodologia baseada na conexão de blocos de sinal misto [Kao1996, Laakkonen2009]. Blocos com características especiais de funcionamento (analógicos ou de sinal misto), temporização e/ou potência, podem ser integrados com blocos digitais constituindo um sistema que realiza funções complexas, em pouco tempo. Dado que sua disposição, roteamento e propriedades, não podem ser alteradas durante o fluxo de integração, os blocos com características especiais, anteriormente mencionados, são conhecidos como “hardbloks”. Já outros blocos ou IPs podem ser configurados dependendo das restrições de funcionamento do sistema. Para efeitos deste trabalho considerar-se-ão que os “hardblocks” são circuitos de sinal misto ou analógicos.

“Scan” é uma técnica bem conhecida que ajuda a detectar a maior parte das falhas de um circuito digital, através de estruturas especiais que permitem o controle e observação das falhas, e de padrões gerados por ferramentas automáticas de geração de padrões de teste (“Automatic Test Pattern Generation”, ATPG). Com esta técnica, é possível obter, eficientemente, altos valores de cobertura em circuitos digitais. No entanto, existem algumas partes do circuito que inserem restrições dificultando a detecção das falhas e impactando a cobertura global do circuito. Em resumo, a maioria das restrições se localizam na interface entre blocos analógicos e

digitais, a lógica de teste, e de configuração do modo de operação. Dependendo do número de falhas do circuito restante, estas restrições podem afetar em maior ou menor grau, já que o cálculo da cobertura final é feito a partir da razão entre o número de falhas detectadas e o número total de falhas testáveis. Se o circuito é pequeno e existe um grande número de falhas não detectadas devido às restrições anteriormente mencionadas, a cobertura de falhas do circuito é reduzida e se corre o risco de não atingir os parâmetros de qualidade.

Em geral, existem quatro conjuntos de padrões de teste que são utilizados na fabricação de CIs. O primeiro corresponde aos padrões de “scan” que cobrem a maior parte das falhas em circuitos digitais. O segundo são padrões de teste e configuração de memórias, caso o circuito possua uma ou várias. O terceiro são padrões funcionais da parte digital e analógica onde devem ser verificadas todas as funcionalidades dos circuitos analógicos que constituem o CI, junto com as funções primárias do mesmo. Por final, os padrões paramétricos que verificam se as características AC e DC estão de acordo com a especificação. Acredita-se que em sua maioria, as falhas digitais não cobertas pelo “scan” deveriam ser cobertas pelos padrões funcionais e de memórias. Mas, qual é o verdadeiro aporte destes padrões na cobertura final? Será que os padrões utilizados são de qualidade? Quer dizer, conseguem detectar o maior número de falhas no menor tempo possível?

Para responder estas perguntas poderia utilizar-se qualquer uma das técnicas de simulação de falhas (ver capítulo 3), criando uma lista de falhas e executando a simulação para extrair a informação do número de falhas cobertas por outros padrões, além dos gerados para “scan”. A simulação consiste em empregar técnicas matemáticas em computadores com o propósito de analisar um processo ou operação do mundo real, utilizando algum tipo de modelo. O problema desta abordagem para a graduação de falha é o tempo que toma sua execução, o qual aumenta drasticamente com o tamanho do circuito, o número de falhas analisadas e o tamanho do padrão.

Um emulador consegue imitar o funcionamento do sistema real, obtendo os mesmos resultados. Um emulador replica o processo do sistema, enquanto um simulador o modela através de equações matemáticas ou computacionais. Baseado no conceito de que a emulação de hardware é muitas vezes mais rápida que a simulação, foram

criadas uma série de técnicas que se aproveitam deste fato, para realizar a graduação de padrões de teste de forma mais rápida e eficiente (ver capítulo 4). Estas técnicas utilizam unicamente modelos de circuitos digitais e não consideram a interface entre a parte digital e analógica. A ausência de modelos analógicos dificulta a utilização destas técnicas para a graduação de falhas em circuitos integrados reais.

A co-emulação se refere ao processo de simular modelos de software em um nível de abstração alto, que se comunicam simultaneamente com modelos de “hardware” dentro da mesma execução. Neste trabalho se apresenta uma plataforma para a co-emulação de falhas que combina um grupo de modelos heterogêneos em um FPGA para a graduação de falhas de padrões de teste. Tanto os modelos quanto o método e ferramentas de automação que acompanham foram desenvolvidos neste trabalho.

Os objetivos específicos da tese foram:

- Realizar um estudo do estado da arte das técnicas de simulação e emulação de falhas.
- Projetar uma plataforma de co-emulação para o aprimoramento das técnicas de graduação de falhas de padrões de teste funcionais, de forma a incluir o modelamento de blocos analógicos e/o sinal misto.
- Melhorar a eficiência do processo de graduação de falhas.
- Criar ferramentas que ajudem na automatização do processo de co-emulação.

O restante do documento está organizado da seguinte forma: no capítulo dois se apresentam conceitos básicos de testabilidade de circuitos integrados; no capítulo três e quatro foi feita uma revisão bibliográfica das técnicas de simulação e emulação de falhas, respectivamente; no capítulo cinco se introduz a plataforma de co-emulação e o método; no capítulo seis se apresentam os distintos tipos de modelos que podem ser usados na plataforma de co-emulação; no capítulo sete se detalha o processo geração, extração e codificação dos vetores de teste; no capítulo oito se apresentam os resultados do uso da plataforma de co-emulação e aplicação do método proposto para a graduação de padrões em alguns circuitos de “benchmark”, entre os quais se incluem alguns do ISCAS89, um conversor análogo digital, um controlador de memória e uma ULA com pinos configuráveis de

entrada/saída; finalmente no capítulo nove se apresentam as conclusões e trabalhos futuros.

2 Conceitos básicos sobre testabilidade

O processo de verificação está dividido em duas grandes fases, a verificação do projeto (teste funcional) e a verificação da manufatura (teste de falhas) [Bushnell2000]. Na primeira, confere-se que não existem erros de interpretação e implementação da especificação, nem de funcionalidade, são analisados os algoritmos, as conexões e as portas usadas; por outro lado, a verificação da manufatura está associada com os processos de manufatura, os defeitos físicos como curto circuitos, circuitos abertos, pontes e velocidade de resposta [Abramovici1990]. O teste de manufatura é uma das fases finais no ciclo de produção. Ela define a qualidade dos produtos para que o cliente receba unidades funcionais. Nenhum circuito integrado pode ser manufaturado sem uma estratégia eficiente de teste que garanta que a porcentagem de produtos deficientes, não detectados, seja inferior ao limiar aceitável; que é normalmente medido em partes por milhão. A Figura 2-1 ilustra um defeito na fabricação de um circuito integrado, onde duas trilhas de sinal foram curto circuitadas por um elemento residual do processo.

Figura 2-1: Exemplo de falha em circuitos integrados ("bridge")

A testabilidade refere-se à capacidade de validar o hardware produzido, de forma que se garanta que o dispositivo final não possua defeitos que impeçam o seu correto funcionamento. Estes defeitos podem ser causados por diferenças no processo de fabricação, imperfeição do material, entre outros. A testabilidade de um circuito é um conceito abstrato relacionado com os custos associados às provas de manufatura. Estes custos estão constituídos por vários fatores entre os quais se encontram o tempo de teste, a área, o tempo para o desenvolvimento dos padrões e circuitos, entre outros. O incremento da testabilidade de um circuito implica que alguma função de custo foi reduzida, embora, não necessariamente cada custo individual. Por exemplo, o uso de técnicas de scan diminuem o custo da geração dos vetores de teste, mas incrementam o número de pinos de entrada/saída e a área.

A controlabilidade, observabilidade e previsibilidade são as três propriedades mais importantes que determinam a complexidade de auferir um teste para um circuito. A controlabilidade é a habilidade de fixar um valor específico de sinal em cada nó, através da manipulação das entradas do mesmo. A observabilidade é a capacidade de determinar o valor de sinal em cada nó, através da manipulação das entradas e observação das saídas. A previsibilidade é a habilidade de conhecer a resposta na(s) saída(s) de um circuito, dado o estímulo na(s) entrada(s). Um circuito integrado tem alta testabilidade se através da injeção de sinais de entrada, e análises da resposta de saída, for possível conferir a ausência de falhas. Conhecem-se por “Design for Testability” (DFT) a atividade que estuda as técnicas que melhoram a controlabilidade, observabilidade e previsibilidade dos circuitos integrados. Elas incluem os métodos e circuitos utilizados para garantir a testabilidade.

Em geral, as técnicas de DFT são usadas para reduzir o custo, aumentar a qualidade (cobertura de falhas), e então, diminuir a quantidade de peças defeituosas não detectadas. Elas afetam também o tamanho dos vetores de dados, a memória do testador e o tempo da execução da verificação. Normalmente, todos estes atributos estão relacionados diretamente entre eles. Infelizmente, não existe nenhum modelo confiável que ajude a determinar com precisão o número de testes requeridos para atingir um nível específico de cobertura de falhas. As técnicas de DFT automatizam o processo de geração de testes de forma eficiente e eficaz.

A medida da qualidade do processo de manufatura é a razão entre os dispositivos que passam nos testes de manufatura, e o número total de componentes fabricados. Este valor é conhecido como Yield [Bushnell2000].

A maioria das técnicas de DFT tratam com a ressíntese do circuito, ou a adição de hardware ao projeto, aumentando a área, os pinos de entrada/saída, os atrasos, etc. Devido a isso, existe um balance crítico entre a quantidade de estruturas de teste a serem usadas e o ganho adquirido. O acréscimo na área e/ou a complexidade da lógica resulta, também, no aumento da potência consumida e a diminuição do Yield.

A despesa com o desenvolvimento de teste é um gasto realizado uma única vez durante o projeto e pode ser diluído pelo número de unidades testadas. Desta forma, quanto mais peças manufaturadas, menor é o custo de desenvolvimento por unidade. Por outro lado, os equipamentos de testes são extremamente caros e demandam um valor alto para manutenção e operação. Este valor é fixo e está determinado pelo custo por unidade de tempo. Desta forma, o custo do teste por unidade fabricada é diretamente proporcional ao tempo gasto para sua execução no testador. Este custo é agregado ao produto da mesma forma como a despesa de desenvolvimento. Deve encontrar-se a relação ótima de custo/benefício entre o tempo de teste de cada unidade e a cobertura de falhas aceitável para garantir a qualidade dos produtos.

2.1 O paradigma de teste digital

O processo de verificação de manufatura emprega uma técnica bem aceita, conhecida como “o paradigma de teste digital”. O paradigma consiste em quatro componentes básicos: (1) seleção do modelo de falha, (2) geração da lista de falhas de interesse, (3) geração dos padrões para detectar as falhas contidas na lista de falhas, e (4) estimação da percentagem de falhas detectadas através da aplicação dos padrões de teste. Este processo é conhecido como a “construção do teste”; é desenvolvido depois que o dispositivo foi projetado e antes que este seja manufaturado. Os padrões de teste resultantes são usados para classificar os circuitos [Bushnell2000]. A literatura mostra que mais de 20% do tempo de

desenvolvimento do produto é gasto para gerar os padrões de teste, de forma que a cobertura de falhas seja suficiente para garantir a qualidade dos produtos [Wang2008].

2.1.1 Seleção de modelo de falha

Um modelo de falha consiste na representação do efeito de um circuito defeituoso ante um possível estímulo. Em circuitos digitais as falhas físicas são modeladas através de equações lógicas [Abramovici1990]. O objetivo do modelamento de falhas é identificar o comportamento de grande parte dos defeitos físicos que podem ocorrer em um dispositivo, usando o maior nível de abstração possível. Isto reduz o número de defeitos individuais que devem ser considerados, e diminui a complexidade do aparelho computacional necessário para a geração do teste.

O nível de abstração de portas é o mais usado para este tipo de modelamento, visto que este consegue representar a maioria dos defeitos do dispositivo sob teste. O nível RTL (“Register transfer level”) é muito abstrato para representar com exatidão muitos dos tipos de falhas, e o nível de transistor aumenta drasticamente a quantidade de cálculos necessários para prever os efeitos das falhas no circuito.

A seguir se encontra uma descrição dos principais modelos de falhas usados para a geração de testes de circuitos integrados. A seleção do modelo de falhas depende do objetivo do teste, sendo muito comum a prática da geração de testes que cobrem falhas de vários modelos.

2.1.1.1 Modelo de falhas single stuck-at (SSAF)

Este é o modelo de falhas mais popular, publicado pela primeira vez em 1961 [Roth1961 apud Garcia2001, p2]. Parte dos seguintes pressupostos: as células modeladas não possuem defeito; os defeitos só podem aparecer nas linhas de interconexão; existe unicamente um defeito por vez em uma linha da célula ou porta; e a falha introduzida é permanente. O efeito da falha é igual a fixar o nó a Vdd (um

lógico) ou Terra (zero lógico), enquanto as outras portas no circuito não são afetadas pela falha. O modelo SSAF cobre muitos dos possíveis defeitos de manufatura nos circuitos CMOS, como: curtos entre fonte e dreno, contaminação nas difusões, curtos de metal, entre outros.

A cobertura do modelo SSAF pode ser incrementada mapeando outros modelos de falha como “stuck-open” e “bridging” nas sequências das falhas “stuck-at”. A vantagem desta estratégia é que o resultado pode ser aplicado ao nível de portas lógicas, mantendo os requerimentos computacionais razoáveis. O conjunto de falhas aplicando este modelo é moderado e corresponde a duas vezes o número de nós. Adicionalmente, os algoritmos para a geração automática de padrões de teste (ATPG) e a simulação de falhas de redes combinacionais que usam SSAF são bem desenvolvidos e eficientes. Outra característica deste modelo de falhas é ser independente da tecnologia, já que pode ser aplicado a qualquer modelo de circuito estrutural.

Uma das desvantagens do SSAF é, como foi indicado pelo ITRS em 1999, que este modelo só cobre 70% dos possíveis defeitos de manufatura e circuitos CMOS, deixando 30% dos defeitos potenciais sem serem detectados.

2.1.1.2 Modelo de falhas “multiple stuck-at”

O modelo de falhas “multiple stuck-at” (MSAF) usa as mesmas premissas que o SSAF, exceto que ele permite que duas ou mais linhas do circuito estejam simultaneamente com defeito. Embora o MSAF cobre maior número de defeitos, quando comparado com o SSAF, o incremento pode não ser significativo o suficiente para compensar o grande aumento do número de falhas que terão que ser analisadas com esta estratégia (Falhas= $3n-1$, para um circuito com n nós). Adicionalmente, os algoritmos de ATPG e simulação de falhas são muito mais complexos e não estão muito desenvolvidos, além de que os simuladores comerciais não suportam bem o MSAF. Devido a isto, a simulação de falhas usando o modelo MSAF é raramente usada.

2.1.1.3 Modelo de falhas “stuck-open”

O modelo de falhas “stuck-open” (SOF) assume que uma única linha física do circuito foi aberta e que como resultado o nó não está fixado a Vdd ou terra. A vantagem desta abordagem é que cobre defeitos que não podem ser detectados pelo SSAF nem pelo MSAF, mas que podem ser testados com sequências de vetores dos testes SAF [Millman1988].

Sabe-se que a análise de um circuito utilizando o modelo de falhas SOF ao nível de portas lógicas pode não ser suficiente, já que não se consideram os efeitos das falhas que ocorrem dentro da porta. As falhas “stuck-open” que ocorrem nas linhas internas da porta resultam em efeitos, completamente diferentes aos considerados no circuito modelado em nível de abstração de portas lógicas. Um destes efeitos é o de memória, no qual a saída da porta mantém o valor do resultado anterior; dependendo da ordem dos vetores de teste e o valor programado para a saída, um circuito com este tipo de falha pode passar nos testes de manufatura. Para a detecção deste tipo de falhas é necessária a criação de pares de vetores que forcem uma sequência de sinais de saída opostos. Outro defeito de fabricação, representado com este modelo, é o que acontece nas portas de transmissão, já que elas podem conduzir nas duas direções. Uma porta de transmissão com um circuito aberto em um dos transistores tipo P ou N, muito provavelmente, irá passar nos testes dos padrões gerados pelos modelos SAF ou SOF, já que a funcionalidade da célula se mantém, embora a velocidade de resposta dela diminua.

Comparado com o SSAF, os modelos SOF requerem maior número de sequências de vetores de teste para detectar cada falha. Os algoritmos para a geração e simulação de falhas com o modelo SOF são muito complexos e demandam grande quantidade de recursos computacionais e de memória. Para a criação da lista de falhas completa, a maioria dos potenciais defeitos SOF precisam de uma descrição mais detalhada do circuito em um nível de abstração mais baixo, como o nível de transistor. Alternativamente, um “modelo de teste” ao nível de portas poderia ser mapear SOFs em SAFs. Esta estratégia resulta em um circuito com dois ou mais vezes o número de portas que no projeto original.

Existem reportes que afirmar que a múltipla detecção de falhas SAF por caminhos diferentes pode fazer com que se aumentam a chance de pegar defeitos de circuitos abertos [McCluskey2000]. Esta técnica é conhecida como “N-detect” ou detecção “N-way”. Normalmente, os algoritmos de ATPG removem a falha da lista de falhas quando um vetor de teste que a detecta foi criado e simulado; desta forma não se “gasta” tempo de CPU adicional tentando encontrar vetores que cobrem a mesma falha. Na detecção “N-way”, a ferramenta de ATPG é configurada para não remover a falha da lista, produzindo vetores adicionais que a cubram. Neste esquema a falha é removida da lista de falhas quando ela é detectada pelo menos “N” vezes. A esperança é que os vetores adicionais incluam pares de vetores que simulem as condições da falha. Por outro lado, muitos dos vetores adicionais podem não gerar os resultados desejados, aumentando os requerimentos de memória e tempo de uso dos equipamentos de teste (ATE).

2.1.1.4 Modelo de falhas “bridging”

Este modelo inclui as falhas onde existe um curto entre dois nós do circuito que se encontram fisicamente perto. Existem três tipos de falhas consideradas por este modelo:

- curto dentro dos elementos lógicos, como aqueles entre as portas dos transistores, fontes ou drenos;
- curto entre nós lógicos, entre as entradas ou saídas dos elementos lógicos sem realimentação; e
- curto entre os nós lógicos com realimentação.

Usualmente o modelo “bridging” não considera curtos entre elementos lógicos e não lógicos. Este modelo cobre uma grande parte dos defeitos físicos não considerados pelo modelo SSAF, mas os algoritmos de ATPG ficam muito complexos, já que os vetores de teste gerados devem fixar valores opostos em nós que se encontram fisicamente próximos, e observar o resultado desta configuração.

2.1.1.5 Modelo de falhas “transition delay”

Um circuito que não tenha falha lógica pode ainda ter um defeito físico, como aqueles gerados por diferenças no processo de fabricação, onde se cria uma porta com um atraso suficientemente grande para causar problemas de funcionamento. O modelo de falhas “transition delay” (atraso de transição) detecta este tipo de defeitos assumindo que a lógica que vai ser testada não possui erros, mas que uma saída de uma dada porta pode ser lenta para subir ou descer, e que este tempo é maior que um nível predefinido. Se a falha de atraso é suficientemente grande, ela se comporta como a SAF e pode ser modelada usando esse método.

As principais desvantagens deste modelo são: a necessidade de dois padrões sequenciais para lançamento e captura de transição, e a dificuldade para determinar o limiar de atraso mínimo, devido aos riscos temporização (“timing hazard”) [Nowick1995]. Consequentemente, usa-se normalmente um ciclo de relógio para fixar o tempo limiar.

2.1.1.6 Modelo de falhas “path-delay”

Este modelo de falhas é similar ao “transition delay”. Nele se assume que a lógica do circuito está correta; porém, ao invés de modelar a falha como um atraso em uma única porta, considera que o atraso total em um caminho, é maior que o limiar. A geração de vetores para cobrir as falhas “transition delay” podem mascarar falhas reais, já que portas rápidas podem compensar o atraso de uma porta com falha. Este problema é eliminado com o modelo “path-delay”, dado que considera a contribuição de cada um dos possíveis atrasos no caminho.

Com este modelo é possível especificar um grau na velocidade do relógio através da determinação estatística dos níveis máximos de atraso que podem acontecer no mundo real. A desvantagem principal deste modelo é que o número de possíveis caminhos pode crescer exponencialmente com o número de linhas de sinal.

2.1.1.7 Modelo de falhas IDDQ

O modelo de falhas IDDQ propõe uma abordagem completamente diferente dos modelos apresentados anteriormente. Nele, a observação das falhas é feita através da medição da corrente da fonte de alimentação, ao contrário dos valores booleanos obtidos ao submeter o padrão para avaliação. O modelo IDDQ se baseia no fato de que uma porta CMOS estática consome unicamente uma corrente significativa nas transições de sinal, e que a corrente quiescente de um transistor CMOS estático é da ordem dos nA (nano Amperes).

Este modelo cobre outros tipos de falhas que não poderiam ser detectados com o SAF, como por exemplo, o curto resistivo entre a porta e o dreno de um transistor. Neste caso a porta lógica poderia responder adequadamente aos estímulos de entrada, mas o defeito poderia fazer com que o funcionamento do circuito se degrade com o tempo. A falha poderia ser detectada quando se configura um valor adequado na entrada do transistor, e se mede um valor de corrente muito superior ao esperado pela corrente quiescente.

A principal vantagem do teste de IDDQ é que a geração é relativamente simples, devido ao fato do ponto de observação das falhas se encontrar na medida da corrente de alimentação do circuito integrado. Outros tipos de falhas que este modelo pode representar são as “bridging”, os defeitos no óxido das portas dos transistores, e os curtos entre os dois terminais dos transistores. Muitas dessas falhas anteriormente mencionadas poderiam não afetar a lógica do circuito, mas degradam sua confiabilidade.

A desvantagem deste modelo está no tempo entre medidas, já que é necessário esperar a estabilização da corrente para cada um dos vetores de teste. Adicionalmente, o valor de limiar de corrente de IDDQ é fixado empiricamente e não existe uma forma absoluta para calculá-lo.

O modelo de falhas IDDQ é dependente da tecnologia. Na medida em que diminui o tamanho da camada de óxido da porta do transistor, aumenta-se a corrente

quiescente significativamente, dificultando a capacidade de distinguir entre uma parte com ou sem falhas.

2.1.2 Geração da lista de falhas

Cada um dos modelos de falhas possui características importantes que permitem abarcar distintos tipos de defeitos. A seleção do modelo de falhas depende do objetivo e tecnologia utilizada no processo de manufatura. Não existe um modelo de falhas que abarque todos os tipos de defeitos. O modelo de falhas mais comumente utilizado é o “stuck-at”, embora seja uma prática habitual considerar modelos de falhas complementares para a geração dos padrões de teste.

Depois de selecionar o modelo de falhas a ser utilizado, ele é aplicado ao circuito que será estudado ou dispositivo sob teste (“Device Under Test”, DUT). Com isso se gera a lista de falhas relevantes ao processo de manufatura. É importante destacar que a lista de falhas representa um conjunto finito, e que existem algumas técnicas e heurísticas que podem ajudar a diminuir seu tamanho, minimizando o número de cálculos a serem realizados para gerar os padrões de teste.

Uma das principais heurísticas utilizadas para diminuir o tamanho da lista de falhas é a de colapso de falhas. Duas falhas de um circuito Booleano são equivalentes se a minimização do circuito, para cada uma das falhas, produz funções de saída idênticas. Falhas equivalentes são conhecidas também como indistinguíveis e possuem exatamente o mesmo conjunto de vetores de teste. O processo de encontrar equivalências e eliminar da lista de falhas os elementos não representativos se conhece como colapso de falhas (“Fault Collapsing”). Por exemplo, usando o modelo de falhas SSAF, uma porta “AND” de três entradas possui cinco falhas representativas, duas na saída (sa0 e sa1), e uma sa1 para cada entrada. A relação entre o conjunto de falhas colapsadas e o conjunto de todas as falhas se conhece como razão de colapso (“Collapse ratio”); aplicado ao exemplo da “AND” de três entradas, a razão de colapso é 5/8.

Outra heurística que ajuda a diminuir o tamanho da lista de falhas está baseada no conceito de falha dominante. Diz-se que uma falha é dominante se os vetores de

teste necessários para detectar uma determinada falha, incluem os vetores de teste para detectar uma outra falha. Do exemplo anterior, o conjunto de possíveis vetores de teste para detectar a falha sa1 da saída são (110, 001, 000, 010, 100, 101, 011); e para detectar uma falha sa1 na primeira entrada da “AND” o vetor de teste é (011). Como o segundo conjunto está incluso dentro do primeiro, então a falha sa1 na saída da “AND” é dominante. Como a falha dominante pode ser detectada pelo conjunto menor, então é possível eliminá-la. Desta forma, só aparecem as falhas sa0 e sa1 de cada uma das entradas da porta “AND” de três entradas. Depois de aplicar a técnica de falha dominante pode-se aplicar a de colapso, de tal forma a ficarem apenas 4 falhas que serão detectadas, uma sa1 para cada entrada e uma sa0 em uma das entradas, já que esta falha é equivalente à cada uma das falhas sa0 das entradas.

2.1.3 Geração de padrões de teste

O processo de geração de padrões de teste (“Test Pattern Generation”, TPG) se executa após a obtenção da lista de falhas representativas do circuito. Existem três categorias de TPG: automática determinística (“Deterministic Automatic” TPG, DATPG), automática randômica (“Random Automatic” TPG, RATPG), e manual (“Manual” TPG, MTPG). O propósito do conjunto de vetores ou padrões de teste gerados é ativar cada um dos elementos contidos na lista de falhas, para observar divergências entre a resposta do circuito bom (sem falhas) e os DUTs defeituosos. Desta forma, os vetores de teste criados são usados para avaliar a qualidade do processo de manufatura de cada circuito integrado, rejeitando os dispositivos cuja resposta de saída seja diferente da esperada.

O processo de geração de padrões de teste automática determinística (DATPG) pode ser dividido em três possíveis métodos: sensibilização de caminhos, simbólico, e independente de falhas. Depois da geração de cada padrão de teste, tipicamente, são aplicadas técnicas de simulação de falhas para determinar o conjunto completo das falhas detectadas.

O propósito principal do método de sensibilização de caminhos é determinar um vetor de entrada para detectar uma falha específica no DUT. Nesta categoria encontram-se a maioria dos algoritmos de geração de padrões. Conceitualmente, esta técnica realiza uma busca exaustiva no espaço das entradas do DUT, para localizar o vetor de entrada que consegue ativar uma determinada falha (sensibilizar um caminho) e propagar seu valor para as saídas primárias. O espaço de busca cresce exponencialmente com o aumento das entradas, afetando o tempo de execução do algoritmo. Por este motivo, é uma prática comum definir um valor de truncamento, a partir do qual se classifica a falha como difícil de detectar e se passa a analisar o seguinte elemento da lista de falhas. Em geral o método de sensibilização de caminhos consiste em três passos:

1. Ativar a falha fixando as entradas de um componente interno com os valores apropriados;
2. Justificar os valores de sinal internos até as entradas primárias; e
3. Propagar o erro causado pela falha ativada a uma ou mais saídas primárias.

A implementação de cada um destes passos depende do algoritmo utilizado; alguns dos mais conhecidos são: o D [Roth1966]; o 9-V [Cha1978]; o PODEM ("Path Oriented Decision Making") [Goel1981, Goel1981a]; o FAN ("Fanout-Oriented", TPG) [Fujiwara1983], e o SOCRATES ("Structured-Oriented Cost-Reducing Automatic TEST pattern generation system"). Todos estes algoritmos utilizam valores lógicos, além do 0 e 1, que permitem simplificar os processos de justificação e propagação. A eficiência é medida pelo número de processos necessários para obter a resposta que se busca. Dependendo dos elementos lógicos utilizados é possível minimizar o número de iterações para justificar as entradas e propagar o erro.

Um dos principais desafios que estes algoritmos enfrentam é quando o circuito analisado possui caminhos reconvergentes, o que significa que as entradas do DUT não podem ser justificadas independentemente umas das outras. Isto é, existem múltiplas alternativas para justificar um sinal ou propagar um erro, o que faz com que o algoritmo tenha que entrar em um laço de decisão. Cada possível decisão é avaliada, caso se encontre uma resposta satisfatória, o processo é terminado

definindo o vetor de entrada; caso contrario, devem-se restaurar os valores de sinal para o ponto anterior ao início do processo de decisão. Os algoritmos PODEM, FAN e SOCRATES não precisam do processo de restauração, o que faz com que sejam mais eficientes. O SOCRATES chega a ser o algoritmo mais econômico, já que implementa algumas heurísticas que minimizam o número de cálculos e iterações realizadas.

O método simbólico consiste na manipulação de expressões lógicas. O processo começa pela conversão do modelo em nível de portas para um conjunto de expressões Booleanas. Estas expressões são manipuladas usando uma sequencia predefinida de operações, cujo resultado final representa as condições das entradas para detectar uma falha. Uma das principais limitações deste método é que o número de símbolos necessários para representar qualquer DUT, não trivial, é grande, o que dificulta a manipulação, e, como consequência, acaba sendo pouco usado [Bushnell2000, Kung1991, Larrabee1989].

