UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UMA ARQUITETURA AUTOTESTÁVEL PARA CIRCUITOS DIGITAIS BASEADA NO ALGORITMO DE BERLEKAMP-MASSEY E EM SISTEMAS IMUNOLÓGICOS ARTIFICIAIS

CLEONILSON PROTÁSIO DE SOUZA

Tese apresentada à Coordenação de Pós-Graduação em Engenharia Elétrica para obtenção do título de Doutor em Ciências no domínio de Engenharia Elétrica.

Orientadores: Francisco Marcos de Assis e Raimundo Carlos Silvério Freire

Área de Concentração: Processamento da Informação

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S729a Souza, Cleonilson Protásio de
2005 Uma Arquitetura Autotestável para Circuitos Digitais Baseada no Algoritmo de Berlekamp-Massey e em Sistemas Imunológicos Artificiais / Cleonilson Protásio de Souza. — Campina Grande, 2005.
222f. : il.

Inclui bibliografia.
Tese (Doutorado em Engenharia Elétrica) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Orientador: Francisco Marcos de Assis e Raimundo Carlos Silvério Freire.

1— Circuitos Digitais 2— Testes de Circuitos 3— Inteligência Artificial
I— Título

CDU 621.3.049.77

UMA ARQUITETURA AUTOTESTÁVEL PARA CIRCUITOS DIGITAIS BASEADA NO ALGORITMO DE BERLEKAMP-MASSEY E EM SISTEMAS IMUNOLÓGICOS ARTIFICIAIS

CLEONILSON PROTÁSIO DE SOUZA

Tese Aprovada em 05.12.2005

FRANCISCO MARCOS DE ASSIS, Dr., UFCG
Orientador

RAIMUNDO CARLOS SILVÉRIO FREIRE, Dr., UFCG
Orientador

LUIZ PEREIRA CALÔBA, Ph.D., UFRJ
Componente da Banca

CECÍLIO JOSÉ LINS PIMENTEL, Ph.D., UFPE
Componente da Banca

SEBASTIAN YURI CAVALCANTI CATUNDA, D.Sc., UFMA
Componente da Banca

ELMAR UWE KURT MELCHERM, Dr., UFCG
Componente da Banca

BRUNO BARBOSA ALBERT, D.Sc., UFCG
Componente da Banca

CAMPINA GRANDE – PB
DEZEMBRO - 2005

Dedicatória

À minha mãe Cleide Protásio de Souza, por tudo !!!

Agradecimentos

Agradeço a Deus e a minha família que tanto me elogia e me admira em meus feitos acadêmicos e pessoais. Essas pessoas são minha mãe Cleide Protásio de Souza, que me deu de um tudo e que acho que por isso estou aqui, minha irmã Cleosilene Protásio de Souza, a quem chamo carinhosamente de *Ilhê*, e a minha irmã Clíssia Carolina Protásio de Souza, que definitivamente tem vida própria e que odeia ser chamada de Clíssia.

Agradeço a minha noiva e futura esposa Ane Polline Lacerda Araújo por ter uma postura séria e, ao mesmo tempo, ser extremamente carinhosa e compreensiva para comigo e por me dar de graça uma outra família nas pessoas de Dona Reginalda, Ana Luíza, Ana Carolina, Seu Aguinaldo, Dona Dodora e o meu amigo Roberto.

Agradeço ao Professor Francisco Marcos de Assis por todo ensinamento pessoal e acadêmico que me ajudou a me tornar uma pessoa melhor e mais justa e de agora me considerar um pesquisador.

Agradeço ao Professor Raimundo Carlos Silvério Freire por suas iniciativas positivas e visão esclarecedora do *modus operandi* do que acontece de fato na vida acadêmica, profissional e pessoal. E claro, não posso deixar de mencionar, meu profundo agradecimento a sua pessoa por ter aberto o caminho para a realização do meu doutorado.

Agradeço aos amigos Marcelo do Ó Lucena (por me mostrar a força que pode ter uma amizade), Henrique Oliveira, Reginaldo (Sirlene e Thiago), Shiko, Enaldo, Fabrício, Felipe, Gonzaga, Portela, Serrão, Diana, Lívia, Cíntia, Paula, Mônica, Zé Alves, Alfranque, Paulo, Edmarzinho, Edmar Candêia, Rex, Ronaldo, Rinaldo, Guilherme, Lucilene, Ricardo Rachichi, Alynthor, Esivaldo, Ademar, Ivan, Kécio, Leocarlos, Max, Wendell, Valesca, Iguatemi, Higino, Waslon, Will e todos aqueles que não me vem na memória de imediato mas que guardo na memória da eternidade.

Agradeço à professora Maria da Guia da Universidade Federal do Maranhão por acreditar em mim e em achar confiantemente que serei um ótimo pesquisador e ao professor João Viana da Fonseca Neto por sua prestabilidade em ajudar as pessoas.

Agradeço aos Professores Bruno, Glauco, Benemar, João Marques, Marcelo, Marcos Barbosa, Elmar e todos que contribuíram de alguma forma nos conhecimentos por mim adquiridos.

Agradeço a Ângela e a Pedro por seus gratos serviços esclarecedores durante o percurso dos trabalhos acadêmicos.

Agradeço a CAPES pelo fomento às minhas atividades de pesquisa.

Por fim, agradeço a todos que colaboraram de forma direta e indireta a este trabalho.

Resumo

Atualmente, o custo de se testar um circuito integrado é estimado em aproximadamente 25% do custo total de sua produção e é previsto que, em 2015, esse custo atingirá 50%. Esse custo é relacionado diretamente aos custos do uso de **equipamentos de teste automático**. Tais equipamentos são extremamente caros e, com o avanço na tecnologia de fabricação de circuitos integrados e com o aumento da velocidade de operação desses, estão se tornado inexatos na detecção de circuitos falhos. Uma alternativa bastante promissora na redução dos custos de testes é o desenvolvimento de **circuitos integrados autotestáveis** que estão rapidamente se tornando uma técnica de teste amplamente utilizada na indústria para testes de circuitos VLSI. Os principais componentes nessas arquiteturas autotestáveis são os **geradores de testes** e os **analisadores de respostas** que realizam a geração de testes e a análise das respostas do circuito a esses testes no próprio circuito integrado, respectivamente.

O objetivo principal desta tese é apresentar um esquema completo de uma arquitetura autotestável propondo um novo esquema de um **gerador de testes** e de um **analisador de respostas**.

O gerador de testes proposto é baseado principalmente no **algoritmo de Berlekamp-Massey** e em um processo de otimização baseado em algoritmo genético. Tal gerador é totalmente baseado na arquitetura de um **registrador de deslocamento com realimentação linear** (LFSR, da expressão em inglês, *Linear Feedback Shift Register*) e é capaz de gerar tanto testes determinísticos, que detectam as falhas de difícil detecção do circuito, quanto testes pseudo-aleatórios, que detectam as falhas restantes.

No que se refere ao analisador de respostas proposto, propõe-se um esquema baseado no **sistema imunológico humano**. No projeto desse esquema é utilizado o **algoritmo de**

seleção negativa inspirado nesse sistema. Tal processo de seleção negativa proporciona ao corpo a capacidade de discriminação entre células próprias e células estranhas a ele. Dessa inspiração, é proposto um analisador de respostas capaz de detectar se o circuito está com ou sem falha.

Utilizando os métodos de desenvolvimento do gerador de testes e do analisador de respostas propostos, alguns resultados de simulações, que demonstram a eficiência dos métodos, são mostrados utilizando-se os circuitos de verificação de desempenho nos padrões ISCAS85 e ISCAS89.

Abstract

Nowadays, the cost of testing an integrated circuit has been estimated at about 25% of its cost of manufacturing and it is estimated that the testing cost will achieve 50% by 2015. The testing cost is related directly to cost of using Automatic Test Equipments which are generally expensive and are becoming inaccurate in circuit testing due to both the advance in manufacturing technology of integrated circuits and the increase of their operational frequency. A promising alternative is the development of built-in self-test (BIST) techniques which are rapidly becoming a widely used test technique in the design of testing support hardware for VLSI circuits. In the BIST setup both test pattern generation and output response analysis are performed by on-chip hardware.

The main objective of this research work is to present a complete scheme of a self-testable architecture, i.e., to propose a new scheme of a test pattern generator and a new scheme of a circuit response evaluation.

The proposed test pattern generator is based mainly on the Berlekamp-Massey Algorithm and a genetic optimization. Such generator is totally based on the structure of Linear Feedback Shift Register and it is capable to generate both the deterministic test patterns, which detect the hard-to-detect faults of the circuit, and the random test patterns, which detect the remaining faults.

Concerning to circuit response evaluation, it is proposed a scheme based on the human immune system. Such a scheme is based on the application of the negative-selection mechanism from the immune system, which is able to discriminate between the self (body's own cell) and any foreign cell (non-self). From this inspiration, it is proposed circuit response analyzer capable of detecting if the circuit is good or faulty.

Using the proposed test pattern generator scheme and the proposed circuit response

evaluation scheme, some experimental results in ISCAS85 and ISCAS89 benchmarks circuits are presented. Their results show the efficiency of the proposed method.

Lista de Figuras

1.1 Queda da produção devido à inexatidão dos ATE's. 24
1.2 Arquitetura básica dos circuitos integrados autotestáveis (BIST). Em que o TPG é o gerador de testes, CUT é o circuito sob teste e o ORA é o analisador de respostas. 25

2.1 Gerador de testes (TPG). 30
2.2 Caracterização das falhas *stuck-at*. 33
2.3 Percentagem de falhas detectadas em relação à quantidade de vetores pseudo-aleatórios aplicados. 35
2.4 Estrutura necessária nas operações dos geradores de testes mistos (MTPG's). 38
2.5 Forma básica ou canônica de um LFSR. 38
2.6 Gerador de testes pseudo-aleatórios baseado em LFSR. 40
2.7 Circuito *c17*. 40
2.8 LFSR como gerador de testes pseudo-aleatórios para o teste do circuito *c17*. 41
2.9 Gerador de testes determinísticos baseado em ROM. $T_i =$ vetor de teste determinístico. 43

3.1 Estrutura adicionais utilizadas em geradores de testes mistos. 46
3.2 Sinal de relógio necessário nas operações dos geradores pseudo-aleatórios. . 46
3.3 LFSR obtido com o algoritmo de Berlekamp-Massey (BMA) em que Λ_i são os coeficientes do polinômio de realimentação desse LFSR. Esse LFSR é capaz de gerar os elementos $T_1, T_2, \cdots, T_n$ que foram aplicados ao BMA. . 48
3.4 Algoritmo de Berlekamp-Massey. 48

3.5 LFSR que reproduz a seqüência 1 0 1 0 0 1 1 0 1 1 a qual foi aplicada ao BMA. 49
3.6 Algoritmo de Berlekamp-Massey adaptado para operar com bits irrelevantes de forma otimizada. 50
3.7 Configuração do gerador de testes mistos proposto. 52
3.8 (a) Inicialmente, H_1 é aplicado ao circuito. (b) Após q deslocamentos do LFSR, H_2 é aplicado. (c) Finalmente, após $(k-1)q$ deslocamentos, H_k é aplicado. Durante e após isso, outros vetores de testes pseudo-aleatórios são gerados e aplicados ao circuito. 55
3.9 Hipercubos de testes que detectam as falhas de difícil detecção do circuito $c432$. 56
3.10 Testes gerados pelo LFSR sintetizado pelo BMA modificado. 56
3.11 Algoritmo de compactação de hipercubos de testes proposto. 58
3.12 Verificação de sobreposição, sobreposição = 1. 60
3.13 Verificação de sobreposição, sobreposição = 3. 60
3.14 Verificação de sobreposição para os testes do circuito $c432$. 61

4.1 Procedimentos básicos de operação do algoritmo genético proposto. 65

5.1 Somente a área hachurada é considerada no cálculo da sobreárea de *hardware* do esquema proposto. 71
5.2 Sobreárea de *hardware* do esquema proposto por Chiusano em [42]. 71
5.3 Técnica de agrupamento de entradas. 72
5.4 Sobreárea de *hardware* do esquema de múltiplos reagrupamentos proposto por Kavousianos em [17]. 72

6.1 Fase de geração de detectores. As cadeias em D_0 que casam com alguma cadeia em R são rejeitadas. As cadeias que não casam são selecionadas como detectores. 79
6.2 Fase de monitoramento de erro. Se algum detector em D, em algum momento, casar com alguma cadeia em R, então uma mudança em R ocorreu. 79

6.3 Relação entre espaços de detecção com o parâmetro r. As cadeias próprias são representadas por □, detectores por ×, espaços de detecção por ◯. (a) Espaços de detecção com parâmetro r adequado, mas, com ocorrência de sobreposição. (b) Sem ocorrência de sobreposição. (c) Espaços de detecção com $r \rightarrow v$. (d) Espaços de detecção com $r \rightarrow 1$. 82
6.4 Exemplo de casamento parcial baseado no critério r-contínuos. As cadeias x e y casam para $r \leq 5$. 83
6.5 Critério de casamento r-Hamming com parâmetro $r = 9$. As cadeias x e y casam para $r \leq 9$. 84

7.1 Avaliação das respostas do CUT através de um analisador de respostas (ORA). 87
7.2 Técnica de análise de assinatura. A avaliação consiste em comparar a assinatura do circuito sem falha previamente determinada, s^*, com a assinatura do CUT, s, no final do teste. Se $s = s^*$, então o circuito é considerado sem falha. Caso contrário, está falho. 87
7.3 Esquema proposto do analisador imunológico. 88
7.4 Algoritmo de otimização proposto para a redução dos espaços de detecção sobrepostos. A entrada do algoritmo é um conjunto de detectores $D = \{D_1, D_2, \cdots, D_d\}$, gerados pelo algoritmo de seleção negativa, capaz de detectar as falhas $S = \{f_1, f_2, \cdots, f_w\}$ de um circuito. A saída é um conjunto reduzido de detectores $L = \{D_1^*, D_2^*, \cdots, D_m^*\}$ $(m \leq d)$ o qual é capaz de detectar as mesmas falhas em S. 91
7.5 O analisador de respostas com *aliasing* é zero baseado em compressão de espaço e de tempo. O compressor de espaço recebe os bits de saída do CUT e, nesse caso, os bits de saída do TPG para que o *aliasing* seja zero [54], e os comprime em um só bit para cada teste aplicado. A seqüência de bits de saída do compressor de espaço, considerando que foi aplicada uma seqüência de testes ao CUT, é aplicada ao compressor de tempo que comprime essa seqüência binária em uma assinatura de poucos bits. 93

B.1 Estrutura do LFSR. 100
B.2 Esquema de transmissão-recepção utilizando código. 103

C.1 Cruzamento parcialmente mapeado. 106
C.2 Mutação por troca 107

Lista de Tabelas

2.1 Falhas *Versus* Vetores de testes. De T, somente o teste T_2 detecta a falha f_1. A falha f_2 é detectada pelos testes T_2, T_3 e T_{n-1}. Nenhum teste em T detecta f_3. A última falha f_k é detectada por T_3 e T_n. 31

2.2 Vetores de testes pseudo-aleatórios gerados pelo LFSR da Figura 2.8. . . . 41

2.3 Hipercubos de testes para o teste completo do circuito *c17*. 42

2.4 Vetores de testes determinísticos e compactados pelo ATALANTA para o teste completo do circuito *c17*. 43

3.1 Etapas na sintetização de um LFSR através do BMA. L é o comprimento atual do LFSR capaz de reproduzir a seqüência de entrada até seu j-ésimo elemento. 49

4.1 Influência da ordem de aplicação dos testes no BMA no comprimento do LFSR. 63

5.1 Dados dos circuitos utilizados nesta pesquisa. 68

5.2 Comparações entre as sobreárea de *hardware* (GE) e comprimento de teste (l_{teste}) obtidos pelo método proposto e os métodos em [17] e em [42] para atingir 100% de cobertura de falhas. O termo H é o valor da sobreárea de *hardware* consumida pelo bloco de controle do esquema de Chiusano e não relatada no respectivo trabalho. Obs: Alguns valores 0's aparecem na Coluna 9 indicando que o respectivo esquema não exige circuitos extras e somente consome um LFSR oriundo de um registrador já pertencente ao circuito. 74

7.1 Resultados experimentais usando o analisador de respostas proposto em conjunto com o algoritmo de otimização. 92

B.1 Equivalência entre análise de assinatura e teoria da codificação. 104

D.1 Adição módulo 2. 108
D.2 Multiplicação módulo 2. 109

Lista de Abreviaturas

ATE	Equipamento de Teste Automático (da expressão em inglês, *Automatic Test Equipment*)
BIST	Circuitos Integrados Autotestáveis (da expressão em inglês, *Built-In Self-Test*)
BMA	Algoritmo de Berlekamp-Massey (da expressão em inglês, *Berlekamp-Massey Algorithm*)
CI	Circuito Integrado
CUT	Circuito Sob Teste (da expressão em inglês, *Circuit Under Test*)
DTPG	Gerador de Testes Determinísticos (da expressão em inglês, *Deterministic Test Pattern Generator*)
FDD	Falha de Difícil Detecção
ISCAS	Simpósio Internacional em Circuitos e Sistemas (da expressão em inglês, *International Symposium on Circuits and Systems*)
LFSR	Registrador de Deslocamento com Realimentação Linear (da expressão em inglês, *Linear Feedback Shift Register*)
MTPG	Gerador de Testes Mistos (da expressão em inglês, *Mixed Test Pattern Generator*)

ORA	Analisador de Respostas de Saída
	(da expressão em inglês, *Output Response Analyzer*)
PRTPG	Gerador de Testes Pseudo-Aleatórios
	(da expressão em inglês, *Pseudo-Random Test Pattern Generator*)
ROM	Memória Somente de Leitura (da expressão em inglês, *Read Only Memory*)
TPG	Gerador de Testes
	(da expressão em inglês, *Test Pattern Generator*)
VLSI	Integração em escala muito alta
	(da expressão em inglês, *Very Large Scale Integration*)
ORA	Analisador de Respostas
	(da expressão em inglês, *Output Response Analyzer*)
PRTPG	Gerador de Testes Pseudo-Aleatórios
	(da expressão em inglês, *Pseudo-Random Test Pattern Generator*)
ROM	Memória Somente de Leitura (da expressão em inglês, *Read Only Memory*))
TPG	Gerador de Testes
	(da expressão em inglês, *Test Pattern Generator*)
VLSI	Integração em escala muito alta
	(da expressão em inglês, *Very Large Scale Integration*)

Lista de Símbolos

d_i	i-ésimo detector gerado pelo algoritmo de seleção negativa
F	Um dado conjunto de falhas do circuito
f_i	Uma falha do circuito
H_i	i-ésimo hipercubo de teste
h_i^j	j-ésimo bit do i-ésimo hipercubo de teste
p	Número de símbolos de um dado alfabeto finito
q	Número de entradas binárias do circuito
R	Seqüência de respostas do circuito
R^*	Seqüência de respostas do circuito sem falhas
R_i	i-ésimo vetor de resposta do circuito
r_i^j	j-ésimo bit do i-ésimo vetor de resposta do circuito
S_i	Espaço de detecção do detector d_i
T	Seqüência de vetores de teste
T_i	i-ésimo vetor de teste
t_i^j	j-ésimo bit do i-ésimo vetor de teste
v	Número de saídas binárias do circuito

Sumário

Resumo **v**

Abstract **vii**

I APRESENTAÇÃO **21**

1 Introdução **22**

1.1 Relevância 26

1.2 Objetivos 27

1.3 Organização do Trabalho 27

1.4 Conclusão 28

II PROPOSTA DE UM GERADOR DE TESTES **29**

2 Geradores de Testes **30**

2.1 Modelo de Falhas *Stuck-at* 33

2.2 Tipos de Geradores de Testes 34

2.2.1 Geradores de Testes Exaustivos 34

2.2.2 Geradores de Testes Pseudo-Aleatórios 35

2.2.3 Geradores de Testes Determinísticos 36

2.2.4 Geradores de Testes Mistos 36

2.3 Gerador de Testes Pseudo-Aleatórios 38

2.4 Gerador de Testes Determinísticos 42

2.5 Conclusão 44

3 Gerador de Testes Mistos Baseado no Algoritmo de Berlekamp-Massey 45

3.1 Algoritmo de Berlekamp-Massey 47

3.2 Algoritmo de Berlekamp-Massey Adaptado 50

3.3 Projeto do Gerador de Testes Mistos Proposto 52

3.4 Aplicando o BMA Modificado ao Projeto do Gerador Proposto 53

3.5 Exemplo de Aplicação 55

3.6 Compactação de Hipercubos de Testes 57

3.7 Verificação de Sobreposição 59

3.8 Conclusão 61

4 Otimização do Gerador de Testes através de Algoritmo Genético 62

4.1 Exemplo de Aplicação 66

4.2 Conclusão 66

5 Resultados Experimentais Usando o Gerador de Testes Mistos Proposto 67

5.1 Identificação das Falhas de Difícil Detecção de um Circuito 69

5.2 Cálculo da Sobreárea de *Hardware* 70

5.3 Resultados Experimentais 73

5.4 Conclusão 74

III PROPOSTA DE UM ANALISADOR DE RESPOSTAS 75

6 Algoritmo de Seleção Negativa 76

6.1 Breve Descrição do Sistema Imunológico 76

6.2 Algoritmo de Seleção Negativa 78

6.3 Critério de Casamento Parcial 81

6.3.1 Critério de casamento r-contínuos 83

6.3.2 Critério de casamento r-Hamming 84

6.3.3 Probabilidade de Erro de Detecção 84

6.4 Conclusão 85

7 Analisador de Respostas Baseado no Algoritmo de Seleção Negativa 86

7.1 Analisadores de Respostas 86

7.2 Estrutura do Analisador de Respostas Proposto 88

7.3 Algoritmo Proposto para Redução de Detectores Redundantes 90

7.4 Resultados Experimentais usando o Analisador Imunológico 91

7.5 Conclusão 93

IV CONCLUSÃO 94

A Lista de Publicações 97

A.1 Publicações em Conferências Internacionais 97

A.2 Publicações em Conferências Nacionais 98

B Análise de Assinatura 99

B.1 Algumas Estruturas de Analisadores de Assinatura 100

B.2 Modelamento de Compressores utilizando a Teoria de Codificação 102

C Operadores Genéticos Utilizados 105

C.1 Operador de Cruzamento Parcialmente Mapeado (PMX) 105

C.2 Operador de Mutação por Troca (EM) 107

D Campos Finitos 108

D.1 Campo Binário 108

D.2 Grupos 109

D.3 Anéis 110

D.4 Campos 110

D.5 Campos finitos baseados em anéis de inteiros 111

D.6 Campos finitos baseados em anéis de polinômios 112

Referências Bibliográficas 114

Índice Remissivo 121

Parte I

APRESENTAÇÃO

Capítulo 1

Introdução

Na sociedade moderna, sistemas eletrônicos tornaram-se produtos essenciais no nosso dia-a-dia. Nesse contexto, um princípio fundamental emerge: quanto maior é o benefício que esses sistemas oferecem ao nosso bem-estar e a nossa qualidade de vida, maior é o potencial de causarem algum dano quando esses falham na execução de suas funções ou executam-nas de forma incorreta [1]. Exemplos evidentes podem ser observados nas áreas como medicina e aviação. Quando esses sistemas falham, vidas e riquezas podem ser perdidas. Então, em tais aplicações, e em outras que potencialmente podem causar algum dano, faz-se necessário o desenvolvimento de sistemas com alta confiabilidade [2, 3].

Como os circuitos integrados são os componentes principais em tais sistemas [2], aumentar a confiabilidade desses circuitos é um processo chave no aumento da confiabilidade do sistema como um todo. Tal afirmação baseia-se no princípio fundamental das técnicas de tolerância a falha que consistem no fato que a confiabilidade de um sistema depende diretamente da confiabilidade de seus componentes constituintes [3].

Torna-se então de fundamental importância evitar que um circuito integrado defeituoso seja utilizado ao longo do processo de fabricação de um produto e antes de ser disponibilizado ao consumidor final, pois além de tornar o produto mais confiável, tem implicação direta na redução dos custos de produção como será visto a seguir.

Tipicamente existem 3 níveis no processo de manufatura de um produto, a saber: **nível de circuito integrado**, **nível de placas** e **nível de produto final**. Após esse último, o

produto é distribuído para o consumo, chamar-se-á esse de **nível de campo**.

No processo de manufatura, o teste de circuitos integrados é uma das principais formas de se evitar que um circuito com falha atinja o próximo nível no processo de produção, ou seja, o nível de produção de placas, ou até mesmo o nível mais avançado, o de produção do produto final. Este é um dos objetivos do teste de circuitos integrados, o de assegurar que um número mínimo de circuitos defeituosos se propaguem para os próximos níveis de manufatura do produto [4]. De fato, testes nos níveis inferiores têm como finalidade diminuir os custos associados, pois, de acordo com a **regra dos dez** [5], o custo de teste é aproximadamente multiplicado por 10 a cada nível do processo de manufatura, ou seja, se a detecção de uma falha na etapa de manufatura do circuito integrado custa um valor X, então detectá-la na etapa de produção de placas custará $10X$, na etapa final de produção $100X$ e após o produto ser distribuído para o consumo, ou seja, em campo, $1000X$.

No processo de manufatura de produtos eletrônicos, empreendem-se esforços significativos na busca por processos de testes eficientes. O custo desses processos tem sido estimado em aproximadamente 25% do custo do processo de produção total [4]. Entretanto, com o avanço na tecnologia do processo de integração e o aumento da complexidade no projeto de circuitos integrados, os custos associados ao processo de teste desses circuitos aumentarão consideravelmente [6] e é previsto que, em 2015, esse custos sejam iguais aos custos de produção [6].

Normalmente, testes de circuitos integrados são realizados utilizando-se **equipamentos de teste automático** (ATE[1]). Tais equipamentos são usados na aplicação, através de pontas de sondagem, de vetores de testes no circuito e no recebimento das respostas a esses testes com o objetivo de verificar se o circuito está falho ou não [7].

Um problema com os equipamentos de teste automático é o seu alto custo tanto o de aquisição do equipamento quanto os relacionados com a operação, treinamento, manutenção e a compra e/ou desenvolvimento do *software* operacional [8]. Um outro problema é que a maioria dos equipamentos de teste automático tem largura de banda de operação em torno de algumas centenas de MHz enquanto que, atualmente, devido ao crescente desenvolvimento na tecnologia de fabricação e do aumento da velocidade de operação, têm-se circuitos operando na faixa de GHz [7]. Assim, falhas que ocorrem somente

[1]Da expressão em inglês, *Automatic Test Equipment.*

quando o circuito está operando na sua freqüência de operação podem não ser detectadas e, dessa forma, a exatidão dos testes usando tais equipamentos é comprometida [8]. Tal fato, leva a um aumento do número de circuitos com falha e não-detectados a se propagarem ao longo do processo de produção acarretando uma diminuição da produção. Uma estimativa, de acordo com [2], da queda de produção devido à inexatidão dos equipamentos de teste automático é vista na Figura 1.1.