A principal característica do método independente de falhas consiste no fato dos algoritmos não direcionarem a geração dos padrões de teste para a detecção de uma única falha por vez, e sim para encontrar o maior número de falhas com o mesmo padrão. De forma genérica, este método começa pela seleção dos valores de saída do DUT que satisfazem o critério de sensibilização, isto é, a porta que fixa o valor de saída deve ser configurada para ter uma entrada controladora. A entrada de um componente é considerada como controladora se a mudança do seu estado lógico faz com que a saída resultante troque de valor. Desta forma, o método independente de falhas busca a combinação das entradas das portas que causam longos caminhos sensibilizados. Estes caminhos são encontrados através da análise reversa dos sinais de saída do DUT e os sinais de entrada controladores. A mudança no estado de qualquer uma das entradas controladoras de um caminho continuo no DUT deve causar um erro na saída [Waicukauski1989, Schnurmann1975, Wang1975, Benmehrez1983].

O processo de geração de padrões automático randômico (RATPG) possui três passos básicos: seleção de vetor aleatório de entrada, simulação de falhas, e remoção de falhas detectadas da lista de falhas. Desta forma, cada vetor de entrada selecionado, que detecta alguma falha, é adicionado ao conjunto de vetores

resultantes. Estes passos são repetidos até que o número de falhas que não foram detectadas seja menor que o valor prefixado [Larrabee1989, Seth1989].

Na medida que o processo avança, a probabilidade de encontrar um novo elemento para o conjunto de vetores de entradas selecionados se reduz, já que o espaço das falhas diminui. Além disto, existe um grupo de falhas, conhecidas como resistentes aos padrões randômicos, nas quais o número de combinações de entrada que podem ativá-las é muito pequeno, quando comparado com os outros elementos da lista.

Nesta categoria existem basicamente dois métodos: seleção de entradas uniforme e a seleção de entradas com peso. Na primeira, a geração dos vetores de entrada utiliza uma distribuição uniforme para a seleção, o que significa que todos os elementos possuem a mesma possibilidade de serem escolhidos. Isto normalmente é feito com bibliotecas para a geração de números aleatórios. O problema da seleção uniforme é que o número de padrões resultantes pode ser muito grande. No segundo método, analisa-se a estrutura do DUT para determinar uma melhor distribuição das entradas, de forma a reduzir o problema das falhas resistentes aos padrões aleatórios e minimizar o número de padrões resultantes [Debany1991, Waicukauski1989].

Existe uma técnica, conhecida como TPG combinada, que tenta aproveitar o melhor das duas categorias apresentadas anteriormente (determinística e randômica). Nesta técnica se inicia o processo de geração de padrões de entrada aplicando um método de geração RATPG, visto que a relação entre o número de falhas detectadas e o tempo de execução é maior que o da DATPG. Consecutivamente, as falhas que são RATPG resistentes são submetidas ao método DATPG de forma a encontrar, caso exista, o vetor de entrada para detectar estas falhas. Finalmente, o resultado se dá pela eficiência computacional [Abramovici1990, Bushnell2000, Benmehrez1983, Larrabee1989].

No processo de geração de falhas manual (MTPG) os padrões de teste são escolhidos pelo projetista. Tipicamente, os primeiros padrões selecionados são os usados para verificar a correta funcionalidade do circuito. Cada padrão é submetido ao simulador de falhas, as que são detectadas são retiradas da lista. Na lista de

padrões de teste unicamente se incluem aqueles que detectam novas falhas. O processo termina quando se consegue o valor de cobertura desejado.

Para determinar o processo TPG ou a estratégia combinada a ser utilizada, é necessário avaliar o custo de geração, a qualidade do teste e o custo de aplicação do mesmo. Os algoritmos de RATPG são normalmente os mais simples, o que minimiza o custo de geração, mas para conseguir uma qualidade de teste adequada, pode ser necessário gerar uma grande quantidade de vetores, o que repercute no custo da aplicação do teste. Por outro lado, o custo de geração do DATPG pode ser elevado devido à complexidade dos seus algoritmos, e nada garante que o custo de aplicação é menor que o da RATPG. Por estes motivos a estratégia combinada RATPG-DATPG resulta muitas vezes na melhor opção, já que é capaz de diminuir os custos de geração e aplicação de teste, mantendo a qualidade do mesmo. O MTPG é o processo mais ineficiente e custoso de todos, quando usado isoladamente para testar um circuito, mas pode ser utilizado para gerar vetores de testes funcionais que servem para cobrir circuitos que foram modelados comportamentalmente e/ou não possuem elementos de testabilidade [Aramovici1990].

O curso da geração de vetores também depende da complexidade do circuito que se deseja testar. As técnicas de projeto para testabilidade (“Design For Testability”, DFT) são usadas para facilitar a controlabilidade e observabilidade de pontos internos do Circuito Integrado, de forma a reduzir o custo de geração e aplicação do teste. Os sistemas eletrônicos possuem três tipos de componentes: lógica digital, blocos de memória e circuitos analógicos ou mistos (“mixed signal”). Existem métodos de DFT específicos para cada um dos tipos de componentes.

Especificamente para a lógica digital são usadas três técnicas de DFT: Ad-Hoc, “Scan” e BIST. A primeira está baseada nas boas práticas de projeto aprendidas com a experiência, como evitar realimentações assíncronas e “fan-ins” grandes, controlar inicialização de “flip-flops”, etc.

Na técnica de “Scan”, a ideia principal é adicionar controlabilidade e observabilidade em cada “flip-flop”. O circuito é projetado para ter dois modos de funcionamento em teste: deslocamento (“shift”) ou avaliação (“capture”). No primeiro modo, todos os “flip-flops” são configurados para formar um grande registro de deslocamento

(cadeia de “scan”), cuja entrada e saída é mapeada nos pinos do CI (entradas e saídas primárias), junto com os outros sinais de controle (“clock” e “reset”). Em um mesmo dispositivo podem existir várias cadeias de “Scan”. Desta forma, é possível fixar um estado determinado em cada “flip-flop”. Quando configurado para avaliação, o circuito se comporta como o faria em funcionamento normal. Cada pulso de relógio faz com que os elementos de memória capturem o resultado da avaliação dos blocos combinacionais. Consecutivamente, é possível voltar ao modo de deslocamento, e assim extrair o resultado da avaliação em cada ponto de observação. Os resultados obtidos são comparados com os esperados para determinar se o dispositivo possui falhas no seu processo de fabricação. A geração de padrões de teste, usando esta técnica é relativamente simples, visto que o circuito é dividido em pequenas partes que podem ser testadas ao mesmo tempo, aumentando o número de pontos de prova e observação. Existe muita atuação no desenvolvimento de algoritmos eficientes que usam relativamente poucos recursos computacionais, o que acaba resultando na minimização do custo de geração. Por outro lado, o custo de aplicação de teste também é reduzido, já que se diminui o número de vetores de teste. Na Figura 2-2 (a), exemplifica-se uma cadeia de “Scan”, composta por “flip-flops” conectados como um registro de deslocamento. A Figura 2-2 (b), representa um “flip-flop” de “Scan”, cuja entrada D está conectada a um multiplexador. O sinal SE (“Scan Enable”) serve para selecionar o modo de operação: deslocamento ou avaliação. O sinal SDI (“Scan Data Input”) serve para conectar a saída de um “flip-flop” anterior com o próximo, formando assim a cadeia. A entrada D (“Data”) seria então a mesma usada no circuito quando configurado para funcionamento normal.

Figura 2-2: Arquitetura de Scan com multiplexador

Existe outro estilo de projeto de “Scan” que usa “flip-flops” com dois “latches” por célula, e dois sinais de “clock” não sobrepostos. Este estilo é conhecido como “Level-Sensitive Scan Design”. Neste “flip-flop” ao invés de usar um multiplexador para capturar o valor do dado de entrada ou o dado da cadeia de “Scan”, usa-se dois sinais de “clock” diferentes para selecionar qual dado vai ser salvo nos “latches”. A ideia e funcionamento da cadeia são similares para os dois casos de “flip-flops”.

A técnica de “Scan” é a mais usada no projeto para testabilidade, devido ao fato de simplificar a obtenção de alta cobertura, com custo relativamente baixo. No entanto, é uma prática comum aplicar padrões funcionais junto com os de “Scan”, já que estes últimos não cobrem todas as falhas que podem aparecer em dispositivos complexos onde existe lógica digital junto aos circuitos analógicos. Adicionalmente, muitas vezes são agregados circuitos que modificam a funcionalidade do dispositivo quando configurado para teste, de forma a controlar e observar mais pontos. Isto gera o problema de não testar os verdadeiros caminhos de conexão funcional, o que pode afetar ou invalidar os resultados de detecção de alguns tipos de falhas como os “path delay” ou “transition delay”. Dependendo do objetivo de cobertura os padrões funcionais são submetidos a simuladores de falhas para determinar o aporte deles no resultado final. Dependendo da complexidade do circuito, a simulação de padrões funcionais é uma tarefa muito dispendiosa que chega a ser inviável.

Por final, a técnica de projeto para testabilidade BIST (“Built-In Self-Test”) consiste em encontrar a assinatura do circuito, agregar elementos para estimular as falhas e comparar os resultados com os esperados. Para circuitos homogêneos, como as memórias, esta técnica é perfeita, porque consegue obter uma alta cobertura, com um tempo de aplicação reduzido. A arquitetura para teste autônomo de memórias é conhecida como MBIST (“Memory BIST”). Atualmente, dentro das arquiteturas MBIST existem alguns padrões bem conhecidos na indústria, como o algoritmo de elementos “March” ou o de “checkerboard” (tabuleiro de damas). Por outro lado, a BIST Lógica ou LBIST, emprega um gerador de padrões pseudo-aleatórios para produzir as entradas que serão aplicadas às cadeias de Scan internas, e um registro de múltiplas assinaturas de entradas (“Multiple input signature register MISR”) para obter a resposta do dispositivo a estos padrões. Uma saída incorreta do MISR indica um defeito no dispositivo. No caso da LBIST é necessário seguir regras estritas do projeto do circuito digital, de forma que se reduza o número de sinais mascarados, o que aumenta o tamanho do circuito e diminui a cobertura. Em geral, a vantagem das arquiteturas de BIST é que a aplicação dos padrões de teste pode ser realizada com equipamentos econômicos e na velocidade de operação. Embora, existe controvérsia referente a estas vantagens, visto que nem sempre é possível obter a cobertura desejada com as técnicas de BIST, e para a aplicação de testes analógicos ou testes que usem outros modelos de falhas como o IDDQ, é necessário utilizar equipamentos sofisticados, com características especiais.

2.1.4 Classificação de falhas

As ferramentas de geração automáticas de padrões de teste produzem como saídas os vetores que serão usados na produção para verificar a qualidade dos circuitos integrados manufaturados, e a lista de falhas resultante do modelo que foi aplicado, com a sua classificação. De forma genérica, as falhas são classificadas em dois grandes grupos [Wang2008, Mentor2009a]: as não testáveis e as testáveis. Dentro do primeiro grupo se encontram as falhas para as quais não pode existir nenhum padrão que as detecte. Estas falhas não podem causar nenhum erro funcional, de

forma que a ferramenta de ATPG as exclui do cálculo de cobertura de teste. Dentro do grupo de falhas não testáveis se encontram:

- Não usadas: inclui todas as falhas das linhas de circuito não conectadas a um ponto de observação.
- Fixas: corresponde às linhas de sinal que por projeto estão presas em VDD ou GND e que se propagam dentro do circuito. Esta característica gera duas falhas por cada porta das células lógicas onde se propaga, uma que é testável e outra que não é. Por exemplo, para uma linha que é fixada em VDD se deve testar se ela não está presa em GND, mas não é necessário testar se ela não está presa em VDD.
- Bloqueadas: inclui todas as falhas cujos caminhos até pontos de observação são bloqueados por lógica presa em algum valor. Neste grupo também estão inclusas as falhas nas linhas de seleção de multiplexadores, cujas entradas de dados são idênticas.
- Redundantes: esta classe de falhas inclui os pontos em que seja qual for o seu valor, sua mudança não consegue ser observada em nenhum dos pontos de captura.

As falhas testáveis são o restante das falhas, quer dizer, aquelas para as quais não é possível provar a não testabilidade. Este grupo é dividido nas seguintes classes:

- Detectadas: inclui todas as falhas que depois do processo geração foram simuladas e detectadas pelos padrões criados. Também inclui as falhas que foram detectadas por implicação, isto é, as falhas derivadas do processo de aprendizado e análises (normalmente inclui as falhas no caminho das células de scan e as associadas aos clocks das mesmas).
- Possivelmente detectadas: corresponde ao grupo de falhas que o simulador não consegue classificar como completamente detetadas. Resulta da diferença 0-X ou 1-X no ponto de observação.
- Não detectável: esta classe está sub-dividida em duas categorias, as não controláveis e as não observáveis. Na primeira, a ferramenta não consegue fixar um valor específico em um ponto do circuito para alcançar a detecção da falha; a segunda, corresponde às falhas cujo efeito não consegue se propagar a um ponto de observação.

- ATPG não testável: inclui todas as falhas testáveis para as quais o gerador de padrões não conseguiu criar um vetor de teste. Dependendo das restrições que se configuram na ferramenta de ATPG, podem-se aumentar ou diminuir o número das falhas deste tipo.

2.1.5 Cálculo de cobertura

A cobertura corresponde à razão entre as falhas detectadas por um conjunto de vetores de teste em relação ao conjunto total de falhas. O processo conhecido como graduação de falhas (“fault grading”), consiste em determinar o número de falhas detectadas através da simulação dos padrões de teste. A complexidade deste processo é dada pela estrutura do DUT, o modelo de falhas empregado, e o tamanho e qualidade do conjunto de padrões aplicados. A estrutura do DUT é determinada pelas funções que ele desempenha, estratégia de teste, e tecnologia usada. Por exemplo, os algoritmos de simulação de falhas elaborados sobre arquiteturas de “Scan”, que usam o modelo de falhas “stuck-at”, são bastante eficientes, visto que reduzem a complexidade do circuito, limitando-a ao espaço combinacional puro ou sequencial simples. Por outro lado, a simulação de falhas de padrões funcionais aplicada a circuitos complexos é muito custosa.

Existem no mercado diversas ferramentas computacionais especializadas para cada tipo de função de teste, dependendo do modelo de falhas adotado e do objetivo: geração, simulação ou graduação de falhas. Duas ferramentas muito conhecidas são a “Fastscan da Mentor Graphics” e o “Encounter Test da Cadence”. Basicamente estas ferramentas se especializam na geração automática de padrões de teste para circuitos digitais que usam “Scan” como arquitetura de projeto para testabilidade. Elas estão habilitadas para trabalhar com distintos modelos de falhas como “stuck-at”, “transition delay”, “path delay”, IDDQ, entre outros. De forma genérica elas seguem os seguintes passos: (1) criação da lista de falhas aplicando o modelo selecionado, (2) geração de padrão de teste, (3) simulação, (4) graduação das falhas e eliminação da lista, (5) repetição da sequencia desde o passo 2 até ter removido todas as falhas ou até ter atingido o limiar de tentativas.

O cálculo da cobertura das ferramentas de ATPG está dado pela seguinte expressão, onde o crédito é um valor estatístico de 0 a 1 que corresponde à porcentagem de falhas possivelmente detectadas que são realmente detectadas. Normalmente as ferramentas de ATPG consideram 0,5 como o valor do crédito por defeito.

$$Cobertura = \frac{Falhas\ detectadas + (\%crédito * falhas\ possivelmente\ detectadas)}{número\ total\ de\ falhas\ testáveis}$$

Para a simulação de padrões funcionais existem várias ferramentas, entre as quais destacam-se a “Verifault da Cadence”, “FlexTest da MentoGraphics” e etc., por serem as mais usadas dentro da indústria. Estas ferramentas partem de padrões que já foram criados, seja de forma manual, ou usando algum tipo de heurística para a geração automática. O algoritmo de simulação empregado por cada ferramenta varia dependendo do vendedor e da configuração que é feita. Detalhes sobre algoritmos de simulação de falhas serão cobertos no próximo capítulo. De forma genérica, estas ferramentas executam os seguintes passos: (1) criar lista de falhas aplicando o modelo “stuck-at”, (2) simular o padrão com o circuito sem falhas, (3) introduzir uma falha e simular novamente, (4) se existir divergência entre as respostas do circuito com e sem falhas, quer dizer que o padrão pode detectar a falha e, conseqüentemente, ela é eliminada da lista, (5) repetir o laço desde o passo 3 até que sejam simuladas todas as falhas, com todos os vetores de entrada, ou até que se esvazie a lista de falhas. Como se pode notar, na descrição anterior, o processo de simulação de falhas de padrões funcionais pode ser extremamente complexo e custoso, chegando a ser inviável em muitos casos [Giangarra2008]. O suporte técnico, fornecido pelos vendedores, às ferramentas de simulação é precário, já que pequenos circuitos podem resultar em grandes desafios para elas, levando-as a uma sobrecarga computacional e de memória. Além disso, os modelos de blocos analógicos ou “hardblocks” que são inseridos dentro dos dispositivos, devem possuir uma descrição compatível com a suportada por estas ferramentas. Como não há desenvolvimento delas, isso pode resultar em uma re-elaboração do modelo ou até mesmo na impossibilidade da simulação.

Na medida que a tecnologia avança e novas aplicações são desenvolvidas, cresce a exigência por qualidade dos produtos desenvolvidos. Principalmente áreas como a

automotiva, de segurança e médica, possuem normativas onde se exige um número que indique a confiabilidade do processo de produção, e que garanta a funcionalidade total das peças distribuídas para os usuários. Dependendo da exigência, são aplicados vários modelos de falhas para determinar a lista das mesmas, e são calculadas as coberturas destas listas usando os padrões de teste aplicados na produção. Atualmente, por exemplo, na área automotiva se exige uma cobertura de falhas stuck-at de 98%, e se recomenda que a cobertura transition delay seja de 80% e IDDQ de 70%; adicionalmente todos os blocos analógicos ou hardblocks devem ter cobertura de 100%. Quanto maior são estes números e mais modelos de falhas são utilizados, maior é a confiabilidade dos produtos elaborados, mas, ao mesmo tempo, maior é o custo do desenvolvimento e da aplicação do teste.

Existem alguns casos onde o projeto para testabilidade usando a técnica de “Scan” não é suficiente para atingir o valor adequado de cobertura, exemplos destes são:

- circuitos complexos onde existe um grande número de blocos analógicos ou de sinal misto, cujas entradas/saídas não foram modeladas adequadamente, não podem ser estimuladas, ou foram fixadas em um valor estável durante todo o tempo de execução do teste;
- circuitos onde o a razão entre o número de falhas da lógica que configura o dispositivo para teste, dividido pelo número de falhas testáveis, corresponde a uma porcentagem maior do máximo de cobertura que se pode perder;
- quando existe mudança nos caminhos funcionais, por causa de realimentações, sinais assíncronos, ou definição de sinais de controle (“clocks” ou “resets”);

Nestes casos é possível utilizar a simulação de falhas para aproveitar os padrões funcionais e determinar seu aporte na cobertura final. Como foi mencionado anteriormente, realizar esta tarefa é dispendioso e pode ser muito caro em relação a recursos computacionais, humanos e de tempo. Uma alternativa é utilizar a emulação de falhas, para diminuir o tempo de gradação de padrões funcionais.

3 Simulação de falhas em circuitos integrados

Neste capítulo se introduzirão os conceitos fundamentais da simulação de falhas que servem como base para a co-emulação. Adicionalmente, serão abordados alguns dos principais métodos para a simulação de falhas.

3.1 Conceitos de simulação de falhas

A simulação de falhas é usada para determinar o efeito de uma falha em um dispositivo específico, dada uma série de vetores de entrada. Um simulador de falhas genérico utiliza como entradas o modelo do dispositivo em teste (“Device Under Test”, DUT), os padrões entrada/saída e a lista das falhas. O processo de simulação começa com a injeção de uma ou mais falhas no DUT, produzindo assim, um conjunto de circuitos com defeito. Cada circuito com defeito é simulado usando como estímulo os padrões de teste. Os vetores de saída obtidos são comparados com a resposta do circuito sem defeitos. Uma falha é detectada se a resposta do circuito com defeito for diferente da resposta do circuito sem falhas. A saída de um simulador de falhas é a lista dos padrões de teste, associados com as falhas que cada um destes detecta. A Figura 3-1 ilustra um diagrama de blocos do processo de simulação de falhas. Nesta figura, as elipses representam as entradas do processo e os retângulos as funções de transformação.

Figura 3-1: Processo de simulação de falhas

Para que o simulador de falhas seja útil, deve ser computacionalmente eficiente. Como se apresentará posteriormente neste texto, o limite superior associado à complexidade computacional da simulação de falhas é uma função quadrática. No melhor dos casos, a simulação de falhas requer $O(G^2)$ de tempo computacional, onde G é o número de componentes do dispositivo em teste.

O processo de simulação de falhas pode ser representado no espaço tridimensional, onde cada dimensão está definida como: o número de componentes no DUT (G), o número de falhas do circuito (F), e o número de vetores necessários para testar o dispositivo (I). O número de falhas no DUT é diretamente proporcional a G , e é fornecido pela expressão:

$$F = \alpha (G)$$

Da mesma forma, o número de padrões de teste em um DUT é função de G :

$$I = \beta(G)$$

O volume que deve ser avaliado pelo simulador de falhas, conhecido como “volumem de teste”, é formado pelos eixos F , I e G [Harel1987 apud Krishnaswamy1997]. Desta forma:

$$Volumen_{teste} = O(G^3)$$

Os simuladores de falhas empregam várias técnicas para reduzir o volume de $O(G^3)$ para $O(G^2)$ [Goel1980]. Por exemplo, uma forma de reduzir o número de simulações de falhas é eliminar as falhas detectadas da lista de falhas depois de que cada padrão de teste é avaliado. Este processo é conhecido como descarte de falhas ou “fault dropping”.

Para o cálculo do custo computacional da simulação de falhas se consideraram circuitos combinacionais sem laços de realimentação. O custo computacional associado à avaliação de máquinas de estados finitos é maior do que o dos circuitos combinacionais.

Outro fator que influencia o custo computacional é a forma como são avaliados os circuitos. Existem duas formas de avaliação genéricas: a simulação compilada e a simulação por eventos. Na primeira são consideradas todas as portas (G) do circuito para cada simulação de falhas. O processo de avaliação começa pela análise de cada porta do circuito sem falhas, até obter o sinal de saída; seguidamente, uma a uma, as falhas da lista são introduzidas no circuito, e para cada uma delas se executa uma simulação calculando a saída de cada porta do circuito. Na segunda forma de avaliação genérica, são analisadas unicamente as portas que produzem uma mudança no sinal que propaga um evento. Da mesma forma, como no caso anterior, o primeiro passo é avaliar todas as portas do circuito, depois introduzir cada uma das falhas e realizar a simulação, mas unicamente se avaliam as portas cujo sinal de propagação é modificado. Sendo assim, a simulação por eventos se mostra mais eficiente já que realiza menos cálculos para obter os resultados de classificação [Wang2008].

3.1.1 Simulação de falhas serial

Esta é a técnica de simulação de falhas mais simples. O primeiro passo é simular o circuito sem falhas para o conjunto de vetores de entrada, armazenando os vetores de saída. Seguidamente, são criados modelos de circuito com defeitos através da injeção das falhas da lista, onde cada circuito criado possui uma única falha. Um a

um, os circuitos com defeitos são simulados, utilizando como estímulo o conjunto de padrões de teste. Na medida em que a simulação avança, os valores de saída são comparados dinamicamente com as respostas armazenadas do circuito sem falhas. O propósito da simulação é relacionar as falhas detectadas com determinado padrão de teste. Isto se realiza ao comparar as respostas do circuito com e sem falha. Se existem diferenças na resposta de saída destes circuitos então a falha injetada pode ser detectada com o padrão de teste simulado. Todos os circuitos com falhas são simulados serialmente.

Um método utilizado para reduzir o tempo de simulação consiste em eliminar da lista as falhas detectadas, de forma que não seja necessário simular todos os padrões de teste para todas as falhas. Este método é comumente conhecido como descarte de falhas ou “fault dropping”.

Um simulador serial pode trabalhar com qualquer falha que possa ser introduzida na descrição do circuito. Desta forma, pode utilizar os modelos de falhas “stuck-at”, “stuck-open”, “bridging”, “transition delay” e analógicos. Por este motivo, falhas analógicas são usualmente simuladas com este método. Da mesma forma, o simulador de falhas serial pode facilmente representar todo tipo de condições de falhas em circuitos sequenciais, difíceis de repetir com outro tipo de simuladores. Exemplos destas condições são: falhas induzidas por corridas, glitches, perdas de inicialização, etc.

Uma das principais desvantagens desta técnica é que G portas são avaliadas para cada padrão de teste aplicado durante a simulação, embora a falha injetada, tipicamente, afete um pequeno subconjunto de G . Para F falhas, o tempo de CPU de um simulador serial pode ser cerca de F vezes o tempo de simulação do circuito sem falhas. Quando é usado o método de “fault dropping”, o tempo de CPU pode ser reduzido significativamente, especialmente se muitas falhas são detectadas por um pequeno sub-grupo de padrões de teste na parte inicial. A utilização de memória para este caso é praticamente a mesma que a da simulação do circuito sem falhas. Este tipo de simulador tem sido utilizado com sucesso em simuladores de hardware e ambientes de multiprocessamento.

3.1.2 Simulação de falhas em paralelo

A simulação de falhas paralela foi projetada para aproveitar as características da estrutura paralela dos processadores onde se executa a simulação. Para esta técnica, supõe-se que a simulação é executada em nível de portas lógicas e que o circuito é puramente combinacional. Assume-se que cada porta possui atraso zero e que todos os sinais possuem valor digital '0' ou '1'. Estes tipos de simulações são conhecidas como simulações de dois valores e atraso zero ("two-value zero-delay simulations"). A eficiência da simulação de falhas paralela é obtida através do mapeamento das portas lógicas na máquina anfitriã. Por exemplo, cada porta de duas entradas e uma saída, por exemplo as portas "AND", "OR" e "XOR", são tipicamente mapeadas em uma única instrução da máquina anfitriã. A instrução desempenha a operação Booleana desejada usando operandos de entrada e saída com tamanhos de W-bit; isto é W-bit posições de memória. Unicamente um bit da palavra de tamanho W-bit é necessário para avaliar uma única porta lógica. Conceitualmente, a execução de uma instrução pode ser vista como a avaliação de W portas em paralelo. A simulação de falhas em paralelo aproveita o tamanho de palavra da máquina para avaliar W DUTs em paralelo.

O DUT tipicamente requer G instruções Booleanas para avaliar o circuito, assumindo que cada porta no DUT possui duas entradas e uma saída. Unicamente um bit da palavra de largura W-bits seria usado para simular um único DUT. Assim, é possível simular W DUTs em paralelo com o mesmo custo computacional da simulação de um único DUT. Este tipo de simulação é conhecido como simulação de falhas em paralelo.

Existem duas técnicas para simulação de falhas em paralelo. A simulação do padrão paralelo propagando uma falha ("Parallel Pattern Single Fault Propagation", PPSFP) avalia DUTs com W padrões de teste diferentes. Quer dizer que se simulam W padrões de teste para W circuitos idênticos. Na primeira parte deste algoritmo se coletam os vetores de resposta do circuito sem falhas. Estas respostas são, posteriormente, comparadas com as obtidas na simulação dos circuitos com falhas. Cada circuito defeituoso é criado a partir da injeção de uma falha no circuito sem defeitos. Depois da comparação, determina-se qual ou quais padrões simulados

detectaram a falha. Este tipo de simulação normalmente usa a forma de avaliação por eventos de maneira que se reduz o tempo de simulação.

A segunda técnica de simulação em paralelo é conhecida como padrão único propagando múltiplas falhas (“Single Pattern Multiple Fault Propagation”, SPMFP). Esta técnica avalia W-1 dispositivos defeituosos e um dispositivo sem falhas simultaneamente. O dispositivo sem falhas é usado como referência para determinar se alguma das W-1 falhas são detectadas. Os circuitos com falhas, da mesma forma como no caso anterior, são construídos a partir da injeção de uma falha no circuito sem falhas.

Tipicamente a simulação supracitada usa a forma de simulação compilada, o que faz com que seja menos eficiente que a simulação do padrão paralelo propagando uma falha, mas W vezes mais eficiente que a simulação serial. A forma de simulação por eventos não é usada neste caso, por causa da alta complexidade e sobrecarga do vetor e eventos, visto que diferentes falhas podem causar propagações distintas em diversas partes do circuito.