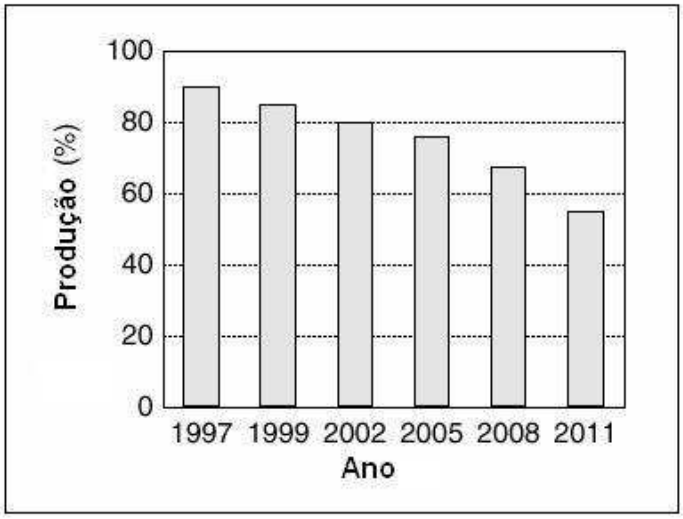


Figura 1.1: Queda da produção devido à inexatidão dos ATE's.

Uma solução encontrada para esses problemas é o desenvolvimento de Circuitos Integrados Autotestáveis (BIST's[2]) [9, 10, 11, 12, 13].

Na arquitetura BIST, a geração de vetores de testes e a avaliação das respostas do circuito a esses vetores são realizadas dentro do próprio circuito integrado [11, 13]. A estrutura básica dos BIST's pode ser visualizada na Figura 1.2, na qual o **gerador de testes** (TPG[3]) é usado para aplicar uma seqüência de n vetores de testes $T = (T_1, T_2, \cdots, T_n)$ ($T_i \in \{0,1\}^q$ em que $1 \leq i \leq n$ e q é o número de entradas do circuito) ao circuito sob teste (CUT[4]). A seqüência de respostas a esses testes, $R = (R_1, R_2, \cdots, R_n)$ ($R_i \in \{0,1\}^v$ em que $1 \leq i \leq n$ e v é o número de saídas do circuito), é aplicada ao **analisador de respostas** (ORA[5]) que avalia se o circuito sob teste está falho ou não [14, 15].

[2]Da expressão em inglês, *Built-In Self-Test.*
[3]Da expressão em inglês, *Test Pattern Generator.*
[4]Da expressão em inglês, *Circuit Under Test.*
[5]Da expressão em inglês, *Output Response Analyzer.*

Figura 1.2: Arquitetura básica dos circuitos integrados autotestáveis (BIST). Em que o TPG é o gerador de testes, CUT é o circuito sob teste e o ORA é o analisador de respostas.

Em geral, dois tipos de geradores de testes são bastantes usados em arquiteturas BIST's, a saber: os **geradores de testes pseudo-aleatórios** e os **geradores de testes determinísticos**. Como o próprio nome sugere, os geradores pseudo-aleatórios aplicam um seqüência de vetores pseudo-aleatórios ao circuito e os geradores determinísticos aplicam uma seqüência de vetores de testes previamente especificada.

Normalmente, a seqüência de vetores de testes T é composta de centenas ou até de milhares de vetores e, conseqüentemente, a seqüência de respostas aos testes R também apresenta essa mesma quantidade de vetores.

A forma mais trivial de se testar um circuito seria a de comparar a seqüência de respostas do circuito sob teste, R, com a seqüência de respostas do circuito sem falha, R^*. Entretanto, armazenar R^* no próprio circuito integrado seria altamente dispendioso. Dessa forma, geralmente, usa-se alguma técnica de compressão para evitar que a área consumida pelo analisador seja elevada.

Uma técnica de compressão bastante utilizada em analisadores usados em arquiteturas BIST's é a da **análise de assinatura**. Essa técnica consiste em comprimir a seqüência de respostas, R, em uma assinatura (um vetor com dimensões bastantes reduzidas) de poucos bits [15] e a avaliação do circuito consistiria em comparar a assinatura resultante com a assinatura do circuito considerado sem falha e previamente determinada [13]. Dessa forma, se essas assinaturas forem iguais, o circuito é considerado bom, caso contrário, é considerado com falha.

Uma desvantagem usando-se tal abordagem, é que, devido ao processo de compressão, podem ocorrer perdas de informação que podem levar a um diagnóstico errado do circuito e, conseqüentemente, esse pode ser declarado sem falha estando de fato com falha. Tal deficiência no processo de detecção é chamada de *aliasing* [13].

Algumas vantagens dos circuitos integrados autotestáveis são:

a) Em geral, operam na freqüência de operação do circuito [12];

b) Atingem alta cobertura de falhas[6] [12];

c) Provêem funcionalidades *on-line*. Dessa forma, BIST's podem ser usados tanto em **testes de produção**, quanto em **testes em campo**, ou seja, em uso pelo consumidor final [16];

d) Estão sendo considerados promissores em teste de circuitos integrados com alta complexidade [17] e de alta velocidade de operação.

d) E, principalmente, as arquiteturas de circuitos integrados autotestáveis são efetiva na redução de custos de testes [18, 11].

Dessa forma, o desenvolvimento de novos esquemas de teste baseados nessa técnica é de grande importância para a melhoria da eficiência do processo de teste e, conseqüentemente, na redução de custos e no aumento da produção de sistemas eletrônicos.

1.1 Relevância

No processo de manufatura de circuitos integrados, a busca por qualidade e de aumento de produção com menor custo passa necessariamente pela escolha do mais econômico e mais eficiente processo de teste. Nas arquiteturas de circuitos integrados autotestáveis, os principais parâmetros na determinação de sua eficiência e economia são:

a) **Cobertura de Falhas**: é a razão entre o número de falhas detectadas pelo processo de teste e o número total de falhas consideradas.

b) **Sobreárea de *hardware***: é o percentual de área ocupada pelos componentes responsáveis pelo teste em relação à área total do circuito[7].

[6] Detectam uma grande quantidade de falhas do circuito

[7] Por exemplo, o microprocessador 80386 da Intel emprega aproximadamente $1,8\%$ de sobreárea de *hardware* para as funções de testes baseadas em BIST [16, pp. 459]

c) **Tempo de teste**: é o tempo necessário para a aplicação de todos os vetores de testes necessários para a realização do teste.

Na prática, o que se deseja em um processo de teste é que a sobreárea de *hardware* e o tempo de teste sejam os menores possíveis e que tenha uma cobertura de falhas ideal, ou seja, de 100%. Como descrito na seção anterior, as arquiteturas de circuitos integrados autotestáveis são promissoras na tentativa de atingir esses objetivos citados e, em conseqüência, proporcionar uma redução nos custos associados. Dessa forma, o desenvolvimento de arquiteturas autotestáveis é de grande valia tanto para a redução de custos quanto no desenvolvimento de sistemas tolerantes à falha.

1.2 Objetivos

O objetivo principal deste trabalho de pesquisa é apresentar um esquema completo de uma arquitetura autotestável, ou seja, propor um novo esquema de um **gerador de testes** e de um **analisador de respostas**.

O gerador de testes proposto é baseado principalmente no **algoritmo de Berlekamp-Massey** e em um processo de otimização baseado em algoritmo genético.

No que se refere ao analisador de respostas, propõe-se um esquema baseado no **sistema imunológico humano**. No projeto desse esquema é utilizado o **algoritmo de seleção negativa** inspirado desse sistema.

1.3 Organização do Trabalho

Este trabalho está dividido em quatro partes. Na **parte I** são descritos os objetivos do trabalho e sua relevância.

Na **Parte II** é descrito um método de sintetização de um gerador de testes baseado no Algoritmo de Berlekamp-Massey adaptado aos requisitos impostos pela configuração proposta na arquitetura de teste. Nessa parte, também é mostrado um processo de otimização via algoritmo genético.

Na **Parte III** é descrito um método de projeto de um analisador de respostas baseado em Sistemas Imunológicos Artificiais.

Por fim, na **Parte IV** as conclusões do trabalho são relatadas.

Para mostrar a eficiência de cada método proposto, resultados experimentais são apresentados utilizando-se os circuitos de verificação de desempenho nos modelos ISCAS85[8] [19] e ISCAS89 [20] [9]

Considerando a natureza multidisciplinar da pesquisa foram adicionados apêndices com resumos dos principais tópicos para a leitura do trabalho para leitores não-especialistas em determinados tópicos.

1.4 Conclusão

Neste capítulo, foram descritos os objetivos principais deste trabalho de pesquisa assim como os principais conceitos relacionados com arquiteturas de circuitos integrados autotestáveis. Em adição, foram evidenciados os fatores que estão levando os circuitos integrados autotestáveis a serem altamente aplicados em testes de produção. Sendo a redução dos custos de produção o principal deles.

[8] No Simpósio Internacional sobre Circuitos e Sistemas de 1985(ISCAS'85), um conjunto de circuitos combinacionais foram usados como circuito de verificação de desempenho (*benchmark circuit*) na avaliação dos geradores de padrões de testes propostos nesse evento. Esses circuitos são descritos no formato de *netlist* e são amplamente usados na área de testes de circuitos digitais.

[9] No Simpósio Internacional sobre Circuitos e Sistemas de 1989(ISCAS'89) foram apresentados novos circuitos de verificação de desempenho para a área de testes de circuitos. Entretanto, tais circuitos são seqüenciais e contém flip-flop do tipo D.

Parte II

PROPOSTA DE UM GERADOR DE TESTES

Capítulo 2

Geradores de Testes

Como descrito na seção anterior, no processo de teste de circuitos integrados é aplicado uma seqüência de vetores de testes ao circuito de modo a possibilitar a detecção de possíveis falhas através da avaliação da seqüência de respostas de saída do circuito. Nas arquiteturas autotestáveis, o componente responsável pela aplicação dos vetores de testes, ou simplesmente, testes, é o gerador de testes (TPG), como pode ser visto na Figura 2.1.

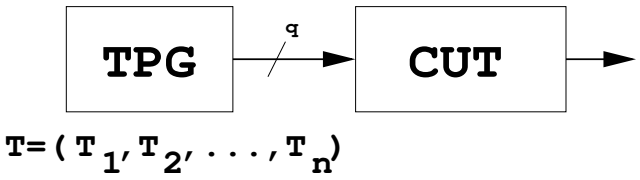


Figura 2.1: Gerador de testes (TPG).

Em geral, as falhas de um circuito são modeladas usando-se algum **modelo de falha** que representará as falhas físicas do circuito. O modelo adotado neste trabalho é o denominado de ***stuck-at***[16] e será descrito na Seção 2.1. Baseando-se no modelo de falha adotado, então as falhas de um circuito podem ser representadas pelo conjunto $F = \{f_1, f_2, f_3, \cdots, f_k\}$ em que k é o número de falhas modeladas do circuito.

Definição 1 *Suponha um circuito com uma dada falha. Essa falha é possível de ser detectada se e somente se a resposta do circuito com essa falha difere da resposta do circuito sem falha quando um dado vetor de teste é aplicado ao circuito.*

Aplicando-se ao circuito uma seqüência de n vetores de testes $T = (T_1, T_2, \cdots, T_n)$, então cada falha do circuito pode ou não ser detectada por um ou mais vetores de testes em T. Por exemplo, na Tabela 2.1, tem-se uma possível configuração a esse respeito.

Tabela 2.1: Falhas *Versus* Vetores de testes. De T, somente o teste T_2 detecta a falha f_1. A falha f_2 é detectada pelos testes T_2, T_3 e T_{n-1}. Nenhum teste em T detecta f_3. A última falha f_k é detectada por T_3 e T_n.

Falha	Teste que a detecta
f_1	T_2
f_2	T_2, T_3 e T_{n-1}
f_3	$-$
$\vdots$	$\vdots$
f_k	T_3 e T_n

Da Tabela 2.1, pode-se perceber que existem falhas que são detectadas por vários vetores de testes em T, assim como, existem falhas que são detectadas por poucos vetores, e falhas que não são detectadas por nenhum, por exemplo, a falha f_3.

Suponha agora que o circuito[1] sob teste tenha q linhas de entradas binárias e que a seqüência de teste aplicada seja formada por todas as 2^q combinações de vetores de entrada possíveis. Nesse contexto, pode-se obter a definição de detectabilidade de uma falha, como é dado a seguir.

Definição 2 *A **detectabilidade** de uma falha é o número de vetores de testes que detectam essa falha dentre os 2^q vetores de entrada possíveis de serem aplicados ao circuito.*

As falhas, do ponto de vista da detectabilidade, podem ser classificadas em falhas de fácil detecção e falhas de difícil detecção [21].

Definição 3 ***Falhas de fácil detecção** são aquelas que têm alta detectabilidade, ou seja, um elevado número de vetores de testes são capazes de detectá-las.*

Definição 4 ***Falhas de difícil detecção** são aquelas que têm baixa detectabilidade, ou seja, poucos vetores de testes são capazes de detectá-las.*

[1] Serão considerados neste trabalho somente os circuitos digitais combinatórios.

Dessas definições, pode-se chegar à seguinte conclusão: escolhendo um vetor de teste ao acaso é mais provável que esse detecte falhas de fácil detecção que falhas de difícil detecção. Esse fato é bastante útil no projeto de geradores de testes como será visto na Seção 2.2.

Uma outra definição relativa às especificações de um teste é a da cobertura de falhas atingida pela seqüência de teste aplicada ao circuito.

Definição 5 *A **cobertura de falhas** de uma seqüência de teste é a razão entre o número de falhas detectadas por essa seqüência e o número de falhas consideradas no processo de teste.*

Dessa forma, a especificação da seqüência de testes é de extrema importância na cobertura de falhas atingida pelo teste. Entretanto, um problema ainda maior é como gerar essa seqüência de teste no próprio circuito integrado de forma econômica. Essa questão é abordada na Seção 2.2. Outros parâmetros importantes que dizem respeito aos geradores de testes são definidos a seguir.

Definição 6 ***Comprimento de teste** é o número de vetores de testes aplicados pelo gerador de testes ao circuito.*

Definição 7 ***Tempo de teste** é o tempo necessário para se aplicar os vetores de testes gerado pelo gerador de testes ao circuito.*

Definição 8 ***Sobreárea de hardware** é a área de silício consumida pelos componentes responsáveis pelo processo de autoteste do circuito, ou seja, pelo gerador de teste e pelo analisador de respostas individualmente*[2]*.*

Em geral, a qualidade de um teste depende muito da seqüência de vetores de testes aplicados ao circuito sob teste e de como essa seqüência é gerada. Dessa forma, um projeto deficiente do gerador de testes pode comprometer significativamente a eficiência da arquitetura BIST como um todo. Pois, de nada adianta otimizar o analisador de respostas se a seqüência de testes gerada pelo gerador de testes não atinja uma alta cobertura de falhas, tenha tempo de teste proibitivo, e/ou consuma uma alta sobreárea de *hardware*.

A seguir, ir-se-á descrever em detalhes o *modelo de falhas* adotado neste trabalho de pesquisa.

[2]O termo "individualmente" nessa expressão indica que pode-se considerar a sobreárea de *hardware* dos componentes de forma separada, como por exemplo, sobreárea de *hardware* do gerador de teste.

2.1 Modelo de Falhas *Stuck-at*

O modelo adotado neste trabalho é o denominado *stuck-at*, também conhecido como modelo clássico ou padrão, e é amplamente utilizado na área de testes de circuitos digitais [16, pg. 110]. Nesse modelo, considera-se que as falhas ocorrem sempre nas linhas de conexão entre as portas lógicas e nas linhas de entrada e saída do circuito. Dessa forma, uma dada linha está com falha quando essa "fixa-se" em algum nível lógico, 0 ou 1, de forma anormal. Têm-se então dois tipos de falhas no modelo *stuck-at*, a saber: falha ***stuck-at*-0**, que representa que uma linha específica do circuito está fixa em nível lógico "zero", e falha ***stuck-at*-1**, que representa que uma linha está fixa em nível lógico "um".

Por exemplo, na Figura 2.2, pode-se verificar que a linha **d** apresenta uma falha *stuck-at*-1 (representada por *d/1*) e que a linha de entrada **c** apresenta uma falha *stuck-at*-0 (*c/0*). De fato, por exemplo, a falha *d/1* não indica que exista obrigatoriamente um meio conductor entre a linha **d** e o terminal positivo da fonte de tensão, mas que existe alguma falha física[3] no(s) transistor(es) de saída da porta lógica *AND* e/ou no(s) transistor(es) de entrada da porta lógica *OR*.

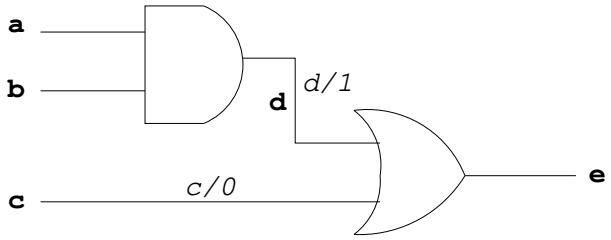


Figura 2.2: Caracterização das falhas *stuck-at*.

As vantagens do modelo *stuck-at* podem ser sumarizadas como a seguir:

a) Como as falhas *stuck-at* representam vários outros tipos de falhas físicas diferentes, então testes que detectam as falhas *stuck-at* também podem detectar esses outros tipos de falhas [16];

b) É independente da tecnologia, pois o conceito de uma linha "fixar-se" em algum valor lógico pode ser aplicado em qualquer modelo estrutural [16];

[3]Transistor em aberto ou em curto, por exemplo.

c) Falhas *stuck-at* podem ser usadas para modelar falhas em algum outro modelo de falhas como, por exemplo, no modelo *stuck-open* que considera que as falhas ocorrem nos transistores que compõem as portas lógicas do circuito. Dessa forma, em tal modelo, as falhas *stuck-open* podem ser caracterizadas por meio de falhas *stuck-at* [16].

Como visto, existem vetores de testes que detectam uma determinada falha e outros que não. Por exemplo, supondo que somente a falha *d/1* esteja presente no circuito da Figura 2.2, tem-se que o vetor $(1,1,1)$, ($a = 1$, $b = 1$ e $c = 1$), não é um teste para *d/1*, pois a resposta do circuito sem falha (no caso, $e = 1$) é a mesma do circuito com a falha. Um teste para a falha *d/1* pode ser qualquer vetor que "force" a linha **d** ser 0, ou seja, $(0,0,0), (0,1,0), (1,0,0)$. Observe que a linha **c** tem que ser 0 para que a falha seja observada na saída do circuito. Considerando somente a falha *c/0* presente no circuito, os testes para essa falha podem ser $(0,0,1), (0,1,1), (1,0,1)$.

Na próxima seção, os principais tipos de geradores de testes serão descritos.

2.2 Tipos de Geradores de Testes

Como visto na seção introdutória do Capítulo 2, é importante a especificação da seqüência de testes a fim de obter uma alta cobertura de falhas. Entretanto, em contrapartida, o problema maior é o seguinte: como gerar essa seqüência no próprio circuito integrado de forma econômica.

As principais formas de geração de testes aplicados em circuitos autotestáveis extraídos da bibliografia pesquisada são descritos a seguir.

2.2.1 Geradores de Testes Exaustivos

Nos geradores de testes exaustivos todas as combinações possíveis de vetores de testes são geradas e aplicadas ao circuito. Dessa forma, são aplicados 2^q vetores binários para um circuito com q entradas. Essa técnica garante cobertura de falhas de 100%. Entretanto, o uso de tais geradores é impraticável em circuitos que contenham um número elevado de

entradas (por exemplo, >20) [2, 22, 23]. Na prática, em aplicações industriais, os circuitos apresentam um número de entradas bastante elevado e bem maior que 20 e isso indica que esses tipos de geradores são raramente usados.

2.2.2 Geradores de Testes Pseudo-Aleatórios

Nos geradores de testes pseudo-aleatórios (PRTPG's, da expressão em inglês, *Pseudo-Random Test Pattern Generator*) ocorrem a geração de uma seqüência de vetores pseudo-aleatórios. Tais geradores têm, normalmente, como base os **registradores de deslocamento com realimentação linear** (LFSR) por sua capacidade de geração de números pseudo-aleatórios, por terem uma estrutura bastante compacta e por consumirem pouca área de silício quando integrados. Dessa forma, os geradores de testes pseudo-aleatórios consomem pouca sobreárea de *hardware*, sendo essa sua grande vantagem.

Uma informação bastante importante é que a maioria das falhas de um circuito são falhas de fácil detecção. Especificamente, em torno de 90% das falhas de um circuito são de fácil detecção [2] e, como visto na Figura 2.3, poucos vetores de teste aleatórios são necessário para atingir essa marca.

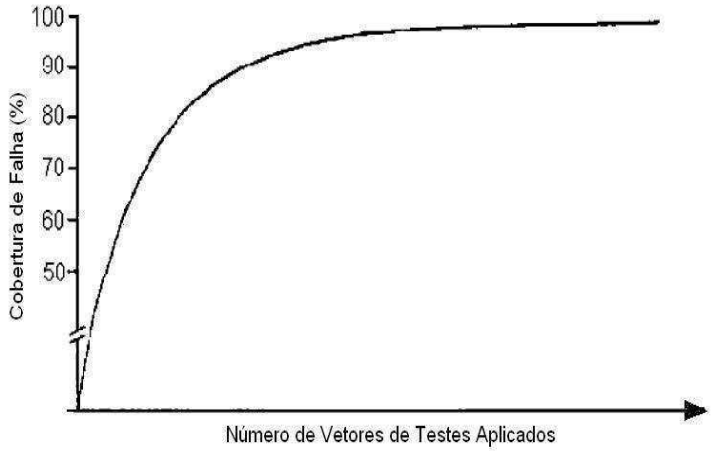


Figura 2.3: Percentagem de falhas detectadas em relação à quantidade de vetores pseudo-aleatórios aplicados.

Dessa forma, geradores de testes pseudo-aleatórios são efetivos na detecção da grande maioria das falhas do circuito e com baixo consumo de sobreárea de *hardware*.

Para que seja possível a detecção das falhas de difícil detecção presentes no circuito e com isso aumentar a cobertura de falhas é necessário aumentar a quantidade de vetores pseudo-aleatórios gerados. Entretanto, isso eleva consideravelmente o tempo de teste do circuito pois são necessários centenas ou milhares de testes para a detecção desse tipo de falha. Tal fato é a grande desvantagem no uso desse tipo de gerador.

Como na maioria das aplicações, uma cobertura de falhas maior que 98% é necessária [24], então, os geradores de testes pseudo-aleatórios são uma opção bastante dispendiosa em relação ao tempo de teste.

2.2.3 Geradores de Testes Determinísticos

Os geradores de testes determinísticos (DTPG's, da expressão em inglês, *Deterministic* TPG) são capazes de gerar vetores de teste previamente especificados usando-se alguma ferramenta algorítmica computacional, tal como as descritas em [25], [26] e [27]. Tais vetores são chamados de testes determinísticos.

A grande vantagem dos geradores de testes determinísticos é que com um número razoável de testes atinge-se uma alta cobertura de falhas e, além disso, o **tempo de teste** é bastante reduzido. Esse número razoável de testes determinísticos é possível graças ao uso de um processo de compactação, ou seja, são determinados vetores de testes para cada falha do circuito e, em seguida, tais vetores são compactados em um conjunto reduzidos de vetores.

A desvantagem no uso dos geradores de testes determinísticos é que, como não há nenhuma lógica que interrelaciona os vetores determinísticos, a **sobreárea de *hardware*** consumida, em geral, ocupa muito espaço, pois é necessário prover um modo de gerar esses testes no próprio circuito integrado. Normalmente, esses são armazenados em memória.

2.2.4 Geradores de Testes Mistos

Os geradores de testes mistos (MTPG, da expressão, *Mixed* TPG), como o próprio nome diz, geram tanto testes pseudo-aleatórios quanto testes determinísticos [28], e, em

geral, combinam certas estruturas utilizadas nos geradores de testes pseudo-aleatórios com as utilizadas nos geradores de testes determinísticos.

Nesses geradores, são usados testes determinísticos para a detecção das falhas de difícil detecção e testes pseudo-aleatórios para a detecção da falhas restantes (de fácil detecção) que são a grande maioria das falhas do circuito. Dessa forma, consegue-se um compromisso entre a sobreárea de *hardware* e o tempo de teste e, além do mais, atinge-se alta cobertura de falhas.

Alguns exemplos de geradores de testes mistos são descritos a seguir.

1. **Geradores baseados em reinicialização de LFSR**. Em tais geradores, em vez de se armazenar os vetores de testes determinísticos em uma memória, armazenam os conteúdos iniciais do LFSR para que esse gere tanto testes pseudo-aleatórios quanto os testes determinísticos de forma otimizada, ou seja, com o menor número de conteúdos iniciais armazenados [26];

2. **Geradores pseudo-aleatório ponderados**. Usam um gerador de vetores de testes pseudo-aleatórios que, usando alguma lógica combinacional, mudam a probabilidade de ocorrência do bit 1 (por exemplo, de $0,5$ para $0,75$ ou $0,25$) no intuito de que os vetores gerados produzam vetores determinísticos [16];

3. **Geradores baseados em fixação de bits**. Geram vetores de testes pseudo-aleatórios e fixam as posições de certos bits em algum valor lógico no intuito de que os vetores gerados produzam vetores determinísticos [29].

Em geral, os geradores de testes mistos necessitam de alguma estrutura de **controle**, de **memória** e/ou de alguma **lógica combinacional** para realizar sua função como visto na Figura 2.4.

Cada uma dessas estruturas consome sobreárea de *hardware* e dependendo da estrutura da lógica combinacional empregada pode-se ter algum tempo de atraso entre a geração do teste e a efetiva aplicação no circuito. Esse fato pode ser uma desvantagem em testes na freqüência de operação do circuito sob teste (*at-speed testing*).

Nas próximas seções, ir-se-á descrever com mais detalhes os geradores de testes pseudo-aleatórios e geradores de testes determinísticos, cujos conceitos e características serão utilizados no decorrer do trabalho e úteis na compreensão do esquema que será proposto.

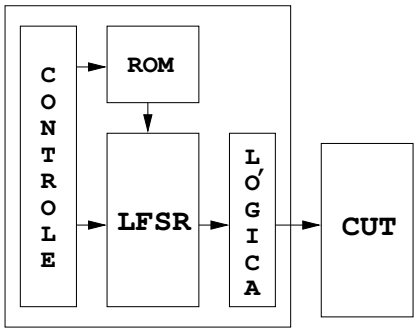


Figura 2.4: Estrutura necessária nas operações dos geradores de testes mistos (MTPG's).

2.3 Gerador de Testes Pseudo-Aleatórios

O gerador de testes pseudo-aleatórios, na maioria das vezes[4], é baseado em LFSR devido ao fato desse ter uma estrutura bastante compacta e consumir pouca sobreárea de *hardware*. Na Figura 2.5 é visto um LFSR na sua forma básica ou canônica.

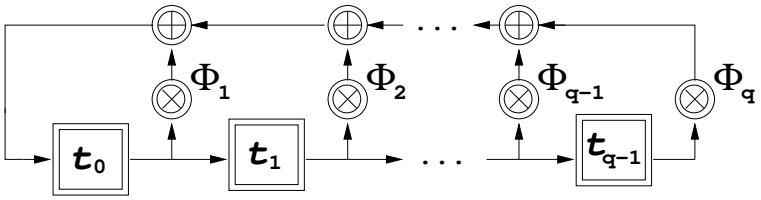


Figura 2.5: Forma básica ou canônica de um LFSR.

Um LFSR pode ser caracterizado por um polinômio chamado de **polinômio de realimentação**, ou polinômio de conexão, $\Phi(x)$, dado por:

$$\Phi(x) = \Phi_q x^q + \Phi_{q-1} x^{q-1} + \cdots + \Phi_2 x^2 + \Phi_1 x + 1 \tag{2.1}$$

em que todos os coeficientes Φ_i, $i = 1, \ldots, q$, pertencem a um determinado campo finito ou infinito[5] e que representam as conexões de realimentação do LFSR. O i-ésimo coeficiente Φ_i define o multiplicador para a i-ésima conexão de realimentação.

[4]Existem alguns esquemas de geradores pseudo-aleatórios baseados em **autômatos celulares** que são modelos computacionais em que um conjunto de células, ou autômatos, identicamente programadas interagem entre si e tentam imitar o comportamento dinâmico das células de um tecido [30]. Em geral, o estado de cada célula depende do seu estado atual e de suas vizinhanças e é computado via regras simples, como por exemplo usando-se a operação lógica ou-exclusivo.

[5]Em geral, LFSR's podem operar em qualquer campo mas, neste trabalho, adotar-se-á o campo binário como campo de operação.

Suponha que o conteúdo do LFSR no i-ésimo deslocamento seja representado pelo vetor

$$T_i = \begin{bmatrix} t_{q-1} \\ \vdots \\ t_2 \\ t_1 \\ t_0 \end{bmatrix}^{(i)} . \tag{2.2}$$

Então, a operação do LFSR pode ser representada por $T_i = A \cdot T_{i-1}$ em que o vetor T_{i-1} é o vetor que representa o conteúdo do LFSR no deslocamento anterior e a matriz A é a matriz de transição de estados. Dessa forma, tem-se que

$$\begin{bmatrix} t_{q-1} \\ \vdots \\ t_2 \\ t_1 \\ t_0 \end{bmatrix}^{(i)} = \begin{bmatrix} 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \\ \Phi_q & \Phi_{q-1} & \cdots & \Phi_2 & \Phi_1 \end{bmatrix} \begin{bmatrix} t_{q-1} \\ \vdots \\ t_2 \\ t_1 \\ t_0 \end{bmatrix}^{(i-1)} . \tag{2.3}$$

Dessa forma, a cada deslocamento do LFSR, todos os registros recebem os valores dos registros imediatamente anterior, exceto o registro t_0 que recebe o valor dado pela seguinte equação recursiva:

$$t_0 = \sum_{i=1}^{q} \Phi_i t_{i-1}.$$

Portanto, a cada deslocamento, uma nova palavra é gerada de acordo com $\Phi(x)$ e com o conteúdo inicial dos registros, T_0.

Em geral, como gerador de vetores pseudo-aleatórios, o LFSR é projetado no Campo de Galois $GF(2)$, ou simplesmente campo binário, no intuito de se economizar sobreárea de *hardware* pois é sabido que operações aritméticas em campos finitos[6], tais como em $GF(2^q)$, em que $q > 1$, são muitos mais dispendiosas em termos da área de circuito do que operações em $GF(2)$ [31, 32]. Assim, operando-se em $GF(2)$, os somadores são simples portas OU-EXCLUSIVAS de duas entradas e cada multiplicador é substituído por uma simples conexão. Assim, se $\Phi_i = 1$, há conexão, caso contrário, $\Phi_i = 0$, não há conexão.

[6]Uma pequena introdução aos campos finitos, $GF(n^q)$ pode ser vista no Apêndice D.

A configuração de um LFSR operando como gerador de vetores de testes é vista na Figura 2.6.

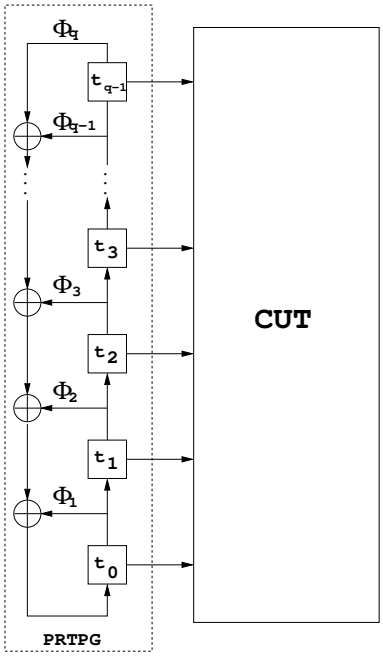


Figura 2.6: Gerador de testes pseudo-aleatórios baseado em LFSR.

Por exemplo, suponha-se que se deseja projetar um gerador de testes pseudo-aleatórios para o circuito *c17*, visto na Figura 2.7, que pertence ao conjunto de circuitos para verificação de desempenho no padrão ISCAS85 [19]. Esse circuito tem 5 entradas binárias, 2 saídas binárias e 22 falhas *stuck-at*.

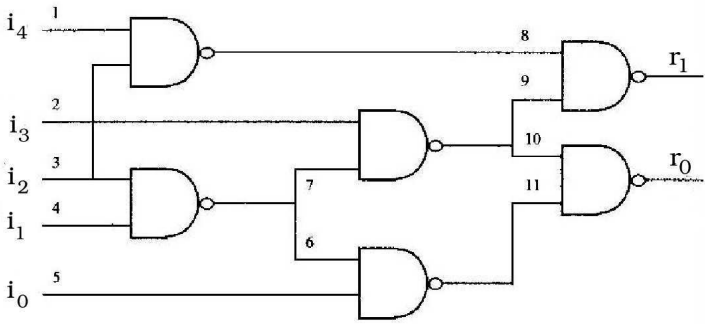


Figura 2.7: Circuito *c17*.

Um LFSR de comprimento 5 e polinômio de realimentação $\Phi(x) = x^5 + x^2 + 1$, visto na Figura 2.8, pode ser empregado para testar tal circuito. Suponha que o conteúdo inicial

desse LFSR seja o vetor $T_0 = (1, 1, 0, 0, 1)$. Na Coluna 2 da Tabela 2.2 são vistos os vetores de testes gerados, T_i, e aplicados ao circuito considerando 8 deslocamentos do LFSR. Na Coluna 3, são vistas as coberturas de falhas acumuladas (rotulada como FC[7]) para cada vetor de teste aplicado. Por exemplo, o vetor de teste T_0 detecta $22,7\%$ das falhas do circuito. Quando o vetor de teste T_1 é aplicado ao circuito, então tem-se que $27,2\%$ das falhas são detectadas por esses dois primeiros vetores de testes, e assim por diante. Dessa forma, quando o vetor de teste T_8 é aplicado tem-se $95,4\%$ de cobertura de falhas.

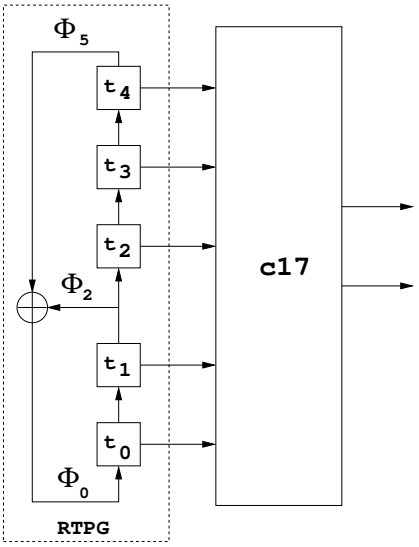


Figura 2.8: LFSR como gerador de testes pseudo-aleatórios para o teste do circuito *c17*.

Tabela 2.2: Vetores de testes pseudo-aleatórios gerados pelo LFSR da Figura 2.8.

i	T_i	FC_i (%)
0	11001	22,7
1	10011	27,2
2	00110	31,8
3	01101	50
4	11000	68,1
5	10001	77,2
6	00011	86,3
7	00111	90,9
8	01111	95,4

[7] A cobertura de falhas foi computada utilizando-se um programa simulador de falhas desenvolvido nesta pesquisa e que simula o circuito com cada falha e verifica se o teste aplicado detecta essa falha ou não.

2.4 Gerador de Testes Determinísticos

Um gerador de testes determinísticos é aquele que gera testes previamente especificados por alguma ferramenta computacional algorítmica. Tais ferramentas, em geral, simulam uma determinada falha e de acordo com a estrutura do circuito computam um ou vários vetores de testes que detectam essa dada falha.

Existem falhas que necessitam somente que algumas linhas de entrada do circuito sejam especificadas, em 0 ou 1, e que as linhas de entrada restantes podem assumir qualquer valor lógico, ou seja, o valor é irrelevante (X), que a detecção da falha não é comprometida. Normalmente, as ferramentas de especificação de testes determinísticos usam esse fato e dão como resultado vetores de testes em que cada bit pode assumir os valores 0, 1 ou X. Vetores de testes desse tipo são denominados de **hipercubos de testes**.

Definição 9 *Define-se hipercubo de teste como sendo um conjunto de vetores binários dado por* $H_a = [h_1, h_2, \ldots, h_q]$ *em que* $h_i \in \{0, 1, X\}$, $1 \leq i \leq q$ *e que o elemento* X*, bit irrelevante, pode assumir os valores* 0 *ou* 1*. Suponha que* H_a *testa uma falha* f_a*, então qualquer um dos vetores especificados e pertencentes de* H *testam* f_a*.*

Por exemplo, na Tabela 2.3, são vistos 22 hipercubos de testes que detectam, respectivamente, as 22 falhas do circuito *c17* do exemplo visto na seção anterior.

Tabela 2.3: Hipercubos de testes para o teste completo do circuito *c17*.

	Hipercubo de Teste			Hipercubo de Teste
f_1	X00X0		f_{12}	X111X
f_2	XX111		f_{13}	00XXX
f_3	X10X0		f_{14}	010XX
f_4	X10XX		f_{15}	100XX
f_5	X00X1		f_{16}	101XX
f_6	X0XX0		f_{17}	001XX
f_7	0110X		f_{18}	100XX
f_8	0111X		f_{19}	010XX
f_9	X101X		f_{20}	1X1XX
f_{10}	X10XX		f_{21}	101XX
f_{11}	000XX		f_{22}	00XXX

Da Tabela 2.3, pode-se pode ser observado que a falha f_1 é detectada pelo hipercubo

$H_1 = [X00X0]$ e, então, é detectada pelos seguintes vetores de testes: [00000], [00010], [10000] e [10010].

Um das vantagens do uso de hipercubos de testes é que esses podem ser usados na obtenção de um número reduzido de testes usando-se alguma ferramenta computacional. Tal procedimento é chamado de compactação. Algumas ferramentas computacionais, como por exemplo o ATALANTA [33], podem gerar conjuntos de testes determinísticos compactados e, dessa forma, reduzir a quantidade de testes que detectam as mesmas falhas detectadas pelo conjunto não-compactado.

Por exemplo, para o circuito *c17*, o conjunto de testes, visto na Tabela 2.4, atinge 100% de cobertura de falhas, ou seja, detectam as 22 falhas consideradas.

Tabela 2.4: Vetores de testes determinísticos e compactados pelo ATALANTA para o teste completo do circuito *c17*.

i	Vetor de Teste Determinístico
1	00101
2	11111
3	10010
4	11010

A desvantagem em se utilizar gerador de testes determinísticos (DTPG's) é que esses consomem uma sobreárea de *hardware* relativamente alta pois tem-se que armazenar os testes determinísticos em memória, normalmente em ROM, como pode ser visto na Figura 2.9, o que geralmente é proibitivo [34], ou prover alguma forma de gerá-los no próprio circuito integrado.

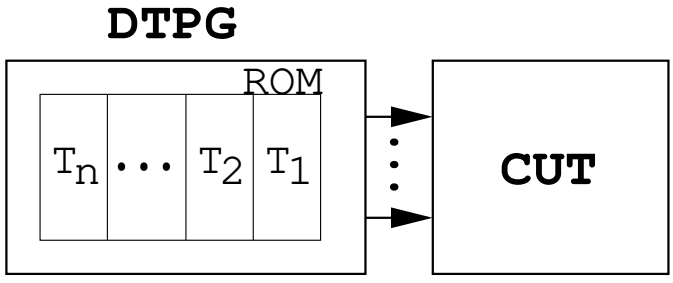


Figura 2.9: Gerador de testes determinísticos baseado em ROM. $T_i =$ vetor de teste determinístico.

2.5 Conclusão

O grande desafio no projeto de arquiteturas BIST's, no que se refere à geração de testes, é o desenvolvimento de técnicas que possibilitem aumentar a cobertura de falhas, diminuir a sobreárea de *hardware* e reduzir o tempo de teste. Os geradores de testes mistos são bastantes promissores nessa tentativa de otimização pois agregam as principais vantagens dos dois principais tipos de geradores, o gerador de testes pseudo-aleatórios e o gerador de testes determinísticos.

No próximo capítulo, serão apresentados os procedimentos para o desenvolvimento de um gerador de testes mistos baseado no algoritmo de Berlekamp-Massey.

Capítulo 3

Gerador de Testes Mistos Baseado no Algoritmo de Berlekamp-Massey

Como visto nas seções anteriores, em geral, dois tipos de geradores de teste são bastantes usados, a saber: os geradores de testes pseudo-aleatórios e os geradores de testes determinísticos.

Em síntese, os geradores pseudo-aleatórios consomem uma pequena sobreárea de *hardware*, porém o tempo de teste é extremamente alto para atingir uma cobertura de falhas adequada devido à presença de falhas de difícil detecção. Já nos geradores determinísticos, o tempo de teste é bem reduzido pois são necessários poucos testes para obter uma cobertura de falhas desejada e têm como desvantagem que a sobreárea de *hardware* normalmente é alta devido ao fato da necessidade de gerar esses testes no próprio circuito integrado.

Na tentativa de aproveitar as vantagens dos dois tipos de geradores de teste descritos, foi idealizado um gerador de testes mistos o qual gera testes determinísticos para detectar as falhas de difícil detecção e testes pseudo-aleatórios para detectarem as falhas restantes [34]. Porém, nas arquitetura mistas, são necessárias algumas estruturas adicionais para **controle de operação**, para **armazenamento** e/ou para alguma **lógica combinacional**, além da estrutura de geração principal, normalmente um LFSR, como podem ser vistos na Figura 3.1. Além do LFSR, todas essas estruturas adicionais consomem bastante sobreárea

de *hardware* e, em adição, pode não ser possível a realização de **teste em velocidade**[1] devido ao atraso causado pelo lógica combinacional entre o gerador e o circuito.

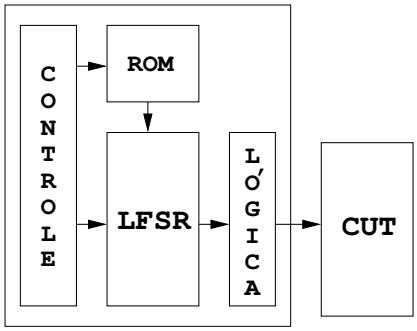


Figura 3.1: Estrutura adicionais utilizadas em geradores de testes mistos.

O gerador de testes pseudo-aleatórios é o que consome menor sobreárea de *hardware*, pois não utilizam nenhuma dessas estruturas adicionais, e o controle de suas operações baseia-se somente no sinal de relógio usado para a operação de deslocamento do LFSR, como pode ser visualizado na Figura 3.2.

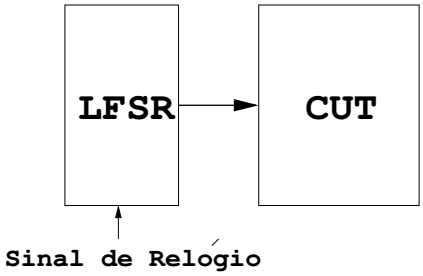


Figura 3.2: Sinal de relógio necessário nas operações dos geradores pseudo-aleatórios.

O objetivo principal desta parte do trabalho é descrever os procedimentos de desenvolvimento de um gerador de testes mistos baseado totalmente em um único LFSR.

A idéia principal do projeto proposto baseia-se no Algoritmo de Berlekamp-Massey (BMA) que sintetiza o menor LFSR capaz de gerar uma seqüência pré-definida de elementos em um dado alfabeto.

Dentro do conhecimento do autor, constitui-se uma inovação o uso do algoritmo de Berlekamp-Massey na área de testes de circuitos.

[1] Termo oriundo da expressão em inglês, *at-speed testing* ou *at-clock-speed testing*, que denomina o teste realizado na mesma freqüência de operação do circuito.

O método de projeto proposto é orientado pelas falhas de difícil detecção presentes no circuito, dessa forma, especificam-se os testes determinísticos que detectam essas falhas e sintetiza-se, através do BMA, um LFSR que gera esses testes. Desta forma, a princípio, o LFSR comporta-se como um gerador determinístico, gerando esses testes em uma dada ordem. Porém, o LFSR não gera somente esses testes, mas, de fato, gera diversos outros testes "residuais", que podem ser utilizados no teste das falhas restantes. Portanto, o gerador de testes proposto funciona como um gerador de testes mistos [35].

Considerando-se os circuitos combinacionais, a ordem de aplicação dos testes determinísticos não influencia no processo de teste. Entretanto, essa ordem influencia no comprimento do LFSR sintetizado pelo BMA. Dessa forma, uma otimização via algoritmo genético é utilizada para a obtenção da ordem de aplicação desses testes que proporciona uma redução do comprimento do LFSR. É importante ressaltar que nenhuma estrutura adicional faz-se necessária ao esquema proposto, propiciando a redução da sobreárea de *hardware* consumida e a realização de teste em velocidade.

Na próxima seção, serão descritos o algoritmo de Berlekamp-Massey e uma modificação sugerida para adequar e otimizar esse algoritmo na sintetização do LFSR base do gerador proposto.

3.1 Algoritmo de Berlekamp-Massey

O algoritmo de Berlekamp-Massey (BMA) foi inicialmente proposto para a localização de erros em sistemas de comunicações [36], de fato, esse algoritmo provê uma solução geral para a sintetização do LFSR mais curto capaz de gerar uma dada seqüência finita de elementos $T = (T_1, T_2, \cdots, T_n)$ em qualquer campo algébrico seja esse finito ou infinito [37, 38]. Uma definição bastante útil, oriunda do advento do BMA, é a de que a **complexidade linear** de uma seqüência é dada pelo comprimento do LFSR sintetizado quando essa seqüência é aplicada ao BMA [39, pp.108].

Quando uma seqüência T é aplicada ao BMA, tem-se como resultado os coeficientes do polinômio de realimentação, $\Lambda(x) = \Lambda_0 + \Lambda_1 x + \Lambda_2 x^2 + \cdots + \Lambda_L x^L$, de grau L, do LFSR mais curto capaz de gerar a seqüência T. Note que o problema solucionado pelo BMA é

equivalente ao da obtenção de um filtro de resposta infinita ao impulso capaz de reproduzir (predizer) a seqüência T a partir dos seus L valores iniciais. Dessa forma, carregando os L primeiros elementos de T no LFSR, este é capaz de gerar toda a seqüência T, como pode ser visto na Figura 3.3. O BMA é reproduzido na Figura 3.4.

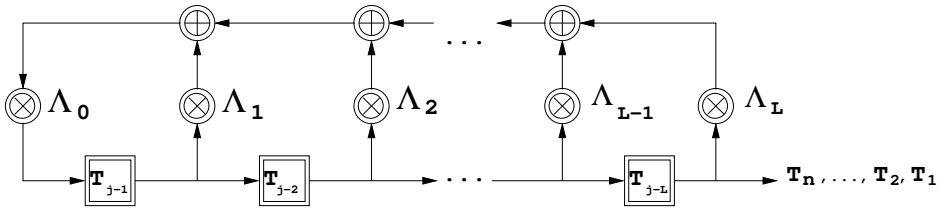


Figura 3.3: LFSR obtido com o algoritmo de Berlekamp-Massey (BMA) em que Λ_i são os coeficientes do polinômio de realimentação desse LFSR. Esse LFSR é capaz de gerar os elementos $T_1, T_2, \cdots, T_n$ que foram aplicados ao BMA.

ENTRADA: $T_1, T_2, \cdots, T_n$
SAÍDA: $\Lambda(x)$
INICIALIZAÇÃO:
$\Lambda(x) \leftarrow 1$: polinômio de realimentação do LFSR
$L \leftarrow 0$: comprimento do LFSR
$B(x) \leftarrow 1$: polinômio temporário
$d \leftarrow 1$, $b \leftarrow 1$: variáveis temporárias
$j \leftarrow 1$: contador
Δ : discrepância
ITERAÇÃO:
Passo 1. Se $j = n$, pare. Caso contrário, faça
$\Delta = T_j - \hat{T}_j = T_j + \sum_{i=1}^{L} \Lambda_i T_{j+1-i}$
Passo 2. Se $\Delta = 0$, então $d \leftarrow d + 1$, vá para o passo 5.
Passo 3. Se $\Delta \neq 0$ e $2L > j$, então
$\Lambda(x) \leftarrow \Lambda(x) - \Delta b^{-1} x^d B(x)$
$d \leftarrow d + 1$, vá para o passo 5.
Passo 4. Se $\Delta \neq 0$ e $2L \leq j$, então
$Temp(x) \leftarrow \Lambda(x)$
$\Lambda(x) \leftarrow \Lambda(x) - \Delta b^{-1} x^d B(x)$
$L \leftarrow j + 1 - L$
$B(x) \leftarrow Temp(x)$ $b \leftarrow \Delta$ $d \leftarrow 1$
Passo 5. $j \leftarrow j + 1$, retorne ao passo 1.

Figura 3.4: Algoritmo de Berlekamp-Massey.

Em termos gerais, o BMA opera da seguinte forma: após a devida inicialização, em cada iteração é computada a discrepância Δ a qual é definida como a diferença entre o valor

"atual" e o valor "predito" do j-ésimo elemento de T, i.e., $\Delta = T_j - \hat{T}_j$ em que $1 \leq j \leq n$ e $\hat{T}_j = -\sum_{i=1}^{L} \Lambda_i T_{j+1-i}$. Dessa forma, em cada iteração j, se $\Delta = 0$, então o comprimento do LFSR não muda e o seu polinômio de realimentação fica igual àquele definido na última mudança. Entretanto, se $\Delta \neq 0$, então o comprimento do LFSR pode mudar ou não de acordo com as seguintes situações: se $2L > j$, então o comprimento não muda (observe que, o polinômio de realimentação pode mudar). Caso contrário, é computado um novo LFSR capaz de gerar a seqüência até essa iteração.

Exemplo: Considere a seguinte seqüência com 10 elementos: $T = (1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1)$. Aplicando essa seqüência ao BMA, é obtido o polinômio de realimentação $\Lambda(x) = 1 + x^2 + x^4 + x^5$ o qual é o polinômio do menor LFSR capaz de gerar T. Os cinco primeiros elementos de T, $(1, 0, 1, 0, 0)$, formam o conteúdo inicial do LFSR como mostrado na Figura 3.5. O processo de síntese desse LFSR é mostrado em detalhes na Tabela 3.1.

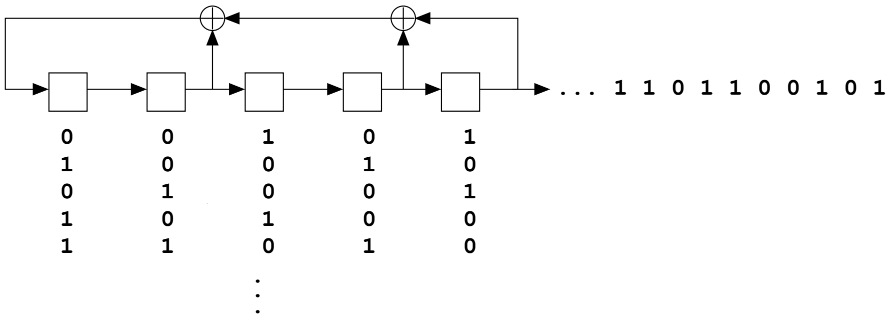
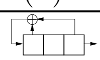
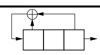
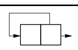
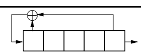
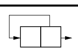
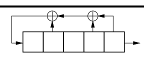
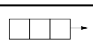
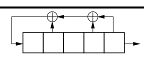


Figura 3.5: LFSR que reproduz a seqüência 1 0 1 0 0 1 1 0 1 1 a qual foi aplicada ao BMA.