As duas técnicas de simulação em paralelo têm sido entendidas de várias maneiras. Uma das extensões corresponde à simulação de múltiplos valores lógicos, por exemplo, 0, 1, Z e X, os quais podem ser codificados em dois bits. É necessário o uso do formato codificado de maneira que seja possível o modelamento das portas lógicas com as operações Booleanas disponíveis no computador. Com esta adição é possível modelar barramentos bidirecionais, portas de alta impedância, entre outros. Adicionalmente, ajudam em algoritmos que aumentam a eficiência da simulação de circuitos sequenciais. Outro tipo de extensão corresponde à implementação de distintos modelos de falha. Originalmente a simulação em paralelo foi projetada para o modelo de falhas “stuck-at”, mas foi estendida para outros modelos como conversão funcional, “bridging” e “transition delay”.

3.1.3 Simulação de falhas dedutiva

Na técnica de simulação de falhas dedutiva assume-se que o circuito foi modelado usando portas Booleanas com atraso zero ou unitário, e que os sinais podem ter

estados binários 1 ou 0. Neste método unicamente se simula o circuito sem falhas, e todos os sinais dos circuitos com falhas são deduzidos a partir desta simulação [Armstrong1972, Bushnell2000]. Como a estrutura do circuito é a mesma para os circuitos com falhas, todas as deduções são executadas simultaneamente. Desta forma, o simulador de falhas dedutivo processa todas as falhas uma única vez, fato que o faz extremadamente veloz, mas apenas quando as condições de modelamento podem ser satisfeitas.

Para o processo de simulação pode usar-se tanto a mecanismo de código compilado, quanto o direcionado por eventos. Antes de simular o vetor seguinte, aplica-se o processo dedutivo a todas as linhas seguindo a ordem de nível, das entradas para as saídas. Neste processo se gera uma lista de falhas para cada sinal. Cada lista de falhas da saída se deriva da correspondente lista das entradas da porta. Em um circuito com realimentação, a lista de falhas do sinal pode mudar várias vezes. Quando a lista permanece estável, o simulador procede com o próximo vetor.

A lista de falhas, para um sinal em qualquer momento da simulação, contém os nomes de todas as falhas em um circuito que podem mudar o estado de uma linha. A Tabela 3-1 apresenta as regras para a lista de propagação de falhas para portas de duas entradas “AND” e “OR” e a porta de uma entrada “NOT”. A lista de falhas de saída de uma porta é gerada usando as operações entre conjuntos união, interseção e complemento. As regras de propagação de falhas das portas complexas, como as “XOR” ou de portas com mais entradas, podem ser deduzidas a partir da Tabela 3-1 combinando as regras apresentadas na mesma [Menon1978].

Porta tipo	Entradas		Saídas	Lista de falhas de saída
	a	b	c	L_c
AND	0	0	0	$[L_a \curvearrowright L_b] \cup c_1$
	0	1	0	$[L_a \curvearrowright \bar{L}_b] \cup c_1$
	1	0	0	$[\bar{L}_a \curvearrowright L_b] \cup c_1$
	1	1	1	$[L_a \cup L_b] \cup c_0$
OR	0	0	0	$[L_a \cup L_b] \cup c_1$
	0	1	1	$[\bar{L}_a \curvearrowright L_b] \cup c_0$
	1	0	1	$[L_a \curvearrowright \bar{L}_b] \cup c_0$
	1	1	1	$[L_a \curvearrowright L_b] \cup c_0$
NOT	0	-	1	$L_a \cup c_0$
	1	-	0	$L_a \cup c_1$

Tabela 3-1: Lista de propagação em simulador de falhas dedutivo

Na Figura 3-2 se apresenta um exemplo do método de simulação de falhas dedutivo. O vetor de entrada corresponde aos valores A:0 e B:0, e a saída a Z:0. Cada segmento de linha está associado com um conjunto de falhas. Cada falha está identificada por uma letra, por exemplo 'a', seguido por um subíndice que indica o tipo de falha, “stuck-at-1” ou “stuck-at-0”. A lista de falhas de cada segmento é formada por todas as falhas que podem mudar o estado de uma determinada linha. Na lista de falhas da saída das portas aparecem unicamente as falhas que sensibilizam determinado caminho. Do exemplo pode se ver que na lista de falhas de 'e' só aparecem as falhas de 'f' para 'e', já que 'f' é uma entrada controladora da porta “AND”. Uma entrada é considerada como controladora se uma mudança no seu valor lógico causa uma mudança na saída da porta. No caso da porta “AND” a entrada controladora corresponde ao '0' quando suas entradas possuem os valores lógicos '1' e '0' [Bushnell2000].

entradas dos elementos de atraso. Isto completa uma iteração. Posteriormente, os valores de sinais livres de falhas e as listas de falhas nas entradas dos elementos de atraso são transferidos as suas respectivas saídas. Se isto causar alguma mudança nas entradas da lógica combinacional, executa-se outra iteração da simulação e da propagação da lista de falhas. As iterações se repetem até que todos os sinais e listas de falhas se estabilizem. Existem duas alternativas de tratamento para os sinais com comportamento oscilatório: fixar o sinal oscilante a um estado desconhecido (X), ou desconsiderar a falha que causa o problema.

Este processo chega a ser mais rápido para circuitos síncronos, quando todos os elementos de memória são implementados como “flip-flops” com relógio. Se considerar as falhas do relógio, então a simulação será similar à feita para circuitos assíncronos, explicada anteriormente. Porém, quando as falhas do sinal de relógio não são simuladas, pode-se assumir que o “flip-flop” trabalha como um elemento ideal de memória. O relógio é tratado como um sinal implícito que permite a transferência de dados através dos elementos de memória.

A técnica dedutiva requer significativamente mais memória que as técnicas de simulação em paralelo. O incremento em memória é devido à lista de falhas associada com as linhas de sinal. Adicionalmente, como a lista de falhas cresce dinamicamente é muito difícil determinar o tamanho de memória requerido para uma simulação dedutiva [Johnson1997]. A atualização da lista de falhas também introduz um gasto computacional adicional que não está presente na simulação de falhas paralela [Bushnell2000]. O custo associado com as operações entre conjuntos é proporcional ao número de entradas. De qualquer forma, a simulação de falhas dedutiva precisa de uma única passada por vetor de entrada para localizar todas as falhas detectáveis.

3.1.4 Simulação de falhas concorrentes

A simulação de falhas concorrente corresponde ao método mais genérico de simulação de falhas [Ulrich1988, Bushnell2000, Abramovici1990]. Este pode trabalhar com vários tipos de modelos, falhas, estados de sinais, e modelos

temporais. Basicamente estende, da maneira mais eficiente, a simulação dirigida por eventos para incluir a simulação de falhas. Da mesma forma como a simulação dedutiva possui a capacidade de determinar as falhas detectáveis em uma única simulação. Esta técnica pode requerer grande quantidade de memória para uma dada simulação. Adicionalmente, é difícil determinar com antecedência a quantidade de memória que será necessária para sua completa execução, esta técnica suporta a simulação de circuitos combinacionais, síncronos e assíncronos. A principal vantagem sobre a simulação de falhas dedutiva é o suporte a modelos que incorporam níveis mistos de abstração [Abramovici1990, Abramovici1977, Bose1992, Lee1992].

A técnica concorrente armazena a lista de falhas ativas na forma de componentes defeituosos, sendo que cada um destes componentes pode estar em diferentes níveis de abstração. A forma de calcular e propagar esta lista de falhas é como se define a técnica. O simulador de falhas concorrente modela o problema da seguinte forma:

Eventos: considera uma linha (sinal) do circuito sem falhas. Um evento nesta linha se conhece como evento bom. Três atributos especificam um evento bom: o nome do sinal, o tipo de mudança (ex. de 0 para 1), e o tempo em que esta ocorre. Nos circuitos com falhas a mesma linha de sinal existe, e os eventos nesses circuitos quando diferem do evento bom, são chamados eventos de falha. Um evento de falha se define através dos mesmos três atributos requeridos para o evento bom, mais o ponto e tipo da falha.

Estrutura: considera o circuito sem falhas eliminando as hierarquias. No caso, os módulos podem estar em qualquer nível de abstração. Em geral, um módulo pode ter variáveis de estado internas e múltiplas saídas. A estrutura contém a informação de conectividade (“fanouts” e “fanins”) e as funções da porta (ou do módulo). Cada porta se chama porta boa. Uma lista de falhas, usualmente na forma de uma lista de enlaces, está associada com cada porta boa. Os elementos desta lista são chamados portas ruins. Uma porta ruim não é defeituosa em si, mas é afetada por alguma falha. Pelo menos um dos seus sinais nos terminais de entrada ou saída diferem dos valores correspondentes da porta boa. A porta ruim deriva seu nome da falha que causa a diferença do seu sinal de resposta. Uma porta ruim possui a

mesma relação entre as entradas e saídas da porta boa, dado que as falhas são modeladas nos sinais e não nas funções. A estrutura resultante de portas boas com a lista de falhas associada é uma completa e compacta representação dos $n+1$ circuitos com falhas.

Falhas: Assume-se que estas são permanentes e afetam os valores de sinal nos terminais ou nos estados internos dos módulos. A informação da falha, local e tipo, armazena-se com a porta boa conectada no ponto da falha. As falhas nas entradas e saídas primárias podem ser modeladas indexando “buffers”. Quando os valores de sinal de uma porta boa ativam uma falha, insere-se uma porta ruim na lista de falhas dessa porta boa.

O processo de simulação ocorre através do agendamento e processamento de eventos, registrando a atividade do circuito. Usualmente se usa o modelo de atraso unitário, embora seja possível usar qualquer modelo de atraso. Ativa-se uma falha quando o valor do sinal, no ponto de análise, difere do estado defeituoso. Neste ponto se introduz instantaneamente uma porta ruim com o nome da falha e os valores dos sinais que diferem.

Todos os eventos bons e os eventos de falha ativam portas boas para que sejam avaliadas. Os eventos bons também ativam portas ruins. Agora, os eventos de falha unicamente ativam portas ruins com o mesmo nome de falha. Todas as portas ativas são avaliadas e podem potencialmente gerar novos eventos. Eventos bons só podem ser gerados quando a ativação de uma porta boa é causada por um evento bom. Todos os eventos que ativam portas ruins são eventos de falha. Durante a avaliação, são removidas as portas ruins cujos sinais se tornem idênticos aos correspondentes da porta boa. Este processo se conhece como convergência e se diz que a porta ruim convergiu à porta boa. Divergência é o processo de criar novas portas ruins, tanto pela ativação de uma falha ou pela avaliação de um evento de falha em uma porta boa.

O exemplo da Figura 3-3 ilustra o algoritmo de simulação de falhas concorrentes. Nesta figura se pode ver o circuito da função “XOR”. As portas boas são as que estão conectadas no circuito e os seus valores simulados estão em cada um dos seus terminais. As portas ruins estão na lista de falhas de cada porta boa. Cada porta ruim está identificada com o nome da falha que a originou: o nome do sinal

com o subíndice do tipo de falha (“stuck-at-0” ou “stuck-at-1”). Por exemplo, a falha a1 corresponde à falha s-a-1 do sinal 'a'.

O algoritmo de simulação concorrente é similar à simulação por eventos. Os vetores de entrada disparam eventos que serão avaliados nas portas ligadas às entradas primárias. Na medida em que cada porta é avaliada, esta cria a lista de falhas com as portas ruins. As portas ruins que produzem saídas diferentes propagam eventos em todos os componentes que usam esse sinal como entrada.

Na Figura 3-3 pode-se ver a lista de falhas para cada porta. Dentro desta lista existe um subgrupo que corresponde às falhas locais. As portas ruins que possuem um asterisco na frente, correspondem às portas que geram uma saída diferente e que propagam a falha. O algoritmo termina quando não existem eventos a serem avaliados na lista de eventos e o resultado é a lista de falhas detectadas com esse vetor de entrada.

Comparando os resultados obtidos na simulação de falhas dedutiva com os da concorrente, nota-se que estes são coerentes, mas que a lista de falhas da primeira é menor. Na simulação dedutiva a lista de falhas está associada aos sinais, contendo unicamente os valores que afetam esse sinal. Na concorrente a lista de falhas está associada às portas, incluindo, inclusive, os eventos de falha que afetam as entradas dessas portas, dando como resultado listas de falhas maiores. Assim sendo, pode se dizer que as listas de falhas da simulação concorrente consistem em um conjunto que abrange a lista de falhas dedutivas [Abramovici1990]. A vantagem disto, embora não muito claro na simulação de falhas lógica, tem significado quando se simulam módulos funcionais mais complexos como memórias e modelos comportamentais [Bushnell2000].

No exemplo foram ilustradas algumas características do algoritmo de simulação de falhas concorrente. Outra técnica útil é a lista múltipla transversal (“multi-list traversal”, MLT), a qual permite a simulação de funções com múltiplas saídas que podem ter estados internos [Bose1992]. Suas vantagens são a eficiência e flexibilidade nos modelos. Alguns simuladores de falhas concorrentes notáveis são o acelerador de “hardware” MARS [Agrawal1990, Bose1992], Creator [Gai1994], MOTIS [Lo1987], e FMOSSIM [Gunes2005]. Com exceção de FMOSSIM, que

modela o circuito em nível de interruptores, todos usam níveis de modelamento misto (transistor, lógico, e funcional) [Bushnell2000].

Figura 3-3: Exemplo de simulação de falhas concorrente

3.1.5 Simulação de falhas diferencial:

Esta técnica é usada principalmente para avaliar circuitos sequenciais, expandindo o método de simulação de falhas serial. A simulação de falhas diferencial usa os conceitos de atraso zero de propagação e sinais binários.

A principal vantagem deste algoritmo é eliminar a restauração explícita do circuito, depois de cada simulação. Para isso, utiliza a ordenação da lista de falhas de forma que falhas próximas ativem as mesmas portas, e a simulação dirigida por eventos para modificar o valor do mínimo número de portas possíveis entre cada execução. Para exemplificar o algoritmo considera-se o seguinte procedimento para circuitos combinacionais:

1. Ordena-se a lista de falhas de forma que falhas subsequentes ativem quase as mesmas portas. Em [Cheng1989, Cheng1990] explica-se várias maneiras de organização de falhas.
2. Simula-se o circuito sem falhas usando o primeiro vetor de entrada junto com o vetor do estado atual, utilizando a técnica de simulação por eventos. O resultado deve ser armazenado.
3. Ativa-se uma falha de forma que o valor real no local da falha é oposto ao valor de falha, agendando um evento nesse ponto. Por exemplo, se o valor real do local é 0 e está se avaliando a falha s-a-1, então agrega-se o evento $0 \rightarrow 1$ na tabela de eventos de simulação.
4. Propaga-se o evento até que o circuito se estabilize. Se qualquer nova saída difere do valor salvo do resultado do circuito sem falhas, então a falha é detectada e pode ser removida da lista de falhas ("fault dropping").
5. Uma a uma as falhas devem ser introduzidas no circuito para simulação, voltando ao passo número três. Concorrentemente, a falha anterior deve ser removida invertendo seu valor.
6. Depois que foram exercitadas todas as falhas do circuito, muda-se o vetor de entrada com seu correspondente estado e executa-se novamente o algoritmo desde o passo três.
7. O algoritmo termina quando todos os vetores de entrada foram exercitados.

Para a simulação dos circuitos sequenciais se avalia o conjunto de vetores de entrada. Cada um destes vetores produz como resultado os sinais de saída do circuito e o estado do mesmo. Para a simulação de falhas diferencial, se utilizam como entradas o modelo do circuito e os conjuntos de vetores de entrada ($i[n]$), dos estados ($s[n]$) e das falhas ($f[n]$). Antes de iniciar o algoritmo deve-se interromper a realimentação dos sinais do próximo estado. Estes sinais são tratados independentemente, de forma que a simulação se inicia com os sinais de estado atual, associados a cada um dos vetores de entrada, e se armazena o valor do próximo estado.

A técnica de propagação singular de falhas oferece a máxima velocidade para circuitos combinacionais, enquanto que a simulação de falhas concorrentes oferece a máxima velocidade para circuitos sequenciais [Cheng1989].

Como a diferença física entre um circuito com defeito e um sem se encontra unicamente no ponto da falha, o estado dos dois circuitos é muito similar. Então, ao invés de armazenar o estado de todos os circuitos com falhas, um simulador de falhas concorrente armazena o estado das diferenças entre cada circuito com falhas e o sem falhas.

A simulação concorrente de falhas (“concurrent fault simulation”) possui uma alta eficiência na simulação de circuitos sequenciais, quando comparada com as outras técnicas descritas anteriormente. Ela se vale do fato que a diferença entre o estado do circuito com falhas e sem é mínima, portanto a simulação só precisa modificar o estado do circuito a partir do local da falha. Apesar da simulação concorrente de falhas ser tão eficiente, a manipulação das células da lista de falhas dinâmica acaba gerando um custo alto de desempenho. A inserção, restauração e remoção de falhas necessitam de algoritmos especializados de procura e classificação, como também de gerenciamento dinâmico de memória. Esta sobrecarga no desempenho pode ser excessiva quando o tamanho da lista de falhas é grande. Se a lista de falhas excede a memória disponível, ela deve ser particionada e as falhas simuladas separadamente. A simulação do circuito sem falhas deve, então, ser repetida para cada uma das partições.

A propagação singular de falhas (“single fault propagation”) é a técnica de simulação mais rápida para circuitos combinacionais. Ela usa uma porção mínima de memória.

Similar à simulação concorrente de falhas, a propagação singular de falhas simula o efeito de cada uma nos circuitos com defeito. Como cada máquina é simulada separadamente, existe uma sobrecarga no momento de restaurar o estado do circuito sem falha antes de cada nova simulação. Para circuitos combinacionais existe uma série de heurísticas que permitem diminuir o número de circuitos defeituosos simulados [Antreich1987, Maamari1988]. Nos circuitos sequenciais, por outro lado, estas heurísticas não podem ser utilizadas devido ao fato do estado do circuito depender de valores desconhecidos ou as falhas podem afetar os resultados em diferentes tempos. Desta forma, cada circuito defeituoso deve ser simulado explicitamente e a sobrecarga para restaurar o estado do circuito sem falhas, antes de cada nova simulação, acaba sendo considerável.

A simulação de falhas diferencial (“differential fault simulation”, DSIM) tenta combinar a eficiência da técnica de simulação concorrente de falhas com o uso mínimo de memória da propagação singular de falhas. O sucesso das duas técnicas, explicadas anteriormente, encontra-se no fato de que unicamente são simuladas as diferenças entre os circuitos com falhas e o circuito bom. Agora, no caso da simulação concorrente de falhas, o estado do circuito com a falha $k+1$ com o vetor de entrada $i+1$ se constrói a partir do mesmo circuito com o vetor de teste anterior (i). Isto funciona bem com exceção do uso excessivo de memória e a sobrecarga na manipulação das células da lista de falhas. Por outro lado, a propagação singular de falhas constrói o estado do circuito com a falha $k+1$ a partir do estado do circuito sem falhas, usando o mesmo vetor de entrada. Este método funciona bem para circuitos combinacionais, mas para sequenciais, introduz uma sobrecarga para restaurar o estado do circuito bom antes de executar a simulação de um circuito defeituoso.

A simulação de falhas diferencial propõe a construção do estado do circuito com falha $k+1$ a partir do circuito com falha k , usando o mesmo vetor de entrada (i). Desta forma, o circuito com falha $k+1$ pode ser gerado a partir do circuito com falha k através da simulação das diferenças dos pontos de falha e as diferenças concorrentes dos seus estados. Esta técnica utiliza a simulação orientada a eventos que introduz sobrecarga mínima.

A continuação segue a descrição do algoritmo da simulação de falhas diferenciais:

Para cada vetor de teste V_i

{

/* Computar o estado do circuito sem falhas */

Se (V_i é o primeiro vetor)

 inicializar o estado do circuito;

senão

 remover a falha previamente injetada;

restabelecer o estado atual;

introduzir o vetor V_i nas entradas primárias;

executar a simulação dirigida a eventos;

armazenar os valores das saídas primárias;

configurar o contador de saídas sensibilizadas para 0;

para cada circuito com defeito não detectado, B_j

{

 remover a falha previamente injetada;

 restabelecer os estados atuais;

 injetar a falha atual;

 executar simulação dirigida a eventos;

 se o contador de saídas sensibilizadas possui um valor positivo

 descartar a falha;

 }

}

Desta forma sendo $B_{m,i}$ o circuito com falha m avaliado com o vetor de entrada i , e sendo G_i o circuito sem falhas avaliado com o vetor i , a sequência de simulação será: ... G_i ; $B_{1,i}$; $B_{2,i}$; ..., $B_{m,i}$; G_{i+1} ; $B_{1,i+1}$, ...

O contador de saída sensibilizada é usado para reduzir o custo de comparar todos os valores das saídas primárias depois de cada simulação. Ele é inicializado em 0 depois da simulação do circuito sem falhas para cada vetor de teste. Durante a simulação do circuito defeituoso, o contador se ajusta para armazenar o número de saídas primárias que diferem do circuito sem falhas. Depois da simulação de cada circuito com falhas, se o contador não é zero, a falha é detectada.

Como não existe a necessidade de usar o circuito sem falhas como referência para determinar e armazenar as diferenças reduz-se altamente o uso de memória para a simulação atual. O total de memória requerida é para armazenar o estado do circuito (a cópia que está sendo simulada), as saídas primárias do circuito sem falhas (para verificar as falhas detectadas) e a informação da diferença de estado de todos os circuitos. A simulação concorrente de falhas requer a informação da diferença em cada nó do circuito, enquanto que a simulação de falhas diferencial requer o valor de diferença nos elementos de estado. Como o número de elementos de estado é muito menor que o número total de nós, a DSIM possui o menor custo de memória quando comparado com outros métodos explicados anteriormente.

Para aumentar a eficiência do algoritmo, algumas técnicas de ordenação da lista de falhas foram desenvolvidas, de forma que falhas subsequentes causem aproximadamente as mesmas mudanças no circuito.

3.1.6 Simulação de falhas hierárquica:

Esta técnica se vale do fato de que a quantidade de recursos computacionais necessários para simular um circuito está relacionada com o nível de abstração com que foi modelado. De forma geral, quanto maior o nível de abstração, menor é o custo computacional para a simulação. Por outro lado, o aumento no nível de abstração, diminui a quantidade de informação fornecida [Johnson1997].

Adicionalmente, a metodologia para o desenvolvimento de circuitos integrados complexos considera a divisão do chip em vários submódulos, que podem ser projetados de forma independente. Com o estabelecimento da hierarquia modular se reduz a complexidade do desenvolvimento.

A simulação de falhas hierárquica propõe uma forma de explorar a estrutura inerente do projeto de circuitos integrados, executando uma simulação mista de falhas. Todos os submódulos são modelados com o nível de abstração mais alto, nível funcional, com exceção do submódulo onde se analisa a falha, o qual é tratado em nível de portas lógicas ou de transistor [Meyer1993, Motohara1988]. Conceitualmente, a simulação de falhas hierárquica reduz o custo computacional de simulação de cada falha, como se explica a seguir.

Seja T o tempo requerido para avaliar uma porta lógica, e g_i o custo equivalente de número de portas no submódulo i no nível de abstração de portas lógicas; o tempo total de simulação do DUT, modelado em nível de portas lógicas T_g é:

$$T_g = \sum_{i=0}^n g_i T$$

Seja h_i o custo equivalente de número de portas no submódulo i no nível de abstração funcional. O tempo total de simulação do DUT usando a técnica de simulação híbrida T_h , onde o submódulo j é simulado em nível de portas lógicas e os módulos restantes em nível funcional, é igual a:

$$T_h(j) = g_j T + \sum_{i \in (1,2,...,n) \forall (j \neq i)} h_i T$$

Então, a relação entre os tempos de simulação resulta no fator multiplicador de velocidade de simulação S_u , assim:

$$S_u = \frac{T_g}{Th}$$

$$S_u = \frac{\sum_{i=0}^n g_i T}{g_j T + \sum_{i \in (1,2,\dots,n) \forall (i=j)} h_i T}$$

Esta equação pode ser simplificada usando como aproximação os custos médios de simulação para os níveis de abstração de portas lógicas e funcional, g e h , respectivamente. Desta forma:

$$S_u = \frac{ng}{g + (n - 1)h}$$

Para exemplificar o ganho de velocidade de simulação, imagine um circuito com 10 submódulos. Suponha também que o custo médio para simular um submódulo em nível de portas lógicas é 1000 e que o custo médio para simular um submódulo em nível funcional é 50. Desta forma, o ganho de tempo na simulação hierárquica é igual a 6,9 vezes o tempo gasto pela simulação em nível de portas lógicas.

É interessante salientar que se o número de submódulos é grande, o ganho da velocidade na simulação hierárquica se aproxima da relação entre g e h , que é o valor máximo do aumento da velocidade. No exemplo anterior, o fator de aumento na velocidade de simulação máximo é de 20 vezes a de uma simulação de falhas usando outra técnica.

A estratégia de simulação hierárquica é normalmente combinada com a simulação de falhas concorrente, já que esta última suporta qualquer nível de abstração.

4 Emulação de falhas

O objetivo da emulação de falhas, da mesma forma como a simulação, é determinar quais defeitos físicos dos circuitos integrados podem ser detectados com a execução de determinado padrão de teste. As falhas são detectadas através da comparação do resultado da aplicação de um vetor de teste nos circuitos sem e com falhas.

As alternativas de simulação de falhas baseadas em software consomem muitos recursos computacionais e tempo de projeto. O crescente aumento da complexidade dos circuitos integrados, a demanda por qualidade nos produtos e processos, e o esforço para diminuir o tempo entre a análise de um produto e sua disponibilização para a venda, são fatos motivadores para o desenvolvimento de novas técnicas de graduação de falhas. A emulação de hardware se apresenta como uma alternativa que pode ser estudada e explorada, de forma que se acelere o processo de testabilidade no projeto de CIs e se diminua seu custo [Sedaghat1999a].

Um emulador lógico consiste em uma plataforma computacional reprogramável que pode ser configurada para implementar a função de um circuito como se fosse um protótipo [Hauck2007]. Esta implementação de “hardware” é criada para conseguir resultados de avaliação, imitando com precisão o funcionamento do circuito projetado, antes mesmo de ser fabricado. Da mesma forma que a simulação, a emulação pode facilitar a detecção e isolamento de falhas no projeto, mas sua execução chega a ser de 10^3 a 10^6 vezes mais rápida [Khan1993, Hauck2007].

A simulação de falhas é uma tarefa que demanda muitos recursos computacionais, visto que compara o resultado da aplicação de cada vetor de teste em todos os possíveis circuitos com falhas. Dependendo da complexidade do circuito e a técnica de projeto para testabilidade utilizada, a simulação de falhas pode ser extremamente dispendiosa ou até mesmo impossível [Giangarra2008]. A emulação de falhas, então, apresenta-se como uma alternativa interessante para acelerar a obtenção de resultados, principalmente quando se deseja avaliar a contribuição de cobertura de padrões funcionais à cobertura total do “chip” [Giangarra2008].

Neste capítulo se apresentarão os principais conceitos e o estado da arte das técnicas de emulação de falhas.

4.1 Emulação de falhas baseada em FPGAs

De forma geral, as plataformas de emulação de falhas utilizam dois tipos de dispositivos eletrônicos como base: “Field Programmable Gate Arrays” (FPGAs) ou lógica de múltiplos processadores. Um FPGA é um circuito integrado que permite criação de chips de aplicação específica, eliminando o tempo do processo de fabricação. Ele pode ser visto como um conjunto de blocos lógicos reconfiguráveis, interconectados através de uma matriz programável.

Em princípio, é possível implementar em um FPGA qualquer circuito digital, enquanto existam recursos disponíveis, utilizando uma configuração apropriada. O objetivo de uma plataforma reconfigurável é se aproximar à eficiência de um circuito lógico especializado, enquanto oferece a flexibilidade da reutilização dos processadores de propósito geral [Vassiliadis2007].

Esta arquitetura é ideal para a emulação de “hardware”, já que permite replicar as características e o funcionamento de um circuito digital antes mesmo de ser fabricado, de forma similar à simulação, mas várias vezes mais rápido. Igualmente, é possível usar este tipo de arquitetura para a emulação de falhas disponibilizando ferramentas para o estudo do comportamento do circuito defeituoso, através da injeção de circuitos que imitem as falhas de fabricação, de forma rápida e eficiente.

Como se pode ver na Figura 4-1, os componentes principais de um FPGA são: blocos lógicos, recursos de interconexão e células de entrada/saída. A implementação de um circuito digital é feita através da programação de uma pequena porção da lógica requerida, em cada um dos blocos lógicos. Os recursos de interconexão são programados para fazer as conexões necessárias entre os distintos blocos lógicos, e entre estes e as células de entrada/saída. A complexidade dos blocos lógicos pode variar desde uma operação booleana simples entre duas entradas, até operações complexas entre múltiplos bits.