Tabela 3.1: Etapas na sintetização de um LFSR através do BMA. L é o comprimento atual do LFSR capaz de reproduzir a seqüência de entrada até seu j-ésimo elemento.

j	L	$\Lambda(x)$	j	L	$\Lambda(x)$
1	0		6	3	
2	1		7	3	
3	2		8	5	
4	2		9	5	
5	3		10	5	

3.2 Algoritmo de Berlekamp-Massey Adaptado

Como visto no Capítulo 2, testes podem ser obtidos na forma de hipercubos e, dessa forma, os bits podem ser especificados em 0, 1 ou X. Considere a seguinte seqüência:

$$(0\ 1\ 1\ X\ 0\ X\ X\ 1\ 1\ X\ X\ X\ X\ X\ X\ 0\ X\ 1\ 1\ X\ 1\ X\ 1\ X\ 0\ 0\ X\ 1\ X\ X\ X).$$

Suponha que se deseja projetar um LFSR, através do BMA, que reproduza tal seqüência. Entretanto, o BMA não opera diretamente com bits irrelevantes, então tem-se que de alguma forma especificar tais bits. Porém, uma questão surge, a saber: *quais os valores especificados aos X's que diminuem o comprimento do LFSR resultante?*

Para resolver tal questão, é proposta uma adaptação do BMA para que esse opere com X's e que, além disso, otimize o comprimento do LFSR resultante. Essa modificação é executada no **passo 1** do algoritmo original e consiste no fato que se $T_j = X$, então T_j é computado como $T_j = \sum_{i=1}^{L} \Lambda_i T_{j+1-i}$ e, no próximo passo, a discrepância Δ é garantida ser zero, pois, nesse caso, $\Delta = T_j - \hat{T}_j = \sum_{i=1}^{L} \Lambda_i T_{j+1-i} - \sum_{i=1}^{L} \Lambda_i T_{j+1-i} = 0$. Assim, o comprimento L é garantido não aumentar. A modificação proposta [40] é vista na Figura 3.6.

INICIALIZAÇÃO:
$\vdots$
ITERAÇÃO:
Passo 1. Se $j = n$, pare. Caso contrário compute
$\hat{T}_j = \sum_{i=1}^{L} \Lambda_i T_{j+1-i}$
Se $T_j = X$**, faça** $T_j = \hat{T}_j$
$\Delta = T_j - \hat{T}_j$
Passo 2. Se $\Delta = 0$, então $d \leftarrow d + 1$, e vá para o passo 5.
Passo 3. Se $\Delta \neq 0$ e $2L > j$, então
$\Lambda(x) \leftarrow \Lambda(x) - \Delta b^{-1} x^d B(x)$
$d \leftarrow d + 1$, e vá para o passo 5.
Passo 4. Se $\Delta \neq 0$ e $2L \leq j$, então
$Temp(x) \leftarrow \Lambda(x)$
$\Lambda(x) \leftarrow \Lambda(x) - \Delta b^{-1} x^d B(x)$
$L \leftarrow j + 1 - L$
$B(x) \leftarrow Temp(x) \qquad b \leftarrow \Delta \qquad d \leftarrow 1$
Passo 5. $j \leftarrow j + 1$ e retorne ao passo 1.

Figura 3.6: Algoritmo de Berlekamp-Massey adaptado para operar com bits irrelevantes de forma otimizada.

Exemplo 2: Considere a seguinte seqüência com 31 elementos:

$$(0\ 1\ 1\ X\ 0\ X\ X\ 1\ 1\ X\ X\ X\ X\ X\ X\ 0\ X\ 1\ 1\ X\ 1\ X\ 1\ X\ 0\ 0\ X\ 1\ X\ X\ X)$$

Aplicando essa seqüência ao BMA modificado, é obtido o polinômio de realimentação $\Lambda(x) = 1 + x^3 + x^5$ que é o polinômio do LFSR capaz de gerar a seqüência:

$$(0\ 1\ 1\ \mathbf{0}\ \ 0\ \mathbf{1}\ \ \mathbf{1}\ 1\ 1\ \mathbf{1}\ \ \mathbf{0}\ \ \mathbf{0}\ \ \mathbf{0}\ \ \mathbf{1}\ \ \mathbf{1}\ 0\ \ \mathbf{1}\ 1\ 1\ \mathbf{0}\ 1\ \mathbf{0}\ \ 1\ \mathbf{0}\ 0\ 0\ \mathbf{0}\ 1\ \mathbf{0}\ \ \mathbf{0}\ \ \mathbf{1})$$

em que os X's foram especificados de maneira a não incrementar o comprimento do LFSR quando o algoritmo está sendo executado.

É importante observar que, sob o ponto de vista teórico, a solução encontrada através do uso da modificação do BMA não seja a melhor, mas essa demonstra ser, baseando-se em experimentos realizados durante a pesquisa, uma solução bastante efetiva na redução do comprimento do LFSR resultante.

3.3 Projeto do Gerador de Testes Mistos Proposto

Nesta seção será apresentada uma descrição detalhada dos procedimentos propostos para o projeto de um gerador de testes mistos baseado no algoritmo de Berlekamp-Massey. Como visto no Capítulo 3, o gerador de testes mistos proposto é baseado somente em um único LFSR sintetizado pelo BMA. Dessa forma, esse LFSR deve ser capaz de gerar tanto testes determinísticos, para detectar as falhas de difícil detecção do circuito, quanto vetores de testes pseudo-aleatórios para detectarem as falhas restantes. A configuração utilizada para autoteste de circuito usando o gerador de testes mistos proposto é vista na Figura 3.7.

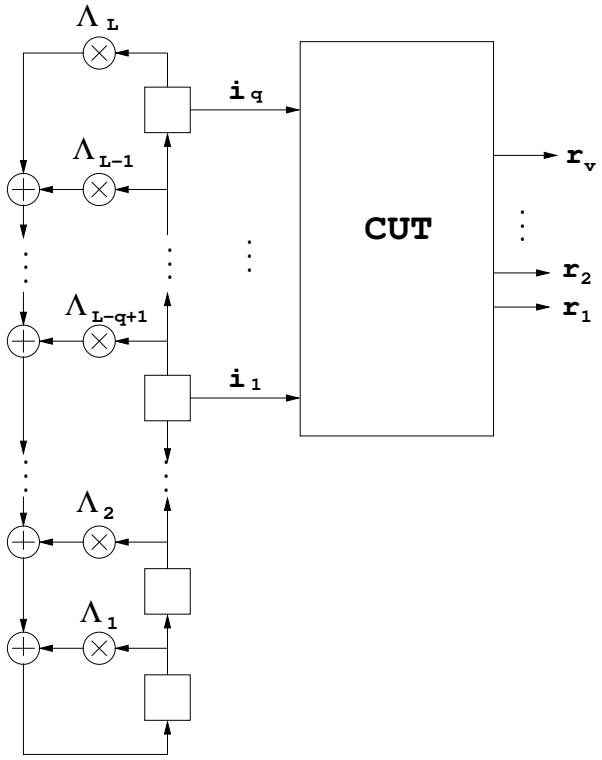


Figura 3.7: Configuração do gerador de testes mistos proposto.

O parâmetro principal para a verificação de desempenho do esquema proposto é o comprimento L do LFSR obtido pelo BMA, que é diretamente proporcional à sobreárea de *hardware* consumida pelo esquema. Dessa forma, tentativas de se diminuir o comprimento do LFSR torna-se de grande valia. Um dos procedimentos usado para diminuir esse comprimento é a utilização do BMA modificado como será visto na próxima seção. Outros procedimentos serão visto no decorrer do trabalho.

3.4 Aplicando o BMA Modificado ao Projeto do Gerador Proposto

Suponha que em um circuito com q entradas foram identificadas[2] k falhas de difícil detecção (FDD's) formando o conjunto $F = \{f_1, f_2, \cdots, f_k\}$ e que a seqüência de k hipercubos de testes determinísticos dada por $T = (H_1, H_2, \cdots, H_k)$ detectam as falhas em F, respectivamente. Cada hipercubo em T pode ser dado por:

$$H_i = \begin{bmatrix} h_i^q \\ \vdots \\ h_i^2 \\ h_i^1 \end{bmatrix}$$

em que $h_i^j \in \{0, 1, X\}$. Dessa forma, T pode ser caracterizado pela seguinte matriz:

$$T = \begin{bmatrix} h_1^q & h_2^q & \cdots & h_k^q \\ \vdots & \vdots & \vdots & \vdots \\ h_1^2 & h_2^2 & \cdots & h_k^2 \\ h_1^1 & h_2^1 & \cdots & h_k^1 \end{bmatrix}$$

É possível aplicar $T \in (GF(2^q))^k$ diretamente no BMA, entretanto, um fato conhecido é que operações aritméticas em campos finitos, tais como em $GF(2^q)$, em que $q > 1$, são muitos mais dispendiosas em termos da área de circuito necessária para efetivar sua execução do que operações em campo binário, $GF(2)$ [31, 32]. Então, para reduzir a área de *hardware*, uma **processo de serialização** é proposto a fim de operar o BMA em $GF(2)$.

Tal processo de serialização consiste em formar a seqüência $t \in GF(2)$ da seguinte forma: $t = ([H_1]', [H_2]', \cdots, [H_k]')$, em que $[H_i]'$ é o transposto de $[H_i]$. Dessa forma, a seqüência serializada t é dada por:

$$t = (h_1^1, \cdots, h_1^q; h_2^1, \cdots, h_2^q; \cdots; h_k^1, \cdots, h_k^q) \tag{3.1}$$

[2]Para a identificação das falhas de difícil detecção de um circuito, é proposto um procedimento algoritmico como é descrito na Seção 5.1.

Após esse processo de serialização, é importante perceber que todos os hipercubos de testes determinísticos em T ainda estão presentes em t.

Exemplo: Suponha que para a detecção da falhas de difícil detecção de um circuito sejam necessários os hipercubos de testes abaixo:

$$H_1 = \begin{bmatrix} 0 \\ X \\ 0 \\ 1 \\ X \end{bmatrix} \; H_2 = \begin{bmatrix} 1 \\ 1 \\ X \\ X \\ 0 \end{bmatrix} \; H_3 = \begin{bmatrix} X \\ 0 \\ 0 \\ X \\ 1 \end{bmatrix} \; H_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ X \\ 0 \end{bmatrix}$$

Então, a serialização da seqüência T resultará na seguinte seqüência t:

$$t = (\,0\,, X\,,\,0\,,\,1\,, X\,,\,1\,,\,1\,, X\,, X\,,\,0\,, X\,,\,0\,,\,0\,, X\,,\,1\,,\,0\,,\,0\,,\,1\,, X\,,\,0)$$

em que

$$t = (h_1^1, h_1^2, h_1^3, h_1^4, h_1^5, h_2^1, h_2^2, h_2^3, h_2^4, h_2^5, h_3^1, h_3^2, h_3^3, h_3^4, h_3^5, h_4^1, h_4^2, h_4^3, h_4^4, h_4^5).$$

Aplicando-se a seqüência serializada t ao algoritmo de Berlekamp-Massey modificado é obtido um LFSR capaz de gerar tal seqüência cujos bits irrelevantes são especificados de forma a minimizar o comprimento do LFSR resultante. Conectando o LFSR resultante como no esquema visto na Figura 3.7, então tem-se um gerador capaz de aplicar os hipercubos de testes determinísticos $H_1, H_2, \cdots, H_k$ ao circuito.

A operação do esquema consiste em que a cada q deslocamentos do LFSR (q **ciclos de relógio**), um hipercubo[3] de teste determinístico, presente na seqüência T, é aplicado ao circuito, como pode ser visto nas Figuras 3.8(a), 3.8(b) e 3.8(c).

Especificamente, no início da operação do LFSR, o hipercubo H_1 é aplicado ao circuito, como é visto na Figura 3.8(a). Após q deslocamentos, como visto na Figura 3.8(b), H_2 é aplicado ao circuito e assim por diante. Após $(k-1)q$ deslocamentos do LFSR, o último hipercubo em T, ou seja, H_k é aplicado ao circuito, como é visto na Figura 3.8(c). Dessa forma, após $(k-1)q$ deslocamentos, todas as k falhas de difícil detecção do circuito são detectadas.

[3]O LFSR gera hipercubos totalmente especificados pelo BMA modificado. Assim, dessa forma, gera testes determinísticos.

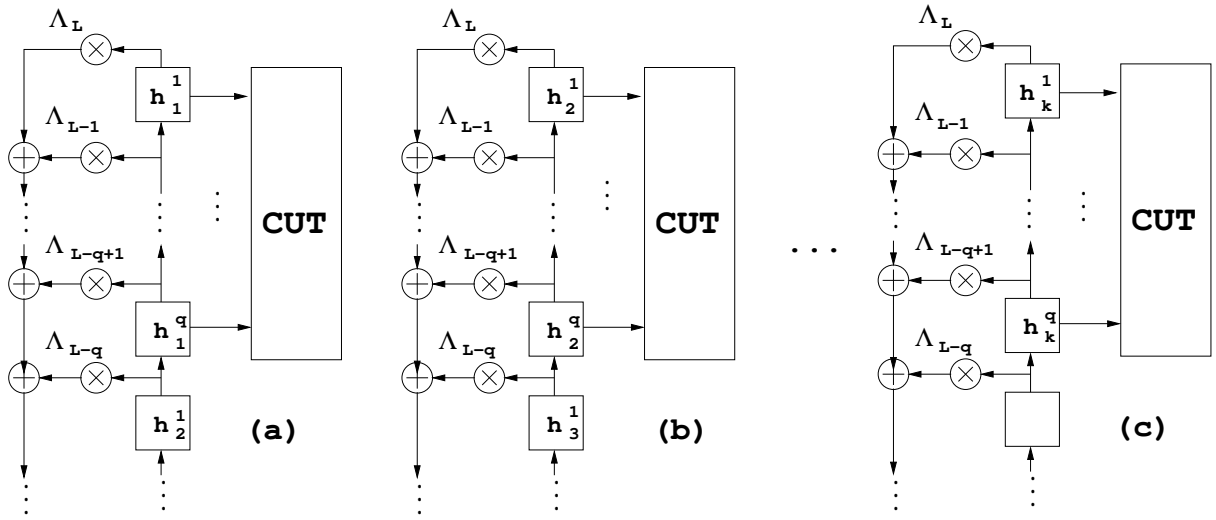


Figura 3.8: (a) Inicialmente, H_1 é aplicado ao circuito. (b) Após q deslocamentos do LFSR, H_2 é aplicado. (c) Finalmente, após $(k-1)q$ deslocamentos, H_k é aplicado. Durante e após isso, outros vetores de testes pseudo-aleatórios são gerados e aplicados ao circuito.

Pode-se verificar que entre a geração dos hipercubos de testes determinísticos e após isso, outros vetores de testes "residuais" são também gerados pelo LFSR. Tais vetores são suficientes[4] na detecção das falhas restantes do circuito, que são de fácil detecção. Portanto, o LFSR sintetizado é capaz de gerar tanto os hipercubos de testes determinísticos em T, que detectam as falhas de difícil detecção, quanto outros testes residuais que detectam as falhas restantes, sendo assim, um gerador de testes mistos.

3.5 Exemplo de Aplicação

Considere o circuito $c432$ (um decodificador de prioridade) da família de circuitos do padrão *ISCAS85*, o qual tem 36 entradas, 7 saídas e 524 falhas. Usando um procedimento de identificação proposto nesta tese e descrito na Seção 5.1, foram identificadas 12 falhas de difícil detecção que são detectadas, respectivamente, pelos hipercubos de testes determinísticos vistos na Figura 3.9.

Aplicando esses hipercubos, após passarem pelo processo de serialização, ao BMA modificado, é sintetizado um LFSR de comprimento igual a 215. Os vetores de testes gerados

[4]Esse fato foi observado nos experimentos realizados durante a pesquisa.

por esse LFSR são vistos na Figura 3.10. Observe que todos os teste determinísticos, provenientes dos hipercubos de testes, estão nessa seqüência (em negrito). Como pode ser visto, outros vetores residuais são também gerados. Assim, as falhas de difícil detecção do circuito são detectadas pelos testes determinísticos e as falhas restantes pelos testes residuais. Nesse exemplo, foram necessários 535 deslocamentos do LFSR, ou seja, 535 testes para que todas as falhas do circuito fossem detectadas (cobertura de falhas = 100%).

```
111x00xxx0xxx0xxx0xxx0xxx0xxx0xxx0xx
x0x1x11x00xxx0xxx0xxx0xxx0xxx0xxx0xx
x0xxx0xxx0xxx0xxx0x1x11x00xxx0xxx0xx
x0xxx0x1x11x00xxx0xxx0xxx0xxx0xxx0xx
x0xxx0xxx0xxx0xxx0xxx0xxx0xxx0x1x110
x0xxx0xxx0xxx0xxx0xxx0xxx0x1x11x00xx
x0xxx0xxx0xxx0x1x11x00xxx0xxx0xxx0xx
x0xxx0xxx0xxx0xxx0xxx0xxx0xxx0x1x100
x0xxx0xxx0xxx0xxx0xxx0x1x11x00xxx0xx
x0xxx0xxx0x1x11x00xxx0xxx0xxx0xxx0xx
1101010x00xxx0xxx0xxx0xxx0xxx0xxx0xx
x0xxx0xxx0xxx0xxx0xxx0xxx0x1x10x00xx
```

Figura 3.9: Hipercubos de testes que detectam as falhas de difícil detecção do circuito $c432$.

```
  1: 11110011100010001001101111000101110111  217: 00110011000010110110000010001000010000
  2: 11100111000100010011011100010111011O    218: 01100110000101101100001000100001000O
                 :                            219: 11001100001011011000010001000010000O
 35: 11000101100001001100111001001000000O                :
 36: 10001011000010011001110010010000000I    251: 00000010111000000100101001001010010I
 37: 00010110000100110011100100100000001O    252: 00000101110000001001010010010100101O
 38: 00101100001001100111001001000000010O    253: 00001011100000010010100100101001010O
 39: 01011000010011001110010010000000100O    254: 00010111000000100101001001010010100I
                 :                            255: 00101110000001001010010010100101001O
 71: 10001100010011101100110111001000111O                :
 72: 00011000100111011001101110010001110O    287: 00101010010010100010001001111000010O
 73: 00110001001110110011011100100011100I    288: 01010100100101000100010011110000100O
 74: 01100010011101100110111001000111001I    289: 10101001001010001000100111100001000I
 75: 11000100111011001101110010001110011O    290: 01010010010100010001001111000010001O
                 :                            291: 10100100101000100010011110000100010O
107: 01100010011111000000010001000110100O                :
108: 11000100111110000000100010001101000I    323: 01001110001011111000001010001010010O
109: 10001001111100000001000100011010001O    324: 10011100010111110000010100010100100I
110: 00010011111100000001000100011010001O    325: 00111000101111100000101000101001001O
111: 00100111110000000100010001101000110O    326: 01110001011111000001010001010010010I
                 :                            327: 11100010111110000010100010100100101O
143: 11000010010010100010011000100100111O                :
144: 10000100100101000100110001001001111I    359: 10110101010001100110110011101010011O
145: 00001001001010001001100010010011111O    360: 01101010100011001101100111010100110O
146: 00010010010100010011000100100111110O    361: 11010101000110011011001110101001100I
147: 00100100101000100110001001001111100O    362: 10101010001100110110011101010011001O
                 :                            363: 01010100011001101100111010100110010O
179: 10001100110010001110101000011011100O                :
180: 00011001100100011101010000011011100I    395: 01001100011000100010101001000101010O
181: 00110011001000111010100000110111000O    396: 10011000110001000101010010001010100I
182: 01100110010001110101000001101110000O    397: 00110001100010001010100100010101001I
183: 11001100100011101010000011011100000O    398: 01100011000100010101001000101010011O
                 :                            399: 11000110001000101010010001010100110O
215: 00001100110000101101100001000100001O                :
216: 00011001100001011011000010001000010O
```

Figura 3.10: Testes gerados pelo LFSR sintetizado pelo BMA modificado.

Na seção seguinte, será apresentado um método de compactação de hipercubos de testes.

3.6 Compactação de Hipercubos de Testes

Como é descrito na seção anterior, o LFSR base do gerador misto proposto é sintetizado pelo BMA modificado o qual tem como entrada uma seqüência de hipercubos de testes determinísticos. É claro que, se o número de hipercubos for menor, então o comprimento do LFSR resultante é provavelmente menor. Propõe-se, então, um método de redução do número de hipercubos de testes sem afetar o teste das falhas que esses detectam.

Em síntese, o procedimento de redução, ou de compactação, tem como entrada uma seqüência de hipercubos $T = (H_1, H_2, \ldots, H_k)$ que detectam, respectivamente, as falhas do conjunto $F = \{f_1, f_2, \ldots, f_k\}$, e obtém, como resultado, uma seqüência $T^* = (H_1^*, \ldots, H_c^*)$, em que $c \leq k$, e que detectam as mesmas falhas em F.

Definição 10 *Considere o hipercubo de teste $U = [u^1, \cdots, u^q]'$ em que $u^i \in \{0, 1, X\}$ para $1 \leq i \leq q$. Define-se o suporte de U como sendo $Sup_U = \{i : u_i \neq X\}$, ou seja, o conjunto das posições em que cada elemento de U é diferente de X.*

Por exemplo, sendo $U = [\mathrm{X1XX0X11XX}]'$, então $Sup_U = \{2, 5, 7, 8\}$.

Definição 11 *Considere dois hipercubos de testes $U = [u^1, \cdots, u^q]'$ e $V = [v^1, \cdots, v^q]'$. Então é dito que os hipercubos U e V são **consistentes**, se e somente se $\forall i \in (Sup_U \cap Sup_V)$ tem-se que $u_i = v_i$.*

Por exemplo, suponha um outro hipercubo $V = [\mathrm{XXX10XX1XX}]'$, então, como $Sup_V = \{4, 5, 8\}$, tem-se que $Sup_U \cap Sup_V = \{5, 8\}$. Assim, como $u^5 = v^5$ e $u^8 = v^8$ então U e V são consistentes. Agora suponha $Z = [\mathrm{0X1XXXX0XX}]'$, como $Sup_Z = \{1, 3, 8\}$ e $Sup_U \cap Sup_Z = \{8\}$, então U e Z não são consistentes pois $u^8 \neq z^8$.

Definição 12 *Considere os elementos u e $v \in \{0, 1, X\}$. Define-se $u \star v$ como sendo uma operação binária dada pela tabela abaixo:*

$v \star u$	0	1	X
0	0	N	0
1	N	1	1
X	0	1	X

sendo N uma situação proibida.

Teorema 1 *Suponha que os hipercubos de testes U e V detectam as falhas f_U e f_V, respectivamente. Se U e V são consistentes, então um hipercubo dado por $H = [h^1, \cdots, h^q]'$, em que $h^i = u^i \star v^i$ para $1 \leq i \leq q$, detecta as falhas f_U e f_V.*

Prova 1 *Sendo U e V consistentes e $H = [h^1, \cdots, h^q]'$, em que $h^i = u^i \star v^i$, então todas as posições especificadas e necessárias para o teste da falha f_U estão também presentes em H e, dessa forma, H detecta f_U. De forma análoga, H também detecta a falha f_V.*

Exemplo. Considere os hipercubos $U = [\text{X1XX0X11XX}]'$ e $V = [\text{XXX10XX1XX}]'$ que detectam as falhas f_U e f_V, respectivamente. Como $Sup_U = \{2, 5, 7, 8\}$ e $Sup_V = \{4, 5, 8\}$, então U e V são consistentes. Dessa forma, tem-se que o hipercubo dado por $H = [\text{X1X10X11XX}]'$, em que $h^i = u^i \star v^i$ para $1 \leq i \leq 10$, detecta as falhas f_U e f_V, pois para a detecção de f_U é necessário que as posições $\{2, 5, 7, 8\}$ estejam especificadas respectivamente em $\{1, 0, 1, 1\}$. Da mesma forma, f_V é detectada por H pois as posições $\{4, 5, 8\}$ estão especificadas respectivamente em $\{1, 0, 1\}$.

O algoritmo de compactação proposto baseia-se na verificação de consistências entre os elementos de uma dada seqüência de hipercubos, ou seja, suponha uma seqüência de k hipercubos dada por $T = (H_1, H_2, \ldots, H_k)$, se, por exemplo, H_1 for consistente com H_2 então T pode ser reduzido para $T = (H_{1,2}, \ldots, H_k)$ sendo $H_{1,2} = [h^1, \cdots, h^q]'$ em que $h^i = h_1^i \star h_2^i$ para $1 \leq i \leq q$. Procedendo dessa forma e verificando a consistência entres os outros hipercubos pode-se reduzir o número de hipercubos de T. O algoritmo de compactação é visto na Figura 3.11.

INÍCIO: n = 0
    ENQUANTO n < k FAÇA
        PARA m = n + 1 ATÉ k
            Se H_n for consistente com H_m
                Então faça $H_n = [h^1, \cdots, h^q]'$ em que $h^i = h_n^i \star h_m^i$ para $1 \leq i \leq q$.
        FIM DO PARA
        n = n + 1
    FIM DO ENQUANTO
FIM

Figura 3.11: Algoritmo de compactação de hipercubos de testes proposto.

Exemplo. Considere $T = (H_1, H_2, H_3, H_4, H_5, H_6)$ em que $H_1 = [\text{X1XX0X11XX}]'$, $H_2 = [\text{XXX10XX1XX}]'$, $H_3 = [\text{0X1XXXX0XX}]'$, $H_4 = [\text{XXX00XXXXX}]'$, $H_5 = [\text{XX1XXXX0XX}]'$ e $H_6 = [\text{X101X1XXX0}]'$. Executando-se o algoritmo de compactação proposto, pode-se verificar que H_1 é consistente com H_2 e, dessa forma, pode ser reduzido em $H_{1,2}$. Por sua vez, $H_{1,2}$ é consistente com H_6 reduzido-se assim em $H_{1,2,6}$. Por outro lado, H_3 é consistente com H_4, reduzindo-se em $H_{3,4}$. Por sua vez, $H_{3,4}$ é consistente com H_5 reduzindo-se em $H_{3,4,5}$. Assim, a seqüência de hipercubos compactados é dada por $T^* = (H_{1,2,6}, H_{3,4,5})$.

Até agora, tem-se dois métodos de otimização do comprimento do LFSR, a saber: o uso do BMA modificado e a compactação de hipercubos proposta. Na próxima seção, um outro procedimento usado para reduzir o comprimento do LFSR será descrito.

3.7 Verificação de Sobreposição

Considere uma seqüência de hipercubos de testes $T = (H_1, H_2, \cdots, H_k)$. A partir da obtenção da seqüência de testes serializados $t = ([H_1]', [H_2]', \cdots, [H_k]')$, pode-se observar que uma porção de H_i pode ser igual a uma porção do teste seguinte H_{i+1} $(1 \leq i < k)$. Por exemplo, sendo uma porção de t dada por:

$$\cdots; h_i^1, \cdots, h_i^{q-2}, h_i^{q-1}, h_i^q; \mathbf{h_{i+1}^1}, \mathbf{h_{i+1}^2}, \mathbf{h_{i+1}^3}, \cdots, \mathbf{h_{i+1}^q}; \cdots$$

em que o vetor $(h_i^1, \cdots, h_i^{q-2}, h_i^{q-1}, h_i^q)$ detecta a falha f_i e o vetor $(\mathbf{h_{i+1}^1}, \mathbf{h_{i+1}^2}, \mathbf{h_{i+1}^3}, \cdots, \mathbf{h_{i+1}^q})$ detecta a falha f_{i+1}.

Suponha agora que o bit $h_i^{q-1} = \mathbf{h_{i+1}^1}$ e que o bit $h_i^q = \mathbf{h_{i+1}^2}$. Removendo os elementos $\mathbf{h_{i+1}^1}$ e $\mathbf{h_{i+1}^2}$ de t, então tem-se que t reduzido é dado por:

$$\cdots, h_i^1, \cdots, h_i^{q-2}, h_i^{q-1}, h_i^q; \mathbf{h_{i+1}^3}, \cdots, \mathbf{h_{i+1}^q}; \cdots$$

Observe que após essa redução em t, tem-se que a falha f_i é ainda detectada pelo vetor $h_i^1, \cdots, h_i^{q-2}, h_i^{q-1}, h_i^q$. Já a falha f_{i+1} agora é detectada pelo vetor $h_i^{q-1}, h_i^q, \mathbf{h_{i+1}^3}, \cdots, \mathbf{h_{i+1}^q}$. Dessa forma, o teste não é prejudicado, porém, o comprimento da seqüência t é reduzido.

Exemplo 1. Suponha que uma parte de t está na forma mostrada na linha (1) da Figura 3.12. Observe que o último elemento de H_i serializado é igual ao primeiro elemento H_{i+1} serializado. Nesse caso, a sobreposição será 1. Após eliminar a sobreposição, é obtida a seqüência t^* reduzida como vista na linha (2). Observe que as falhas detectadas por H_i e por H_{i+1} ainda serão detectadas pois esses vetores não foram efetivamente retirados de t^*.

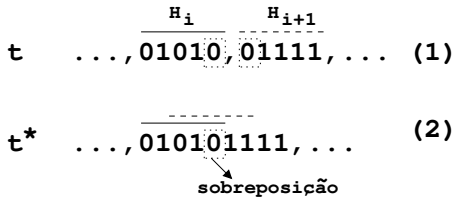


Figura 3.12: Verificação de sobreposição, sobreposição = 1.

Exemplo 2. Na porção de t vista na Figura 3.13, os 3 últimos elementos de H_j serializado "casam" como os 3 primeiros elementos de H_{j+1}. Nesse caso, a sobreposição é 3. Observe que, neste caso, ocorrem substituições de X´s.

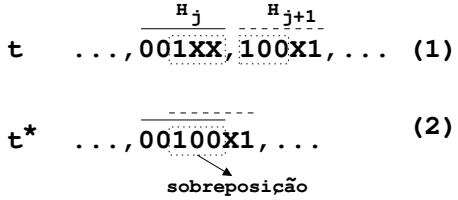


Figura 3.13: Verificação de sobreposição, sobreposição = 3.

Como, após a verificação completa de sobreposição, é garantido que todos os testes em T estejam ainda presente em t^*, então o processo de verificação de sobreposição descrito realiza uma redução no comprimento da seqüência de testes serializada e que será aplicada ao BMA. Isso pode acarretar em uma diminuição do comprimento do LFSR sintetizado pois, de acordo com [38], quanto menos elementos tem uma seqüência arbitrária, menor pode ser comprimento do LFSR.

Exemplo 3. Considerando os hipercubos de testes para o circuito $c432$ visto na Figura 3.9, tem-se que, após o processo de verificação de sobreposição, o comprimento da seqüência serializada passou de $12 \cdot 36 = 432$ para 114 bits como pode ser visto na Figura 3.14. Esse processo de verificação possibilitou a redução do comprimento do LFSR de 215 para 72 (redução de aproximadamente 66%).

Usando o LFSR sintetizado, é notado que, não usando o processo de verificação de sobreposição, em cada q deslocamento um teste determinístico é aplicado ao circuito como visto na Figura 3.8. A diferença de operação, usando o processo de verificação de sobreposição, é que menos deslocamentos são necessários entre a geração de um teste e um outro, conseqüentemente, o tempo para gerar todos os testes determinísticos, é reduzido. Dessa forma, o processo de verificação de sobreposição é efetivo tanto na redução da sobreárea de *hardware*, pois diminui o comprimento do LFSR, quanto na diminuição do tempo de teste do circuito.

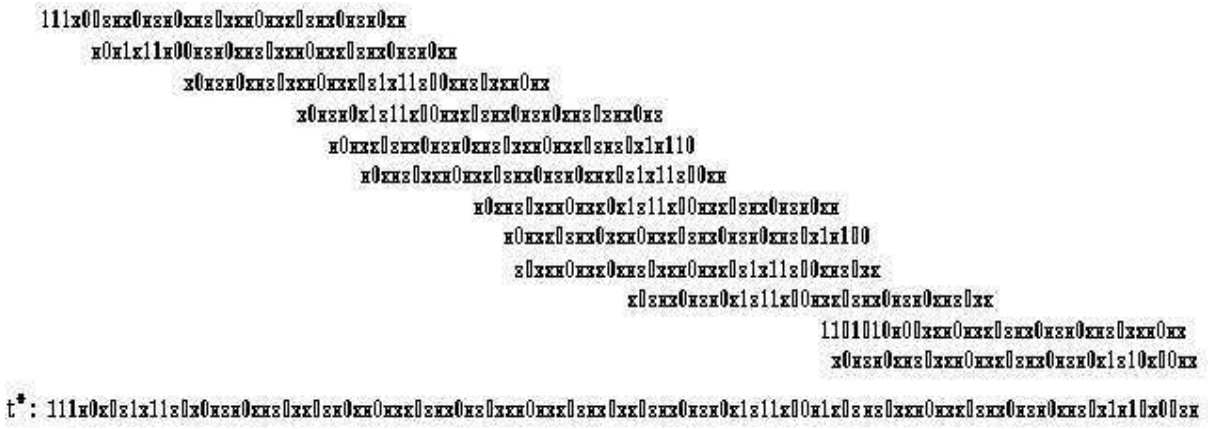


Figura 3.14: Verificação de sobreposição para os testes do circuito $c432$.

3.8 Conclusão

Neste capítulo foi descrito o algoritmo de Berlekamp-Massey modificado e os procedimentos gerais para a síntese do LFSR base para o gerador de testes mistos proposto. Além disso, foi descrito um método de compactação de hipercubos de testes e um processo de verificação de sobreposição que são efetivos na redução do comprimento da seqüência de entrada do BMA modificado. Tal redução proporciona uma redução tanto na sobreárea de *hardware* do gerador proposto quanto no tempo de teste do circuito.

No próximo capítulo, será descrito um procedimento baseado em algoritmo genético capaz de reduzir ainda mais o comprimento do LFSR sintetizado pelo BMA modificado.

Capítulo 4

Otimização do Gerador de Testes através de Algoritmo Genético

É observado que aplicando-se uma seqüência de testes $T = (T_1, T_2, \cdots, T_k)$ ao circuito sob teste e considerando o modelo de falhas *stuck-at*[1], uma mudança na ordem de aplicação desses testes, como por exemplo, $T = (T_2, T_k, \cdots, T_1)$, não altera o conjunto de falhas detectadas por eles.

Por outro lado, foi observado que, considerando o esquema de geração de testes proposto no capítulo anterior, uma mudança na ordem de aplicação dos testes no algoritmo de Berlekamp-Massey (BMA) pode alterar significativamente o comprimento do LFSR sintetizado. Por exemplo, suponha que se tenha uma seqüência de testes dada por $T = ([00111]', [11010]', [01001]')$. Aplicando-se a seqüência T serializada ao BMA, obtém-se um LFSR de comprimento igual a 9. Modificando-se a ordem dos testes para, por exemplo, $T^* = ([00111]', [01001]', [11010]')$, obtém-se um LFSR de comprimento igual a 3. Dessa forma, trocando-se somente os dois últimos testes em T, foi obtido uma redução no comprimento do LFSR de 9 para 3, ou seja, uma redução de $66,6\%$.

Outras modificações na ordem de aplicação dos testes podem ser vistas na Tabela 4.1 juntamente com o comprimento do LFSR sintetizado pelo BMA, visto na Coluna 3, respectivamente.

[1]No teste de falhas de atraso, por exemplo, uma mudança na ordem de aplicação dos testes altera o resultado.

Tabela 4.1: Influência da ordem de aplicação dos testes no BMA no comprimento do LFSR.

Item	Ordem de Aplicação	L
1	$([00111]', [01001]', [11010]')$	3
2	$([00111]', [11010]', [01001]')$	9
3	$([11010]', [01001]', [00111]')$	9
4	$([11010]', [00111]', [01001]')$	8
5	$([01001]', [00111]', [11010]')$	9
6	$([01001]', [11010]', [00111]')$	9

Como o comprimento do LFSR é proporcional à sobreárea de *hardware* consumida pelo esquema de geração de testes proposto, então a busca por uma ordem de aplicação de testes no BMA que resulte em um LFSR de menor comprimento é de grande importância.

Como descrito na Seção 3.3, têm-se k vetores ou hipercubos de testes determinísticos responsáveis pela detecção das falhas de difícil detecção do circuito que formam a seqüência $T = (T_1, T_2, \cdots, T_k)$. Dessa forma, existem $k!$ possíveis ordens de aplicação, ou **permutações**, dos elementos da seqüência T. Tem-se assim que a complexidade computacional é de ordem fatorial tornando qualquer busca por exaustão impraticável.

A definição do problema é a seguinte: *dado uma seqüência T com k vetores de testes, qual permutação em T resulta em um LFSR de comprimento mínimo quando sintetizado pelo BMA?* Para resolver esse problema uma otimização via algoritmo genético é proposta.

> ***Breve descrição sobre a operação dos algoritmos genéticos.*** *A solução de um problema através de algoritmos genéticos utiliza um processo evolucionário inspirado na teoria da seleção natural de Charles Darwin [41]. O algoritmo é inicializado, normalmente de forma aleatória, com um conjunto de soluções (representadas por indivíduos) chamado de população. Os indivíduos (possíveis soluções) de uma população são utilizados na composição de uma nova população (uma nova geração). A esperança é que, em cada nova geração, a população tenha soluções "melhores" que as anteriores pois seus indivíduos são selecionados de acordo com sua adequação, ou seja, quanto melhores, mais chances de reprodução terão. Esse processo é repetido até que alguma condição de parada seja satisfeita, como por exemplo, um certo número de gerações seja atingido.*

Os parâmetros da otimização proposta através de um algoritmo genético são descritos a seguir.

A cada geração, a **população** será formada por um número ρ de **indivíduos** e cada indivíduo é representado por uma permutação dos índices da seqüência de vetores de testes original T. Dessa forma, tem-se que a seqüência $(T_1, T_2, \cdots, T_k)$ é representada pelo indivíduo dado por $(1, 2, \cdots, k)$. Uma outra seqüência, por exemplo, $(T_k, T_1, \cdots, T_2)$, i. e., uma permutação da seqüência original T, é representada pelo indivíduo $(k, 1, \cdots, 2)$, e assim por diante.

Essa representação dos indivíduos é denominada de **representação por percurso**, muito usada em problemas do tipo do caixeiro viajante, e foi escolhida, ao invés da **representação binária** normalmente utilizada, pois, nesta última, as operações de **cruzamento** e **mutação** podem não constituir operações fechadas, ou seja, o resultado obtido por essas operações pode não ser um "percurso" (conjunto de índices) válido [42].

Por exemplo, suponha que uma representação binária de 4-bits seja utilizada para representar os indivíduos da seqüência $(T_1, T_2, \cdots, T_9)$ e que o indivíduo $(1, 2, 3, 4, 5, 6, 7, 8, 9)$ sofreu uma mutação no terceiro elemento $(3_d = 0011_b)$ na qual troca o primeiro bit de '0' para '1' resultando em 11_d (1011_b). Observe que o elemento 11_d não faz parte do conjunto de índices válidos[2].

Existem vários operadores genéticos que podem ser utilizados na representação por percurso, por exemplo: cruzamento parcialmente mapeado (PMX[3]), cruzamento cíclico (CX[4]), cruzamento por mudança de ordem (OX[5]), mutação por deslocamento (DM[6]), mutação por troca (EM[7]), dentre outros [42].

Neste trabalho foi utilizado o cruzamento parcialmente mapeado, que consiste em passar uma porção mapeada dos **pais** para os seus **descendentes**, e a mutação por troca, que seleciona aleatoriamente dois elementos em um indivíduo e os troca de posição. A operação detalhada desses operadores são descritos no Apêndice C.

[2]Vale ressaltar que, usando-se a representação binária, pode-se realizar certas operações de correção a fim de eliminar percursos não válidos.

[3]Da expressão em inglês, *Partially-Mapped Crossover.*

[4]Da expressão em inglês, *Cicle Crossover.*

[5]Da expressão em inglês, *Order Crossover.*

[6]Da expressão em inglês, *Displacement Mutation.*

[7]Da expressão em inglês, *Exchange Mutation.*

Suponha que cada indivíduo da população seja representado por π_i e corresponda a uma permutação ocorrida na seqüência original de testes $T = (T_1, T_2, \cdots, T_k)$. Defina $bma(\pi_i)$ como sendo a função que retorna o comprimento do LFSR sintetizado quando a seqüência de testes permutados representado por π_i é aplicado ao algoritmo de Berlekamp-Massey.

A função $bma(\cdot)$ descrita é a **função de avaliação** do algoritmo genético proposto. Dessa forma, quanto menor o valor retornado pela função $bma(\cdot)$ maior é a chance do indivíduo ser selecionado. Tem-se então um problema de minimização.

Em cada geração, tem-se uma população dada por:

$$P = (\pi_1, \pi_2, \cdots, \pi_\rho)$$

sendo que, as funções de avaliação são dadas por:

$$E = (bma(\pi_1), bma(\pi_2), \cdots, bma(\pi_\rho))$$

em que n é o número de indivíduos da população.

Na Figura 4.1, são vistos os procedimentos básicos da operação do algoritmo genético proposto.

INÍCIO. Gere uma população aleatória de ρ indivíduos.

SELEÇÃO. Faça ρ amostragem na população em que a probabilidade de qualquer indivíduo da população ser selecionado seja inversamente proporcional a sua função de avaliação. Assim, quanto menor a avaliação, maior a chance de ter mais cópias na próxima geração.

CRUZAMENTO. Para cada par de pais selecionado, e de acordo com a probabilidade de cruzamento, cruze esses pais para formar dois novos descendentes. Se não realizar cruzamento, os dois novos descendentes serão uma cópia exata dos pais.

MUTAÇÃO. De acordo a probabilidade de mutação, altere os descendentes.

VERIFICAÇÃO. Compare o melhor resultado já obtido com o melhor resultado desta geração e armazene este, caso seja menor.

TESTE. Se a condição final foi atingida, pare, e retorne o melhor resultado obtido.

REPITA. Retorne ao passo SELEÇÃO.

Figura 4.1: Procedimentos básicos de operação do algoritmo genético proposto.

A técnica usada no passo de seleção, visto na Figura 4.1, é chamada de técnica da roleta [41] e é responsável pela multiplicidade das cópias dos pais com menor valor da função de avaliação. É importante observar que o melhor indivíduo obtido, ou melhor, o menor LFSR sintetizado, não é obtido somente da última geração mas sim em qualquer geração como é visto no passo de verificação. Por fim, a condição final ou de parada utilizada é a de que um número de gerações foi atingido.

4.1 Exemplo de Aplicação

Considere o circuito $c432$ e os hipercubos de testes vistos na Figura 3.9. Sintetizando um LFSR na ordem apresentada e sem utilizar o processo de verificação de sobreposição, o comprimento do LFSR resultante seria de 215. Utilizando o processo de verificação de sobreposição seria de 72.

Executando o algoritmo genético proposto foi obtido um LFSR de comprimento igual a 38. A redução foi de aproximadamente 82%, não usando o processo de verificação de sobreposição, e de 47%, usando o processo de verificação de sobreposição.

Nesse exemplo, o polinômio de realimentação do LFSR obtido é dado por $\Lambda(x) = 1+x+x^5+x^9+x^{10}+x^{16}+x^{18}+x^{23}+x^{24}+x^{25}+x^{26}+x^{27}+x^{28}+x^{30}+x^{31}+x^{34}+x^{35}+x^{36}+x^{38}$.

4.2 Conclusão

Este capítulo descreveu um processo de otimização através da aplicação de um algoritmo genético cuja finalidade é obter uma ordem de aplicação dos vetores de testes de forma a minimizar o LFSR sintetizado pelo algoritmo de Berlekamp-Massey. Foi apresentado um exemplo que mostra a efetividade da otimização proposta.

Capítulo 5

Resultados Experimentais Usando o Gerador de Testes Mistos Proposto

De maneira a validar o gerador de testes proposto, algumas simulações experimentais foram realizadas através do uso de um simulador desenvolvido nesta pesquisa, chamado de **LFSRMAKER**. No desenvolvimento desse simulador foi usado a linguagem de programação *C++* no ambiente de desenvolvimento gráfico *Kylix 3.0* da *Borland*© operando sobre o sistema operacional *LINUX*. Foram também utilizados na obtenção dos resultados experimentais, o simulador ATALANTA [33], para a especificação dos hipercubos de testes determinísticos, e o simulador FSIM [43], como simulador de falha.

Nas pesquisas sobre desenvolvimentos de esquemas de testes, normalmente são usados circuitos que oferecem alta complexidade de teste. Dois padrões de circuitos se destacam nesse contexto, a saber: os circuitos no padrão *ISCAS85* e os circuitos no padrão *ISCAS89*[1].

Na Tabela 5.1 são vistos alguns dados sobre os circuitos, no padrão *ISCAS85* (linhas rotuladas por *ISCAS85*) e padrão *ISCAS89* (linhas rotuladas por *ISCAS89*), utilizados nos experimentos. Nas Colunas 3 e 4, são vistos o número de entradas e de saídas binárias de cada circuito, respectivamente. O número de falhas do tipo *stuck-at* para cada circuito é visto na Coluna 5 e na Coluna 6 é visto o número de falhas redundantes (falhas não-observáveis) para cada circuito.

[1]Os circuitos no padrão *ISCAS89* são seqüenciais e, nesta pesquisa, é utilizada somente a parte combinacional desses circuitos.

Tabela 5.1: Dados dos circuitos utilizados nesta pesquisa.

Padrão	Circuito	Número de Entradas	Número de Saídas	Número de Falhas	Número de Falhas Redundantes
ISCAS85	c880	60	26	942	0
	c1355	41	32	1574	8
	c1908	33	25	1879	9
	c3540	50	22	3428	137
ISCAS89	s420	35	17	430	0
	s641	54	42	463	0
	s713	54	42	581	41
	s1196	32	32	1242	0

Os procedimentos experimentais seguidos nesta pesquisa foram os seguintes:

1. Identificar o conjunto $F = \{f_1, f_2, \cdots, f_k\}$ das falhas de difícil detecção do circuito.
2. Obter k hipercubos de testes determinísticos $T = (H_1, H_2, \cdots, H_k)$ que detectem as falhas em F, respectivamente.
3. Usando o método de compactação proposto, compactar a seqüência T em $T^* = (H_1^*, \ldots, H_c^*)$.
4. Aplicar os hipercubos compactados em T^* ao procedimento de otimização através do algoritmo genético proposto.

   É importante salientar que é executada a verificação de sobreposição para cada seqüência permutada de hipercubos, ou seja, para cada indivíduo, no cálculo da sua respectiva função de avaliação que retorna o comprimento do LFSR sintetizado pelo BMA modificado. Por exemplo, suponha o seguinte indivíduo: $T_i^* = (H_c^*, H_1^*, \ldots, H_2^*)$. Então é executada a verificação de sobreposição da seqüência T_i^* que retorna uma seqüência serializada reduzida t_i^* e esta é aplicada ao BMA modificado que, por sua vez, retorna o comprimento L_i do LFSR (De fato, também retorna o polinômio de realimentação desse LFSR).
5. A partir do LFSR otimizado via o algoritmo genético, determinar a sobreárea de *hardware* consumida e o comprimento de teste que proporcione 100% de cobertura de falhas.

Dois desses procedimentos, a identificação das falhas de difícil detecção e o cálculo da sobreárea de *hardware* consumida pelo gerador obtido, são descritos em detalhes a seguir.

5.1 Identificação das Falhas de Difícil Detecção de um Circuito

Como o desenvolvimento do gerador de testes mistos proposto é direcionado, de início, à detecção das falhas de difícil detecção do circuito sob teste, então é de grande importância a correta identificação dessas falhas dentre todas as falhas presentes no circuito. Dessa forma, é proposto um procedimento de identificação. Esse procedimento é baseado no método de Monte Carlo que consiste em realizar vários experimentos a partir de amostras aleatórias a fim de inferir os valores de parâmetros desejados. O procedimento de identificação proposto é baseado no seguinte algoritmo:

1. Aplique uma seqüência de testes pseudo-aleatórios ao circuito.
2. Salve as falhas não detectadas pela seqüência de testes na lista.
3. Se um número predefinido de experimentos foi atingido, vá ao passo 4. Caso contrário, retorne ao passo 1.
4. Faça o ordenamento das falhas presentes na lista de acordo com o número de vezes que essas aparecem nessa lista.
5. Identifique como falha de difícil detecção as que mais aparecem na lista ordenada.

No passo 1 é esperado que uma grande parte das falhas de fácil detecção seja detectada pela seqüência pseudo-aleatória aplicada ao circuito. Dessa forma, as falhas não-detectadas por essa seqüência têm grande probabilidade de serem falhas de difícil detecção. Assim, aplicando-se ao circuito várias seqüências pseudo-aleatórias e salvado em uma lista as falhas não-detectadas por cada uma delas, é muito provável que as falhas de difícil detecção do circuito apareçam em maior número. Dessa forma, pode-se, através do ordenamento (passo 4), identificar as falhas de difícil detecção do circuito em questão.

5.2 Cálculo da Sobreárea de *Hardware*

Para o cálculo da sobreárea de *hardware* dos esquemas de testes propostos na bibliografia pesquisada, em geral, é adotado o método da **equivalência de portas** [44]. Esse método consiste em considerar que a cada 4 transistores utilizados em um bloco lógico seja equivalente a 1 GE (O termo GE é oriundo da expressão em inglês *Gate Equivalent*). Dessa forma, tem-se que uma porta *NAND* ou uma porta *NOR* consome 1 GE pois são construídas usando somente 4 transistores cada.

O esquema de geração de testes proposto é baseado somente na estrutura de um LFSR que por sua vez só usa portas *XOR* e *flip-flop*'s. De acordo com [17] cada *flip-flop* consome $3,5$ GE. Uma porta *XOR* consome $1,5$ GE pois tal porta pode ser construída usando-se um multiplexador de duas entradas e uma saída, $MUX(2 \rightarrow 1)$, que consome 1 GE, e uma porta *NOT*, que consome $0,5$ GE.

Por exemplo, o LFSR base do gerador de testes projetado para o circuito $c432$ com 36 entradas visto na Seção 4.1 tem o seguinte polinômio de realimentação:

$$\Lambda(x) = 1+x+x^5+x^9+x^{10}+x^{16}+x^{18}+x^{23}+x^{24}+x^{25}+x^{26}+x^{27}+x^{28}+x^{30}+x^{31}+x^{34}+x^{35}+x^{36}+x^{38}.$$

O cálculo da sobreárea de *hardware* consumida por esse LFSR baseia-se nos seguintes fatos:

1. Normalmente, o circuito sob teste faz parte de um macrocircuito que contém registradores e alguns outros circuitos independentes, por exemplo, um microprocessador. Para fins de economia de área de chip, esses registradores são modificados de forma a operarem como LFSR's. Dessa forma, como o registrador/LFSR já faz parte do macrocircuito, esse não é considerado no cálculo da sobreárea de *hardware*. Portanto, somente a circuitaria extra ao LFSR é considerada no cálculo da sobreárea.
2. Do exposto anteriormente, é considerado para o cálculo da sobreárea de *hardware* do gerador de testes proposto somente a parte do LFSR que excede o número de entradas do circuito sob testes, como é visto na Figura 5.1.

3. Dessa forma, como o circuito $c432$ tem 36 entradas e o polinômio de realimentação do LFSR é dado por $\Lambda(x) = 1 + x + \cdots + x^{35} + x^{36} + x^{38}$, então a parte do LFSR que excede 36 é dada por $x^{36} + x^{38}$. Essa parte excedente, $x^{36} + x^{38}$, consome 3 *flip-flop*'s, FF_{36}, FF_{37} e FF_{38}, e uma porta *XOR*. Portanto, a sobreárea de *hardware* é igual a $3 \cdot 3,5 + 1 \cdot 1,5 = 12GE$.

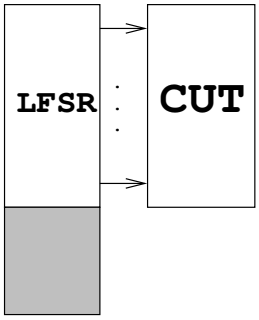


Figura 5.1: Somente a área hachurada é considerada no cálculo da sobreárea de *hardware* do esquema proposto.

Como será descrito na Seção 5.3, os resultados experimentais são comparados a dois trabalhos anteriormente publicados na área de testes e cujos esquemas são baseados em LFSR's. O primeiro desses trabalhos consiste em um gerador de testes baseado em reinicialização de LFSR e é visto em [45]. O projeto desse gerador é baseado em um algoritmo genético que busca o mínimo número de pares de **conteúdos iniciais** e **polinômios de realimentação** do LFSR que proporcione a máxima cobertura de falhas. Tais conteúdos iniciais e polinômios de realimentação são armazenados em memória. Somente essa memória e o bloco de controle são considerados nesse trabalho para o cálculo da sobreárea de *hardware* desse gerador, como pode ser visto na parte hachurada da Figura 5.2.

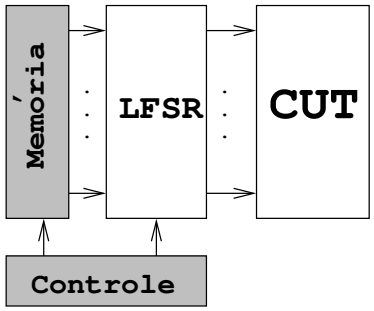


Figura 5.2: Sobreárea de *hardware* do esquema proposto por Chiusano em [42].

O outro trabalho usado para comparação é o de Kavousianos e descrito em [17]. Nesse trabalho é usado o método de agrupamento. Esse método é baseado nos seguintes fatos: (1) considere um circuito com q entradas, (2) projete um LFSR com n saídas ($n < q$), (3) forme n grupos (um grupo é um conjunto de entradas do circuito interligadas e que recebem um mesmo sinal), e (4) aplique o sinal de cada saída do LFSR ao grupo correspondente, como é visto na Figura 5.3. Em síntese, cada saída do LFSR pode ser ligada a várias entradas do CUT.