Na Figura 4-2 se apresenta o diagrama de blocos genérico de um bloco lógico reconfigurável. Usualmente, é constituído por “Lookup Tables” (LUTs), lógica de “vai um”, “flip-flops” e multiplexadores programáveis. Os multiplexadores são usados para formar os caminhos de dados (“data-paths”) dentro do bloco lógico e para conectar o mesmo com os recursos de interconexão. A parte combinacional pode ser registrada no “flip-flop” da saída e caminhos de realimentação da lógica podem ser ativados. Além da lógica especializada em “vai um”, alguns FPGAs possuem recursos adicionais que permitem aumentar o seu desempenho.

Figura 4-1: Elementos básicos de um FPGA

Figura 4-2: Diagrama genérico de um bloco lógico reconfigurável

Um bloco lógico reconfigurável é responsável por implementar a função projetada para cada aplicação em nível de portas. É definido pela sua estrutura e granularidade. A estrutura indica os distintos tipos de lógica que podem ser implementadas no bloco, enquanto a granularidade define o máximo tamanho de palavra (W) da função implementada. Uma função lógica é configurada através do controle da conectividade de algumas portas lógicas básicas ou através da programação das LUTs. A vantagem de uma LUT de ' k ' entradas é que pode realizar qualquer função combinacional de ' k ' entradas [Vassiliadis2007].

A interconexão entre blocos lógicos, e entre os mesmos e as células de entrada/saída, é efetuada através de chaves programáveis. O método de disponibilização da conectividade possui um grande impacto nas características da arquitetura do FPGA. Os tipos de interconexão podem ser classificados em cinco grupos: estilo de ilhas, baseado em colunas, “sea-of-gates”, hierárquico e estruturas unidimensionais [Vassiliadis2007].

A tecnologia de programação determina o método de armazenamento da informação de configuração, e possui um grande impacto na área e desempenho do dispositivo (potência consumida, tempo de reconfiguração, etc). As tecnologias de programação mais usadas são: “Static Random Access Memory” (SRAM), “antifuse”, e tecnologias não voláteis [Vassiliadis2007].

Cada elemento configurável em um FPGA requer um bit de armazenamento para manter a programação definida pelo usuário. Para um FPGA baseado em LUTs, estes espaços de programação usualmente estão constituídos pelo conteúdo dos blocos lógicos e a conectividade dos recursos de roteamento. A configuração de um FPGA é feita através da programação dos bits de armazenamento das partes programáveis, de acordo com a definição de usuário. Para as LUTs, isto se traduz em preenchê-las com 1s e 0s; e para os recursos de roteamento, habilitar ou desabilitar as chaves dos caminhos de interconexão [Hauck2007].

A configuração pode ser feita através de um arquivo binário cujo conteúdo mapeia, bit a bit, os locais programáveis do FPGA. Este arquivo é conhecido como “bitstream”, e é gerado usando uma ferramenta específica do distribuidor do FPGA. O formato do “bitstream” não é público, já que ele revela a informação da propriedade intelectual da arquitetura do FPGA [Hauck2007]. O carregamento de distintos “bitstreams” é o que se conhece como reconfiguração, e através dela é possível mudar a funcionalidade do “chip”.

Existem vários modelos de reconfiguração usados, dependendo da aplicação de usuário que se deseja, que são: estático, dinâmico, mono-contexto, múltiplos-contextos, parcial e “pipeline”.

A reconfiguração estática, conhecida também como reconfiguração em tempo de compilação, é a mais simples e comum para a implementação de aplicações com lógica reconfigurável. Ela envolve mudanças no “hardware” a taxas relativamente baixas: horas, dias ou semanas. Nesta estratégia, cada aplicação consiste em uma configuração. Muitos dos sistemas reconfiguráveis existentes são estaticamente reconfiguráveis. Para reconfigurar este tipo de sistemas, a sua execução deve ser detida enquanto a reconfiguração está em processo, e restaurada com o novo programa [Vassiliadis2007].

Por outro lado, a reconfiguração dinâmica [ADRIATIC2002], também conhecida como reconfiguração em tempo de execução (“run-time reconfiguration”), usa um esquema de alocação dinâmica que realoca “hardware” em tempo de execução. Nesta técnica existe uma relação direta entre tempo e espaço. Esta, por sua vez, pode aumentar o desempenho do sistema através do uso de circuitos altamente otimizados que são carregados e descarregados dinamicamente durante a

operação. A reconfiguração dinâmica está baseada no conceito de “hardware” virtual, o qual é similar à ideia de memória virtual. Neste caso, o “hardware” físico é muito menor que a soma dos recursos requeridos para todas as configurações. Desta forma, ao invés de reduzir o número de configurações que são mapeadas, é preferível a instalação e desinstalação do “hardware”, na medida em que este é necessitado.

Os FPGAs mono-contextuais possuem uma única configuração por vez. Eles podem ser programados usando uma sequência serial da informação de configuração. Visto que é permitido, unicamente, o acesso sequencial, cada mudança de configuração requer a reprogramação do “chip” inteiro. Apesar de que isto simplifica o “hardware” de configuração, agrega-se uma sobrecarga e demora, quando unicamente muda uma pequena porção da memória que precisa ser atualizada.

Os FPGAs de múltiplos contextos incluem múltiplos bits de memória para cada bit de programação [Trimberger1997, Kapre2006]. Estes bits de memória são considerados como múltiplos planos de informação de configuração, que podem ser ativos em determinado momento, mas estruturados de forma tal que o dispositivo pode rapidamente trocar entre um plano ou contexto já programado. Estes dispositivos permitem o carregamento de uma nova configuração, enquanto outro plano está ativo e em execução. Desta forma, o tempo de troca entre planos é da ordem de nano segundos, ao invés de mili segundos ou mais, como ocorre nos outros tipos de reconfiguração.

Em alguns casos, uma configuração não ocupa completamente o “hardware” reconfigurável, ou apenas uma parte da configuração requer modificação. Nas duas situações, é necessário uma reconfiguração parcial da matriz, ao invés de uma configuração completa como ocorre nos FPGAs mono ou de múltiplos contextos. Em um FPGA parcialmente reconfigurável, a camada de configuração opera como uma memória RAM, de forma que é possível endereçar seletivamente uma parte da sua configuração com o objetivo de mudar seu conteúdo. Frequentemente, as partes da matriz que não são acessadas podem continuar com sua execução normal, permitindo sobrepor a computação com a reconfiguração. Adicionalmente, algumas aplicações requerem a atualização de uma única parte do circuito mapeado, enquanto o resto deve se manter intacto. Usar esta reconfiguração seletiva pode

diminuir consideravelmente o tamanho dos dados de configuração que devem ser transferidos para o FPGA.

A reconfiguração “pipeline” é uma modificação da reconfiguração parcial, na qual a reconfiguração ocorre por estágios [Schmit1997]. Cada estágio é configurado como um todo. Este estilo de reconfiguração é usado principalmente no estilo de computação com “datapaths”, onde são empilhadas mais partições das que poderiam ser encaixadas simultaneamente no hardware disponível.

Uma plataforma de emulação de circuitos lógicos que usa FPGAs, normalmente consiste em uma placa com múltiplos FPGAs conectados, e um “software” que se encarrega do particionamento, mapeamento, síntese, configuração e monitoramento dos distintos dispositivos usados. Os monitores são automaticamente introduzidos, dependendo da configuração dada pelo usuário; e o controle da execução é centralizado no computador anfitrião. As tarefas de particionamento, roteamento, etc. podem levar horas, dependendo da complexidade do circuito que se deseja emular; mas a execução da emulação, introduzindo os vetores de entrada ao circuito, é extremadamente rápida, já que a avaliação do resultado está mapeada diretamente no hardware. Esta característica faz com que a emulação de lógica seja uma tecnologia interessante na avaliação de vetores de teste funcionais, onde se deseja determinar a sua contribuição na cobertura total do circuito projetado.

Como se verá neste capítulo, existem várias técnicas de emulação de falhas desenvolvidas, as quais usam alguns dos métodos de reconfiguração apresentados anteriormente. Basicamente, o princípio sobre a emulação de falhas é fazer réplicas do circuito original, introduzindo elementos que imitem cada um dos possíveis defeitos da lista de falhas extraída, aplicando um determinado modelo de falhas ao circuito avaliado. Seguidamente, submetem-se a cada um dos circuitos os respectivos padrões de teste e se compara os resultados com o obtido no circuito sem falhas. As diferenças entre as respostas obtidas indicam a detecção de uma falha com o padrão avaliado. As falhas detectadas divididas entre o universo de falhas determinam a contribuição em cobertura dos padrões de teste. A vantagem da emulação de falhas, em comparação com as técnicas de simulação, é reduzir o tempo de avaliação para determinar o número de falhas detectadas por determinado padrão.

4.2 Emulação de falhas baseada em múltiplos-processadores

A emulação de falhas usando múltiplos processadores é muito parecida com a que usa FPGAs; a diferença se encontra principalmente na forma como o circuito projetado é mapeado diretamente sobre os processadores, e o tempo que estes levam para avaliar o resultado. Geralmente, a frequência da emulação usando uma plataforma baseada em FPGAs é de duas a três vezes mais rápida do que a baseada em múltiplos processadores [Sedaghat1999a].

Um emulador lógico baseado em múltiplos processadores opera com uma arquitetura dedicada, baseada em processamento paralelo. Os componentes básicos destes processadores são também blocos lógicos programáveis, os quais são consideravelmente mais simples do que os blocos lógicos configuráveis dos FPGAs. O emulador baseado em múltiplos processadores é composto por uma matriz de circuitos integrados adaptados para operar coletivamente e emular circuitos lógicos complexos. Cada processador emula uma pequena porção do circuito através da avaliação de operações sequenciais durante cada ciclo de relógio do sistema.

O circuito a ser emulado é dividido em pequenas funções lógicas que serão associadas a cada bloco programável. As funções lógicas distribuídas entre os processadores correspondem à divisão do circuito em passos temporais, de forma que cada bloco programável avalie funções cujas mudanças ocorrem na mesma fração de tempo. As frações de tempo são agrupadas e avaliadas sequencialmente durante a emulação, de forma que os resultados de um passo ou fração de tempo sejam usados como entradas do passo seguinte.

O máximo número de passos temporais que podem ser processados em paralelo é determinado pelo maior caminho combinacional do projeto. Um grande número de passos temporais leva a uma frequência menor de emulação. Atualmente uma plataforma de emulação baseada em múltiplos processadores suporta circuitos com até 256 milhões de portas lógicas [Cadence2009]. Este tipo de arquitetura pode

avaliar milhões de ciclos de circuito por segundo, devido ao fato das funções do circuito completo serem calculadas em paralelo.

4.3 Estado da arte da emulação de falhas em circuitos integrados

As técnicas para emulação de falhas foram desenvolvidas dada a necessidade de aumentar a eficiência da classificação de falhas de padrões de teste. No capítulo dois foram apresentados vários algoritmos de simulação de falhas, os quais, embora compactos e eficientes, consomem muito tempo para a gradação de falhas de circuitos complexos.

Adicionalmente, foram desenvolvidos algoritmos que permitem a paralelização das tarefas de simulação de falhas, distribuindo estas em clusters ou redes de múltiplos processadores [Narayanan1989, Patil1991]. Para reduzir a quantidade de cálculos dos algoritmos de simulação de falhas, algumas técnicas de projeto para testabilidade foram desenvolvidas [Crouch1999]. Nestas, basicamente, diminui-se a complexidade do circuito através do particionamento do mesmo em pequenas unidades combinacionais, aumentando assim a controlabilidade e observabilidade. As técnicas de DFT combinadas com as de paralelização de tarefas, aumentam drasticamente a eficiência da geração de vetores de teste e simulação dos mesmos para a classificação de falhas. Embora, na medida em que aumenta o tamanho dos caminhos, o número das unidades combinacionais e a complexidade do circuito, diminui-se a capacidade da simulação de falhas e se aumenta a necessidade de recursos computacionais.

Dependendo das características ou requerimentos do projeto, existem casos onde não é possível agregar as estruturas de teste das técnicas de DFT, o que aumenta consideravelmente o trabalho computacional da simulação de falhas, já que se diminui a possibilidade de paralelizar tarefas, de controlar e observar. Nestes casos são utilizados padrões funcionais para detectar falhas. Dependendo da complexidade do circuito, a classificação das falhas pode chegar a ser uma tarefa inviável e de extrema complexidade [Giangarra2008].

É possível que mesmo tendo adaptado o circuito seguindo as técnicas de DFT, não se consiga um valor de cobertura que esteja de acordo com as exigências da regulamentação [AEC2008]. Alguns dos motivos mais comuns para que os valores de cobertura desejados não sejam atingidos são:

- 1) O número de falhas da parte que configura o circuito para teste não é desprezível, em relação ao número de falhas do restante do circuito. É de se esperar que a parte que configura o circuito para teste se mantenha estática durante todo o tempo em que este é realizado, e não inclui nenhuma estrutura adicional para teste, já que não se deseja que os sinais de configuração sejam alterados.
- 2) Os caminhos de realimentação e circuitos assíncronos são adaptados para se ajustar às regras de projeto para testabilidade; da mesma forma, todos os sinais de relógio e de reset devem ser controlados por entradas primárias. Isto significa que, em teste, vários caminhos funcionais de usuário são alterados ou desativados, impossibilitando a verificação de algumas falhas e mudando sua classificação.
- 3) Modelos incompletos de “hardblocks” onde todas as falhas da lógica combinacional associadas aos seus “fan-ins/fan-outs” não possuem uma relação adequada com as suas saídas, de forma que estas falhas possam ser observadas e/ou controladas a partir de outros pontos do circuito. Além disso, qualquer bloqueio tanto nos sinais de entrada, quanto saída ou até mesmo internos, podem fazer com que não sejam exercitados distintos valores de sinal.
- 4) A existência de qualquer sinal de configuração que bloqueie o exercício de algum caminho ou lógica. Têm vários destes sinais na configuração dos pinos de entrada/saída do “chip”, se o número de pinos é reduzido o impacto na cobertura devido ao número de sinais fixos pode ser considerável.

Uma solução para aumentar a cobertura de falhas, é a utilização de padrões funcionais, onde se exercite a detecção e classificação adequada da lógica que configura o circuito para os distintos modos de operação. Dessa forma, é possível realizar uma melhor classificação das falhas, determinando a contribuição dos padrões funcionais na cobertura total. Adicionalmente, os modelos funcionais de “hardblock”, embora comportamentais, normalmente são mais completos que os usados para a geração de padrões de teste usando técnicas de DFT. Como foi

mentionado anteriormente, a tarefa de simulação de falhas de padrões funcionais é complicada e demanda tanto tempo quanto recursos computacionais.

Assim sendo, foram desenvolvidas algumas alternativas baseadas em sistemas de emulação lógica para a classificação de falhas.

4.3.1 Emulação de falhas serial.

Wieler et al [Wieler1995] e Burgun et al. [Burgun1996] propuseram técnicas similares onde se extrapola o algoritmo de simulação de falhas serial para a emulação. O processo descrito nestes algoritmos começa pela extração da lista de falhas, aplicando o modelo “single stuck-at” colapsado. Depois, são gerados os arquivos de configuração do circuito sem falhas que serão carregados na matriz de FPGAs, para o qual, a lista de conexão original é mapeada nos blocos lógicos dos FPGAs. A continuação, o algoritmo entra em um laço onde é gerado um novo “bitstream” para cada falha da lista; para isto é necessária a recompilação do circuito, de forma que seja modificado para se comportar da mesma forma como se comportaria se a falha existisse. A emulação de uma falha “stuck-at” em um FPGA corresponde simplesmente à modificação do circuito associado a um bloco lógico; desta forma, uma heurística aplicada a este método para diminuir o tempo de geração, é evitar a recompilação global do circuito e fazer este procedimento unicamente no FPGA onde se encontra a falha a emular [Wieler1995]. As técnicas elaboradas nestes trabalhos usam a reconfiguração estática total, mas ao mesmo tempo parcial, já que consegue endereçar o FPGA que se deseja mudar e reconfigurar o mesmo. Os “bitstreams” gerados são usados para reconfigurar a matriz de FPGAs, para depois aplicar os padrões de teste. A resposta do circuito a estes padrões é comparada com a obtida na avaliação do circuito sem falhas, previamente armazenada. Se existirem diferenças entre estas respostas significa que essa sequência de teste pode detectar a falha. Este processo continua até ter analisado e emulado todas as falhas da lista. Finalmente, efetua-se o cálculo da cobertura de falhas dos padrões analisados, dividindo o número das falhas detectadas entre o número de falhas da lista.

O tempo total da execução do processo de emulação está dividido basicamente em dois fatores: 1) o tempo de recompilação e reconfiguração. Para cada falha injetada, são necessários alguns cálculos para determinar o que deve ser modificado nos FPGAs. O tempo utilizado para esta computação se conhece como tempo de recompilação. A reconfiguração do FPGA também considera um tempo, que é constante, correspondente à duração da configuração de um FPGA completo. Fazendo 'T' como a soma do tempo de recompilação e reconfiguração para uma falha, o tempo total para esta tarefa é ' $n \cdot T$ ', onde 'n' é o número total de falhas da lista de falhas. 2) O tempo de emulação. Tipicamente um emulador lógico pode trabalhar em frequências na ordem de MHz, de forma que é possível avaliar uma grande quantidade de vetores de teste em curtos períodos de tempo. Isto significa que o tempo de recompilação e reconfiguração dominam a maior parte da execução do algoritmo de emulação de falhas serial.

4.3.2 Método de Cheng-Huang&Dai

Este método faz duas melhorias ao algoritmo de emulação de falhas serial: emulação simultânea de falhas independentes e a injeção dinâmica de falhas [Cheng1999]. A primeira melhora consiste em determinar grupos de falhas independentes, as quais podem ser emuladas simultaneamente. Isto se faz buscando as falhas cujo efeito nas saídas primárias é disjuntivo. Quer dizer que se for possível determinar com precisão as saídas primárias que são afetadas por duas falhas, elas são independente se a intercepção entre os conjuntos de saídas primárias afetadas é o conjunto vazio. Cheng et al [Cheng1999] melhorou o algoritmo apresentado em [Iyengar1988] para encontrar conjuntos de falhas independentes em circuitos combinacionais.

Com a técnica de emulação simultânea de falhas independentes é possível reduzir o número de recompilações e reconfigurações, caso as falhas estejam no mesmo FPGA, e o número de vezes em que são avaliados os vetores de teste. No entanto, o número de falhas independentes para circuitos seqüenciais é normalmente baixo.

Devido a isto, foi desenvolvida a técnica de injeção dinâmica de falhas. Esta técnica permite a injeção de um grande número de falhas dependentes em uma mesma configuração de FPGA, através da inclusão de uma lógica adicional.

Na Figura 4-3 (a) é apresentado o mapeamento da lista de conexão original nos blocos lógicos reconfiguráveis dos FPGAs. Na Figura 4-3 (b) se ilustra um exemplo do circuito agregado para injetar falhas com uma única configuração. Neste exemplo deseja-se emular duas falhas dependentes, cujo controle está dado pelo circuito de ativação. O controlador de ativação de falhas é um circuito que possui uma única saída em alto, enquanto as outras estão com nível lógico zero; isto quer dizer que por cada rodada de avaliação dos padrões de teste, uma única falha deve ser ativada.

Figura 4-3: Técnica de injeção dinâmica de falhas. (a) circuito original, (b) falhas dinâmicas, sinal 'a' SA1 e sinal 'g' SA0 injetadas por G1 e G2.

O controlador de ativação pode ser um registro de descolamento circular com entrada de relógio, de forma que cada pulso de relógio faz com que se ative uma nova falha e se desative a anterior. Do exemplo também pode se ver que para a ativação de uma falha “stuck-at” 1 se utiliza uma porta 'OU' e para a ativação de uma falha “stuck-at” 0 se utiliza uma porta 'E' com a entrada de controle negada. A porta 'OU' 'G1', quando ativa, faz com que a entrada 'a' fique travada em alto, independentemente do verdadeiro valor do sinal 'a'. Do mesmo modo, a porta 'E' 'G2', quando ativa, faz com que o sinal 'g' fique travado em baixo, independentemente do resultado da operação lógica 'd' E 'e'. Esta estratégia faz com que se reduza o tempo de recompilação e reconfiguração, de forma que podem ser avaliadas distintas falhas usando uma mesma configuração e emulando cada uma delas de forma seqüencial, como é estabelecido no algoritmo de simulação serial original. O custo desta estratégia é a lógica adicional que se necessita introduzir para a emulação de cada uma das falhas a ser avaliada e para o controlador de ativação. No caso extremo, onde o emulador possui recursos ilimitados de capacidade para a emulação lógica, é possível introduzir todas as falhas em uma única configuração. Dependendo da complexidade, é provável que sejam necessárias várias configurações, mas em cada uma delas um número considerável de falhas podem ser analisadas ao mesmo tempo.

Adicionalmente, no trabalho de Cheng et al [Cheng1999], foi proposta uma técnica para reduzir o tempo de recompilação, que envolve as tarefas de reparticionamento, substituição e roteamento. O objetivo final é permitir a injeção de um conjunto de falhas dinâmicas, unicamente mudando o conteúdo dos blocos lógicos reconfiguráveis que foram afetados, como é feito no método de emulação serial de falhas estático. Esta técnica parte da política de reservar uma entrada de cada bloco lógico reconfigurável, a qual será usada como sinal de ativação ou desativação da falha. Em princípio, independentemente do valor do sinal reservado, as saídas dos blocos lógicos reconfiguráveis correspondem às respostas corretas do circuito sem falhas, a seguir, na medida em que os conjuntos de falhas forem sendo introduzidos no FPGA, o sinal reservado permitirá a ativação de cada falha para ser analisada. Depois de mapear o circuito nos blocos lógicos reconfiguráveis, considerando a restrição do sinal de ativação de falhas reservado, é introduzido o registro de deslocamento circular. Cada saída deste registro deve ser conectada a um conjunto

de entradas reservadas dos blocos lógicos reconfiguráveis, levando em consideração que para cada saída mapeada unicamente deve existir um bloco lógico capaz de ativar uma falha. Estas novas conexões devem ser mapeadas no FPGA, para posteriormente descarregar o “bitstream” gerado. Depois destas modificações, a injeção de falhas pode ser feita mudando unicamente o conteúdo dos blocos lógicos afetados. Desta forma se evita a recompilação completa, já que as conexões entre blocos lógicos são mantidas durante todo o processo de emulação.

Para projetos de circuito grandes que precisem mais de um FPGA para implementar a emulação se deve considerar que se necessita particionar o circuito de forma que o controlador de ativação acione a emulação de unicamente uma falha por vez.

Em [Bernardi2007] se estende o método de Cheng-Huang&Dai para ser utilizado na emulação de falhas “path-delay”. Para isso, são incluídas certas estruturas “saboteurs” que permitem injeção e emulação do comportamento dos tipos de falhas “path-delay”. O método foi idealizado para circuitos microcontrolados onde o padrão de teste consiste em um programa funcional que executa uma série de instruções. Um controlador é encarregado de ativar cada uma das falhas a serem analisadas e comparar a resposta obtida com a esperada; também, encarrega-se da comunicação com o computador anfitrião, para receber o comando de início de emulação e transmitir os dados que resultaram da comparação. A comunicação com o computador é mínima o que faz com que o método seja rápido e eficiente.

4.3.3 Método para manipulação de lógica de três valores

Na simulação de falhas, tipicamente se usam sistemas de lógica com três valores: zero (0), um (1) e desconhecido (x). Na emulação lógica, da mesma forma como na implementação do circuito real, não existe o valor lógico desconhecido, o que pode trazer a desvantagem de não poder determinar se uma falha é realmente detectada. Por exemplo, quando se comparam as respostas dos circuitos com e sem falhas e se observam diferenças nas saídas, fala-se que a falha foi detectada, caso contrário isto não ocorre. O valor desconhecido, no circuito real, assume um dos

estados: zero ou um. Se a resposta do circuito sem falhas é um valor fixo (zero ou um), e a resposta do circuito com uma determinada falha faz com que apareça um valor desconhecido na saída, depois de avaliar determinado padrão, significa que com esse padrão não é possível efetivamente detectar a falha em 100% dos circuitos a serem testados. Como na emulação lógica não existe o valor desconhecido, o mesmo exemplo apresentado anteriormente, poderia causar um resultado errado do cálculo de cobertura, já que a comparação das saídas dos circuitos com e sem falhas podem, casualmente, serem diferentes, mas essa diferença não prevalece em todos os casos de chips avaliados.

Uma solução é usar lógica de duplo trilho (“dual-railed logic”) e codificar os valores um, zero e desconhecido em '11', '00' e '01', respectivamente. O código '10' representa um não importa que pode ser utilizado para minimizar o circuito. Esta codificação pode ser implementada através da simples duplicação da lista de conexão original. Por exemplo, uma porta 'E' de duas entradas pode ser mapeada como duas portas 'E', como se mostra na Figura 3.4 [Cheng1999].

Figura 4-4: (a) Tabela verdade da porta 'E' considerando o valor lógico desconhecido. (b) lógica de trilho dual para a porta 'E'

De forma genérica para a lógica combinacional, introduzir o valor desconhecido equivale a duplicar o circuito e quando aparecer um inversor, trocar as saídas entre os circuitos duplicados. Os “flip-flops” também devem ser duplicados, e projetados com características de “set” e “reset”, de forma que no início da emulação eles sejam inicializados com o código '01' que representa o valor desconhecido.

4.3.4 Emulação de falhas usando reconfiguração parcial

A principal desvantagem do método de Cheng-Huang&Dai [Cheng1999] é que a quantidade de “hardware” adicionado para fazer a emulação de falhas, aumenta drasticamente com o número de falhas do circuito que se deseja analisar, e conseqüentemente limita seu tamanho. Em [Burgun1996] é apresentada uma técnica de emulação de falhas que usa a reconfiguração parcial e não precisa que se adicione “hardware”. Esta técnica provou ser duas ordens de magnitude mais rápida que as técnicas de simulação de falhas. Seguindo a mesma linha, outras técnicas para a injeção de falhas foram propostas usando o programa JBITS [Antoni2000, Antoni2002, Antoni2001, Guccione1999]. Este programa facilita a tarefa de injeção de falhas através da modificação do conteúdo das LUTs (“Look-Up-Tables”) e a geração de “bitstreams” para a configuração dos FPGAs.

Em [Parreira2003] se apresenta uma metodologia eficiente para a emulação de falhas usando reconfiguração parcial. Nesta metodologia são criados pequenos arquivos de reconfiguração para a injeção das falhas, os quais são gerados através da manipulação direta do arquivo de configuração total, sem requerer nenhuma ferramenta adicional de software. Utiliza como tecnologia base os FPGAs Virtex da Xilinx. Em [XAPP1512004] se encontra a informação necessária para a extração do conteúdo das LUTs do FPGA. Cada bloco lógico configurável destes FPGAs, possui dois “slices”. Cada slice possui dois LUTs, que implementam uma lógica de até quatro entradas, dois “flip-flops” e lógica de saída. Uma LUT corresponde a uma memória de 16 bits, onde cada bit corresponde ao resultado da combinação das entradas, denotado como: $y_{abcd} = \{y_{0000}, y_{0001}, y_{0010}, \dots, y_{1111}\}$. Se uma das entradas foi fixada em um valor, resulta em um vetor de 8 bits. Por exemplo, se a entrada 'a' está presa em '0' o vetor resultante é $y_{abcd} = \{y_{0000}, y_{0001}, y_{0010}, y_{0011}, y_{0100}, y_{0101}, y_{0110}, y_{0111}\}$. Desta forma, do arquivo de configuração total do FPGA, são extraídas as LUTs relevantes ao circuito a ser analisado. Depois, determina-se o número de entradas efetivas que cada uma das LUTs utiliza. Para a injeção de falhas então, o único que se necessita é duplicar no vetor conjugado, o vetor de saída resultante, quando a

falha assume um determinado valor. Por exemplo, para injetar uma falha SA0 na entrada 'a' se copia o valor y_{0000} em y_{1000} , y_{0001} em y_{1001} , etc. De forma que o vetor resultante seja o apresentado a seguir:

$$Y_{abcd} = \{y_{0000}, y_{0001}, y_{0010}, y_{0011}, y_{0100}, y_{0101}, y_{0110}, y_{0111}, y_{1000}, y_{1001}, y_{1010}, y_{1011}, y_{1100}, y_{1101}, y_{1110}, y_{1111}\} = \{y_{0000}, y_{0001}, y_{0010}, y_{0011}, y_{0100}, y_{0101}, y_{0110}, y_{0111}, y_{0000}, y_{0001}, y_{0010}, y_{0011}, y_{0100}, y_{0101}, y_{0110}, y_{0111}\}$$

Assim sendo, a única informação requerida por este método é a contida no arquivo de configuração (“bitstream”) do FPGA. Depois de extrair cada LUT do “bitstream”, avalia-se a relevância de cada entrada i_x ($x=0, 1, 2$ e 3) comparando os vetores resultantes de 8bits de $i_x=0$ e $i_x=1$. Se estes dois vetores são idênticos então a entrada i_x é ativa. Desta forma é como se extraem cada um dos tipos de LUTs do circuito de acordo com o número de entradas relevantes (LUT2, LUT3, LUT4) [Parreira2003].