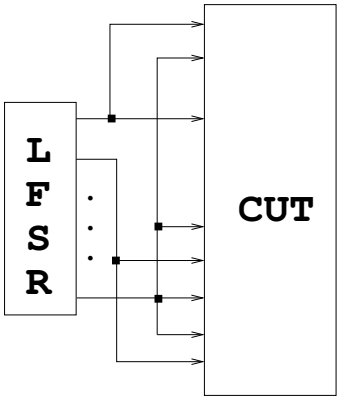


Figura 5.3: Técnica de agrupamento de entradas.

Especificamente nesse trabalho de Kavousianos é proposto um método baseado em múltiplos agrupamentos que consiste em realizar vários reagrupamentos ao longo do teste do circuito de maneira a melhorar a sua qualidade (aumentar a cobertura de falhas, diminuir o tempo de teste e/ou diminuir a sobreárea de *hardware*). Em relação à sobreárea de *hardware* consumida por esse esquema, o autor considerou somente os blocos de controle e de agrupamento no cálculo, como pode ser visto na parte hachurada da Figura 5.4.

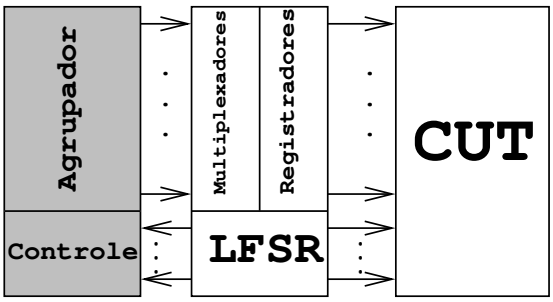


Figura 5.4: Sobreárea de *hardware* do esquema de múltiplos reagrupamentos proposto por Kavousianos em [17].

5.3 Resultados Experimentais

Como descrito na seção anterior, os resultados experimentais obtidos foram comparados com os resultados obtidos pelos esquemas propostos no trabalho de Kavousianos [17] e no de Chiusano [45]. O esquema de gerador de testes proposto em [17], de acordo com estudos de casos feitos nesta pesquisa, é o que apresenta melhor compromisso entre o comprimento de teste e a sobreárea de *hardware*. Já no esquema em [45], é considerado o que apresenta a menor sobreárea de *hardware* [10].

Na Coluna $\#FDD$ na Tabela 5.2 é visto o número de falhas difíceis consideradas em cada circuito e na Coluna $\#HC$, o número de hipercubos de testes compactados que detectam essas falhas, respectivamente.

Na Tabela 5.2 são relatados o comprimento de teste e a sobreárea de *hardware* requeridos por cada esquema para atingir 100% de cobertura de falhas para cada circuito, respectivamente. Os comprimentos de teste para cada circuito são vistos nas Colunas 6, 8 e 10, rotuladas como l_{teste}.

Em relação à sobreárea de *hardware*, os resultados podem ser vistos nas Colunas 5, 7 e 9, rotuladas como *GE*. Entre esses resultados, o símbolo (-) indica que o autor não forneceu a informação requerida. Além disso, alguns valores 0's aparecem na Coluna 9 indicando que o respectivo esquema não exige circuitaria extra, ou seja, o gerador é formado somente pelo LFSR base que não é considerado no cálculo da sobreárea de *hardware*.

Pelo dados relatados na Tabela 5.2, pode-se observar que o método proposto neste trabalho proporciona comprimentos de testes menores que os resultados apresentados em [45] para todos os circuitos. E que, também para todos os circuitos, o método proposto consome menos sobreárea de *hardware* que os resultados apresentados em [17].

Portanto, pelos resultados apresentados, o esquema do gerador de testes proposto oferece um compromisso entre o comprimento de teste e a sobreárea de *hardware* em relação aos resultados em [17] e em [45].

Tabela 5.2: Comparações entre as sobreárea de *hardware* (GE) e comprimento de teste (l_{teste}) obtidos pelo método proposto e os métodos em [17] e em [42] para atingir 100% de cobertura de falhas. O termo H é o valor da sobreárea de *hardware* consumida pelo bloco de controle do esquema de Chiusano e não relatada no respectivo trabalho. Obs: Alguns valores 0's aparecem na Coluna 9 indicando que o respectivo esquema não exige circuitos extras e somente consome um LFSR oriundo de um registrador já pertencente ao circuito.

Circuito	# Falhas	# FDD	# HC	Método proposto		Método do Agrupamento [17]		Método da reinicialização de LFSR [45]	
				GE	l_{teste}	GE	l_{teste}	GE	l_{teste}
c880	942	45	11	64,5	494	159,5	248	0	1829
c1355	1574	16	7	143,5	991	281	465	0	1334
c1908	1879	23	18	92	2883	147	1397	0	3759
c3540	3428	16	11	202	2724	-	-	0	4505
s420	430	19	12	138,5	2336	141,5	630	97 $+H$	10843
s641	463	16	10	100,5	1996	106,5	248	94,5 $+H$	2430
s713	581	32	12	51,5	1877	-	-	108 $+H$	2759
s1196	1242	35	21	114	14321	-	-	40 $+H$	18776

5.4 Conclusão

Neste capítulo foi descrito os procedimentos experimentais utilizados na obtenção de resultados usando o esquema de gerador de testes proposto. De acordo com os resultados experimentais obtidos, pôde-se observar que o método oferece um compromisso entre a sobreárea de *hardware* consumida e do comprimento de teste necessários para obter cobertura de falhas completa.

Parte III

PROPOSTA DE UM ANALISADOR DE RESPOSTAS

Capítulo 6

Algoritmo de Seleção Negativa

Neste capítulo, será apresentado um algoritmo inspirado no sistema imunológico humano que será o componente principal no desenvolvimento de um analisador de respostas para circuitos autotestáveis proposto nesta tese.

6.1 Breve Descrição do Sistema Imunológico

O sistema imunológico compreende todos os mecanismos pelos quais um organismo multicelular se defende de invasores, tais como bactérias, vírus ou parasitas. Para tanto, esse sistema é dotado de diversos mecanismos para o reconhecimento e destruição de tais invasores do organismo.

Existem dois tipos de mecanismos de defesa: os inatos ou não específicos, tais como; a proteção da pele, a acidez gástrica, a segregação de lágrimas, e o adquirido ou específico, como por exemplo, a ação direcionada dos linfócitos e a produção de anticorpos específicos.

Entre as células do sistema imunológico específico, encontramos os glóbulos brancos, ou linfócitos, que podem ser de dois tipos: T e B.

De modo simplificado, pode-se dizer que as células do tipo B são produzidas na medula óssea e são responsáveis por uma modalidade de defesa chamada **imunidade humoral** e atua diretamente na produção de anticorpos (imunoglobulinas) os quais são especificamente

capazes de identificar agentes estranhos através da detecção de antígenos presentes na superfície desses agentes e, além disso, podem destruir tais agentes através de uma ação direta ou indireta. Uma informação interessante é que, mesmo tendo sido eliminados os agentes portadores dos antígenos considerados estranhos, uma memória imunológica permanece e, durante anos, anticorpos específicos estarão circulando pelo sistema vascular do organismo e protegendo-o contra novos ataques daqueles mesmos agentes que, num primeiro contato, teriam sido detectados.

Já por sua vez, as células do tipo T são maturadas no Timo[1] durante a vida fetal e em alguns anos após o nascimento. Essas células são responsáveis por uma modalidade de defesa chamada **imunidade celular** baseada na formação de clones das células T antígeno-específicos para a identificação e combate aos agentes estranhos.

Em geral, os procedimentos iniciais de defesa do organismo partem da identificação de células **não-próprias** ao corpo. Um mecanismo de identificação, de interesse neste trabalho de pesquisa, é o mecanismo de maturação das células T que as tornam tolerantes às células **próprias** do organismo, tornando-se capazes de reconhecer somente antígenos nas células **não-próprias**.

Esse mecanismo de **discriminação próprio/não-próprio** consiste nos seguintes procedimentos: (1) durante o seu período de maturação, as células T passam por um processo de seleção genético-aleatório no Timo, chamado de **seleção negativa**. Esse processo de seleção negativa promove a eliminação das células T que têm na sua superfície epítetos (proteínas) que reagem com os antígenos presentes nas células próprias do corpo. (2) Após esse processo de maturação, somente as células T que não casam com nenhuma célula própria do corpo são permitidas sair do Timo e circular através do corpo para realizar suas funções imunológicas [46, 47]. Em síntese, a discriminação próprio/não-próprio baseia-se no fato que se uma célula T maturada casar (reagir) com alguma célula no organismo, então essa é considerada como não-própria, pois nenhuma célula T maturada casa com uma célula própria.

[1] O Timo é uma pequena glândula situada na porção antero-superior da cavidade torácica na altura da parte superior dos pulmões e cuja função é produzir células do tipo T.

6.2 Algoritmo de Seleção Negativa

Baseado no processo de maturação das células T, Forrest et al. [47] propuseram o **algoritmo de seleção negativa** (ASN) para a detecção de erros em dados de computador. Esse algoritmo funciona de forma similar ao processo de discriminação próprio/não-próprio do corpo, gerando detectores candidatos (células T imaturas) aleatoriamente e eliminando aqueles que casam (detectam) com elementos que são considerados próprios, de maneira que os detectores restantes possam detectar praticamente qualquer elemento não-próprio [46].

O algoritmo de seleção negativa é composto de três fases, a saber: (1) fase de definição dos dados a serem protegidos, (2) fase de geração de detectores, similar ao processo de seleção negativa que ocorre no Timo, e (3) fase de monitoramento de erro, similar ao processo de discriminação realizado pelas células T maturadas através do corpo. Essas fases são descritas em detalhes a seguir.

Definição dos dados a serem protegidos. Definem-se como elementos próprios de um sistema, ou seja, os dados a serem monitorados, uma coleção $R = \{R^1, R^2, \cdots, R^n\}$ de cadeias de comprimento s sobre um alfabeto finito.

Fase de geração de detectores. Gera-se um conjunto D de **detectores**, no qual cada um desses não detecta nenhuma cadeia própria em R, mas que, entretanto, sejam capazes de detectar qualquer cadeia diferente das de R, ou seja, sejam capazes de detectar cadeias não-próprias.

A fase de geração é ilustrada na Figura 6.1. Os detectores candidatos são gerados aleatoriamente e, então, testados para verificar se eles casam com alguma cadeia própria. Se um casamento é encontrado, o candidato é rejeitado. Esse processo é repetido até que um número desejado de detectores seja alcançado.

Ao invés de se usar um processo de casamento exato, o método usa critérios de casamento parcial (por exemplo, um critério de casamento parcial pode consistir em comparar duas cadeias da seguinte forma: essas duas cadeias casam se e somente se elas forem idênticas em pelo menos r posições contínuas, em que r é um parâmetro escolhido adequadamente).

Fase de monitoramento de erro. Supondo que o conjunto R, de cadeias próprias, seja passível de ocorrência de erros. Então, para verificar se um erro ocorreu, verificar-se continuamente se detectores em D casam com alguma cadeia em R. Se qualquer detector em algum momento casar com algum elemento de R, então um erro em R ocorreu, pois os detectores foram projetados de forma a não casar com nenhuma cadeia original de R. A fase de monitoramento de erro é ilustrada na Figura 6.2.

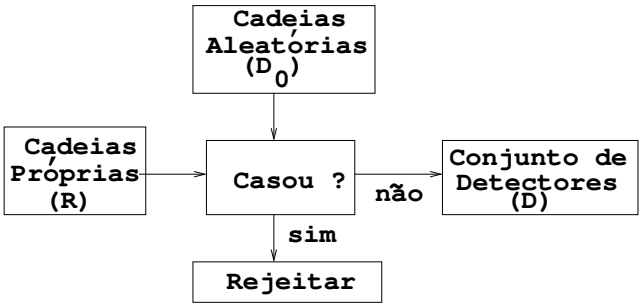


Figura 6.1: Fase de geração de detectores. As cadeias em D_0 que casam com alguma cadeia em R são rejeitadas. As cadeias que não casam são selecionadas como detectores.

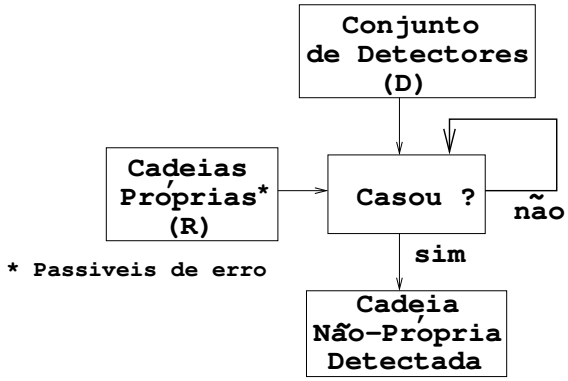


Figura 6.2: Fase de monitoramento de erro. Se algum detector em D, em algum momento, casar com alguma cadeia em R, então uma mudança em R ocorreu.

A fase de geração dos detectores é a que consome mais recursos computacionais e existem vários conjuntos de detectores para cada conjunto de dados a ser protegido, pois cada vez que essa fase é executada um novo conjunto de detectores é selecionado devido à natureza aleatória da geração dos detectores candidatos. Entretanto, essa fase é realizada de forma *off-line* [47]. Por outro lado, a fase de monitoramento de erro é a menos dispendiosa e realizada de forma *on-line* [47].

A maior limitação na geração dos detectores é devido ao fato de ser computacionalmente difícil gerar detectores válidos e com o aumento no número de cadeias em R torna-se ainda mais difícil a geração pois o número de detectores que não casam com nenhuma dessas cadeias decresce exponencialmente. Alguns algoritmos para geração de detectores aplicados ao ASN foram propostos a fim de aumentar a eficiência na geração de detectores válidos, como pode ser visto em [48] e [49].

Em síntese, o algoritmo de seleção negativa propõe-se a gerar detectores para (quase) todas as cadeias que não pertençam ao conjunto de dados original e é algoritmicamente viável no sentido de que um pequeno conjunto de detectores tem uma alta probabilidade de verificar uma mudança aleatória no conjunto original [47]. Além do mais, a solução dada por esse algoritmo é robusta pois um sistema de detecção baseado nesse tipo de abordagem detecta (probabilisticamente) qualquer atividade estranha ao invés de procurar por padrões específicos de erros ou mudanças conhecidos [46].

Exemplo: Considere que foram definidas como cadeias próprias, dados a serem monitorados, as seguintes 8 cadeias de 4 bits: $R = \{0010,\ 1000,\ 1101,\ 0000,\ 0100,\ 0010,\ 1001,\ 0011\}$.

A fase de geração de detectores consistirá em gerar cadeias aleatórias (conjunto D_0) e, então, verificar se as cadeias em D_0 casam com alguma cadeia em R. Caso alguma cadeia em D_0 casar com qualquer cadeia própria em R, então essa é rejeitada. As cadeias que não foram rejeitadas tornam-se membros da coleção de cadeias detectoras (D).

Por exemplo, suponha que $D_0 = 0111\ 1000\ 0101\ 1001$. Dessa forma, D consistirá de duas cadeias, 0111 e 0101, pois as cadeias 1000 e 1001 são rejeitadas porque cada uma delas casam com uma cadeia em R. Na prática, o procedimento consiste em gerar cadeias aleatórias continuamente até que o conjunto D tenha um número[2] adequado de membros.

Considerando-se que em R é possível a ocorrência de erros, na fase de monitoramento de erro verificar-se continuamente se detectores em D casam com alguma cadeia em R.

Por exemplo, suponha que um bit da última cadeia própria (0011) mude, produzindo a cadeia 0111. Então, em algum ponto da fase de monitoramento, é verificado que a cadeia não-própria 0111 casa com algum dos detectores (neste caso, o detector 0111), e um erro em R será relatado.

[2]Quanto maior o número de detectores, menor será a probabilidade de erro na detecção (não detectar uma cadeia não-própria)(Ver Seção 6.3).

6.3 Critério de Casamento Parcial

No algoritmo de seleção negativa é necessário definir um critério de casamento parcial para a execução das fases de geração de detectores e de monitoramento de erro devido ao fato de que um casamento perfeito entre duas cadeias de comprimentos iguais significa que, em cada posição de ambas as cadeias, seus elementos têm o mesmo valor. Nesse caso, o casamento é completamente especificado e, então, cada detector detectará somente uma única cadeia. Por outro lado, usando-se um critério de casamento parcial, cada detector tem sua capacidade de detecção aumentada como será visto em seguida.

Dado um espaço U_v que contenha todas as cadeias de comprimento v em um dado alfabeto de cardinalidade p, considere o conjunto $R \subset U_v$ como sendo o conjunto das cadeias próprias. Dessa forma, o conjunto das cadeias não-próprias é dado por $N = U_v \setminus R$.

Suponha que r é um valor inteiro e que $0 \leq r \leq v$. Considere $C_r(x, y)$ como sendo uma medida de correlação entre duas cadeias x e y. Um critério de casamento parcial baseado em $C_r(x, y)$ prover a um detector x a capacidade de detectar cadeias $y \in N$ em sua vizinhança de acordo com o valor de r.

Definição 13 *Um critério de casamento parcial dado pela função indicadora $M_r(x, y)$ é definido por:*

$$M_r(x, y) = \begin{cases} verdadeiro, & se\ C_r(x, y) \geq r \\ falso, & caso\ contrário. \end{cases}$$

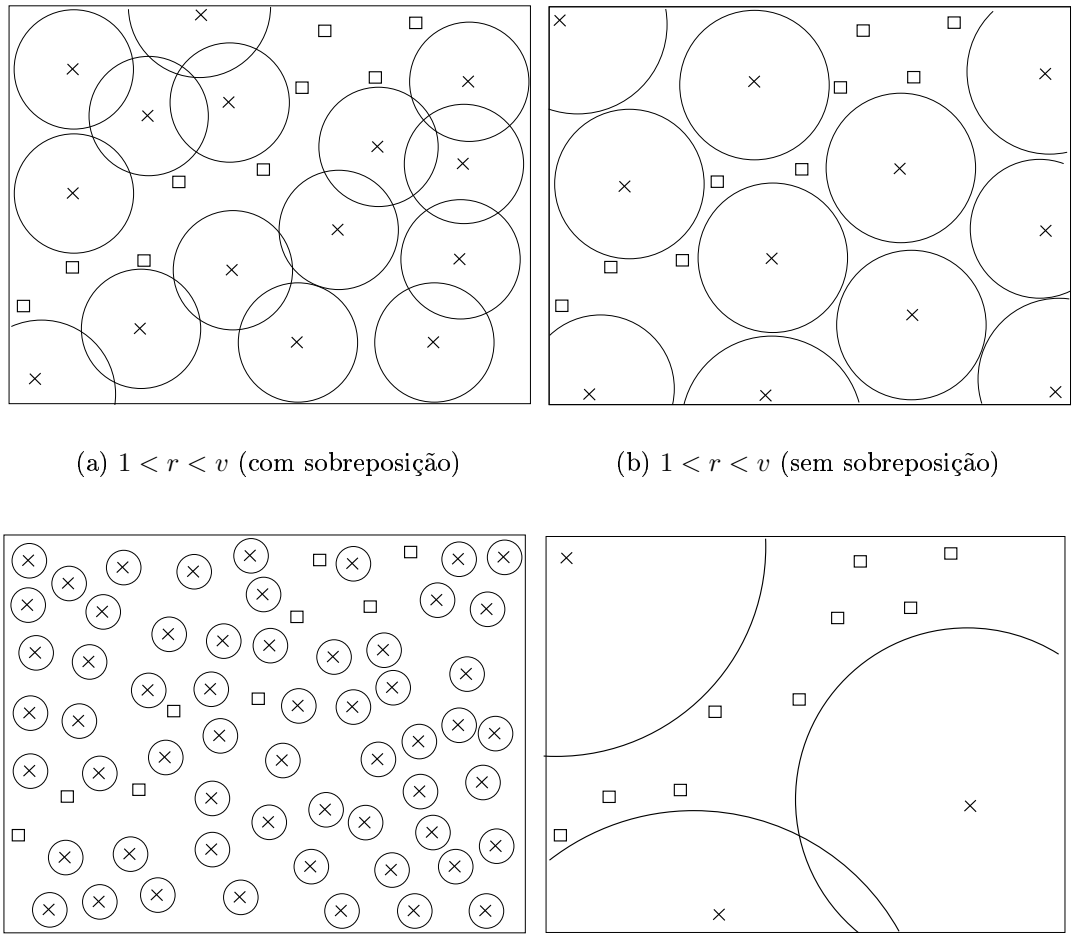
Definição 14 *O espaço de detecção S de um detector x e de parâmetro r é dado por:*

$$S(x, r) = \{y : M_r(x, y) = verdadeiro\}.$$

Na Figura 6.3(a) é ilustrada, pictoricamente em um espaço bidimensional, a idéia básica acerca de espaços de detecção e sua relação com o parâmetro r. Nessa figura, os quadrados representam as cadeias próprias, os símbolos $\times$ representam os detectores, e os espaços de detecção são representados por círculos. Pode-se observar que sobreposições de espaços de detecção podem ocorrer. Tal fato indica que múltiplos detectores são capazes de detectarem

uma mesma cadeia não-própria, ou seja, existem detectores redundantes para cobrir o espaço não-próprio N. Um caso ótimo, em relação ao número de detectores, é mostrado na Figura 6.3(b) em que não há sobreposição de espaços de detecção.

Casos extremos são verificados quando $r \to v$ e $r \to 1$ como mostrado na Figura 6.3(c) e na Figura 6.3(d), respectivamente. No primeiro caso, o espaço de detecção de cada detector é bastante reduzido resultando em um grande número de detectores para a detecção das cadeias não-próprias. No segundo caso, cada detector tem um grande espaço de detecção, entretanto, a geração de tais detectores é computacionalmente difícil.



(a) $1 < r < v$ (com sobreposição) (b) $1 < r < v$ (sem sobreposição)

(c) $r \to v$ (d) $r \to 1$

Figura 6.3: Relação entre espaços de detecção com o parâmetro r. As cadeias próprias são representadas por □, detectores por ×, espaços de detecção por ◯. (a) Espaços de detecção com parâmetro r adequado, mas, com ocorrência de sobreposição. (b) Sem ocorrência de sobreposição. (c) Espaços de detecção com $r \to v$. (d) Espaços de detecção com $r \to 1$.

Uma conseqüência no uso de critérios de casamentos parciais é que há um compromisso entre o valor do parâmetro r com o número de detectores necessários para cobrir o espaço de cadeias não-próprias N.

Dois critérios de casamentos parciais bastante utilizados em sistemas de monitoramento de erros baseados no ASN são os critérios **r-contínuos** e o **r-Hamming**. Esses critérios são controlados pelo parâmetro r e são descritos a seguir.

6.3.1 Critério de casamento r-contínuos

O critério de casamento r-contínuos foi proposto por Forrest et al. [47] e consiste em verificar se duas cadeias de comprimentos iguais casam em pelo menos r posições contínuas respectivamente.

Por exemplo, suponha que duas cadeias binárias de comprimento igual a 16 sejam dadas, como visto na Figura 6.4. Essas duas cadeias casam para $r \leq 5$ usando o critério r-contínuos.

```
x:  1 1 0 0 1 0 1 0 1 0 1 1 1 0 1 0
y:  0 0 1 0 1 0 1 0 0 1 1 0 1 0 1 1
```

Figura 6.4: Exemplo de casamento parcial baseado no critério r-contínuos. As cadeias x e y casam para $r \leq 5$.

Em geral, dadas duas cadeias x e y compostas de v símbolos de um alfabeto de cardinalidade p cada uma, defina $C_r^C(x, y)$, a medida de correlação baseada no critério r-contínuos, como sendo o máximo número de símbolos de x e y os quais essas cadeias casam continuamente em suas posições correspondentes. Dessa maneira, a respectiva função indicadora $M_r^C(x, y)$ é dada por:

$$M_r^C(x,y) = \begin{cases} verdadeiro, & \text{se, } C_r^C(x,y) \geq r \\ falso, & \text{caso contrário.} \end{cases}$$

6.3.2 Critério de casamento r-Hamming

O critério de casamento r-Hamming consiste em verificar se duas cadeias x e y de comprimentos iguais casam em r posições correspondentes mas não necessariamente de forma contínua, ou seja, consiste na verificação do complemento da distância de Hamming entre elas.

Por exemplo, como visto na Figura 6.5, as cadeias x e y casam para $r = 9$ pois têm símbolos iguais em 9 posições correspondentes. De fato, para qualquer $r \leq 9$, as cadeias x e y casam.

```
x:  1 1 0 0 1 0 1 0 1 0 1 1 1 0 1 0
y:  0 0 1 0 1 0 1 0 0 1 1 0 1 0 1 1
```

Figura 6.5: Critério de casamento r-Hamming com parâmetro $r = 9$. As cadeias x e y casam para $r \leq 9$.

Para o critério r-Hamming, define-se $C_r^H(x, y) = \sum_i^v \overline{x_i \oplus y_i}$ como sendo a sua respectiva medida de correlação. Dessa forma, $M_r^H(x, y)$, sua respectiva função indicadora, é dada por:

$$M_r^H(x, y) = \begin{cases} verdadeiro, & \text{se, } C_r^H(x, y) \geq r \\ falso, & \text{caso contrário.} \end{cases}$$

6.3.3 Probabilidade de Erro de Detecção

Considere que se tenha um conjunto de d detectores e deseja-se saber qual a probabilidade que esses detectores não detectem um elemento não-próprio do sistema, ou melhor, qual a probabilidade de erro na detecção de uma mudança ocorrida em algum elemento próprio do sistema não ser relatada. Para responder a essa questão, faz-se necessário saber a probabilidade de que duas cadeias aleatórias casem de acordo com um dado critérios de casamento como será vista a seguir.

Para o critério de casamento r-contínuos, a probabilidade P_M de que duas cadeias aleatórias casem em pelo menos r posições contínuas é dada por:

$$P_{M_r^C} \approx p^{-r} \left[\frac{(v-r)(p-1)}{p} + 1 \right] \tag{6.1}$$

em que v é o comprimento da cadeia própria, p é o número de símbolos do alfabeto e $p^{-r} \ll 1$ [50].

Para o critério de casamento r-Hamming, a probabilidade P_M de que duas cadeias aleatórias casem em pelo menos r posições por esse critério [51] é dada por:

$$P_{M_r^H} = p^{-v} \left(\sum_{i=r}^{v} \binom{v}{i} (p-1)^{v-i} \right). \tag{6.2}$$

Em geral, usando-se qualquer critério de casamento, como a probabilidade de um detector não casar com uma cadeia não-própria é dada por $1 - P_{M_r}$ e a probabilidade de dois detectores não casarem com uma cadeia não-própria é dada por $(1 - P_{M_r})^2$ e assim por diante. Então, a probabilidade que d detectores falhem em detectar uma cadeia não-própria, isto é, a probabilidade de erro de detecção P_e será dada por:

$$P_e = (1 - P_{M_r})^d. \tag{6.3}$$

Dessa forma, quanto maior for o número de detectores menor será a probabilidade de erro de detecção P_e. Além disso, quanto maior a probabilidade de casamento P_M menor será a probabilidade de erro de detecção P_e.

6.4 Conclusão

Neste capítulo foi descrito o algoritmo de seleção negativa que foi inicialmente usado para verificação de erros em dados, assim como, na verificação de virus de computador. Foram descritos também as fases de execução desse algoritmo e os critérios de casamentos parciais utilizados por nesse. No próximo capítulo será apresentado o esquema de um analisador de respostas baseado no algoritmo de seleção negativa.

Capítulo 7

Analisador de Respostas Baseado no Algoritmo de Seleção Negativa

Neste capítulo será apresentada a segunda contribuição desta tese e que consiste na utilização do **algoritmo de seleção negativa** inspirado do sistema imunológico humano no projeto de um analisador de respostas aplicado em circuitos autotestáveis.

Na próxima seção será feita uma breve descrição acerca dos analisadores de respostas convencionais baseados em análise de assinatura. Em seguida, na Seção 7.2, será descrito o analisador proposto.

7.1 Analisadores de Respostas

Como visto no Capítulo 1, em circuitos autotestáveis, uma seqüência de testes $T = (T_1, T_2, \cdots, T_n)$ é aplicada ao circuito sob teste (CUT) por um **gerador de testes** (TPG) e a seqüência de respostas a esses testes, $R = (R_1, R_2, \cdots, R_n)$, é avaliada por um **analisador de respostas** (ORA) cujo sinal de saída indica se o CUT está falho ou não, como é visto na Figura 7.1.

Considerando $R^* = (R_1^*, R_2^*, \cdots, R_n^*)$ a seqüência de respostas do circuito sem falha, uma possível avaliação do CUT consiste em comparar bit-a-bit os elementos de R com os

de R^*. Entretanto, isso é impraticável em circuito autotestáveis pois R^*, que normalmente é composta de centenas a milhares de vetores, teria que ser armazenada no próprio circuito integrado.

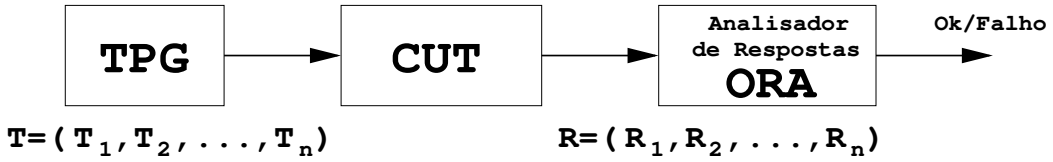


Figura 7.1: Avaliação das respostas do CUT através de um analisador de respostas (ORA).

Por exemplo, supondo que foram aplicados 100.000 vetores de teste a um circuito com 50 saídas, então ter-se-á $100.000 \cdot 50 = 5.000.000$ bits de respostas que torna o custo de armazenamento inviável. Uma solução encontrada para esse problema é a utilização de técnicas de compressão.

Uma técnica de compressão bastante utilizada é a da **análise de assinatura** que consiste em comprimir R em um vetor de poucos bits s, denominado de **assinatura**, e a avaliação do circuito consiste em se comparar a assinatura do CUT com a assinatura do circuito sem falha, s^*, no final do teste [13], como pode ser visto na Figura 7.2.

Em geral, em analisadores de respostas baseados em assinatura, a partir da seqüência de respostas do circuito sem falha, R^*, é determinada a sua assinatura, s^*, e, por ser um vetor de dimensões bastantes reduzidas, é viável seu armazenamento no próprio circuito integrado. Já a avaliação do CUT consiste na compressão da sua seqüência de respostas, R, em uma assinatura, s, que é comparada com s^* no final da aplicação dos testes. Assim, se $s = s^*$ então o CUT é considerado sem falha, caso contrário, esse tem alguma falha.

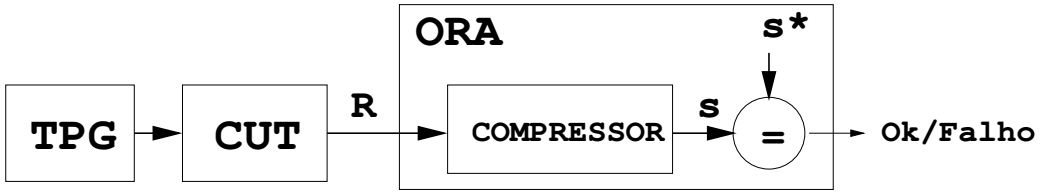


Figura 7.2: Técnica de análise de assinatura. A avaliação consiste em comparar a assinatura do circuito sem falha previamente determinada, s^*, com a assinatura do CUT, s, no final do teste. Se $s = s^*$, então o circuito é considerado sem falha. Caso contrário, está falho.

Entretanto, devido ao processo de compressão na obtenção das assinaturas, perdas de informação podem ocorrer e podem levar a um erro de diagnóstico do circuito. Dessa

forma, o circuito pode ser declarado sem falha estando, de fato, com falha. Tal deficiência no processo de detecção de falha é denominada de *aliasing* [13].

Por exemplo, suponha que o CUT esteja falho e que sua seqüência de respostas aos testes, R, seja diferente da resposta do circuito sem falha, R^*. Suponha também que, após passar pelo compressor, a assinatura do CUT, s, seja igual a do circuito sem falha, s^*. Então, dessa forma, *aliasing* ocorreu, visto que, o circuito de fato está com falha, pois $R \neq R^*$, mas pela comparação das assinaturas, $s = s^*$, o circuito foi considerado sem falha. Um estudo mais aprofundado sobre análise de assinatura pode ser visto no Apêndice B.

Em geral, as técnicas de compressão são bastantes empregadas em circuitos autotestáveis mas, entretanto, têm como principais desvantagens o fato que a probabilidade de erro de diagnóstico (*aliasing*) não é zero e que a verificação do circuito é realizada somente no final da aplicação dos testes.

Na próxima seção, será apresentado um algoritmo inspirado no sistema imunológico humano que será o componente principal no desenvolvimento de um esquema de analisador de respostas o qual obtém probabilidade de *aliasing* zero e identifica se o circuito está falho durante o teste e não somente no final deste.

7.2 Estrutura do Analisador de Respostas Proposto

Neste capítulo é apresentado um analisador de respostas baseado no algoritmo de seleção negativa (ASN) inspirado no sistema imunológico como visto na Seção 6.2. O esquema do analisador de respostas proposto [51], denominado de **analisador imunológico**, pode ser visto na Figura 7.3.

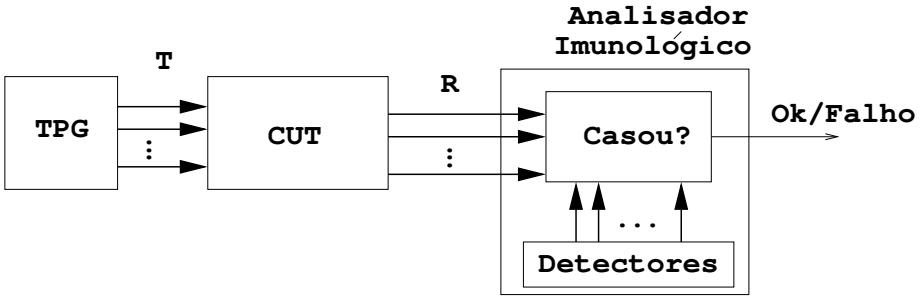


Figura 7.3: Esquema proposto do analisador imunológico.

O funcionamento básico do analisador de resposta proposto é baseado nos seguintes procedimentos: (1) na etapa de projeto, gera-se um conjunto de detectores através do algoritmo de seleção negativa considerando como cadeias próprias as respostas do circuito sem falha, e (2) o teste do circuito pode ser realizado usando-se esses detectores e executando-se a fase de monitoramento de erro do algoritmo de seleção negativa usando o esquema visto na Figura 7.3. A detecção de falha é possível, através desse procedimento, pois se algum detector casar com alguma resposta do CUT, então é sabido que esse está falho pois a resposta do circuito com falha difere da resposta do circuito sem falha[1], sendo assim, um cadeia não-própria passível de ser detectada [52].

Os procedimentos em detalhes do projeto do analisador de respostas proposto são descritos abaixo:

1. Considere $R^* = (R_1^*, R_2^*, \cdots, R_n^*)$ a seqüência de n respostas do circuito sem falhas com v saídas binárias quando uma seqüência de testes é aplicada.
2. Considere cada resposta R_i^* ($R_i^* \in \{0,1\}^v$ e $1 \leq i \leq n$) como sendo uma cadeia própria, ou seja, como sendo um dado a ser protegido.
3. Aplique essas cadeias próprias no algoritmo de seleção negativa e gere um conjunto de detectores $D = \{D_1, D_2, \cdots, D_d\}$ em que d é o número de detectores obtidos e $D_i \in \{0,1\}^v$.
4. Verifique se os d detectores obtidos detectam todas as falhas do circuito. Caso contrário, gere mais detectores e verifique novamente.
5. Use os detectores gerados para o teste do CUT usando o esquema visto na Figura 7.3. Dessa forma, se em algum momento do teste, um detector casar, então é sabido que o CUT está falho.

Em geral, usando o esquema de analisador de respostas e os procedimentos propostos, foi possível detectar todas as falhas do circuito com um número razoável de detectores [53]. Entretanto, como descrito na Seção 6.3, detectores redundantes podem ocorrer. Dessa forma, considerando que quanto maior é o número de detectores maior é a sobreárea de

[1]Essa declaração é verdadeira somente para falhas observáveis, ou seja, aquelas que a resposta do circuito com essa falha difere da resposta do circuito sem falha.

hardware, na próxima seção será apresentado um algoritmo para reduzir o número de detectores redundantes.

7.3 Algoritmo Proposto para Redução de Detectores Redundantes

Há uma peculiaridade no uso do algoritmo de seleção negativa em testes de circuitos, a saber: é possível obter previamente as respostas do circuito com falha usando-se um simulador de falhas, ou seja, é possível obter o espaço das cadeias não-próprias. Tal fato, torna possível, após a execução da fase de geração de detectores, a eliminação dos detectores redundantes usando o algoritmo de otimização descrito a seguir.

Considere $D = \{D_1, D_2, \cdots, D_d\}$ como sendo o conjunto de detectores gerados pelo algoritmo de seleção negativa e que tais detectores detectam w falhas de um circuito formando o conjunto $S = \{f_1, f_2, \cdots, f_w\}$.

Considere $S_i = \{f_1^{'}, \cdots, f_k^{'}\}$, $S_i \subset S$, como sendo o espaço de detecção do detector D_i para $1 \leq i \leq d$.

O algoritmo de otimização, em síntese, funciona da seguinte maneira: para cada detector D_i, são verificadas as falhas detectadas por esse e determinado o seu espaço de detecção S_i. Em seguida, verifica-se se S_i contém algum espaço de detecção S_j anteriormente determinado ($1 \leq j < i$). Se verdadeiro, o detector D_j é eliminado. Caso contrário, D_i é salvo no conjunto de detectores ótimos L. Dessa forma, a saída desse algoritmo é um conjunto mínimo de detectores $L = \{D_1^*, \cdots, D_m^*\}$ ($m \leq d$) o qual é capaz de detectar todas as falhas em S. O algoritmo em detalhes é descrito na Figura 7.4.

Pode-se mostrar que o tempo de execução do algoritmo proposto é da ordem de $O(wd^2)$ desde que para verificar se um espaço de detecção S_i "cobre" um espaço previamente verificado são necessários w comparações, e o loop ENQUANTO é executado $d(d-1)/2$ vezes.

Na próxima seção, alguns resultados de simulações experimentais usando o analisador imunológico juntamente com o algoritmo de otimização proposto serão mostrados.

ENTRADA: $D = \{D_1, D_2, \cdots, D_d\}$: conjunto inicial de detectores

SAÍDA: $L = \{D_1^*, \cdots, D_m^*\}$: conjunto ótimo de detectores

INICIALIZAÇÃO:
$i \leftarrow 1$ e j : contadores
$L \leftarrow D_1, D_2, \cdots, D_d$: lista de detectores
$S \leftarrow S_1, S_2, \cdots, S_d$: conjunto de espaços de detecção

ITERAÇÃO:
Passo 1. ENQUANTO i $<$ d, FAÇA
PARA j = i + 1 ATÉ j $<$ d, FAÇA
SE $S_i \subset S_j$, ENTÃO delete D_i de L e vá ao Passo 2.
$j \leftarrow j + 1$
Passo 2. FIM do PARA
Passo 3. $i \leftarrow i + 1$
Passo 4. FIM do ENQUANTO

Figura 7.4: Algoritmo de otimização proposto para a redução dos espaços de detecção sobrepostos. A entrada do algoritmo é um conjunto de detectores $D = \{D_1, D_2, \cdots, D_d\}$, gerados pelo algoritmo de seleção negativa, capaz de detectar as falhas $S = \{f_1, f_2, \cdots, f_w\}$ de um circuito. A saída é um conjunto reduzido de detectores $L = \{D_1^*, D_2^*, \cdots, D_m^*\}$ $(m \leq d)$ o qual é capaz de detectar as mesmas falhas em S.

7.4 Resultados Experimentais usando o Analisador Imunológico

As simulações experimentais foram implementadas em um simulador de falhas desenvolvido neste trabalho de pesquisa, denominado de FSIA, executado no sistema operacional LINUX e utilizando o ambiente gráfico Kylix da Borland baseado na linguagem C++. Os resultados foram obtidos utilizando-se os circuitos no padrão ISCAS85.

Foi utilizado nos experimentos o critério de casamento r-Hamming devido ao fato de que o esquema baseado nesse critério consome menos sobreárea de *hardware* que quando baseado no critério r-contínuos. Tal fato é verdadeiro pois o esquema baseado no critério r-Hamming utiliza somente portas lógicas OU-EXCLUSIVAS e um contador combinacional baseado **meio-somadores** que, por sua vez, são baseados em portas OU-EXCLUSIVAS e em portas E.

Nas Colunas 1, 2 e 3 da Tabela 7.1 são vistos os circuitos utilizados, o número de saídas e o número de falhas de cada circuito, respectivamente. O número de vetores de testes n aplicados a cada circuito é visto na Coluna 4 o qual atinge 100% de cobertura de falhas.

O experimento consistiu em gerar um conjunto de detectores que detectem todas as falhas do circuito e eliminar os detectores redundantes usando o algoritmo de otimização proposto. Nas Colunas 5 (# D) e 6 (# DC) da Tabela 7.1 são relatados o número de detectores iniciais (d) e o número de detectores após a compactação (m), respectivamente. O parâmetro r utilizado no ASN na geração dos detectores nesses experimento são vistos na Coluna 7. A sobreárea de *hardware* do analisador imunológico é vista na Coluna 8. Para efeitos de comparação, é visto na Coluna 9 o valor da sobreárea de *hardware* de um esquema de um analisador de respostas, que atinge *aliasing* zero, baseado em compressão de espaço [54] e de tempo [13] visto na Figura 7.5.

Tabela 7.1: Resultados experimentais usando o analisador de respostas proposto em conjunto com o algoritmo de otimização.

Circuito	# saídas	# falhas	# testes (n)	# D (d)	# DC (m)	r	Immuno ORA (GE)	[54] [13] (GE)
c499	32	758	52	10	1	13	119	272
c880	26	942	47	26	3	12	78	204
c1355	32	1574	84	147	1	19	119	440
c1908	25	1879	112	56	4	16	72	470
c2670	140	2747	100	13	2	73	2424	2144
c3540	22	3428	141	75	4	13	540	529
c5315	123	5350	106	89	5	62	1869	2003
c6288	32	7744	27	14	3	16	119	142
c7552	108	7550	198	78	6	58	1438	3296

Dos resultados obtidos, pode-se verificar que o algoritmo de otimização proposto é bastante eficiente na eliminação de detectores redundantes. Além do mais, esses detectores detectam todas as falhas e, dessa forma, não ocorre erro de detecção, ou seja, o *aliasing* é zero.

Uma outra informação importante a ser considerada é que uma falha pode ser detectada durante o teste. Dessa forma, não é necessário ser executado todo o teste para que seja verificado se o circuito está falho.

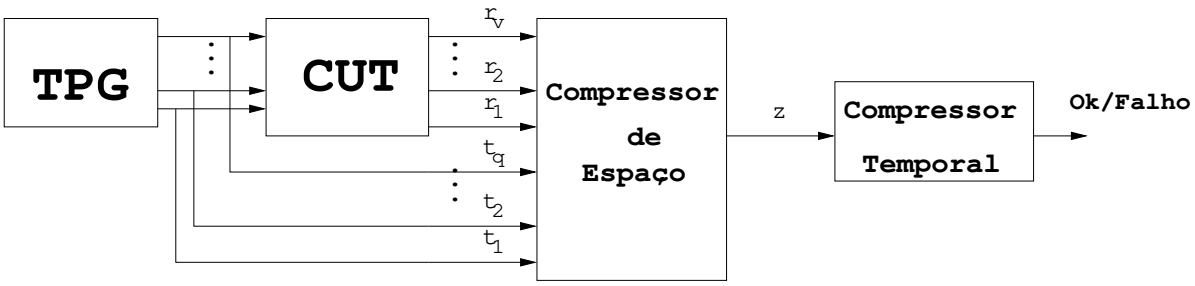


Figura 7.5: O analisador de respostas com *aliasing* é zero baseado em compressão de espaço e de tempo. O compressor de espaço recebe os bits de saída do CUT e, nesse caso, os bits de saída do TPG para que o *aliasing* seja zero [54], e os comprime em um só bit para cada teste aplicado. A seqüência de bits de saída do compressor de espaço, considerando que foi aplicada uma seqüência de testes ao CUT, é aplicada ao compressor de tempo que comprime essa seqüência binária em uma assinatura de poucos bits.

7.5 Conclusão

As principais aplicações do algoritmo de seleção negativa encontradas na bibliografia pesquisada foram em detecção de virus de computador [47], em detecção de falhas em motores de indução [55, 56], e em sistemas tolerantes à falhas [50]. Baseando-se na bibliografia pesquisada, constitui uma inovação a aplicação desse algoritmo em circuitos autotestáveis.

Neste capítulo foi descrito os procedimentos adotados no projeto de um analisador de respostas de circuito baseado no algoritmo de seleção negativa. Foi proposto um algoritmo para eliminação de detectores redundantes que, pelos resultados experimentais obtidos, mostrou-se bastante eficiente. Em geral, utilizando-se o esquema de analisador de respostas proposto em conjunto com o algoritmo de otimização, um número bastante reduzido de detectores são necessários para testar todas as falhas do circuito e com erro de detecção zero.

Parte IV

CONCLUSÃO

Utilizando-se o algoritmo de Berlekamp-Massey, amplamente aplicado em sistemas de comunicações, um método de desenvolvimento de um gerador de testes mistos foi apresentado. O gerador proposto é baseado totalmente em um único e simples LFSR sintetizado pelo algoritmo de Berlekamp-Massey e otimizado por meio de um algoritmo genético. Esse LFSR é capaz de gerar testes determinísticos, que detectam as falhas de difícil detecção do circuito, e testes residuais que detectam as falhas restantes. De acordo com os resultados experimentais obtidos, o método mostrou-se efetivo na redução da sobreárea de *hardware*, além de permitir uma operação de controle bastante simples na geração dos testes e permitir a realização de teste em velocidade. Com base na bibliografia pesquisada, é uma inovação o uso do algoritmo de Berlekamp-Massey em projetos de geradores de testes para circuitos autotestáveis.

Foram propostos também neste trabalho, uma modificação no algoritmo de Berlekamp-Massey, a fim de adaptá-lo aos requisitos de geração de vetores de testes, e um método de compactação de hipercubos de testes que se mostraram bastantes efetivos na redução do comprimento do LFSR base do gerador de testes proposto.

Um outro objetivo do trabalho, é a apresentação de um esquema de analisador de respostas baseado no sistema imunológico humano. Com a aplicação do algoritmo de seleção negativa inspirado desse sistema, o analisador de respostas proposto foi capaz de discriminar a resposta do circuito livre de falha (própria) de qualquer outra resposta considerando o circuito falho (não-própria). Além disso, foi proposto um algoritmo para redução do número de detectores que, pelos resultados experimentais obtidos, mostrou-se bastante eficiente e utilizando-se o esquema de analisador de respostas proposto em conjunto com esse algoritmo de otimização, um baixo número de detectores são necessários para testar todas as falhas do circuito e com erro de detecção zero.

Dessa forma, usando o gerador de testes e o analisador de respostas propostos, pode-se projetar um circuito autotestável de forma eficiente comprovada pelos resultados experimentais obtidos.

Para trabalhos futuros, propõe-se os seguintes estudos:

1. A princípio, a modificação no algoritmo de Berlekamp-Massey é uma solução local, pois nada garante que as substituições dos bits irrelevantes proposta otimize o LFSR de forma global. Dessa forma, faz-se necessário um estudo para verificar tal fato;

2. Aplicar o gerador proposto no teste de circuitos seqüenciais;

3. Aplicar o analisador de resposta imunológico proposto no teste de sistema-em-um-só-chip;

4. E, a partir de um circuito específico, desenvolver e conceber um circuito autotestável usando as técnicas aqui apresentadas.

Apêndice A

Lista de Publicações

A.1 Publicações em Conferências Internacionais

SOUZA, C. P., FREIRE, R. C. S. and ASSIS, F. M. Testing System-on-a-Chip using Artificial Immune System IEEE International SOC Conference. September, 2005. Washington, DC. USA.

SOUZA, C. P., FREIRE, R. C. S. and ASSIS, F. M. Circuit Testing Using Self-Nonself Discrimination. 2005 IEEE Instrumentation And Measurement Technology Conference (IMTC 2005). Ottawa, Ontario - CANADA 2004. May, 2005.

SOUZA, C. P., FREIRE, R. C. S. and ASSIS, F. M. A Mixed-Mode BIST Based on LFSR Synthesized Using Berlekamp-Massey Algorithm and Aided by Genetic Optimization. 6th IEEE Latin-American Test Workshop. Salvador - Brazil. March, 2005.

SOUZA, C. P., FREIRE, R. C. S. and ASSIS, F. M. A BIST scheme Based on Self-Nonself Discrimination of Immune System. Proceedings of the 2004 IEEE Signal Processing Society Workshop. São Luís - Brazil. September 29 - October 1, 2004.

SOUZA, C. P., FREIRE, R. C. S. and ASSIS, F. M. A Zero-Aliasing Test Response Analyzer Based on Berlekamp-Massey Algorithm. 9th IEEE European Test Symposium. Ajaccio, Corsica - France May 23-26, 2004.

SOUZA, C. P., FREIRE, R. C. S. and ASSIS, F. M. A Test Response Analyzer Based on Berlekamp-Massey Algorithm for BIST. 5th IEEE Latin-American Test Workshop. Cartagena - Colombia. Março, 2004.

A.2 Publicações em Conferências Nacionais

SOUZA, C. P., FREIRE, R. C. S. and ASSIS, F. M. Usando o Algoritmo de Berlekamp-Massey em Testes de Circuitos Integrados. Submetido ao XXII Simpósio Brasileiro de Telecomunicações (SBrT'05).

SOUZA, C. P., FREIRE, R. C. S. and ASSIS, F. M. Testing System-On-a-Chip using Artificial Immune System. Submetido ao VII Congresso Brasileiro de Redes Neurais.

Apêndice B

Análise de Assinatura

A técnica de análise de assinatura é um esquema de compressão linear baseado em divisão polinomial [13] e suportada por estruturas baseadas em LFSR. Tem-se que $GF(2^q)[x]$ é um anel de polinômios com coeficientes pertencentes ao **campo de Galois** $GF(2^q)$. Em [14], um LFSR é usado para comprimir a seqüência de resposta aos testes em uma assinatura compacta. A estrutura desse LFSR é caracterizada por um polinômio de realimentação $\Phi(x) \in GF(2^q)[x]$ e definido por:

$$\Phi(x) = x^m + \Phi_{m-1}x^{m-1} + \cdots + \Phi_1 x + \Phi_0 \tag{B.1}$$

em que todos os coeficientes pertencem a $GF(2^q)$ e representam as conexões de realimentação do LFSR. O i-ésimo coeficiente Φ_i define o multiplicador para a i-ésima conexão de realimentação, como é vista na Figura B.1.

A seqüência de resposta de q-bits aos testes de um circuito, com q saídas binárias, é representada pelo polinômio, $R(x)$, definido como:

$$R(x) = R_n x^n + R_{n-1}x^{m-1} + ... + R_1 x + r_0, \quad (R_i \in GF(2^q)) \tag{B.2}$$

em que ocorre um deslocamento elemento por elemento para o LFSR. No final de n deslocamentos, o conteúdo do LFSR, $S(x)$, é a assinatura do circuito, e é calculada como o resíduo $S(x)$ de acordo com o algoritmo de divisão euclidiano dado por:

$$R(x) = Q(x)\Phi(x) + S(x) \tag{B.3}$$

onde grau$(S(x)) <$ grau$(\Phi(x))$ e o polinômio $Q(x)$ é o quociente da operação que é deslocado elemento por elemento pela saída (último estágio) do LFSR e, normalmente, são descartados.

A assinatura $S(x)$, obtida pela equação B.3, é comparada com uma predeterminada assinatura do circuito considerado sem falha $S^*(x)$ dada por:

$$R^*(x) = Q^*(x)\Phi(x) + S^*(x) \tag{B.4}$$

em que $R^*(x)$ representa a seqüência de resposta do circuito considerado sem falha.

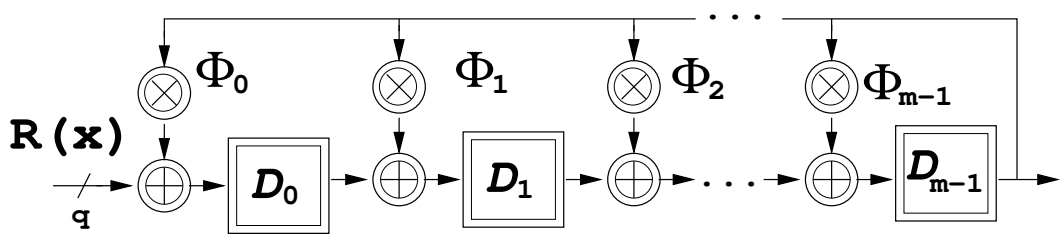


Figura B.1: Estrutura do LFSR.