Depois de extrair o número de entradas ativas para cada LUT se cria a lista de falhas, agregando duas falhas “stuck-at” por cada uma das entradas/saídas. Os elementos da lista são injetados um a um em circuitos sem falhas, gerando um “bitstream” por cada elemento. O método de injeção que se utiliza é o que foi explicado anteriormente, onde o novo vetor de configuração de uma LUT específica $\{y_{abcd}\} = \{\{y_{1bcd}\}, \{y_{0bcd}\}\}$ é substituído por $\{y_{abcd}\} = \{\{y_{1bcd}\}, \{y_{1bcd}\}\}$, para analisar a falha SA1 da entrada 'a'. Este processo se repete até que não existam mais elementos na lista de falhas. Para melhorar a eficiência, é possível realizar colapso de falhas, agrupando as falhas que possuem o mesmo vetor de reconfiguração. Para ilustrar isto considere a função $F=a.b+c.d$. A Tabela 3.1 mostra o vetor de configuração sem falhas para a LUT (F) e os vetores de configuração que modelam todas as falhas “stuck-at” desta LUT (i/v significa sinal i preso em v). Analisando esta tabela é possível identificar dois pares de falhas que podem ser colapsados usando a metodologia apresentada (a/0, b/0) e (c/0, d/0). As células cinza da tabela indicam a combinação das entradas ou vetor de teste que pode ativar a falha dada para a respectiva coluna.

<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>F</i>	<i>a/0</i>	<i>b/0</i>	<i>c/0</i>	<i>d/0</i>	<i>a/1</i>	<i>b/1</i>	<i>c/1</i>	<i>d/1</i>	<i>F/1</i>	<i>F/0</i>
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	1	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	1	1	0
0	0	1	1	1	1	1	0	0	1	1	1	1	1	0
0	1	0	0	0	0	0	0	0	1	0	0	0	1	0
0	1	0	1	0	0	0	0	0	1	0	1	0	1	0
0	1	1	0	0	0	0	0	0	1	0	0	1	1	0
0	1	1	1	1	1	1	0	0	1	1	1	1	1	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1	0
1	0	0	1	0	0	0	0	0	0	1	1	0	1	0
1	0	1	0	0	0	0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	1	0	0	1	1	1	1	1	0
1	1	0	0	1	0	0	1	1	1	1	1	1	1	0
1	1	0	1	1	0	0	1	1	1	1	1	1	1	0
1	1	1	0	1	0	0	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Tabela 4-1: Vetores de reconfiguração de falhas para uma LUT que implementa $F=a.b+c.d$

Os “bitstreams” gerados correspondem à configuração de uma coluna do FPGA, de forma que o tempo de reconfiguração é reduzido. Para cada falha injetada, são executados todos os vetores de teste. Se existe alguma diferença entre a resposta obtida e a esperada se diz que o vetor executado consegue detectar a falha injetada. Este processo se repete até acabar com os elementos da lista de “bitstreams”.

O tempo de emulação de falhas pode ser aproximado à seguinte expressão:

$$t_{emulação} = t_{comp} + Falhas * \left(t_{reconf} + \frac{Vetores}{f_{emulação}} \right)$$

Onde t_{comp} é o tempo de sínteses e configuração do FPGA completo, t_{reconf} é o tempo de reconfiguração e $f_{emulação}$ é a frequência de aplicação dos vetores de teste.

Em [Parreira2003] se apresentam os resultados da comparação do método de emulação de reconfiguração parcial, de Cheng-Huang&Dai e a simulação de falhas de uma ferramenta comercial. Neste trabalho se concluiu que o método de Cheng-Huang&Dai tem melhor desempenho do que a simulação de falhas unicamente quando o número de vetores de teste excede certo valor que basicamente depende do tempo de compilação do FPGA. Depois de que o circuito é compilado, o custo de aplicar um vetor de teste novo é muito pequeno comparado com o custo da simulação. Por outro lado, o método de emulação de falhas que usa reconfiguração parcial é mais rápido que a simulação para um grupo de poucos vetores. Isto se deve ao fato que o tempo de compilação e reconfiguração é muito menor que o tempo de compilação do método de Cheng-Huang&Dai.

Em [Parreira2003a] se ilustra como o método de emulação de falhas usando reconfiguração parcial pode ser estendido para detectar outros tipos de falhas como “bridge” e as falhas “stuck-at” internas à LUT. Adicionalmente, propõe a criação de um novo modelo de falhas “Combination Stuck-At” (CSA) que reúne vários tipos de falhas como “stuck-at” e “bridge” em um único modelo funcional e não de integridade do sinal. Ele permite correlacionar de forma mais adequada as falhas reais que possam aparecer no circuito, gerando uma métrica de cobertura unificada e facilitando a injeção de falhas. O modelo CSA define que por cada possível combinação das entradas da LUT existe uma falha, quer dizer uma falha por cada bit ativo da LUT. Então para a injeção das falhas simplesmente se inverte, um a um, dos bits ativos da LUT, para cada novo “bitstream” se executa a emulação.

5 Plataforma de co-emulação de falhas

A simulação de falhas é um processo custoso, pois consome grandes recursos de memória e poder computacional. Em capítulos anteriores, foram apresentadas algumas alternativas para aumentar a eficiência do processo de detecção de falhas, através da emulação de hardware, dado que ela oferece velocidades que podem ser até cinco ordens de magnitude mais rápida que a simulação. Essas alternativas estavam direcionadas unicamente a modelar circuitos digitais, porém a maioria de sistemas estão constituídos por circuitos mistos: digitais e analógicos. A plataforma proposta nesta tese corresponde a um “hardware” que permite a co-emulação de um circuito para determinar as falhas detectadas por um padrão funcional dado, junto com o “software” que o controla. A co-emulação se refere ao processo de modelar simultaneamente sistemas representados de forma mista, quer dizer uma parte em “hardware” e outra em “software”, sobre uma plataforma de emulação de “hardware”. Neste capítulo se apresentará a metodologia para a co-emulação de falhas, os elementos da plataforma e a relação entre os mesmos.

5.1 Metodologia

A Figura 5-1 apresenta um fluxograma da metodologia para a co-emulação de falhas. Cada um destes passos está constituído por uma série de atividades que processam dados de entrada e entregam uma saída que será utilizada por um processo posterior, até obter a resposta final.

O primeiro passo corresponde à geração da lista de falhas. A ideia final da metodologia proposta é determinar o conjunto de falhas que são detectadas por padrões funcionais, a partir do conjunto de falhas testáveis que não foram detectadas pelo “scan”, usando ATPG. Como foi explicado em capítulos anteriores, o processo de geração automática de padrões de teste é feito a partir de uma lista de falhas que é gerada após a aplicação de um modelo de falha específico. Neste trabalho se utilizou o modelo “stuck-at”, pois com ele se consegue detectar a maior parte dos defeitos que podem aparecer na fabricação de circuitos integrados; no

entanto, a maioria dos métodos aqui apresentados podem ser aplicados utilizando outros modelos de falha. Após aplicar o modelo de falha à lista de interconexão, as ferramentas de ATPG criam a lista de falhas, as quais são inicialmente classificadas em testáveis e não testáveis. Depois, são aplicados algoritmos de geração de padrões automáticos randômicos ou determinísticos, para criar um conjunto de padrões que são avaliados através de simulação de falhas. O resultado final é uma lista de padrões de teste, junto com a classificação das falhas testáveis, divididas em vários grupos: detectadas, não detectadas, possivelmente detectadas, etc. A lista de falhas não detectadas será usada em um passo posterior.

Figura 5-1: Fluxograma da metodologia para co-emulação de falhas

O fluxo de verificação de fabricação de circuitos integrados compreende principalmente quatro conjuntos de padrões de teste. O primeiro corresponde aos

padrões de gravação e teste de memórias; o segundo são os padrões de “scan” que verificam o estado da lógica digital; o terceiro são os padrões funcionais que exercitam as funções principais do sistema e cem por cento dos circuitos analógicos; e o quarto que analisa os valores paramétricos do circuito em determinadas configurações. Estes conjuntos de padrões podem variar de “chip” para “chip” devido à ausência de alguma das estruturas mencionadas.

Atualmente só os padrões de “scan” são utilizados para determinar a cobertura de falhas da parte digital. Os outros três conjuntos de padrões também aportam a esta cobertura; só que sua contribuição não é quantificada devido ao alto custo computacional, humano e de tempo. Como já foi discutido anteriormente, grande parte das falhas que normalmente não são cobertas pelo “scan” estão relacionadas à arquitetura e circuitos de teste, ao pobre modelamento de blocos de sinal misto, entre outros. Acredita-se que grande parte das falhas não cobertas pelo “scan” são cobertas pelos outros padrões, mas como sua contribuição não é quantificada, fica difícil saber quais circuitos não foram suficientemente exercitados e se é necessário a criação de padrões adicionais. Portanto, foi proposto neste trabalho a análise e utilização de todos os padrões de teste, excetuando “scan”, para que sejam usados como entrada da nova plataforma de co-emulação de falhas aqui apresentada. Adicionalmente novos padrões podem ser criados e analisados posteriormente, caso seja necessário o exercício de alguma parte de circuito não coberta. De cada padrão deve se extrair os vetores de teste a serem utilizados, de forma que o controlador de co-emulação leia e aplique cada um. Para isto foi criada uma nova codificação de sinais que facilita a criação do arquivo de vetores e a aplicação dos mesmos. Adicionalmente foram criados processos, drivers e monitores no “testbench”, que capturam os sinais de entrada/saída configurados e salvam os vetores de teste codificados em um arquivo. Maiores detalhes serão apresentados no capítulo sete. Os arquivos com os vetores de teste codificados serão utilizados em um passo posterior desta metodologia.

Em geral os circuitos integrados estão compostos por uma série de blocos interligados como sistema. Estes blocos podem ser circuitos puramente digitais, analógicos, de sinal misto ou matrizes de memória. Em outros trabalhos encontrados na literatura [Wiler1995, Burgun1996, Cheng1999, Antoni2002, Parreira2003, Bernardi2007], apresentam-se métodos de emulação de falhas considerando

circuitos puramente digitais, já que modelos de falhas como “stuck-at” foram idealizados unicamente para este tipo de lógica. Blocos analógicos e memórias possuem outros modelos de falhas. Porém, observa-se que grande parte das falhas não cobertas pelo “scan” estão localizadas em circuitos de interface entre a parte digital e o restante. Isto indica que todos os blocos do circuito integrado devem ser considerados e modelados para a adequada graduação de falhas. Adicionalmente, similar a [Cheng1999], considera-se importante o modelamento dos tipos de sinal lógicos considerados em simulação: um, zero, indeterminado, alta impedância e níveis lógicos com distinta força. Devido a todo o anterior foi proposta a metodologia para a criação de modelos digitais (“dual railed logic” modificado) e memórias. Adicionalmente foi proposto a metodologia para modelamento de sinal misto através da descrição estruturada usando lógica de quatro níveis, e descrição comportamental. Desta forma o DUT pode estar constituído por uma mistura destes modelos, motivando o uso do nome de “co-emulação”, também conhecida como “co-modelamento”. A descrição destas propostas, detalhes de modelamento e aplicação serão discutidos no capítulo seis.

Os modelos devem ser aplicados de forma que se crie uma nova lista de conexão. Depois, devem ser inseridos os elementos da lista de falhas e criada uma nova lista de conexão. Do mesmo modo como em [Cheng1999], a proposta é colocar de forma estática todas as falhas no circuito, e de forma dinâmica ativar uma a uma através do controlador de falhas. Assim sendo, foi criado um script que lê a lista de falhas e as insere segundo o modelo de falha e de circuito que foram escolhidos, junto com o respectivo registro de ativação do controlador de falhas. Este passo será apresentado novamente no capítulo sete, depois de que se tenha uma melhor compreensão da plataforma e dos modelos.

A parte de “hardware” da plataforma de co-emulação termina com a interconexão do DUT que tem o controlador de falhas, com o controlador de co-emulação. Este último é sistema com o processador NIOSII. Foi agregado ao controlador uma Uart JTAG, um “timer”, memória e alguns barramentos de entrada ou saída paralelos. A este conjunto se denominou “controlador de co-emulação”. Na Figura 5-2 se vê um diagrama que apresenta as distintas partes que constituem o “hardware” da plataforma de co-emulação e sua relação de comunicação. Destaca-se que o primeiro barramento de saída tem alguns sinais que vão para o controlador de falhas

que permitem a reinicialização do circuito, assim como também o sinal de deslocamento do registro de ativação de falhas.

Figura 5-2: Hardware da plataforma de co-emulação

Foi criado um programa para o processador NIOSII que lê da memória os vetores de teste codificados, e os utiliza para estimular o DUT e comparar os resultados obtidos com os esperados. De forma geral a sequência de teste começa com a co-emulação do circuito bom, para verificar que todos os passos anteriores foram executados corretamente. Depois disto, se insere uma falha e se executa o vetor de teste via o controlador de co-emulação. Caso exista diferença entre a resposta observada e a esperada significa que a falha é detectada. Neste ponto a co-emulação pode terminar sem necessidade de executar completamente o vetor de teste. Para cada falha detectada se deve incrementar o contador de falhas e salvar o mesmo. Depois se remove a falha anterior e injeta-se outra para poder executar o padrão novamente. Este processo se repete até que o controlador de falhas termine de ativar todas as falhas do circuito. Lembrado que as falhas injetadas pertencem unicamente ao conjunto de falhas não detectadas pelo “scan”, o que significa que

deve ser uma porcentagem muito menor do que o total de falhas testáveis do circuito.

Após esta etapa, o emulador envia para o computador a informação das falhas que foram detectadas. Cada número do contador pode ser mapeado diretamente com a falha através da lista das mesmas. Adicionalmente o resultado das falhas detectadas deve ser incluído no novo cálculo de cobertura como se apresenta na equação 1.

$$Cobertura_{total} = \frac{falhas_{dt_{scan}} + falhas_{dt_{funcional}}}{Total_de_falhas_testáveis} \quad (1)$$

O processo de co-emulação deve ser repetido para cada um dos vetores de teste escolhidos. Veja que não é necessário recompilar o sistema para o FPGA cada vez que se deseje testar um novo padrão; é necessário apenas atualizar a memória com o novo conjunto de vetores.

A Figura 5-3 apresenta um diagrama de blocos simplificado de cada uma das partes da plataforma de co-emulação de falhas e como elas se relacionam e comunicam. Veja que o modelo estrutural e o comportamental são opcionais e dependem exclusivamente do circuito que se esteja testando. Adicionalmente, a figura apresenta os elementos relacionados com a camada de “software” e os relacionados com a camada de “hardware”. O controlador de co-emulação tem as duas partes “hardware” e “software”. O maior trabalho de desenvolvimento esteve concentrado na parte de “software” do mesmo, uma vez que a outra parte foi somente integração, usando as facilidades que o programa para FPGAs da Altera (“Quartus”) oferece.

Figura 5-3: Relação de comunicação entre elementos da plataforma de co-emulação

A Figura 5-4 apresenta um diagrama com todas as entradas e saídas de cada um dos processos realizados na metodologia proposta para a co-emulação de falhas. Desta forma dá para ver os processos que podem ser executados concorrentemente.

Figura 5-4: Diagrama de processos com entradas e saídas

6 Modelamento

O processo de co-emulação proposto nesta tese começa pela criação e substituição de cada um dos elementos da lista de conexão, por modelos que possam ser utilizados na plataforma com FPGA. Neste capítulo explicar-se-ão os tipos de modelos que podem ser utilizados, o processo de geração e aplicação de cada um destes.

6.1 Lógica digital

Na lista de conexão se descreve o grupo de instâncias que constituem um circuito e sua interconexão. As instâncias podem ser classificadas em três tipos: lógica digital, memórias e “hardblocks”. A lógica digital pode ser modelada de forma direta ou codificada, como se apresentará a continuação. Destaca-se que um dos resultados deste trabalho é a codificação modificada, que permite o modelamento de qualquer tipo de circuito lógico e sinal, incluindo sinais de relógio e de controle assíncrono.

6.1.1 “Single railed logic”

Os circuitos digitais de uma lista de conexão estão constituídos por um grupo de células padrão para uma determinada tecnologia. Para a simulação deste tipo de circuitos é necessário um conjunto de modelos, em forma de primitivas, que explicam para o simulador o comportamento de cada uma das células que o constituem. Para a emulação de “hardware” é necessário criar o mesmo conjunto de modelos, mas, usando uma mistura entre descrição estrutural e primitivas da plataforma de FPGA adotada. Como a plataforma utilizada neste projeto está baseada em um FPGA da Altera, o conjunto de células primitivas disponíveis foi: DFF, LATCH, “verilog standard cells”, entre outros.

O trabalho de criação de modelos de uma biblioteca de células padrão pode ser monumental, dada grande quantidade de elementos que pode compreender. Foi criado um “script” para automatizar a tarefa de criação destes modelos. O “script” não é genérico, pois deve ser ajustado para cada tipo de biblioteca ou tecnologia; dadas as características particulares que cada uma delas possui. Este “script” foi utilizado para a transformação de duas bibliotecas de 0.25 e 0.18 micrômetros com resultados satisfatórios. As mudanças no “script” entre uma biblioteca e outra foram poucas.

Os simuladores de falhas normalmente podem trabalhar com vários tipos de sinal digital: ‘1’, ‘0’, ‘Z’ e ‘X’. O sinal ‘Z’, alta impedância, pode ser usado em barramentos onde existem vários drivers e o valor que este assume corresponde à combinação entre os valores dos sinais que o comandam. O resultado da combinação é dado pela Tabela 6-1. O sinal tipo ‘X’ representa indeterminado ou desconhecido, onde um sinal pode assumir tanto o nível lógico ‘1’, quanto ‘0’. Quando uma das entradas de uma porta lógica é comandada por um sinal em estado ‘Z’, este é tratado como ‘X’.

	0	1	X	Z
0	0	X	X	0
1	X	1	X	1
X	X	X	X	X
Z	0	1	X	Z

Tabela 6-1: Resultado da combinação de sinais em um barramento

Desconsiderando problemas de “timing” (“setup” e “hold”) as fontes de ‘X’ podem ser de dois tipos: 1- originadas por entradas primárias ou sinais de saída de “hardblocks”; ou 2- elementos de memória (“flip-flop, latch”, etc) não inicializados. Normalmente a emulação de “hardware” não considera estes múltiplos tipos de sinais digitais, já que são projetados circuitos que não possuem o risco de sinal indeterminado, ou que sua ocorrência não compromete o funcionamento da máquina. Desta forma são utilizados modelos lógicos “single railed”. Para a

emulação de falhas, a ausência do tipo de sinal 'X' pode resultar em um erro na classificação das falhas. Para exemplificar a afirmação anterior, considere o circuito da Figura 6-1, na qual é feito uma análise do circuito com uma falha "stuck-at 0" (SA0) na entrada reset do "flip-flop". Os sinais dado, "clock", e "reset" fazem parte do vetor de teste de entrada, e a "saída boa" o resultado esperado do circuito bom. O sinal "strob" indica os pontos onde se realiza a comparação entre o circuito com falha e o bom. O sinal "Saída falha" corresponde ao resultado do circuito após realizar uma simulação com a falha SA0 no "reset" do "flip-flop". Agora, na emulação do circuito usando modelo "single railed logic", existem dos possíveis resultados: Saída FPGA1 e Saída FPGA2. A diferença entre eles é o possível valor que assume o "flip-flop" na inicialização. Comparado com a simulação esse valor deveria ser indeterminado, mas como esse tipo de sinal não existe no FPGA, então o circuito assume o valor '1' ou '0'. Dessa forma, quando se realize a emulação da falha, a "Saída FPGA 1" indica que a falha não é detectada com o padrão de entrada utilizado; enquanto que a "Saída FPGA 2" indica o contrário, que a falha é detectada com esse padrão. As duas respostas seriam incorretas ou imprecisas. Quando se compara o resultado da simulação de falhas do circuito bom com o defeituoso, se nos pontos de observação se registra sinais em X, quer dizer que a falha é possivelmente testável com o padrão avaliado. Segundo [AEC2008], se a observação de 'X' se repete pelo menos dez vezes com padrões diferentes, significa que a falha pode ser classificada como possivelmente testável. Dependendo da rigorosidade, é permitido dar um crédito de até 50% para as falhas possivelmente detectadas. Com o exemplo anterior se vê como a emulação de falhas com lógica "single railed" é frágil por não utilizar o tipo de sinal 'X'. É possível que para alguns circuitos isto não comprometa o resultado obtido, mas diminui sua confiabilidade.

Figura 6-1: Exemplo de importância do tipo de sinal 'X'

6.1.2 “Dual railed logic”

No capítulo quatro foi introduzido o método para a manipulação de lógica de três valores, da forma como foi apresentada por [Cheng1999] nos seus artigos. Através deste método é possível solucionar o problema do tipo de sinal 'X' (indeterminado) na emulação de circuitos digitais. Nesta seção será analisado o método de forma mais detalhada, apresentando alguns exemplos de utilização e sua aplicação, destacando prós e contras.

Este método se baseia na codificação de cada um dos tipos de sinal como se apresenta na Tabela 6-2. Estes códigos foram selecionados desta forma pela facilidade que tem para sua aplicação. Assim, a proposta diz que o circuito original deve ser duplicado e que os trilhos onde a saída é negada, devem ser trocados entre sim, para manter a codificação do 'X' (01). Segundo o artigo, este método de duplicação aplica-se também para portas complexas. Os “flip-flops” devem ser duplicados também, e inicializados com o código do 'X' (01) através de um sinal a ser agregado.

Sinal	Código
0	00
1	11
X	01
Simplificar	10

Tabela 6-2: Codificação de tipos de sinal “dual railed logic”

Na Figura 6-2 se apresentam as portas lógicas simples (“AND, OR, NAND, NOR e NOT”) modeladas com “dual railed logic”. Como se vê, as tabelas da verdade respeitam as funções lógicas, incluído o sinal de tipo ‘X’. Vale a pena salientar que no caso de funções que incluem a função “NOT”, foram trocadas entre as saídas. A aplicação do método para este tipo de lógica é simples; coisa que não é necessariamente verdade para portas complexas. Se a função da porta complexa usa unicamente lógica que exclui a função “NOT”, a aplicação do método é direta. Quer dizer, simplesmente deve-se duplicar a porta complexa e re-fazer a conexão. Agora, para portas complexas que incluem o “NOT” é necessário analisar o circuito a partir da sua representação em portas básicas e aplicar individualmente o método.

Figura 6-2: Modelos “dual railed logic” para portas simples

Para exemplificar a forma como é criado o modelo de uma porta complexa, pode-se analisar o caso de um multiplexador, ver Figura 6-3. Esta figura está dividida em três modelos com sua correspondente tabela da verdade: a parte identificada pela letra 'a' corresponde ao multiplexador, junto com sua representação em portas básicas; a 'b' o modelo de multiplexador aplicando diretamente o método de "dual railed logic"; e a 'c' o modelo resultante de aplicar o método a partir da representação em portas básicas do multiplexador. Note que três campos da tabela da verdade de 'b' não equivalem às saídas de 'a'; inclusive um destes campos corresponde a um código não permitido dentro do conjunto de valores desta lógica. Portanto o modelo de 'b' não é apropriado. Por outro lado, observa-se em 'c' que todos os campos, excetuando um, equivalem ao especificado pelo modelo original. Existem duas formas de observar esta diferença: 1- Por definição do modelo de porta 'E', a única forma de obter um valor diferente de indeterminado, quando uma das entradas é indeterminada, é que a outra entrada seja zero. Aplicando esta idéia no modelo de porta lógica básica do multiplexador se deduz que quando a entrada de seleção da porta é indeterminada, a única forma de obter um valor diferente de indeterminado na sua saída é que as outras duas entradas (D0 e D1) sejam zero. Portanto apesar da diferença o modelo está correto. 2- Por outro lado, a definição de um multiplexador diz que o valor da entrada de seleção indica qual das entradas (D0 ou D1) deve ser direcionada para a saída. Agora, se as duas entradas (D0 e D1) possuem o mesmo valor, quando a entrada de seleção é indeterminada, a saída deve ser a mesma indicada por D0 e D1. O que é mais parecido ao resultado que se observaria no circuito real. Poderia criar-se um terceiro modelo que utilize a codificação definida pelo "dual railed logic", gerado a partir da síntese do circuito com a tabela da verdade. De qualquer forma o modelo de 'c' pode ser usado para a emulação de falhas, já que ele não acusa a detecção de falhas de forma errada pelo padrão aplicado.

Figura 6-3: Modelo de multiplexador em dual railed logic

De forma análoga, o modelo do “flip-flop” tipo D poderia ser criado a partir da sua representação em portas lógicas. Porém, é uma forma ineficiente e inviável de utilizar os recursos do FPGA; além de colocar lógica no caminho dos relógios. Adicionalmente, a codificação proposta pelo “dual railed logic” é suscetível ao risco de funcionamento de sistemas assíncronos [Nowick1995] e portanto, sinais de controle assíncrono não deveriam ser modelados usando esta codificação. Isto se aplica aos sinais “set/reset” como também ao sinal de relógio, que é o sinal assíncrono que sincroniza o sistema. Para entender melhor esta premissa, veja Figura 6-4. Toda transição do relógio passa de zero para um e vice-versa, usando os

códigos '00' e '11'. Nesta transição o relógio pode passar por um estado indeterminado ('X'), ou um não codificado. Agora, imagine que é possível criar um modelo "dual reailed logic" do "flip-flop" tipo D de borda positiva, utilizando dois DFF do FPGA e analise os cenários de transições propostos na Tabela 6-3. Esta tabela mostra como deveria comportar-se a saída Q, a partir das entradas D e CLK, na sequência T. Note que o "flip-flop" capturou o dado '1' em algum tempo anterior, e que nos tempos apresentados não existe captura de dado. Se estes sinais forem codificados, a transição do tempo T 2 para 3, do cenário 1 implica que o sinal de relógio passa de '00' para '11'. Dado que existe o risco fundamental nesta codificação, na transição de 2 para 3 o relógio poderia ser '01' por algum instante de tempo. Isto significa que ele é indeterminado ('X'), o que faz com que sua saída seja forçada para 'X', como mostra o cenário 2. O risco fundamental não é problema para os sinais da lógica combinacional, já que o relógio determina o tempo máximo de propagação em que os sinais se resolveram e podem ser capturados pelo registro.

Figura 6-4: Transição de sinal de relógio

	Cenário 1			Cenário 2		
T	D	CLK	Q	D	CLK	Q
1	1	1	1	1	1	1
2	0	1	1	0	1	1
3	0	0	1	0	X	X

Tabela 6-3: Exemplo de risco fundamental

No artigo do Cheng [Cheng1999] se propõe que a lógica do circuito seja analisada de forma que o método de “dual railed logic” não seja aplicado às partes onde não possa existir propagação de sinal indeterminado. Lembrando que as duas únicas fontes de ‘X’ são elementos de memória não inicializados, ou sinais provenientes de entradas primárias ou “hardblocks”. Através da análise do circuito e padrões que se desejem emular, é possível dizer que no circuito o sinal de relógio e sinais de “reset/set” nunca serão indeterminados. Este é um caso particular para o qual é possível implementar um modelo conveniente que utiliza uma única fonte de relógio com filtro de ‘X’ no reset; ver Figura 6-5. Destaca-se que na figura foi incluído um sinal de inicialização de registro (SIM_INI) de forma que todos sejam configurados com o valor indeterminado antes do início da emulação. Este modelo não é aplicável a todos os casos, especificamente ele pode não ser útil em casos onde se deseja analisar circuitos de geração e controle de relógio. Este tipo de circuito normalmente tem baixa cobertura de falhas em “scan” e portanto são alvo da graduação de falhas de padrões funcionais.

Figura 6-5: Modelo de “flip-flop” D usando “dual railed logic” desconsiderando ‘X’ em sinais de controle

6.1.3 “Dual railed logic” modificado:

Para resolver os problemas anteriormente citados, propõem-se algumas modificações na codificação e aplicação do método “dual railed logic”. Assim sendo, a nova codificação é a ilustrada pela Tabela 6-4. Esta codificação é livre do risco fundamental (entre ‘1’ e ‘0’) e portanto pode ser aplicada a qualquer tipo de sinal. Além disso, como se explicará posteriormente, este método oferece outras vantagens como: definição de uma única fonte de relógio, simplificação de aplicação do método usando os recursos do FPGA, entre outros.

Sinal	Código
0	00
1	01
X	10
Simplificar	11

Tabela 6-4: Codificação “Dual railed logic” modificado

Para a geração de modelos de lógica combinacional, portas simples e complexas, basta fazer a função do valor indeterminado (‘X’), cuja resposta será a segunda saída, e fazer uma ‘E’ com a função original negada para criar a primeira saída. Isto se ilustra na Figura 6-6. Note que a função de ‘X’ está constituída por todas as entradas 2, e algumas ou todas as entradas 1.

Figura 6-6: Método de geração de modelo de DRL modificado para lógica combinacional

Figura 6-7: Modelo de “dual railed logic” modificado de portas simples

A Figura 6-7 apresenta os modelos de portas lógicas simples aplicando o método do modelo modificado. Destacou-se em cinza a porta originária do modelo. Lembrando que em uma “Look Up Table” é possível implementar qualquer função digital de quatro entradas, observa-se que tanto para o caso do modelo “dual railed logic” simples como o modificado são necessárias duas LUT para cada porta simples. Porém, na redução lógica é possível que, usando o modelo de “dual railed logic” simples a ferramenta de síntese do FPGA consiga agrupar em uma LUT mais funções do que o modelo modificado. Isto significa que o modelo “dual railed logic” modificado pode chegar a utilizar mais recursos do FPGA.

Na Figura 6-8 se apresenta a forma de aplicar o método para portas complexas, neste caso o multiplexador. Note como é necessário apenas encontrar a função do valor indeterminado.

Figura 6-8: Modelo de multiplexador usando “dual railed logic” modificado

Adicionalmente, no caso do multiplexador, todos os valores da tabela correspondem ao valor esperado pela especificação original (Figura 6-3a).

Para elementos de memória, como “flip-flops” e “latches”, a forma de criar o modelo é duplicando o elemento de memória, utilizar um único relógio e adicionar a função de ‘X’ nas entradas “set/reset”, segundo seja o caso. Para exemplificar isto foi criado o modelo de “flip-flop” D, ver Figura 6-9. Pode-se observar da figura que o sinal CLK1 é o relógio comum para os dois elementos de memória, o que facilita as coisas para o FPGA, e que o CLK2 é usado unicamente para a função de ‘X’. O sinal SIM_INI foi agregado para forçar os “flip-flops” para o estado indeterminado no início da emulação.

Figura 6-9: Modelo de “flip-flop” D usando “dual railed logic” modificado

O modelo proposto possui o risco fundamental entre X e 1, nas outras transições não. Isto não é problema para a lógica combinacional. Para os elementos de memória no entanto, pode fazer com que o mesmo saia de ‘X’ antecipadamente na hora em que os seus sinais de controle “clk/reset/set” vão de ‘X’ para ‘1’, dependendo de como seja feita a transição. Porém, este risco só afeta alguns casos específicos de circuito, sendo que a maior parte dos casos estão cobertos. Para o escopo deste projeto esta codificação é suficiente e garante, em sua maior parte, a correta emulação das falhas. O erro induzido quando o registro sai do estado indeterminado de forma adiantada é mínimo e pode considerar-se desprezível. Ainda assim, este modelo prova ser melhor que o anterior, por ter removido o risco das

transições de ‘0’ para ‘1’ e de ‘1’ para ‘0’, já que estes são os estados de sinal predominantes durante toda a emulação.

No desenvolvimento deste trabalho foram pensadas alternativas como uma nova codificação de três trilhos, ou recodificação dos sinais nos casos específicos onde pode aparecer o problema, valendo-se de artifícios implementados em sistemas assíncronos; mas estas soluções implicam em novas complexidades que fogem do escopo da tese. Desta forma se deixam para posterior análises em trabalhos futuros.

Para a aplicação dos modelos na lista de conexão foi criado um “script” que basicamente troca todas as instâncias por seu correspondente modelo, usando a codificação “dual railed logic” modificado, e depois realiza a conexão das entradas adicionais seguindo o “driver” da entrada análoga. A Figura 6-10 mostra o pseudocódigo que realiza este procedimento.

```
-@instâncias_processadas={}
-@instâncias=Criar lista com todos os elementos da lista de conexão
-Para cada elemento de @instâncias
  -Substituir elemento por modelo do mesmo, conectando cada porto com seu correspondente
  com subíndice 1 e deixando porto com subíndice 2 aberto
  -@instâncias_processadas={@instâncias_processadas, elemento}
-Fim
-Para cada elemento de @instâncias_processadas
  -@portos=lista de portos de elemento com subíndice 1
  -Para cada elemento_porto de @portos
    -elemento_porto_2=elemento_porto substituindo subíndice por 2
    -driver_2=(driver de elemento_porto) substituindo subíndice por 2
    -Conecte elemento_porto_2 com driver_2
  -Fim
-Fim
```

Figura 6-10: Pseudocódigo para a aplicação de modelos à lista de conexão

6.2 Memórias

Dependendo do padrão é possível que não seja necessário o modelamento da memória. Este é o caso, por exemplo, da análise dos modos de operação do dispositivo, no qual a memória pode estar em um estado indeterminado.

Atualmente a maioria dos FPGAs possui blocos de memória embarcados. O FPGA utilizado para o desenvolvimento do trabalho, Altera Cyclone II [Altera2008], possui vários tipos de configuração de memória: “Single-port”, “Simple dual-port”, “True dual-port” (“bidirectional dual-port”), “Shift register”, ROM e FIFO “buffers”. Também, disponibiliza a configuração dos blocos de memória em várias configurações de tamanho, para cada bloco M4K: 4Kx1, 2Kx2, 1Kx4, 512x8, 512x9, 256x16, 256x18, 128x32 e 128x36. Então estas configurações pode ser usadas para a emulação da memória do “chip” (o FPGA usado possui 256 M4K blocos. Caso seja necessário, é possível utilizar memória externa). Qualquer que seja o caso é necessário fazer uma lógica de envoltório que realize a transdução dos sinais gerados pela lógica a ser emulada, para a memória utilizada.

Na criação do modelo pode acontecer que a memória utilizada precise de mais ciclos de relógio para dar a sua resposta; neste caso se recomenda adicionar um relógio de simulação, independente do circuito, e conectar o relógio do circuito emulado na entrada de habilitação de relógio da memória. Quando se esteja injetando o padrão, entre cada mudança de nível do relógio do circuito, deve-se pulsar a memória o número de vezes necessário para obter a resposta da mesa.

As memórias embarcadas do FPGA utilizado podem ser carregadas com dados predefinidos para cada padrão, caso seja este um requerimento. Para isto basta a criação do arquivo ‘MIF’ (“Memory Initialization File”) com a informação do tamanho da memória (profundidade e tamanho de palavra), os tipos usados no endereçamento e dado, e a lista de endereços com o dado correspondente. Esta informação é carregada no momento de realizar a programação do FPGA.

Existem dois tipos de memória que podem ser modeladas: a de leitura, e a de leitura e escritura. Se no processo de emulação se está utilizando modelos single railed logic, para a criação da memória simplesmente se deve criar a lógica de envoltório de forma direta. Agora, em caso no qual se utilize o “modelo dual railed logic” modificado, deve-se duplicar as memórias usadas e realizar a lógica de envoltório seguindo as seguintes regras:

Somente Leitura:

- Os dados devem ser carregados com o arquivo tipo MIF, sendo que o “Quartus” carrega os espaços de memória não preenchidos com ‘0’. Na primeira memória deve-se colocar o programa ou dados de início, enquanto que na segunda só se devem preencher as células não usadas com ‘1’. Isto pode variar de circuito para circuito, já que na aplicação real é muito provável que a memória seja carregada completamente com um valor inicial. Nesses casos, a segunda memória pode ser eliminada.
- Se algum bit do barramento de endereço é indeterminado, todos os bits de saída são forçados para indeterminado até vir o próximo relógio de leitura.
- Por definição o sinal de relógio nunca irá ser indeterminado, mas o circuito de propagação pode ter falhas SA0 e SA1.

Leitura e escritura:

- Se alguma entrada de dado é indeterminada, salvar ‘X’ na célula de memória correspondente à posição do dado. Isto é ‘0’ no primeiro bloco de memória e ‘1’ no outro.
- Se a entrada de habilitação de escritura é indeterminada, salvar ‘X’ na palavra indicada pelo endereço.
- Se algum bit do endereço ou o sinal de habilitação de memória são indeterminados, quando se realize uma escrita, então a saída da memória vai para indeterminado e permanece nesse estado até o próximo padrão (o bloco de memória inteiro é indeterminado).
- Por definição o sinal de relógio nunca irá ser indeterminado, mas o circuito de propagação pode ter falhas SA0 e SA1.
- Deve-se evitar ter que inicializar a memória com valor indeterminado entre cada uma das aplicações do padrão. Isto devido ao tempo que consome esta inicialização. Caso seja necessário, um pequeno circuito pode ser agregado na lógica de envoltório, que utilize os sinais SIM_INI e SIM_CLK, para realizar o procedimento de inicialização com valor indeterminado.
- Memórias inicializadas com algum padrão específico não devem ser corrompidas entre a aplicação de cada padrão.

As regras anteriormente mencionadas foram idealizadas para não afetar a eficiência da emulação de falhas e simplificar a criação da lógica do envoltório de memória. Os

padrões funcionais podem ser projetados considerando estas regras para evitar teste de situações não consideradas. A complexidade do envoltório pode aumentar na medida em que se deseje, de forma que se considerem todas as possibilidades de operação que incluam o valor indeterminado; embora o aumento na complexidade pode não resultar em alguma melhoria na classificação das falhas e sim na diminuição da eficiência da emulação.

6.3 “Hardblocks”

Os “hardblock” normalmente são blocos de circuito com uma parte digital e outra analógica. Eles interagem com a lógica digital recebendo comandos de configuração ou enviando resultados de alguma operação. Sempre a interface entre estes dois é digital e, para avaliação de falhas “stuck-at”, os sinais analógicos são desconsiderados, porém sua funcionalidade não. Nesta seção apresentar-se-ão dois métodos de modelamento de “hardblocks” (modelamento estrutural e comportamental), propostos neste trabalho. Assim também, propor-se-á uma nova forma de modelamento de sinais de quatro estados.

6.3.1 Modelamento estrutural

Este tipo de modelamento se aplica a blocos cujo comportamento pode ser descrito através de operações lógicas. Este é o caso de blocos onde se considerem sinais de alta impedância, força de sinal, chaves analógicas, etc. Existe a preferência em descrever o modelo de “hardblock” de forma estrutural, já que não introduz atraso ao processo de emulação.

6.3.1.1 Lógica de quatro níveis

Basicamente esta é uma extensão ao modelo de “dual railed logic” modificado. A idéia então é adicionar aos três tipos de sinal lógico conhecidos, zero, um e indeterminado, o quarto tipo: alta impedância. Na Tabela 6-5 se apresenta a codificação de cada um destes estados.

Sinal	Código
0	00
1	01
X	10
Z	11

Tabela 6-5: Codificação de lógica de quatro níveis

Com esta codificação é possível criar alguns elementos que facilitam a criação de modelos de hardblocks usando o método estrutural. O primeiro destes elementos é o “buffer tri-state”. A Figura 6-11 mostra o “buffer” e o modelo em lógica de quatro níveis, junto com suas tabela de verdade. O símbolo Φ foi usado para representar qualquer valor ‘0’, ‘1’ ou ‘X’. Para simplificar, foi definido que as entradas nunca devem estar em alta impedância.

Figura 6-11: Modelo de “buffer tri-state” em lógica de quatro níveis

O seguinte elemento é o concentrador ou multiplexador analógico. Este elemento só considera os quatro níveis lógicos. A especificação do modelo é feita a partir de duas entradas e uma saída. Este elemento pode ser colocado em cascata com outros do mesmo tipo, para aumentar o número de entradas, útil para modelar barramentos “tri-state”. A especificação é a seguinte: se uma das entradas é ‘Z’, na saída aparece a outra entrada; se as duas entradas são iguais, na saída aparece o valor comum; e se as entradas são diferentes, na saída aparece ‘X’. Na Figura 6-12 se apresenta a especificação e modelo do concentrador.

Figura 6-12: Modelo do concentrador usando lógica de quatro níveis

Outro elemento que se criou foi um que permite concentrar as entradas, considerando a força do seu sinal. Então, quando o sinal forte é ‘Z’, o sinal fraco assume o controle da saída; em outro caso o sinal forte é quem direciona. Este modelo se vê na Figura 6-13.

Figura 6-13: Modelo de concentrador que considera a força dos sinais de entrada

Finalmente, foi criado um elemento que permite conectar o mundo de sinais lógicos de quatro níveis, com o de três da lógica digital. Isto é, qualquer sinal de saída 'Z' deve ser considerado como 'X' pela lógica digital, como se vê na Figura 6-14.

Figura 6-14: Interface entre lógica de quatro níveis e a de três

A importância dos elementos anteriormente apresentados é que eles se complementam com os modelos de “dual railed logic” modificado, e juntos constituem os elementos básicos para a construção de modelos estruturais de “hardblocks”. Um exemplo de aplicação é na criação de modelos para pinos de entrada e saída configuráveis. Na Figura 6-15 se apresenta um diagrama de blocos de um pino de entrada/saída, e na Figura 6-16 o correspondente modelo em forma estrutural. Como se vê, ele está constituído pelos elementos anteriormente apresentados, de forma que fica simples sua implementação. Observe que o PAD, originalmente bidirecional, foi substituído por dois portos, um de saída que reflete o valor que o PAD assume, e o de entrada que corresponde aos valores que são forçados externamente pelo padrão de entrada. Desta forma fica simples a

administração dos testes, tanto para os sinais a serem injetados, quanto para os que são observados e comparados com o modelo do circuito bom.

Figura 6-15: Modelo de PADs configuráveis de entrada/saída

Figura 6-16: Modelo estrutural do PAD

6.3.2 Modelamento comportamental

Consiste na elaboração de um programa executado pelo processador que administra o teste. Deve ser inicializado para cada padrão avaliado. Possui acesso ao sinal de relógio de simulação, para programar a execução de eventos. Também tem acesso direto a um porto de entrada/saída do processador que serve como

interface com a lógica digital que se está emulando. Além disso, permite a troca de mensagens de configuração, codificados nos padrões avaliados, para a simulação de uma tarefa e estímulo da interface com os circuitos digitais. Este tipo de modelo pode ter sinais de entrada de configuração, barramento de leitura de dados e saídas de interrupção.

A flexibilidade que oferece uma linguagem de programação é grande e por tanto pode ser usado para agir como praticamente qualquer bloco. Porém, tanta flexibilidade traz a penalidade do tempo de execução e administração do teste.

Recomenda-se o uso de modelos comportamentais unicamente em casos onde não é possível a criação de modelos estruturais.

Modelos comportamentais não necessariamente devem refletir todas as operações do circuito original, pois unicamente importa os sinais de interface com a lógica digital. Esta idéia é importante para poder escrever um modelo que seja o mais simples possível.

A execução do processo do modelo é feita depois de que o processador injeta os sinais de entrada do padrão e antes de comparar os resultados com o esperado nas saídas. Maiores detalhes serão vistos no capítulo posterior.

Escolheu-se um conversor analógico-digital para exemplificar a criação deste tipo de modelo. A Figura 6-17 apresenta o diagrama de blocos do circuito. Ele está constituído por vários elementos complexos que na verdade não interessam para a classificação de falhas “stuck-at”. Assim sendo, os únicos sinais de interesse são o sinal de “Start”, que inicializa o processo de conversão; o “End” que indica a finalização do mesmo; e o registro de dados que determina o valor final da conversão. Então, o modelo pode ser um programa que depois de um certo número de ciclos de relógio da simulação, indica o final de conversão, junto com um valor predefinido.

Figura 6-17: Diagrama de blocos de conversor analógico digital

Entende-se que o modelo descrito é muito simples e poderia ser modelado usando descrição estrutural. No entanto, pode-se agregar elementos complicadores, flexibilidade de operação e outros refinamentos difíceis de obter com um modelo estrutural. A escolha depende do problema que se esteja resolvendo e as falhas que se desejem analisar.

7 Detecção de falhas

No capítulo cinco foi introduzido o método e plataforma para a co-emulação de falhas de forma global. No capítulo seis foi ampliado o conceito de modelo para co-emulação de falhas, e apresentados os distintos tipos de modelos dos elementos que podem constituir o circuito sob teste. No presente capítulo será abordada a síntese do circuito de injeção de falhas, a geração e administração dos vetores de teste. Desta forma serão introduzidos alguns conceitos importantes propostos neste trabalho.

7.1 Síntese do circuito de injeção de falhas

Da mesma forma como em [Cheng1999], nesta tese se utilizou a injeção dinâmica de falhas. Como foi apresentado na seção 4.3.2 e exemplificado na Figura 4.3, esta técnica consiste na compilação do circuito sob teste junto com as falhas e o controlador de ativação. As falhas são injetadas dinamicamente na emulação através do controlador de ativação de falhas, que é um registro de deslocamento em anel com uma única saída ativa por rodada de emulação. A saída ativa faz com que seja injetada uma falha através do uso de algum circuito específico. No modelo de falhas “stuck-at” usando a representação “single railed logic”, as falhas são exercidas por uma porta ‘OU’, em caso de “stuck-at high”, ou uma ‘E’ com a entrada de ativação negada, para o “stuck-at low”. Veja que este tipo de estrutura pode ser usada para outros modelos de falha e representações do circuito. As entradas do controlador de ativação são o relógio e o sinal de inicialização.

A Figura 7-1 ilustra os elementos de falha quando se usa o modelo de representação do circuito dual railed logic modificado. Quando a entrada de ativação é ‘um’, o circuito é forçado para o estado ‘zero’ ou ‘um’, dependendo do tipo de falha injetada; caso contrário os elementos de falha são transparentes. Na mesma figura se apresenta também um exemplo de inserção de falhas para uma porta ‘OU’ no

modelo “dual railed logic” modificado. Nota-se que para inserir uma falha SA0 e SA1 no mesmo sinal, é necessário colocar os dois elementos de falha em série.

Figura 7-1: Elementos para inserção de falhas “stuck-at” em modelo “dual railed logic” modificado

Esta técnica traz a vantagem de que não é necessário a recompilação e/ou reconfiguração do FPGA para injetar uma outra falha, o que reduz o tempo de co-emulação.

Para a inserção automática das falhas e síntese do controlador de ativação foram criados dois “scripts”. O primeiro pré-processa a lista de falhas de forma que seja entendida pelo segundo “script”, e gera duas listas separadas das falhas SA0 e SA1. Este “script” basicamente faz uma transformação dos caminhos, extração de informação e agrupação dos dados.

O segundo “script” foi criado para varrer as duas listas de falhas, inserir os elementos de falha e sintetizar o controlador de ativação. Na Figura 7-2 se encontra

um pseudocódigo resumido com a descrição dos principais processos do programa. A essência do mesmo está em introduzir no DUT elementos de circuito que podem ser acionados pelo controlador de ativação, de forma que imitem o comportamento esperado por cada uma das falhas da lista. O controlador de ativação é gerado através da inclusão de um “flip-flop” no registro de deslocamento em anel por cada falha inserida. O primeiro “flip-flop” do registro de deslocamento é acertado para ‘um’ na inicialização da co-emulação, e não está conectado a nenhum elemento de falha. Neste estado se analisa o DUT sem falhas.

```

-criar instancia do controlador de ativação com entradas FC_CLK e SIM_INI (reset)
-agregar flip-flop (registro de deslocamento 0) com sinal de set ligado ao SIM_INI, relógio ao
FC_CLK e entrada de dados a zero
-Para cada item de @falhas
    -porta=célula associada a $item
    - Se $item é sinal de entrada
        -cortar sinal de $item
        -conectar saída de elemento de falha com entrada de porta
        -conectar antigo driver de porta com entrada de sinal de elemento de falha
    -senão
        -cortar sinal de $item
        -conectar entrada de sinal de elemento de falha com saída de porta
        -conectar saída de elemento de falha com driven sinal
    -fim
-criar flip-flop em controlador de ativação ligando reset com sinal SIM_INI, relógio ao
FC_CLK e entrada de dados à saída de ultimo flip-flop do controlador
-conectar saída de novo flip-flop com entrada de ativação do elemento de falha
-fim
-conectar saída de ultimo flip-flop do controlador de ativação de falhas com primeiro

```

Figura 7-2:Pseudocódigo do “script” para inserção de falhas

7.2 Geração de vetores de teste

Comumente, o fluxo de geração de padrões funcionais que rodam no testador começa com a criação do “testbench”. A partir dele é projetado e simulado cada um dos padrões, extraindo suas formas de onda. As formas de onda capturadas são codificadas em uma linguagem intermediária e depois é realizada uma resimulação, para ver que não foram perdidos e/ou adicionado dados que resultem em uma

diferença no comportamento do circuito. Finalmente o resultado intermediário é convertido em uma linguagem que o testador entende, sofrendo uma ou várias transformações. Este processo se ilustra na Figura 7-3.

Figura 7-3: Fluxo de geração de padrões de teste

Da mesma forma foi idealizado um fluxo para a geração de vetores de teste para a co-emulação de falhas, com a diferença de que os vetores são extraídos diretamente da simulação, sem passar por nenhuma transformação e sem necessidade de resimulação. Salienta-se que os mesmos vetores podem ser transformados na linguagem do testador de forma direta, devido à maneira como são extraídos os dados. Isto resulta em uma vantagem para o método, já que não só se diminuem o número de passos para a extração de vetores de teste, como também se garante que tanto a co-emulação como o exercício do vetor no testador, trabalham com exatamente os mesmos dados.

Para facilitar o armazenamento e administração dos vetores foi definida a codificação ilustrada na Tabela 7-1. Cada palavra é constituída pelos quatro bits

mais significativos, que definem a função a aplicar, mais vinte e oito bits de dado. Desta forma, no caso de “single railed logic” é possível controlar vinte e oito sinais de entrada do DUT e observar vinte e oito saídas. Para dual railed logic este número se reduz para quatorze. Em caso de necessidade de mais entradas e/ou saídas se podem utilizar os seguintes quatro bits mais significativos para indicar o porto que se está controlando/observando.

Código(4 MSB)	Sinais(28 bits)	Significado
0x0	0xFFFFFFFF	Exercita as entradas do DUT com o valor de sinal especificado
0xF	0x(máscara)	Não compara saídas do DUT com bit de máscara 1
0x8	0xFFFFFFFF	Valor esperado nas saídas do DUT
0xC	0x(número)	Repete instruções em loop "número" vezes
0xE	0x00000000	Fim de loop
0x1	0x(número)	Mensagem para modelo comportamental "número"
0x2	0x(mensagem)	Dados da mensagem
0xD	0x00000000	Fim de mensagem

Tabela 7-1: Codificação dos vetores de teste

Definiu-se que os sinais extraídos da simulação devem ser do tipo “Non Return to Zero” (NRZ) e que a comparação dos sinais monitorados deve ser feita no início de cada ciclo. Isto significa que o relógio de captura deve ser no máximo meio período do o mínimo relógio do DUT. Também, que o padrão deve ser projetado para realizar a comparação dos sinais monitorados em momentos onde se garanta o estado estável dos mesmos.

Para o “testbench” de simulação foram projetados três elementos sincronizados pelo relógio de captura: “Driver”, “Monitor” e Gerador de vetores. Um “driver” é instanciado para cada entrada do DUT. Ele se encarrega de enviar o sinal de controle para as entradas, sincronizando o dado de entrada com o relógio de simulação. Da mesma forma um monitor é instanciado para cada saída. Ele compara o dado esperado com o observado na saída, no momento em que o relógio de captura sobe. Caso exista diferença no cotejo, aponta o erro. Deve-se usar uma lógica ‘OU’ entre todos os sinais de erro dos monitores, e no final da simulação

garantir que a sinalização de erro é zero, caso contrário deve-se realizar um ajuste no padrão e repetir a geração do vetor de teste. Quando o dado esperado é X não é feita comparação nenhuma para esse sinal monitorado, no ciclo especificado. O gerador de vetores captura as entradas e saídas, e gera um arquivo com a codificação do vetor de teste. Um ciclo de teste corresponde às sequências de entrada, mascara de saídas e saídas. Caso todas as saídas estejam mascaradas não se escreve no arquivo de teste a palavra de máscara nem a de saída. Um ciclo de teste é capturado para cada ciclo de relógio de captura. Cada palavra no arquivo resultante deve estar escrita usando a codificação anteriormente apresentada, quatro bits de instrução mais vinte e oito de dado. Na Figura 7-4 se apresenta um diagrama de blocos do testbench de geração de vetores de teste.

Adicionalmente foi criado um script para reduzir o tamanho do arquivo de vetor de teste. Este “script” busca os ciclos de teste subsequentes equivalentes, e os recodifica usando as instruções de início e fim de loop. Desta forma se poupa memória do controlador de co-emulação.

Figura 7-4: Diagrama de blocos do “testbench” de geração de vetores de teste

7.3 Administração do teste

Com a plataforma de co-emulação pronta, os modelos aplicados e o vetor de teste do padrão extraído, resta apenas aplicação do vetor de teste no DUT injetando uma a uma as falhas e observando se existem diferenças com respeito ao resultado esperado do circuito bom. A diferença indica a detecção da falha que foi injetada. O responsável para a execução desta tarefa é o controlador de co-emulação. Para isso foi feito um programa em C para o NIOSII que realiza as funções de administração do teste e avaliação dos modelos comportamentais. O programa foi feito de forma modular e parametrizável, para que possa ser facilmente adaptado à necessidade da co-emulação. Isto devido aos modelos comportamentais, número de portos de entrada ou saída, entre outras coisas. O resultado final da execução do programa corresponde ao número e as falhas detectadas com o padrão avaliado.

O programa que foi feito para a administração do teste começa pela inicialização do controlador de falhas e dos modelos. Isto é feito utilizando o sinal de reset de emulação e funções específicas para cada modelo. Depois é realizada a co-emulação do circuito bom. Cada ciclo do vetor de teste é aplicado ao DUT e é avaliado se existe alguma diferença da resposta esperada com a observada. O ciclo de teste está constituído por várias palavras codificadas da forma como se apresentou na Tabela 7-1. Se existir alguma diferença com a resposta esperada do circuito bom, significa que algo deve ser ajustado. Depois de corrigir o problema deve-se executar novamente o programa. Caso ainda existam diferenças deve-se ajustar o padrão ou o circuito até que a co-emulação passe sem erros. Entre cada uma das aplicações do ciclo de teste, são avaliados os modelos comportamentais. Eles podem modificar sua saída em função das entradas, ou agendar um evento para um ciclo de teste posterior.

O passo seguinte consiste em injetar uma falha e realizar a co-emulação novamente. Caso sejam detectadas diferenças na resposta esperada pelo DUT, significa que a falha injetada é detectável pelo padrão avaliado. Esta falha é armazenada e se incrementa o contador de falhas detectadas. Caso contrário, a avaliação do padrão continua até que ele termine. Finalizada a aplicação do padrão, remove-se a falha que foi injetada anteriormente, e ativa-se a próxima falha da lista. Uma a uma as falhas são injetadas, avaliadas e removidas, até que se processe toda a lista. No

final o controlador de co-emulação envia um relatório para o computador, através da porta USB, com o número de falhas que foram detectadas e a posição das mesmas na lista de falhas.

Na Figura 7-5 se apresenta o fluxograma do programa para a administração do teste, projetado para o controlador de co-emulação.

Em um passo posterior se deve realizar a reclassificação das falhas para encontrar o novo valor de cobertura. Para isto é importante obter as falhas que foram detectadas por cada padrão, de maneira que seja possível calcular a cobertura de forma incremental utilizando a seguinte equação:

$$cobertura_{final} = \frac{falhas_{dt_{scan}} + falhas_{dt_{padrão1}} + falhas_{dt_{padrão2}} + \dots + falhas_{dt_{padrãon}}}{Total_de_falhas_testáveis}$$

Todos os padrões podem ser gravados na memória ao mesmo tempo, e avaliados através da co-emulação de falhas. Deve-se considerar que para cada padrão as falhas são as mesmas e que falhas detectadas por um padrão podem ser igualmente detectadas por outro, a menos que se realize uma reconfiguração do circuito, removendo o sinal de ativação, ou modificando a execução do teste de forma a desconsiderar falhas já detectadas.

Figura 7-5: Fluxograma do programa do processador de co-emulação

8 Análises de resultados da co-emulação de falhas

Neste capítulo analisar-se-ão os resultados da co-emulação de falhas em circuitos integrados através do uso de dois conjuntos de circuitos. O primeiro, usado para medir a eficiência da plataforma, corresponde a um subconjunto dos circuitos de “benchmark” do ISCAS89; o segundo corresponde a um conjunto de circuitos criados especificamente para ilustrar as características da plataforma de co-emulação utilizando os distintos tipos de modelamento propostos neste trabalho. Adicionalmente, serão apresentados resultados de simulação de falhas serial com o intuito de comparar eficiências.

8.1 Circuitos de “benchmark”

Para caracterizar a plataforma de co-emulação foram realizados dois conjuntos de testes. No primeiro foi utilizado um subconjunto dos circuitos de “benchmark” do ISCAS89, os seguintes parâmetros foram observados:

- porcentagem de ocupação do circuito no FPGA em função do número de portas;
- cobertura de scan e cálculo de cobertura depois da co-emulação das falhas;
- aumento de tamanho do circuito para cada uma das transformações (“single railed logic”, “single railed logic” com falhas, “dual railed logic” modificado e “dual railed logic” modificado com falhas);
- eficiência dada pelo tempo da co-emulação de falhas, em função do tamanho do circuito, o número de falhas injetadas e o tamanho do padrão;
- eficiência dada pelo tempo da simulação de falhas, em função do tamanho do circuito, o número de falhas injetadas e o tamanho do padrão;
- porcentagem de redução do vetor de teste utilizando funções de repetição;

Os circuitos do ISCAS89 são puramente digitais. Na Tabela 8-1 se apresentam algumas características dos circuitos utilizados.

<i>Circuito</i>	<i>Origem</i>	<i>Entradas</i>	<i>Saídas</i>	<i>Clocks</i>	<i>Elementos Sequenciais</i>	<i>Inversores</i>	<i>Lógica</i>
s1196.1	ISCAS89	14	14	1	18	141	388
s1238	ISCAS89	14	14	1	18	80	428
s1423	ISCAS89	17	5	1	74	167	490
s1488	ISCAS89	8	19	1	6	103	550
s5378	ISCAS89	35	49	1	179	1775	1004
s9234.1	ISCAS89	36	39	1	211	3570	2027
s15850.1	ISCAS89	77	150	1	534	6324	3448
s35932	ISCAS89	35	320	1	1728	3861	12204
s38417	ISCAS89	28	106	1	1636	13470	8709
s38584	ISCAS89	38	304	1	1426	7805	11448

Tabela 8-1: Características dos circuitos de “benchmark” utilizados do ISCAS89

Para o outro conjunto de testes foram utilizados circuitos diferentes, que ilustram o potencial de modelamento da plataforma. O primeiro circuito é um conversor análogo digital (ADC) que possui uma parte analógica correspondente à fonte de referência, comparador, conversor digital analógico, etc; e uma parte digital, correspondente ao registro de aproximações sucessivas (“Successive Approximation Register”). O segundo circuito é uma unidade lógica e aritmética (ULA) de quatro bits, com pinos configuráveis de entrada e/ou saída. Finalmente, um terceiro circuito de memória com “wrapper” digital e BIST (“Built In Self Test”). A parte digital destes circuitos foi modelada usando DRLM; o circuito analógico do ADC foi representado usando um modelo comportamental; para os pinos entrada/saída da ULA foi utilizado um modelo estrutural, e para a memória foi utilizado uma memória DRLM. Os parâmetros que se observaram durante a execução do processo de graduação de falhas foram:

- porcentagem de ocupação do FPGA em função do número de portas do circuito;
- cobertura de scan e cálculo de cobertura depois da co-emulação das falhas utilizando padrões de scan como entrada;
- aumento de tamanho do circuito para cada uma das transformações (“single railed logic”, “single railed logic” com falhas, “dual railed logic” modificado e “dual railed logic” modificado com falhas), incluído os modelos de “hardblock”;
- eficiência dada pelo tempo da co-emulação de falhas, em função do tamanho do circuito, o número de falhas injetadas e o tamanho do padrão;

- eficiência dada pelo tempo da simulação de falhas, em função do tamanho do circuito, o número de falhas injetadas e o tamanho do padrão;
- porcentagem de redução do vetor de teste utilizando funções de repetição;
- cálculo de cobertura depois da co-emulação de falhas usando padrões complementares;
- tamanho dos padrões complementares;

Na Figura 8-1 se apresenta um diagrama de blocos do conversor análogo digital. As falhas “stuck at” se encontram unicamente na parte digital, incluindo a interface com a parte analógica. Para a simulação de falhas, o circuito analógico é representado através de um modelo verilog simplificado, como costuma ser feito nas simulações de circuitos digitais com interfaces analógicas. Para a co-emulação de falhas, o circuito analógico é representado usando o modelo comportamental do mesmo; ao passo que a parte digital é modelada usando “single” ou “dual railed logic” modificado.

Figura 8-1: Diagrama de blocos do conversor análogo digital

A Figura 8-2 ilustra a unidade lógica e aritmética com pinos de entrada/saída configurados permanentemente como entradas ou saídas, e pinos configuráveis como entradas ou saídas dependendo da seleção do modo de operação. As funções de alguns pinos são compartilhadas entre entrada, saída de dado e “scan”. Para a

simulação, os pinos de entrada/saída são modelados usando verilog; enquanto que na co-emulação é usado o modelo estrutural.

Figura 8-2: ULA com pinos de entrada/saída configuráveis

Na Figura 8-3 apresenta-se um diagrama de blocos do circuito com memória RAM, BIST, barramento de escrita e leitura de dados. Similar aos casos anteriores, para a simulação, a memória é modelada usando “verilog”. Para a co-emulação é usado o modelo “dual railed logic” modificado de memória.

Figura 8-3: Diagrama de blocos de circuito com memória

Na Tabela 8-2 se apresentam algumas das características do segundo conjunto de circuitos usados para “benchmark”. Para a criação destes circuitos foi usada uma

descrição RTL comportamental, enquanto que os circuitos do ISCAS89 estão descritos de forma estrutural.

<i>Circuito</i>	<i>Origem</i>	<i>Entradas</i>	<i>Saídas</i>	<i>Entrada/ Saída</i>	<i>Clocks</i>	<i>Hardblocks</i>
<i>ADC_SAR</i>	Pesquisa	3	9	--	1	Comparador e DAC
<i>ALU_PAD</i>	Pesquisa	--	--	22	1	PADs
<i>Memory_wrapper</i>	Pesquisa	45	12	--	2	Memória

Tabela 8-2: Algumas características do segundo conjunto de circuitos usados para “benchmark”

8.2 Metodologia: “Benchmark” ISCAS89

O primeiro passo do processo de avaliação foi sintetizar os circuitos, criando uma cadeia de scan para cada caso, usando todos os seus “flip-flops”. Depois, foram gerados padrões de scan usando uma ferramenta de ATPG. Nos “scripts” de ATPG, foi agregada a restrição de mascaramento de saídas na captura, de forma que se limita a cobertura.

Na Tabela 8-3 apresentam-se os resultados da síntese e do ATPG. O número de células de lógica e inversores diminui, comparado com os valores originais, devido ao uso de células complexas. Adicionalmente, os elementos sequenciais correspondem a “flip-flops” de “scan”, isto é, um multiplexador na entrada de dados do “flip-flop” incluído dentro da mesma célula e controlado pelo sinal de habilitação. Os circuitos que têm baixa cobertura possuem bastante lógica combinacional conduzindo as saídas. Nestes casos, provavelmente a ferramenta de ATPG estaria em condições de cobrir 100% das falhas, mas a restrição de mascaramento evita que sejam gerados padrões que tentem cobri-los, assim como também restringe a observação das falhas na simulação. Os valores da Tabela 8-3 são usados como referência para as próximas transformações.

<i>Circuito</i>	<i>Sequenciais</i>	<i>Inversores</i>	<i>Lógica</i>	<i>Falhas testáveis</i>	<i>Falhas DT scan</i>	<i>Cobertura scan</i>	<i>Padrões scan</i>
s1196.1	18	37	357	2744	1738	63,34%	107
s1238	18	39	363	2782	1732	62,26%	106
s1423	74	42	389	3426	3364	98,19%	52
s1488	6	38	388	2894	1622	56,05%	89
s5378	179	142	860	8035	5959	74,16%	77
s9234.1	211	140	1135	10144	9968	98,26%	120
s15850.1	534	206	2126	20063	19449	96,94%	132
s35932	1728	9	6598	58780	57084	97,11%	44
s38417	1636	537	7172	63477	63219	99,59%	144
s38584	1426	797	7762	68826	67710	98,38%	168

Tabela 8-3: Resultados de sínteses e ATPG

O passo seguinte foi compilar os circuitos para o FPGA, usando o modelo “single railed logic” e “dual railed logic” modificado. A Tabela 8-4 ilustra estes resultados. Nos elementos lógicos se incluem tanto os registros quanto a lógica combinacional. O número de flip-flops para o modelo DRLM é exatamente o dobro do que foi usado para o SRL. Por outro lado, quando comparados estes dois tipos de representação, observa-se que os recursos utilizados pela lógica combinacional cresceram em maior proporção. Na média a utilização de elementos lógicos aumentou 520% nos circuitos avaliados.

<i>Circuito</i>	<i>Single Railed Logic</i>		<i>Dual Railed Logic Modificado</i>	
	<i>Elementos Lógicos</i>	<i>% de ocupação</i>	<i>Elementos Lógicos</i>	<i>% de ocupação</i>
s1196.1	208	<1%	1059	2%
s1238	218	<1%	1042	2%
s1423	162	<1%	1227	2%
s1488	257	<1%	1175	2%
s5378	508	<1%	2647	4%
s9234.1	542	<1%	3787	6%
s15850.1	1058	<2%	7471	11%
s35932	2917	4%	17736	26%
s38417	3269	5%	24275	35%
s38584	3860	6%	27523	40%

Tabela 8-4: Resultados de compilação de circuitos para o FPGA usando SRL e DRLM

Depois foram utilizados os scripts para a síntese do controlador de falhas e ativação das mesmas. Para cada tipo de modelo (SRL ou DRLM) foi criado um “script”

diferente. Estes “scripts” usam como entrada a lista de falhas a serem injetadas, correspondente às falhas não detectadas pelo “scan”, e a lista de conexão do circuito sintetizado descrita em “Verilog”. A Tabela 8-5 apresenta os resultados de compilação destes circuitos para o FPGA, junto com o número de falhas que foram adicionadas. Como não existe diferença no controlador de falhas entre os modelos SRL e DRLM, e como a diferença dos circuitos de ativação é de uma porta que pode ser incorporada dentro de um elemento lógico, o delta de utilização de elementos lógicos nos modelos SRL e DRLM é aproximadamente constante. Com a Tabela 8-4 e a Tabela 8-5 calculou-se o aumento da utilização de Elementos lógicos por unidade de falha média ($\overline{U_{EL}}$) através do uso da seguinte equação:

$$\overline{U_{EL}} = \frac{1}{2} \left(\frac{\sum_{i=0}^n \left(\frac{DRLMF_{EL} - DRLM_{EL}}{falhas_i} \right)}{n} + \frac{\sum_{i=0}^n \left(\frac{SRLF_{EL} - SRL_{EL}}{falhas_i} \right)}{n} \right) = 0,985 \frac{EL}{falha}$$

Circuito	Falhas injetadas	Single Railed Logic			Dual Railed Logic Modificado		
		Elementos Lógicos	Registros dedicados	% de ocupação	Elementos Lógicos	Registros dedicados	% de ocupação
s1196.1	1006	1156	1025	2%	2062	1043	3%
s1238	1050	1209	1069	2%	2109	1087	3%
s1423	62	226	137	<1%	1283	211	2%
s1488	1272	1441	1279	2%	2429	1285	4%
s5378	2076	2515	2256	4%	4766	2435	7%
s9234.1	176	735	388	1%	3944	599	6%
s15850.1	614	1671	1149	2%	8068	1683	12%
s35932	1696	4621	3425	7%	19383	5153	28%
s38417	258	3521	1895	5%	24582	3531	36%
s38584	1116	4919	2543	7%	28540	3969	42%

Tabela 8-5: Utilização de recursos do FPGA nos modelos SRL e DRLM, com controlador de falhas e circuitos de ativação

Com o intuito de diminuir o tempo de criação dos padrões de teste, foram utilizados os mesmos padrões gerados pela ferramenta de ATPG como os padrões a serem graduados pela ferramenta de co-emulação de falhas. Para isto foi necessário a criação de um “script” que transforma os padrões codificados em WGL (Waveform Generation Language) para a codificação que foi proposta neste trabalho e apresentada no capítulo sete. Este “script” extrai unicamente os estímulos de

entrada do padrão e gera um arquivo com os vetores de entrada. Estes vetores foram utilizados dentro de um testbench que foi criado para capturar as saídas do “circuito bom” e mascarar os sinais indeterminados, similar ao apresentado no capítulo anterior. O resultado da simulação deste “testbench” é um arquivo com os vetores de entrada, as mascaras e saídas esperadas. Este arquivo é utilizado como entrada de um “script” que foi criado para realizar a transformação da codificação de sinais SRL para DRLM. O resultado é usado por outro “script” que foi criado para dividir os vetores de mais de 28 bits em vetores de 24 bits com a codificação da instrução e número de porta. Adicionalmente foi escrito um “script” que diminui o tamanho do arquivo do padrão, através da codificação de laços de repetição. A Tabela 8-6 contém o número de vetores resultantes para cada um dos circuitos avaliados, em cada uma das suas transformações. Os vetores de entrada correspondem aos vetores de estímulos dados pelo WGL. Os vetores de teste correspondem aos vetores de estímulo, junto com as saídas e as máscaras. Os vetores de teste DRLM são os mesmos vetores de teste, mas depois serem processados pelo script que transforma os sinais SRL em DRLM, e o que fragmenta vetores de mais de 28 bits. Na última coluna da tabela se encontram os números de vetores DRLM depois de serem processados pelo script de compressão.

<i>Circuito</i>	<i>Vetores de entrada</i>	<i>Vetores de teste</i>	<i>Vetores de teste DRLM</i>	<i>Compressão DRLM</i>
s1196.1	4398	8797	17594	16892
s1238	4359	8720	17440	16634
s1423	8358	16717	25076	22384
s1488	1470	2941	4411	4339
s5378	28957	57917	260626	251414
s9234.1	52396	104793	419172	393731
s15850.1	144718	289437	2894367	2743284
s35932	162619	324975	5034088	4639110
s38417	481570	963141	5778843	5173023
s38584	488374	976749	14651224	13789546

Tabela 8-6: Tamanho de vetores de teste

Com a lista de conexões modelada com DRLM e o vetor de teste, foi possível realizar a co-emulação de falhas de cada um dos circuitos. Para isso foi necessário

conectar e compilar o processador de co-emulação junto à lista de conexões com falhas DRLM. Adicionalmente foi utilizado o programa desenvolvido para a co-emulação, que administra o teste e reporta as falhas que foram detectadas pelo padrão. Foi utilizado o circuito SRL com falhas para realizar a simulação de falhas serial e comparar o tempo de avaliação da simulação com a co-emulação. Na Tabela 8-7 se encontram os resultados obtidos. O aumento na cobertura destes padrões é devido à inexistência das restrições de configuração que foram estipuladas para a ferramenta de ATPG. Por outro lado, não se consegue cobertura final de 100% em todos os circuitos devido aos padrões terem sido gerados com restrições que evitaram o direcionamento da detecção de alguns tipos de falhas.

<i>Circuito</i>	<i>Falhas detectadas</i>	<i>Nova cobertura</i>	<i>Tempo simulação de falhas [segundos]</i>	<i>Tempo co-emulação de falhas [segundos]</i>	<i>% Co-emulação / Simulação</i>
s1196.1	910	96,50%	61,3	2,22	3,62%
s1238	889	94,21%	70,2513	2,53	3,60%
s1423	62	100,00%	61,9306	0,09	0,15%
s1488	1264	99,72%	16,0716	0,29	1,80%
s5378	2032	99,45%	2008,92	10,02	0,50%
s9234.1	176	100,00%	93,7617	0,52	0,55%
s15850.1	614	100,00%	1512,92	3,61	0,24%
s35932	1696	100,00%	78661,2	35,72	0,05%
s38417	258	100,00%	5884,8	3,05	0,05%
s38584	1116	100,00%	28687,6	14,35	0,05%

Tabela 8-7: Resultados da simulação e co-emulação de falhas

Dada a natureza do processo de co-emulação de falhas, o tempo de execução deveria ser, no pior dos casos, diretamente proporcional ao tamanho do vetor de teste multiplicado pelo número de falhas injetadas. À possibilidade de interromper a introdução do vetor de teste no momento em que a falha é detectada, introduz uma não linearidade nesta proporção. Quer dizer que circuitos que possuem vetores de teste que detectam a maioria das falhas avaliadas no início da sua execução, conseguem tempos de co-emulação de falhas menores que circuitos com vetores de teste menores. Um exemplo disto pode ser observado comparando-se os resultados

dos circuitos s9234.1 e s1196.1, onde o primeiro gasta pouco mais de 20% do tempo do segundo, sendo que o tamanho do seu vetor de teste multiplicado pelas falhas injetadas é quase 4 vezes maior. Um caso particular que varia de circuito para circuito, dependendo do padrão.

Por outro lado, o tempo da simulação de falhas serial está dado por duas componentes: a primeira é o tamanho do padrão de teste multiplicado pelas falhas, que dá o tempo simulado; e a segunda corresponde ao tempo de avaliação e propagação de eventos, que depende diretamente das características do circuito (tamanho e complexidade) e do padrão ou estímulo utilizado. Assim sendo, por cada unidade de tempo de simulação são avaliados vários eventos em forma serial; enquanto na co-emulação só existe o tempo do estímulo e a avaliação é feita concorrentemente. Estes fatos fazem com que a co-emulação de falhas seja muito mais rápida que a simulação, e a diferença de tempo aumenta com a complexidade e tamanho do circuito, junto com o tempo e características do padrão avaliado.

Nos resultados da Tabela 8-7 pode-se observar que a co-emulação de falhas chegou a ser 2200 vezes mais rápida que a simulação. Esta diferença poderia ser ainda maior se fossem avaliados circuitos maiores e mais complexos.

8.3 Metodologia: “Benchmark” circuitos com “hardbloks”

Para estes tipos de circuitos se seguiu uma metodologia similar à anterior, mas que considera as particularidades dos blocos não digitais. Adicionalmente, foram criados padrões que exercitaram o funcionamento destes blocos e que permitiram a detecção de falhas bloqueadas em “scan”.

Seguindo esta idéia, foram sintetizados os circuitos descritos em RTL e foi inserida a cadeia de “scan”. No caso do circuito ALU_PAD, dois sinais de seleção de modo de operação permitem a configuração dos PADs. Desta maneira, um dos modos de operação foi utilizado para scan, aproveitando os PADs disponíveis e reconfigurando-os para realizar esta função. Foram gerados padrões de “scan” utilizando uma ferramenta de ATPG. Na Tabela 8-8 se apresentam estes resultados. Os valores baixos de cobertura são devidos às interfaces com modelos de

“hardblocks” que bloqueiam a detecção de falhas, ou devido a bloqueios da própria configuração de “scan”.

<i>Circuito</i>	<i>Sequenciais</i>	<i>Inversores</i>	<i>Lógica</i>	<i>Falhas testáveis</i>	<i>Falhas DT scan</i>	<i>Cobertura scan</i>	<i>Padrões scan</i>
ADC_SAR	10	9	39	425	340	80,00%	15
ALU_PAD	72	11	54	899	777	74,21%	20
Mem_wrapper	158	77	825	7730	7364	95,27%	107

Tabela 8-8: Resultados de sínteses e ATPG

O passo seguinte foi compilar os circuitos para o FPGA, tanto para SRL como DRLM. Neste ponto cada circuito tem suas particularidades referentes ao modelo de “hardblock” que utiliza. O caso do ADC_SAR utiliza um modelo comportamental, desta forma o funcionamento da parte analógica é feito pelo controlador de co-emulação. Devido a isto o resultado de Elementos Lógicos utilizados só considera a parte digital. No caso do circuito ALU_PAD, não foi feito um modelo dos PADS para interfaces SRL. O cálculo do número de Elementos Lógicos necessários foi feito removendo os PADS e levando os seus sinais de interface para o topo do circuito. Poderia se modificar o elemento de interface entre a lógica de quatro níveis e a de três, para suportar conversões de quatro para dois, substituindo os estados indeterminado e alta impedância por um estado lógico ‘um’ ou ‘zero’; mas preferiu-se manter as características do circuito, evitando introduzir erros de detecção de falhas. No caso de DRLM se utilizou o modelo estrutural de PAD apresentado no capítulo seis. Desta forma os resultados de utilização de recursos do FPGA para o circuito ALU_PAD DRLM incluem os recursos utilizados por este modelo. Vale a pena salientar que dependendo da característica do circuito, a compilação pode acabar removendo parte de lógica não utilizada do modelo estrutural. No circuito analisado, a redução e simplificação fez com que os modelos de PADS ocupassem entre 2 e 10 elementos lógicos. Finalmente para o circuito “Mem_wrapper”, a compilação foi feita utilizando os blocos de memória disponíveis dentro do FPGA. No caso do SRL foi utilizado um único bloco de memória com 8kB e para o DRLM dois blocos de memória, junto com a lógica necessária para cumprir com as regras de modelamento de memórias. O aumento do uso de elementos lógicos para este circuito em DRLM foi maior que nos casos anteriormente analisados, devido a uma

grande quantidade de lógica combinacional utilizada como rede de observação dos controles de memória quando o circuito se encontra configurado para “scan”. Quanto maior é a lógica combinacional, maior é a proporção de utilização de recursos do FPGA. Adicionalmente, o caso DRLM possui uma lógica que faz parte do modelo da memória e que não está presente no circuito SRL. Veja os resultados de ocupação de recursos do FPGA para cada um dos circuitos na Tabela 8-9.

<i>Circuito</i>	<i>Single Railed Logic</i>		<i>Dual Railed Logic Modificado</i>	
	<i>Elementos Lógicos</i>	<i>% de ocupação</i>	<i>Elementos Lógicos</i>	<i>% de ocupação</i>
ADC_SAR	26 ⁽¹⁾	<1%	224 ⁽¹⁾	<1%
ALU_PAD	70 ⁽¹⁾	<1%	711	1%
Mem_wrapper	491 ⁽¹⁾	<1%	4230 ⁽¹⁾	6%

(1) Só considera a parte digital do circuito

Tabela 8-9: Resultados de compilação de circuitos para o FPGA usando SRL e DRLM

A síntese do controlador de falhas e circuitos de ativação foi feita com os “scripts” descritos na sessão anterior. A Tabela 8-10 apresenta os resultados de compilar os circuitos com falhas para o FPGA. O valor de aumento da utilização de elementos lógicos por unidade de falha média se manteve próximo do valor calculado; a diferença observada se atribui aos circuitos adicionados no DRLM que não existem no SRL.

<i>Circuito</i>	<i>Falhas injetadas</i>	<i>Single Railed Logic</i>			<i>Dual Railed Logic Modificado</i>		
		<i>Elementos Lógicos</i>	<i>Registros dedicados</i>	<i>% de ocupação</i>	<i>Elementos Lógicos</i>	<i>Registros dedicados</i>	<i>% de ocupação</i>
ADC_SAR	85	117 ⁽¹⁾	78	<1%	319 ⁽¹⁾	18	<1%
ALU_PAD	122	183 ⁽¹⁾	156	<1%	819	165	1%
Mem_wrapper	292	817 ⁽¹⁾	527	1%	4554 ⁽¹⁾	686	7%

(1) Só considera a parte digital do circuito

Tabela 8-10: Utilização de recursos do FPGA nos modelos SRL e DRLM, com controlador de falhas e circuitos de ativação

Dois conjuntos de padrões foram avaliados para cobrir as falhas destes circuitos. O primeiro é o obtido a partir da conversão do padrão WGL gerado pela ferramenta de ATPG. O segundo é criado manualmente, de forma que se exercitam as interfaces com hardblocks e se cobrem falhas que normalmente se encontram bloqueadas em scan. Na Tabela 8-11 se encontram os tamanhos dos vetores de teste, depois de passar por cada uma das transformações até sua obtenção.

<i>Circuito</i>	<i>Vetores de entrada</i>	<i>Vetores de teste</i>	<i>Vetores de teste DRLM</i>	<i>Compressão DRLM</i>
ADC_SAR	396	787	787	756
ALU_PAD	1608	3217	6434	6112
Mem_wrapper	35198	70397	175994	156635

Tabela 8-11: Tamanho de vetores de teste

Na Tabela 8-12 se encontram os resultados do tempo empregado para realizar a simulação e co-emulação de falhas. Nota-se que o aumento de cobertura foi pouco, devido à necessidade do exercício da interface entre o “hardblock” e a parte digital; tarefa difícil de se conseguir com “scan”. Adicionalmente, a configuração do circuito neste teste faz com que muitos sinais sejam fixados em um valor.

<i>Circuito</i>	<i>Falhas detectadas</i>	<i>Nova cobertura</i>	<i>Tempo simulação de falhas [segundos]</i>	<i>Tempo co-emulação de falhas [segundos]</i>	<i>% Co-emulação / Simulação</i>
ADC_SAR	24	85,65%	2,90356	0,05	1,72%
ALU_PAD	5	86,99%	21,1	0,75	3,55%
Mem_wrapper	192	97,75%	32,36	0,14	0,43%

Tabela 8-12: Resultados da simulação e co-emulação de falhas

Para a criação de padrões que exercitem as falhas não cobertas, foi necessário analisar independentemente cada circuito e aproveitar as características que o modelo utilizado oferece. No caso do ADC_SAR, a maioria das falhas não cobertas se encontra na interface com a parte analógica. Esta última foi descrita através de um modelo comportamental executado pelo processador de co-emulação durante o

exercício do padrão. Para interagir com esse modelo foi necessário agregar processos no testbench de geração dos vetores de teste para permitir a configuração do comparador através do envio de mensagens, que são codificadas dentro do mesmo padrão. Desta forma, o “testbench” escreve a mensagem dentro do padrão em quanto extrai os vetores de teste através da simulação do circuito “bom”. Depois, no momento da co-emulação, estas mensagens são lidas pelo processador de co-emulação que as direciona para o modelo comportamental endereçado. A leitura e decodificação das mensagens, junto com a avaliação do modelo comportamental introduzem um atraso na co-emulação. No circuito avaliado o modelo utilizado foi muito simples, e mesmo com o atraso introduzido a co-emulação conseguiu ser 15 vezes mais rápida que a simulação, para um circuito pequeno. No circuito ALU_PAD a maior parte das falhas não cobertas se devem a bloqueios de sinal por causa da seleção do modo de operação e à falta de cobertura em alguns sinais de controle dos PADs como os relacionados com a lógica de “PULL”. Foi criado um padrão que consegue avaliar a resposta do circuito a estímulos adequados em cada modo de operação, de forma que se exercitou grande parte da lógica que foi bloqueada durante o “scan”. Cabe salientar que o modelo estrutural utilizado não introduz nenhum atraso no processo de co-emulação de falhas. Finalmente o circuito “Mem_wrapper” possui algumas falhas não cobertas na lógica de seleção de teste e na interface com a memória. Foi feito um padrão simples que realiza algumas escritas e leituras na memória, junto com o exercício do padrão de BIST.

<i>Circuito</i>	<i>Vetores</i>	<i>Falhas detectadas</i>	<i>Nova cobertura</i>	<i>Tempo simulação de falhas [segundos]</i>	<i>Tempo co-emulação de falhas [segundos]</i>	<i>% Co-emulação / Simulação</i>
ADC_SAR	350	73	100%	0,15	0,01	6,67%
ALU_PAD	448	91	96,55%	3,04	0,05	1,64%
Mem_wrapper	42294	292	100%	1234	0,84	0,07%

Tabela 8-13: Resultado de padrões que exercitam blocos não digitais

9 Conclusões

Os fabricantes de circuitos integrados estão na busca constante pela melhora dos processos de fabricação e teste, visando a distribuição de produtos funcionais e sem defeitos. O processo de teste de CIs tem por objetivo a melhora da qualidade com o menor custo, o qual está dado pelos recursos de desenvolvimento, área utilizada e tempo de aplicação. O projeto para testabilidade (DFT) estuda as técnicas que possibilitam o aumento da testabilidade e a geração de testes para a avaliação de circuitos, através da adição de estruturas e a criação de padrões de estímulo. Estas técnicas possibilitam a verificação do processo de fabricação através da inspeção do maior número de falhas e a aplicação de estímulos que permitam a detecção no menor tempo.

Atualmente são projetados sistemas em “chip” complexos constituídos por uma série de blocos digitais, analógicos e de sinal misto que se comunicam e realizam funções em conjunto. Para o teste, esta mistura representa um desafio já que devem ser realizados padrões eficientes que consigam verificar os blocos individualmente, assim como também estimular a interface entre eles.

“Scan” é uma técnica bem conhecida que ajuda a detectar a maior parte das falhas de um circuito digital eficientemente. No entanto, algumas falhas são difíceis de serem detectadas com estes padrões devido às restrições de controle e observação impostas pela configuração e funcionamento do teste. Em resumo, a maioria das restrições se localizam na interface entre blocos analógicos e digitais, a lógica de teste, e de configuração do modo de operação. Dependendo do número de falhas do circuito restante, estas restrições podem afetar o resultado de cobertura em maior ou menor grau. Acredita-se que em sua maioria, as falhas digitais não cobertas pelo “scan” deveriam ser cobertas pelos padrões funcionais e de memórias. Para medir o aporte destes padrões na cobertura, assim como a qualidade dos mesmos, é possível usar técnicas de simulação de falhas. O problema desta abordagem para a graduação de falha é o tempo que toma sua execução. Adicionalmente, empregam processos complexos que exigem poder computacional e de memória. A utilização destes recursos aumenta drasticamente com o tamanho do circuito, o número de falhas analisadas e o tamanho do padrão.

Alguns trabalhos prévios mostraram que através da emulação de “hardware” é possível realizar a graduação de padrões de teste com maior velocidade do que a simulação. Estas técnicas podem ser usadas em circuitos digitais empregando distintos modelos de falha. Apesar de sua eficiência, a falta de elementos para modelar circuitos não puramente digitais, dificulta a utilização destas técnicas para a graduação de falhas em circuitos integrados reais.

Nesta tese foi apresentada uma plataforma e o método para a co-emulação de falhas que combina um grupo de modelos heterogêneos em um FPGA para a graduação de falhas de padrões de teste. Adicionalmente, foram definidos quatro tipos de modelos que podem ser utilizados no FPGA destinados a representar as distintas partes que constituem o DUT. Estes modelos podem representar circuitos analógicos, digitais, de sinal misto ou memórias.

O modelo “dual railed logic” modificado pode ser usado para modelar qualquer tipo de sinal digital. A diferença com o que o antecede é que não possui o risco de função nos estados lógicos onde os circuitos trabalham na maior parte do tempo. Também, o método de geração de modelos resulta em circuitos que respeitam completamente a tabela verdade da célula original. Sua utilização permite a classificação adequada das falhas e disponibiliza um padrão de código que pode ser usado por outros modelos para representar circuitos de sinal misto. Porém, o modelo “dual railed logic” modificado ocupa mais recursos do FPGA do que o “dual railed logic” e o “single railed logic”.

Com as recomendações apresentadas para a criação de modelos de memória DRLM é possível emular o comportamento da memória considerando o sinal indeterminado. A inicialização de modelos de memória de leitura e escritura com estado indeterminado ou com dados específicos deve ser evitada, já que insere atraso no processo de co-emulação. Caso seja uma necessidade e disponha de recursos livres no FPGA, é possível criar um modelo que duplique o modelo DRLM de memória, e junto com um controlador simples preencher os dados requeridos. Neste caso, entre cada uma das execuções de padrão é possível usar um dos modelos enquanto o outro está sendo preenchido.

Com o tipo de modelo estrutural proposto é possível representar grande variedade de circuitos, através do uso das células apresentadas. Ele introduz à codificação

“dual railed logic” modificada, o estado lógico de alta impedância. Além disso, permite o uso de força de sinal. Este tipo de modelamento deve ser preferido ao comportamental, já que não introduz atraso no processamento do padrão avaliado.

O tipo de modelamento comportamental é o que oferece maior flexibilidade para representar qualquer tipo de circuito. Porém, deve-se restringir o seu uso aos casos onde seja estritamente necessário, devido ao atraso que introduz. Além da flexibilidade, ele proporciona a capacidade de receber mensagens de configuração codificadas nos vetores de teste; permitindo a programação de eventos que ocorreriam no teste real.

Neste trabalho foram desenvolvidos vários “scripts”, “testbench” e programas que permitiram a automatização de partes do fluxo de co-emulação, como a síntese do processador de falhas; a geração, extração, codificação e compressão dos vetores de teste; transformação do DUT para DRLM; a administração do processo de co-emulação; entre outros. Adicionalmente, para a avaliação do método e plataforma, foram criados outros “scripts” e “testbench” para a extração de padrões a partir de arquivos WGL; o algoritmo e execução da simulação de falhas serial; entre outros.

A codificação dos vetores de teste proposta provou ser efetiva oferecendo as funções necessárias para a aplicação do teste, como o mascaramento, a configuração de laços de repetição e o envio de mensagens para modelos comportamentais, fora da codificação das entradas e saídas esperadas.

A plataforma de co-emulação foi avaliada usando dois conjuntos de circuitos. O primeiro estava constituído por um sub-grupo dos ISCAS89. Com eles foi constatada a eficiência do processo de co-emulação de falhas comparado com a simulação.

O tempo de execução da co-emulação é, no pior dos casos, diretamente proporcional ao tamanho do vetor de teste multiplicado pelo número de falhas injetadas. A possibilidade de interromper a introdução do vetor de teste no momento em que a falha é detectada, introduz uma não linearidade nesta proporção. Quer dizer que circuitos que tinham vetores de teste que detectam a maioria das falhas avaliadas no início da sua execução, conseguiram tempos de co-emulação de falhas menores que circuitos com vetores de teste menores.

Por outro lado, o tempo da simulação de falhas serial está dado por duas componentes: a primeira é o tamanho do padrão de teste multiplicado pelas falhas, que dá o tempo simulado; e a segunda corresponde ao tempo de avaliação e propagação de eventos, que depende diretamente das características do circuito (tamanho e complexidade) e do padrão ou estímulo utilizado. Assim sendo, por cada unidade de tempo de simulação são avaliados vários eventos em forma serial; enquanto na co-emulação só existe o tempo do estímulo e a avaliação é feita concorrentemente. Estes fatos fazem com que a co-emulação de falhas seja muito mais rápida que a simulação. A diferença de tempo aumenta com a complexidade e tamanho do circuito, junto com o tempo e características do padrão avaliado.

Nos resultados se observou que a co-emulação de falhas chegou a ser 2200 vezes mais rápida que a simulação. Esta diferença poderia ser ainda maior se fossem avaliados circuitos maiores e mais complexos, ou circuitos pequenos nos quais a simulação agenda muitos eventos simultâneos.

Com respeito ao uso de recursos do FPGA, o número de “flip-flops” para o modelo DRLM foi exatamente o dobro do que foi usado para o SRL. Por outro lado, quando comparados estes dois tipos de representações, observou-se que os recursos utilizados pela lógica combinacional cresceram em maior proporção. Na média a utilização de elementos lógicos aumentou 520% nos circuitos avaliados.

A síntese do controlador de falhas resultou numa média de utilização de elementos lógicos por unidade de falha igual a 0,985. Isto se aplica tanto para SRL como para SRLM.

O segundo grupo de circuitos utilizados para fazer “benchmark” estava constituído por elementos com uma parte digital e outra que podia ser representada usando os outros tipos de modelos apresentados neste trabalho. O intuito foi exemplificar casos típicos de circuitos cuja cobertura de “scan” é afetada pela falta de controle e observação da interface com modelos de “hardblocks”, e como através da graduação de padrões funcionais, usando a plataforma de co-emulação, é possível completar a verificação de fabricação. Adicionalmente, desejava-se observar a interação dos distintos modelos e a validação da metodologia.

Para estes propósitos os circuitos criados foram um conversor analógico digital com um controlador de aproximações sucessivas, uma memória RAM com controlador e lógica de BIST, e uma ULA com PADS configuráveis de entrada/saída. Os resultados mostraram que embora os circuitos fossem pequenos, a co-emulação de falhas foi várias vezes mais rápida que a simulação. Além disso, verificou-se o aporte de cobertura dos padrões criados fazendo com que fossem detectadas a maior parte das falhas não cobertas pelo “scan”. Os resultados do número de falhas detectadas pelos padrões, através da graduação com a simulação e a co-emulação foram exatamente os mesmos, o que permitiu concluir que os modelos verilog da simulação tiveram uma total correspondência com os modelos criados usando os tipos propostos.

10 Trabalhos futuros

Propõe-se a continuação da pesquisa através dos seguintes trabalhos futuros:

- Estudar alternativas para eliminar o risco de função que existe entre o sinal lógico '1' e 'X', através do uso de lógica de três trilhos ou automação de artifícios de síntese de sistemas assíncronos em FPGA.
- Confrontar a eficiência do método e plataforma de co-emulação de falhas, comparado com o resultado obtido por uma ferramenta de simulação de falhas comercial.
- Estudar a forma de ampliar a disponibilidade de recursos do FPGA, através da distribuição e compilação do DUT, o controlador de falhas e o controlador de co-emulação em uma matriz de FPGAs.
- Implementar a co-emulação de falhas com outros modelos de falhas distintos do "stuck-at".
- Aproveitar idéias dos algoritmos de simulação de falhas, como o paralelo, hierárquico, etc, para aumentar ainda mais a eficiência da co-emulação.
- Realizar a co-emulação de falhas de um circuito integrado completo.

Bibliografia

- Abramovici1977 Abramovici, M.; Breuer, M. A. & Kumar, K. Concurrent fault simulation and functional level modeling 14 th Design Automation Conference - DAC, 1977, 128-135
- Abramovici1990 Abramovici, M. Breuer, M. A. F. A. D. Digital Systems Testing and Testable Design John Wiley & Sons, 1990, 672
- ADRIATIC2002 ADRIATIC Reconfigurability requirements for wireless LAN products
IMEC:http://www.imec.be/adriatic/deliverables/adriatic_deliverable_D1.1.zip, 2002
- AEC2008 AEC - Q100-007 - Rev-B: Fault Simulation and Test Grading Component Technical Committee,
- Agrawal1990 Agrawal, P. & Dally, W. J. A hardware logic simulation system IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1990, 9, 19-29
- Altera1998 Altera. Cyclone II Memory Blocks. 32p. 1998
http://www.altera.com/literature/hb/cyc2/cyc2_cii51008.pdf
- Altera2009 Altera Nios Embedded Processor 2009.
<http://www.altera.com/products/ip/processors/nios/nio-index.html>
- Antoni2000 Antoni, L.; Leveugle, R. & Feher, B. Using run-time reconfiguration for fault injection in hardware prototypes Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2000, 405-413
- Antoni2001 Antoni, L.; Leveugle, R. & Feher, B. Using run-time reconfiguration for fault injection applications Proc. 18th IEEE Instrumentation and Measurement Technology Conference IMTC 2001, 2001, 3, 1773-1777 vol.3
- Antoni2002 Antoni, L.; Leveugle, R. & Feher, M. Using run-time reconfiguration for fault injection in hardware prototypes Proc. 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems DFT 2002, 2002, 245-253
- Antoni2003 Antoni, L.; Leveugle, R. & Feher, B. Using run-time reconfiguration for fault injection applications IEEE Transactions on Instrumentation and Measurement, 2003, 52, 1468-1473
- Antreich1987 Antreich, K. J. & Schulz, M. H. Accelerated Fault Simulation and Fault Grading in Combinational Circuits #IEEE_J_CAD#, 1987, 6, 704-712
- Armstrong1972 Armstrong, D. B. A Deductive Method for Simulating Faults in Logic Circuits IEEE Transactions on Computers, 1972, 464-

471

- Baucom1998 Baucom, R. J.; DeLong, T. A.; Smith, D. T.; Johnson, B. W. & Hanna, J. P. VHDL-based distributed fault simulation using SAVANT Proc. IEEE 1998 National Aerospace and Electronics Conference NAECON 1998, 1998, 565-573
- Bauer2008 Bauer, L.; Shafique, M. & Henkel, J. Run-time instruction set selection in a transmutable embedded processor DAC '08: Proceedings of the 45th annual Design Automation Conference, ACM, 2008, 56-61
- Bauer2008a Bauer, L.; Shafique, M. & Henkel, J. Efficient resource utilization for an extensible processor through dynamic instruction set adaptation IEEE Trans. Very Large Scale Integr. Syst., IEEE Educational Activities Department, 2008, 16, 1295-1308
- Bauer2009 Bauer, L.; Shafique, M. & Henkel, J. MinDeg: a performance-guided replacement policy for run-time reconfigurable accelerators CODES+ISSS '09: Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis, ACM, 2009, 335-342
- Benmehrez1983 Benmehrez, C. & McDonald, J. Measured Performance of a Programmed Implementation of the Subscripted D-Algorithm Design Automation, 1983. 20th Conference on, 1983, 308-315
- Benson1997 Benson Cheung & Wang, L. T. The Seven Deadly Sins of Scan-Based Designs Integrated system design, 1997
- Bernardi2007 Bernardi, P.; Grosso, M. & Reorda, M. S. Hardware-accelerated path-delay fault grading of functional test programs for processor-based systems GLSVLSI '07: Proceedings of the 17th ACM Great Lakes symposium on VLSI, ACM, 2007, 411-416
- Bernardi2007a Bernardi, P.; Grosso, M.; Sanchez, E. & Sonza Reorda, M. On the Automatic Generation of Test Programs for Path-Delay Faults in Microprocessor Cores Proc. 12th IEEE European Test Symposium ETS '07, 2007, 179-18
- Bose1992 Bose, S. & Agrawal, P. Concurrent Fault Simulation of Logic Gates and Memory Blocks on Message Passing Multicomputers 29th Design Automation Conference Proceedings, 1992, 332-335
- Burgun1996 Burgun, L.; Reblewski, F.; Fenelon, G.; Barbier, J. & Lepape, O. Serial fault emulation Proc. 12th Design Automation, 1996, 801-806
- Bushnell2000 Bushnell, M. L. & Agrawal, V. D. Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Testing Springer ISBN 0792379918, 9780792379911, 2000, Frontiers in Electronic Testing Volume 17, 690 pag.

- Cadence2009 Cadence Incisive Palladium Brsochure Cadence: <http://www.cadence.com>, 2009
- Castelnuovo2002 Castelnuovo, A.; Fin, A.; Fummi, F. & Sforza, F. Emulation-based design errors identification Proc. 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems DFT 2002, 2002, 365-371
- Cha1978 Cha, C. W.; Donath, W. E. & Ozguner, F. 9-V Algorithm for Test Pattern Generation of Combinational Digital Circuits IEEE Trans. Comput., IEEE Computer Society, 1978, 27, 193-200
- Chao2006 Chao, M. C. T.; Wang, S.; Chakradhar, S. T.; Wei, W. & Cheng, K.-T. Coverage Loss By Using Space Compactors in Presence of Unknown Values Proc. Design, Automation and Test in Europe DATE '06, 2006, 1, 1-2
- Cheng1989 Cheng, W.-T. & Yu, M.-L. Differential Fault Simulation - a Fast Method Using Minimal Memory DAC, 1989, 424-428
- Cheng1990 Cheng, W.-T. & Patel, J. H. PROOFS: a super fast fault simulator for sequential circuits Proc. European EDAC Design Automation Conference, 1990, 475-479
- Cheng1999 Cheng, K.-T.; Huang, S.-Y. & Dai, W.-J. Fault emulation: A new methodology for fault grading IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1999, 18, 1487-1495
- Civera2001 Civera, P.; Macchiarulo, L.; Rebaudengo, M.; Reorda, M. S. & Violante, M. FPGA-based fault injection for microprocessor systems Proc. 10th Asian Test Symposium, 2001, 304-309
- Civera2001a Civera, P.; Macchiarulo, L.; Rebaudengo, M.; Sonza Reorda, M. & Violante, M. Exploiting FPGA-based techniques for fault injection campaigns on VLSI circuits Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2001, 250-258
- Civera2001b Civera, P.; Macchiarulo, L.; Rebaudengo, M.; Reorda, M. S. & Violante, M. FPGA-Based Fault Injection Techniques for Fast Evaluation of Fault Tolerance in VLSI Circuits FPL '01: Proceedings of the 11th International Conference on Field-Programmable Logic and Applications, Springer-Verlag, 2001, 493-502
- Civera2001c Civera, P.; Macchiarulo, L.; Rebaudengo, M.; Reorda, M. S. & Violante, M. Exploiting circuit emulation for fast hardness evaluation IEEE Transactions on Nuclear Science, 2001, 48, 2210-2216
- Crouch1999 Crouch, A. Design-For-Test For Digital IC's and Embedded Core Systems Prentice Hall PTR, 1999
- Debany1991 Debany, W.H., J.; Hartmann, C.; Varshney, P. & Mehrotra, K. Comparison of random test vector generation strategies

- Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on, 1991, 244-247
- Eusse2009 Eusse, J. F.; Hübner, M. & Jacobi, R. P. BRICK: a multi-context expression grained reconfigurable architecture SBCCI '09: Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design, ACM, 2009, 1-6
- Fang1999 Fang, W.-J.; Kao, P.-C. & Wu, A. C. H. A multi-level FPGA synthesis method supporting HDL debugging for emulation-based designs Proc. Asia and South Pacific Design Automation Conference the ASP-DAC '99, 1999, 351-354 vol.1
- Fernandes2004 Fernandes, J. M.; Santos, M. B.; Oliveira, A. L. & Teixeira, J. C. A probabilistic method for the computation of testability of RTL constructs Proc. Design, Automation and Test in Europe Conference and Exhibition, 2004, 1, 176-181 Vol.1
- Fujiwara1983 Fujiwara, H. & Shimono, T. On the Acceleration of Test Generation Algorithms IEEE Trans. Comput., IEEE Computer Society, 1983, 32, 1137-1144
- Gai1994 Gai, S. & Montessoro, P. L. Creator: new advanced concepts in concurrent simulation IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1994, 13, 786-795
- Garcia2001 Garcia, R. Rethink fault models for submicron-IC test Test & Measurement World, 2001, p. 35
- Giangarra2008 & Philip Giangarra, J. H. Design for Test Methodology Case Study for Motorola C-5e DCP Using the Cadence Incisive Accelerator/Emulator Cadence - White paper, 2008
- Goel1980 Goel, P. Test Generation Costs Analysis and Projections Proc. 17th Conference on Design Automation, 1980, 77-84
- Goel1981 Goel, P. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits Computers, IEEE Transactions on, 1981, C-30, 215-222
- Goel1981a Goel, P. & Rosales, B. PODEM-X: An Automatic Test Generation System for VLSI Logic Structures Design Automation, 1981. 18th Conference on, 1981, 260-268
- Guccione1999 Guccione, S.; Levi, D. & Sundararajan, P. JBits: Java based interface for reconfigurable computing Proceedings of the 2nd Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD), 1999, 27
- Gunes2005 Gunes, M.; Thornton, M. A.; Kocan, F. & Szygenda, S. A. A survey and comparison of digital logic simulators Proc. 48th Midwest Symposium on Circuits and Systems, 2005, 744-749 Vol. 1
- Gurumurthy2008 Gurumurthy, S.; Vemu, R.; Abraham, J. A. & Natarajan, S. On

- efficient generation of instruction sequences to test for delay defects in a processor GLSVLSI '08: Proceedings of the 18th ACM Great Lakes symposium on VLSI, ACM, 2008, 279-284
- Harel1987 Harel, D. & Krishnamurthy, B. Is There Hope for Linear Time Fault Simulation? 17th Fault. Tolerant Computing Symposium Proceedings., 1987, 28-33
- Hauck2007 Hauck, S. & Dehon, A. Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation (Systems on Silicon) (Systems on Silicon) Morgan Kaufmann, 2007
- Iyengar1988 Iyengar, V. S. & Tang, D. T. On simulating faults in parallel Proc. Eighteenth International Symposium on Fault-Tolerant Computing FTCS-18, Digest of Papers, 1988, 110-115
- Johnson1997 Johnson, B. W.; Smith, D. T. & DeLong, T. A. A Survey of Fault Simulation, Fault Grading and Test Pattern Generation Techniques with Emphasis on the Feasibility of VHDL Based Fault Simulation Virginia Univ Charlottesville, 1997
- Kafka2007 Kafka, L. & Danek, M. Preservation of Circuit Structure and Timing during Fault Emulation in FPGA Design & Reuse Industry Articles, 2007
- Kao1996 Kao, W. H. A new block-based design methodology and CAD toolset for mixed signal ASIC design Proc. nd Int ASIC Conf, 1996, 19-22
- Kapre2006 Kapre, N.; Mehta, N.; deLorimier, M.; Rubin, R.; Barnor, H.; Wilson, M. J.; Wrighton, M. & DeHon, A. Packet Switched vs. Time Multiplexed FPGA Overlay Networks Proc. 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines FCCM '06, 2006, 205-216
- Khan1993 Khan, U. R.; Owen, H. L. & Hughes, J. L. A. FPGA architectures for ASIC hardware emulators Proc. Sixth Annual IEEE International ASIC Conference and Exhibit, 1993, 336-340
- Krishnaswamy1997 Krishnaswamy, D.; Banerjee, P.; Rudnick, E. M. & Patel, J. H. Asynchronous parallel algorithms for test set partitioned fault simulation Proc. 11th Workshop on Parallel and Distributed Simulation, 1997, 30-37
- Kung1991 Kung, C.-P. & Lin, C.-S. Parallel sequence fault simulation for synchronous sequential circuits VLSI Technology, Systems, and Applications, 1991. Proceedings of Technical Papers, 1991 International Symposium on, 1991, 419-423
- Laakkonen2009 Laakkonen, T.; Itkonen, T.; Luukko, J. & Ahola, J. Time-stamping-based synchronization of power electronics building block systems Proc. 35th Annual Conf. of IEEE Industrial Electronics IECON '09, 2009, 925-930
- Larrabee1989 Larrabee, T. Efficient generation of test patterns using Boolean difference Test Conference, 1989. Proceedings.

- Meeting the Tests of Time., International, 1989, 795-801
- Lee1992 ho Lee, D. & Reddy, S. M. On Efficient Concurrent Fault Simulation for Synchronous Sequential Circuits 29th Design Automation Conference - DAC, 1992, 327-331
- Lin2007 Lin, Y.-C.; Lu, F. & Cheng, K.-T. Multiple-Fault Diagnosis Based On Adaptive Diagnostic Test Pattern Generation IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2007, 26, 932-942
- Lo1987 Lo, C.-Y.; Nham, H. N. & Bose, A. K. Algorithms for an Advanced Fault Simulation System in MOTIS IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1987, 6, 232-240
- Lopez-Ongil2005 Lopez-Ongil, C.; Garcia-Valderas, M.; Portela-Garcfa, M. & Entrena-Arrontes, L. Autonomous transient fault emulation on FPGAs for accelerating fault grading Proc. 11th IEEE International On-Line Testing Symposium IOLTS 2005, 2005, 43-48
- Maamari1988 Maamari, F. & Rajski, J. A fault simulation method based on stem regions Proc. IEEE International Conference on Computer-Aided Design ICCAD-88. Digest of Technical Papers, 1988, 170-173
- McCluskey2000 McCluskey, E. & Tseng, C. Stuck-fault tests vs. actual defects 2000
- Menon1978 Menon, P. R. & Chappell, S. G. Deductive Fault Simulation with Functional Blocks IEEE Transactions on Computers, 1978, 689-695
- Mentor2009 Mentor Graphics Corporation. Design-for-Test Common Resources Manual 2009. 392p
- Mentor2009a Corporation, M. G. Scan and ATPG Process Guide (DFTAdvisor™, FastScan™ and FlexTest™) 2009. 400p
- Meyer1993 Meyer, W. & Camposano, R. Fast Hierarchical Multi-Level Fault Simulation of Sequential Circuits with Switch-Level Accuracy Proc. 30th Conference on Design Automation, 1993, 515-519
- Millman1988 Millman, S. & McCluskey, E. Detecting bridging faults with stuck-at test sets Test Conference, 1988. Proceedings. New Frontiers in Testing, International, 1988, 773-783
- Motohara1988 Motohara, A.; Murakami, M.; Urano, M.; Masuda, Y. & Sugano, M. An approach to fast hierarchical fault simulation Proc. th ACM/IEEE Design Automation Conference, 1988, 698-703
- Narayanan1989 Narayanan, V. & Pitchumani, V. A massively parallel algorithm for fault simulation on the connection machine DAC '89: Proceedings of the 26th ACM/IEEE Design Automation Conference, ACM, 1989, 734-737

- Nowick1995 S. M. Nowick, Automatic synthesis of burst-mode asynchronous controllers, Tech. Rep. CSL-TR-95-686, Stanford University, Stanford, Calif, USA, 1995.
- Parreira2003 Parreira, A.; Teixeira, J. P.; Pantelimon, A.; Santos, M. B. & Sousa, J. T. D. Fault simulation using partially reconfigurable hardware In: Proc. of Field-Programmable Logic and Applications, Springer, 2003
- Parreira2003a Parreira, A.; Teixeira, J. P. & Santos, M. A Novel Approach to FPGA-based Hardware Fault Modeling and Simulation Proc. of the Design and Diagnostics of Electronic Circuits and Syst. Workshop, 2003, 17-2
- Parreira2006 Parreira, A. E. M.; dos Santos, M. B. & Teixeira, J. P. C. BIST Design Automation and Fault Emulation: BFE Tool Jornadas Sobre Sistemas Reconfiguráveis (REC), Faculdade de Engenharia da Universidade do Porto, 2006
- Patil1991 Patil, S.; Banerjee, P. & Banerjee, P. Performance Trade-Offs In A Parallel Test Generation/ Fault Simulation Environment 1991
- PeeterEllervee2003 Peeter Ellervee, Jaan Raik, V. T. Fault Emulation on FPGA: A Feasibility Study Proceedings of the Norchip Conference, Riga, Latvia, 2003
- PeeterEllervee2006 Peeter Ellervee, Jaan Raik, K. T. R. U. Environment for FPGA Based Fault Emulation Proceedings of the Estonian Academy of Sciences. Engineering, 2006, pp. 323-33
- Santos2001 Santos, M. B.; Gonçaves, F. M.; Teixeira, I. C. & Teixeira, J. P. RTL design validation, DFT and test pattern generation for high defects coverage Proc. IEEE European Test Workshop, 2001, 99-105
- Santos2003 Santos, M. B.; Fernandes, J. M.; Teixeira, I. C. & Teixeira, J. P. RTL test pattern generation for high quality loosely deterministic BIST Proc. Design, Automation and Test in Europe Conference and Exhibition, 2003, 994-999
- Schmit1997 Schmit, H. Incremental reconfiguration for pipelined applications Proc. 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines, 1997, 47-55
- Schnurmann1975 Schnurmann, H.; Lindbloom, E. & Carpenter, R. The Weighted Random Test-Pattern Generator Computers, IEEE Transactions on, 1975, C-24, 695-700
- Sedaghat1999 Sedaghat, R. Fault Emulation: Reconfigurable Hardware-Based Fault Simulation Using Logic Emulation Systems with Optimized Mapping Institut für Mikroelektronische Systeme, 1999
- Sedaghat1999a Sedaghat, R. Fault Emulation: Reconfigurable Hardware-Based Fault Simulation Using Logic Emulation Systems With Optimized Mapping Dem Fachbereich Elektrotechnik und

- Informationstechnik der Universität Hannover, December 1999
- Sedaghat-Maman1997 Sedaghat-Maman, R. & Barke, E. A new approach to fault emulation Proc. th IEEE International Workshop on Rapid System Prototyping ' Shortening the Path from Specification to Prototype', 1997, 173-179
- Seth1989 Seth, S.; Agrawal, V. & Farhat, H. A theory of testability with application to fault coverage analysis European Test Conference, 1989., Proceedings of the 1st, 1989, 139-143
- Snyder2009 Snyder, W. Verilog-Perl. Visitado em 2009.
<http://www.veripool.org/wiki/verilog-perl>
- Tahoori2002 Tahoori, M. B.; Mitra, S.; Toutounchi, S. & McCluskey, E. J. Fault grading FPGA interconnect test configurations Proc. International Test Conference, 2002, 608-617
- Tendolkar2002 Tendolkar, N.; Raina, R.; Woltenberg, R.; Lin, X.; Swanson, B. & Aldrich, G. Novel techniques for achieving high at-speed transition fault test coverage for Motorola's microprocessors based on PowerPC™ instruction set architecture Proc. 20th IEEE VLSI Test Symposium (VTS 2002), 2002, 3-8
- Trimberger1997 Trimberger, S.; Carberry, D.; Johnson, A. & Wong, J. A time-multiplexed FPGA Proc. 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines, 1997, 22-28
- Ulrich1988 Ulrich, E. G. & Baker, T. The concurrent simulation of nearly identical digital networks Design Automation Conference - DAC, 1988
- Vassiliadis2007 Vassiliadis, S. & Soudris, D. Fine- and Coarse-Grain Reconfigurable Computing Springer Publishing Company, Incorporated, 2007
- Waicukauski1989 Waicukauski, J. A.; Lindbloom, E.; Eichelberger, E. B. & Forlenza, O. P. A method for generating weighted random test pattern IBM J. Res. Dev., IBM Corp., 1989, 33, 149-161
- Wang1975 Wang, D. An Algorithm for the Generation of Test Sets for Combinational Logic Networks Computers, IEEE Transactions on, 1975, C-24, 742-746
- Wang2008 Wang, Laung-Terng.; Stroud, C. E. T. N. A. Laung-Terng Wang, C. E. S. & Toubia, N. A. (ed.) System-on-chip Test Architectures : Nanometer Design for Testability Elsevier ISBN 978-0-12-373973-5, 2008
- Wieler1995 Wieler, R. W.; Zhang, Z. & McLeod, R. D. Emulating static faults using a Xilinx based emulator Proc. IEEE Symposium on FPGAs for Custom Computing Machines, 1995, 110-115
- Wohl1999 Wohl, P. & Waicukauski, J. Using Verilog simulation libraries for ATPG Proc. International Test Conference, 1999, 1011-1020

- XAPP1512004 XAPP151: Virtex Series Configuration Architecture User Guide Xilinx:
http://www.xilinx.com/support/documentation/application_notes/xapp151.pdf, 2004
- XAPP4522008 XAPP452 Spartan-3 FPGA Family Advanced Configuration Architecture Xilinx, 2008
- Xydis2009 Xydis, S.; Economakos, G. & Pekmestzi, K. Designing coarse-grain reconfigurable architectures by inlining flexibility into custom arithmetic data-paths Integr. VLSI J., Elsevier Science Publishers B. V., 2009, 42, 486-503