O problema principal com análise de assinatura é que pode ocorrer que $S(x) = S^*(x)$ mesmo que $R(x) \neq R^*(x)$. Isso resulta em declarar um circuito sem falha, mesmo que esse apresente falha e, dessa forma, ocorre *aliasing*. Algumas técnicas de redução da probabilidade de ocorrência de *aliasing* em analisadores de assinatura foram propostas, entretanto, nenhuma reduz essa probabilidade a zero exceto por comparação elemento por elemento entre $R(x)$ e $R^*(x)$ [14].

B.1 Algumas Estruturas de Analisadores de Assinatura

Existem várias estruturas de analisadores de assinatura. Essas estruturas são particularidades de uma estrutura geral proposta por [13] denominada de GLFSR (da expressão em inglês *generalized linear feedback shift register*) a qual é um LFSR em que todas operações e elementos estão em $GF(2^q)$ e contém m estágios de elementos armazenadores.

Em geral, esse GLFSR pode ser caracterizado por um LFSR(q,m). Por exemplo: para um circuito de 5 saídas binárias, pode-se usar o LFSR(5,1) que é chamado de MISR (da expressão em inglês *multi-input shift register*). Convém observar que q pode ser maior ou igual ao número de saídas binárias do circuito e que m influencia na probabilidade de *aliasing* do compressor e na sobreárea de *hardware* [1]. Em seguida é apresentado alguns desses compressores.

Com a escolha adequada de q e m pode-se reduzir a estrutura LFSR(q,m) para qualquer estrutura convencional, tais como: LFSR, MISR, múltiplos MISR e MLFSR (multi-entrada LFSR) [13].

- $q = 1$, $m = 1$. Compressor de Paridade: essa estrutura é caracterizada pelo LFSR(1,1), ou seja, é utilizada em circuitos com uma única saída binária e a assinatura obtida tem 1 bit de comprimento a qual é simplesmente a paridade da resposta de saída.

- $q = 1$, $m > 1$. LFSR Convencional: essa estrutura é caracterizada pelo LFSR(1,m) e é utilizada em circuitos com uma única saída binária e a assinatura obtida tem m bits de comprimento, ou seja é um LFSR de m estágios.

- $q > 1$, $m = 1$. MISR Convencional: esse caso corresponde ao GLFSR em que dois casos se destacam:

    1. MLFSR. (**multi-entrada** LFSR): caracterizado por utilizar um polinômio primitivo como polinômio de realimentação. Esse de grau m em $GF(2^q)$, desta forma constituindo-se de um LFSR de m estágios e q entradas usado para circuitos com mais de uma saída.

    2. Múltiplo MISR. Caracterizado quando o polinômio de realimentação não for primitivo, entretanto, esse é redutível e fatorável em m fatores lineares, tal como, $(x-a_1)\cdots(x-a_m)$. Desta forma, o Múltiplo MISR é equivalente a usar m MISR's com polinômios de realimentação $(x-a_1), \cdots, (x-a_m)$ respectivamente.

[1] percentagem de área do circuito integrado que efetivamente não faz parte da área que operacionaliza a função principal desse.

B.2 Modelamento de Compressores utilizando a Teoria de Codificação

Nesta seção, aplica-se a teoria de codificação para o modelamento matemático do analisador de assinatura baseado em LFSR(q,m). Usando esse modelo, todos os casos especiais, discutidos previamente, podem ser tratados uniformemente.

1. Códigos Algébricos (síntese).

   Tem-se que C seja um código (n,k) em $GF(2)$ e que a n-úpla $c = c_{n-1} \cdots c_1 c_0$ seja uma palavra código de C em que $c_i \in GF(2)$ para $0 \leq i < n$. A representação polinomial dessa n-úpla é dada por $c(x) = c_{n-1}x^{n-1} + \cdots + c_1 x^1 + c_0$.

   Definição 1: o polinômio $g(x)$ é chamado polinômio gerador do código C se qualquer palavra código em C for divisível por $g(x)$. C é denominado de código(n,k) em que o grau de $g(x)$ é igual a k.

   Definição 2: um código (n, k) C é cíclico se qualquer deslocamento cíclico de uma palavra código pertencente a C resulta em uma palavra código em C.

   Teorema 1: se $g(x)$ é um polinômio de grau k e divide x^{n-1}, então $g(x)$ gera um código cíclico.

   Todas as palavras código em um código C são múltiplas de seu polinômio gerador $g(x)$. Essa propriedade é usada no processo de decodificação utilizado para esse código. Tem-se que $r(x)$ é um polinômio de grau n em $GF(2)$.

   Definição 3: o resto $s(x)$, obtido pela divisão de $r(x)$ por $g(x)$, é denominado como a **síndrome** de $r(x)$. Isto é:

   $$r(x) = a(x)g(x) + s(x)$$

   onde $s(x)$ é o resto da divisão polinomial e tem grau menor que o grau de $g(x)$.

   Para calcular $s(x)$, serialmente, de um código gerado por $g(x)$ pode-se usar um LFSR com polinômio de realimentação igual a $g(x)$. O seguinte lema é uma conseqüência

direta da relação acima e mantém-se mesmo quando o código não for cíclico.

Lema 1: tem-se que $r(x)$ e $r^*(x)$ são dois vetores diferentes em $GF(2)$. Então ambos $r(x)$ e $r^*(x)$ produzem a mesma síndrome, em relação a $g(x)$, se e somente se a soma $r(x) + r^*(x)$ for divisível por $g(x)$, ou seja, essa soma é uma palavra código gerada por $g(x)$.

2. Condição para *aliasing* em LFSR.

   Considerando-se que $R(x)$ e $R^*(x)$ são as representações polinomiais das saídas do circuito e do circuito considerado sem falha, respectivamente, então *aliasing* ocorre se e somente se o polinômio de erro $E(x) = R(x) + R^*(x)$ for divisível pelo polinômio de realimentação do LFSR, $\Phi(x)$.

   Por outro lado, foi mostrado que quaisquer dois vetores $r(x)$ e $r^*(x)$ produzem a mesma síndrome se e somente se a soma $r(x) + r^*(x)$ for divisível pelo polinômio gerador $g(x)$. Com base nisso, uma analogia entre análise de assinatura e teoria de codificação é mostrada na Tabela B.1.

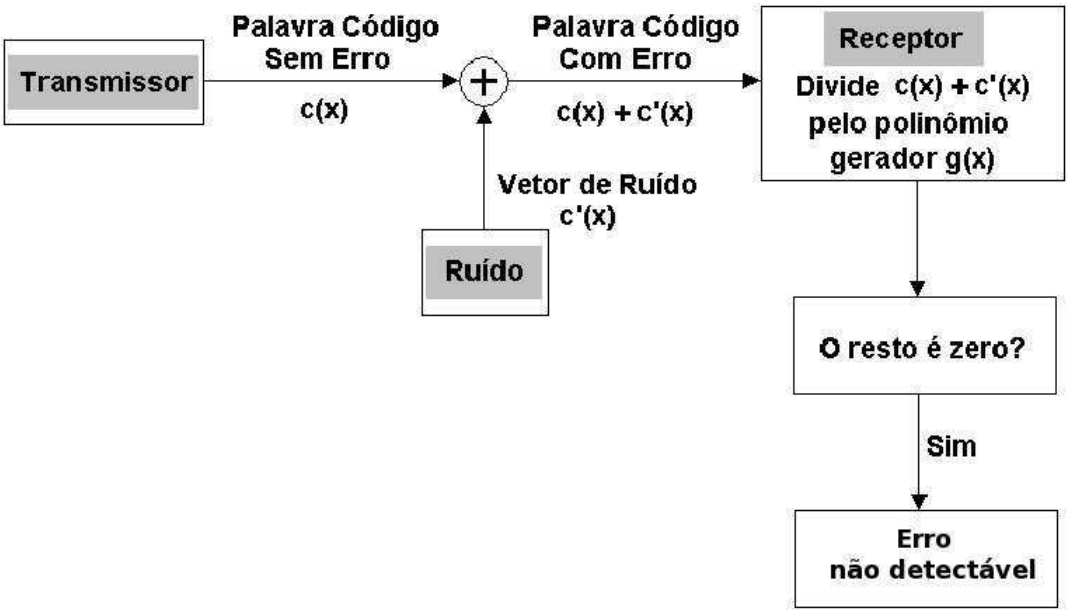


Figura B.2: Esquema de transmissão-recepção utilizando código.

Essa equivalência é oriunda das seguintes observações: primeiro, considerando-se um transmissor e um receptor ligados por um canal ruidoso, como na Figura B.2, sendo

Tabela B.1: Equivalência entre análise de assinatura e teoria da codificação.

Compressão por LFSR	Teoria da Codificação
Resposta de Saída do Circuito	Vetor Recebido pelo Receptor
LFSR	Gerador de Síndromes
Assinatura	Síndrome
Aliasing	Erro Indetectável
Probabilidade de *Aliasing*	probabilidade de Erro Indetectável

que o receptor é, simplesmente, um detector de erro, pode ser visto que o vetor de erro adicionado à palavra código não é detectado pelo receptor somente se esse vetor de erro for também uma palavra código. De maneira a testar se o vetor recebido é uma palavra código, o circuito de detecção de erro pode ser projetado para dividir o vetor recebido pelo polinômio gerador do código. Se o resto dessa divisão for igual a zero assume-se que o vetor recebido é uma palavra código. Então, a probabilidade de erro indetectável é igual, precisamente, à probabilidade de que o vetor de erro seja uma palavra código.

Por outro lado, considerando-se que o analisador de assinatura seja um LFSR, *aliasing* ocorre quando o vetor de erro, no qual é definido como a soma da resposta do circuito sem falha, $R^*(x)$ e da resposta do circuito sob teste, $R(x)$, seja divisível pelo polinômio de realimentação $\Phi(x)$ do LFSR. Usando a analogia do canal de comunicação vista na Figura B.2, pode-se declarar que aliasing ocorre quando o vetor de erro corresponde a um palavra código do código gerado por $\Phi(x)$ do analisador de assinatura. Deste modo, a probabilidade de *aliasing* é igual, precisamente, à probabilidade de erro indetectável no esquema do canal de comunicação equivalente.

Apêndice C

Operadores Genéticos Utilizados

Para a otimização genética proposta no Capítulo 4, foram utilizados o operador de cruzamento parcialmente mapeado e o operador de mutação por troca descritos a seguir.

C.1 Operador de Cruzamento Parcialmente Mapeado (PMX)

O operador PMX foi sugerido por [57]. Esse operador passa informações de valores e de ordem dos **pais** para os seus **descendentes** e consiste no seguinte procedimento: uma porção de um dos **pais** é mapeada no outro **pai** e os dados restantes são trocados. Por exemplo, abaixo tem-se dois **pais**:

$$(1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8)$$
$$(3\ \ 7\ \ 5\ \ 1\ \ 6\ \ 8\ \ 2\ \ 4)$$

O operador PMX cria um **descendente** da seguinte maneira: primeiro, dois pontos de corte são selecionados aleatoriamente. Suponha que o primeiro ponto de corte está entre o terceiro e o quarto elemento e o segundo está entre o sexto e o sétimo, como abaixo.

$$(1\ \ 2\ \ 3\ |\ 4\ \ 5\ \ 6\ |\ 7\ \ 8)$$
$$(3\ \ 7\ \ 5\ |\ 1\ \ 6\ \ 8\ |\ 2\ \ 4)$$

Os elementos entre os pontos de cortes são chamados de elementos mapeadores e indica o seguinte **mapeamento**, $4 \leftrightarrow 1$, $5 \leftrightarrow 6$, $6 \leftrightarrow 8$. Neste ponto, os elementos mapeadores do primeiro **pai** são copiados para o segundo **descendente** e os elementos mapeadores do segundo **pai** são copiados para o primeiro **descendente**, tornando-se:

$$\begin{aligned} \textit{Descendente}\ 1: &\quad (x\ \ x\ \ x \mid 1\ \ 6\ \ 8 \mid x\ \ x) \\ \textit{Descendente}\ 2: &\quad (x\ \ x\ \ x \mid 4\ \ 5\ \ 6 \mid x\ \ x) \end{aligned}$$

Então o **descendente** i $(i = 1, 2)$ é completado copiando-se os elementos do i-ésimo **pai**. No caso que um elemento já estiver presente no **descendente**, esse é trocado de acordo com o mapeamento. Por exemplo, o primeiro elemento do **descendente** 1 deveria ser 1 como o primeiro elemento do primeiro **pai**. Entretanto, já há um 1 presente no **descendente** 1. Então, devido ao mapeamento $1 \leftrightarrow 4$, o primeiro elemento do **descendente** 1 será 4. O segundo, o terceiro e o sétimo elementos são derivados do primeiro pai. Entretanto, o último elemento do **descendente** 1 deveria ser 8, que já está presente, então devido ao mapeamento $8 \leftrightarrow 6$ e $6 \leftrightarrow 5$ torna-se 5. Deste modo, tem-se:

$$\textit{Descendente}\ 1: \quad (4\ \ 2\ \ 3 \mid 1\ \ 6\ \ 8 \mid 7\ \ 5)$$

Analogamente, é encontrado

$$\textit{Descendente}\ 2: \quad (3\ \ 7\ \ 8 \mid 4\ \ 5\ \ 6 \mid 2\ \ 1)$$

Nota-se que, no operador PMX, algumas posições absolutas de alguns elementos de ambos os **pais** são preservadas. Um exemplo do uso do operador PMX é visto na Figura C.1.

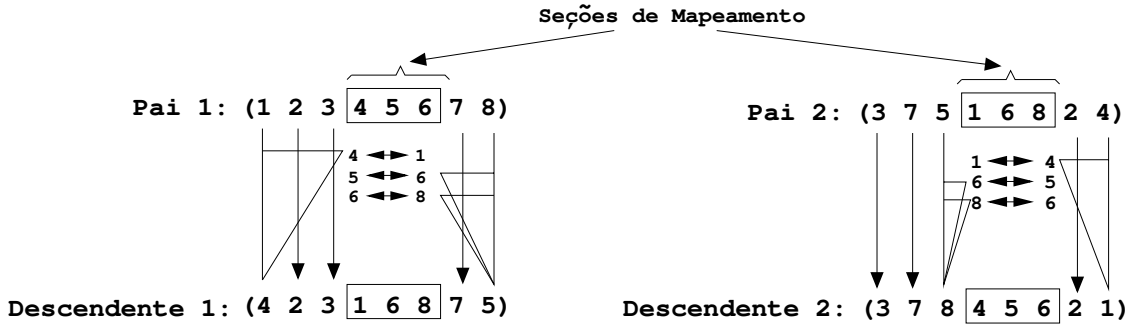


Figura C.1: Cruzamento parcialmente mapeado.

C.2 Operador de Mutação por Troca (EM)

O operador de mutação por troca [58] seleciona aleatoriamente dois elementos em um indivíduo e os troca de posição. Por exemplo, considere o indivíduo (1 2 3 4 5 6 7 8) e suponha que a troca entre o terceiro e o quinto elementos, então o resultado na mutação será então (1 2 5 4 3 6 7 8), como pode ser vista na Figura C.2.

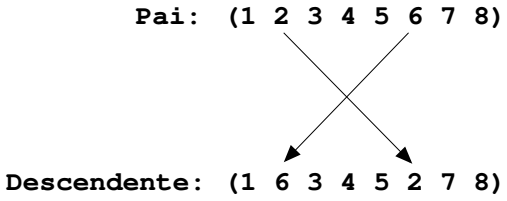


Figura C.2: Mutação por troca

Apêndice D

Campos Finitos

Neste apêndice, uma introdução básica aos campos finitos, também denominados de campos de *Galois*, é descrita. A primeira seção deste apêndice descreve o mais elementar campo finito, o campo binário[36]. As demais seções descrevem formalmente as três estruturas básicas, chamadas grupo, anel e campo, a partir das quais toda a teoria algébrica em campos finitos é desenvolvida. Maiores detalhes e provas dos resultados apresentados aqui podem ser obtidos em Blahut [36, pp. 65–92].

D.1 Campo Binário

O campo finito mais elementar é o campo binário denominado por $GF(2)$ e é formado por somente dois elementos que podem ser denotados pelo conjunto $\{0, 1\}$. Tem-se que, nesse campo, as operações de adição e de multiplicação são definidas como vista nas Tabelas D.1 e D.2, respectivamente. As operações de adição e de multiplicação definidas são denominadas de *adição módulo 2* e de *multiplicação módulo 2*.

Tabela D.1: Adição módulo 2.

+	0	1
0	0	1
1	1	0

Tabela D.2: Multiplicação módulo 2.

·	0	1
0	0	0
1	0	1

Em síntese, os dois elementos, 0 e 1, juntamente com as operações *adição módulo 2* e de *multiplicação módulo 2* é denominado de campo binário.

É importante ressaltar que a operação de *adição módulo 2* em $GF(2)$ pode ser implementada pela operação lógica *OU-EXCLUSIVA* e que a operação de *multiplicação módulo 2* pode ser implementada pela operação lógica *E*. Fato esse, que torna o campo binário amplamente utilizado devido a sua fácil implementação.

Nas próximas seções são descritos formalmente as três estruturas básicas, chamadas grupo, anel e campo, nas teoria de campos finitos.

D.2 Grupos

Um *grupo* G é um conjunto associado a uma operação (denotada por "+") sobre pares de elementos deste conjunto que satisfaz as quatro propriedades seguintes:

1. *Fechamento* $\rightarrow$ Para todo $a, b \in G$, $c = a + b \in G$;
2. *Associatividade* $\rightarrow$ Para todo $a, b, c \in G$, $a + (b + c) = (a + b) + c$;
3. *Identidade* $\rightarrow$ Existe um elemento e chamado identidade, em que $a + e = e + a = a$;
4. *Inversas* $\rightarrow$ Se $a \in G$, então existe $b \in G$ chamado inversa de a, em que $a + b = b + a = e$;

Os grupos com número finito de elementos são ditos *grupos finitos*, e seu número de elementos é chamado de *ordem do grupo*. A *ordem do elemento* $a \in G$ é o menor valor n, tal que

$$\underbrace{a + a + \cdots + a}_{n \text{ vezes}} = e \Rightarrow \underbrace{a + a + \cdots + a}_{n+1 \text{ vezes}} = a.$$

A ordem de um elemento sempre divide a ordem do grupo.

O grupo G que apresenta a propriedade de comutatividade, na qual, para $a, b \in G$, $a + b = b + a$, é dito *grupo comutativo* ou *grupo abeliano*.

Não se deve confundir a operação "+" com a adição convencional. Esta operação sobre os elementos do grupo pode ser definida de diferentes formas, desde que satisfaça as propriedades descritas.

Em todo grupo, o elemento identidade é único. Além disso, a inversa a^{-1} de cada elemento a do grupo é única, e $(a^{-1})^{-1} = a$.

D.3 Anéis

Um *anel* R é um conjunto com duas operações, *adição* ("+") e *multiplicação* (justaposição), que satisfaz os seguintes axiomas:

1. R é um grupo abeliano sob a adição;
2. *Fechamento* $\rightarrow$ Para todo $a, b \in R$, o produto $ab \in R$;
3. *Associatividade* $\rightarrow$ Para todo $a, b, c \in R$, $a\,(bc) = (ab)\,c$;
4. *Distributividade* $\rightarrow$ $a\,(b + c) = ab + ac$ e $(b + c)\,a = ba + ca$, para todo $a, b, c \in R$;

A adição em um anel é sempre comutativa. Um *anel comutativo* é aquele em que a multiplicação é comutativa, ou seja, em que $ab = ba$ para todo $a, b \in R$.

Um anel não necessariamente possui identidade na multiplicação, assim como inversas de seus elementos. Caso um anel tenha identidade, esta identidade é única. Além disso, se $ab = 1$ e $ca = 1$, então $b = c$, e a é dito ter uma única inversa denotada por a^{-1}.

Um elemento que possua inversa em um anel é dito uma *unidade*.

D.4 Campos

Um *campo* F é um conjunto que possui duas operação definidas sobre seus elementos, adição e multiplicação, e que satisfaz os seguintes axiomas:

1. É um grupo abeliano sob a adição;
2. É fechado sob a multiplicação, e $F - \{0\}$ é um grupo abeliano sob a multiplicação;
3. Para todo $a, b, c \in F$, $(a + b)\, c = ac + bc$;

Convenciona-se denotar por 0 a identidade sob a adição, por $-a$ a inversa aditiva de a, por 1 a identidade sob a multiplicação, e por a^{-1} a inversa multiplicativa de a. Entende-se por subtração $(a - b) = a + (-b)$ e por divisão $\left(\frac{a}{b}\right) = b^{-1}a$.

São exemplos de campos os conjuntos $\mathbb{R}$ (números reais), $\mathbb{C}$ (números complexos) e $\mathbb{Q}$ (números racionais). Um campo com número finito de elementos é chamado *campo finito*, ou *campo de Galois*, e denotado por $\mathbb{F}_q$ (campo com q elementos).

Um subconjunto de um campo F é dito *subcampo* de F se constituir um campo sob as operações inerentes a F. O campo F é dito uma *extensão* deste subcampo.

Um campo comporta-se como um anel, que permite a divisão ou cancelamento. Em um campo, se $ab = ac$ e $a \neq 0$, então $a^{-1}ab = a^{-1}ac \Rightarrow b = c$. Existem alguns anéis, como o anel dos inteiros, que permitem o cancelamento, mesmo não sendo campos, ou seja, mesmo não existindo a inversa na multiplicação para todos os seus elementos. Um anel comutativo em que $b = c$ sempre que $ab = ac$, com $a \neq 0$, é chamado de *domínio integral*.

D.5 Campos finitos baseados em anéis de inteiros

O conjunto $\mathbb{Z}$ dos inteiros forma um domínio integral sob as operações de adição e multiplicação usuais, sendo denotado por $\mathbf{Z}$.

Um inteiro s é dito *divisível* pelo inteiro r, ou de modo igual r *divide* s, se $ra = s$ para algum inteiro a. Um inteiro p é dito *primo* se for divisível apenas por $\pm p$ ou ± 1. O *máximo divisor comum* $MDC\,(r, s)$ de dois inteiros r e s é o maior inteiro positivo que divide ambos r e s. O *mínimo múltiplo comum* $MMC\,(r, s)$ de dois inteiros r e s é o menor inteiro positivo que é divisível por ambos r e s.

Em geral, a divisão não é possível em um anel. Pode-se, entretanto, definir uma divisão com resto e um cancelamento (resto igual a zero), razão pela qual o anel de inteiros constitui um domínio integral. O chamado *algoritmo da divisão* estabelece que, para todo par de

inteiros c e d, com $d \neq 0$, existe um único par de inteiros Q (quociente) e s (resto), tal que $c = dQ + s$, em que $0 \leq s < |d|$. O resto s também pode ser escrito como $s = [c]_d$. Outra expressão comum é a de *congruência* $s \equiv c \mod d$. Dizer que s é congruente a c módulo d significa que s e c possuem o mesmo resto na divisão por d, mas s não necessariamente é menor que d.

Considere q um inteiro positivo. O anel dos inteiros módulo q, denotado por $\mathbf{Z}/q$, é o conjunto $\{0, \ldots, q-1\}$ associado à adição e à multiplicação definidos por $a + b = [a + b]_q$ e $ab = [ab]_q$.

Quando q for um inteiro primo, o anel $\mathbf{Z}/q$ constituirá um campo, sendo denotado por $\mathbb{F}_q$. Portanto, tomando-se q inteiro primo, pode-se obter um campo $\mathbb{F}_q$ a partir do anel de inteiros $\mathbf{Z}$ associado à operação de módulo q.

D.6 Campos finitos baseados em anéis de polinômios

Um polinômio em $\mathbb{F}_q$ consiste numa expressão da forma

$$f(x) = f_{n-1}x^{n-1} + f_{n-2}x^{n-2} + \cdots + f_1 x + f_0,$$

em que x é uma variável e $f_{n-1}, \ldots, f_0 \in \mathbb{F}_q$. O polinômio nulo é $f(x) = 0$. Um *polinômio mônico* é um polinômio no qual o coeficiente f_{n-1} é igual a 1. O *grau* $\deg f(x)$ de um polinômio não nulo $f(x)$ é o índice do coeficiente f_{n-1}.

O conjunto de todos os polinômios em $\mathbb{F}_q$ associado à adição e à multiplicação de polinômios (com adição e multiplicação dos coeficientes em $\mathbb{F}_q$) constitui um anel, denotado por $\mathbb{F}_q[x]$.

Um polinômio $s(x)$ é *divisível* por $r(x)$, ou $r(x)$ *divide* $s(x)$, se existir um polinômio $a(x)$, tal que $s(x) = r(x)\,a(x)$. Um polinômio $p(x)$ é dito *irredutível* se for divisível apenas por $\alpha p(x)$ ou α, em que $\alpha \in \mathbb{F}_q$. Um polinômio mônico irredutível não nulo é dito um *polinômio primo*. O *máximo divisor comum* $MDC[r(x), s(x)]$ é o polinômio mônico de maior grau que divide ambos $r(x)$ e $s(x)$. O *mínimo múltiplo comum* $MMC[r(x), s(x)]$ é o polinômio mônico de menor grau que é divisível por ambos $r(x)$ e $s(x)$.

O chamado *algoritmo da divisão de polinômios* determina que, para todo par de polinômios $c(x)$ e $d(x)$, com $d(x) \neq 0$, existe um único par de polinômios $Q(x)$ (quociente) e $s(x)$ (resto), tal que $c(x) = d(x)\,Q(x) + s(x)$, em que $\deg s(x) < \deg d(x)$. O resto $s(x)$ também pode ser escrito como $s(x) = [c(x)]_{d(x)}$. Assim como no caso dos anéis de inteiros, a congruência $s(x) \equiv c(x) \mod d(x)$ indica que $s(x)$ e $c(x)$ possuem o mesmo resto na divisão por $d(x)$.

Assim como em certos casos é útil expressar inteiros como produto de inteiros primos (fatoração), também o é no caso de polinômios. Um polinômio não nulo $p(x)$ em um campo $\mathbb{F}_q$ possui uma fatoração única (exceto pela ordem dos fatores) em um produto de um escalar (elemento de $\mathbb{F}_q$) e de polinômios primos em $\mathbb{F}_q$.

Um polinômio em $\mathbb{F}_q$ pode ser avaliado em qualquer elemento β de $\mathbb{F}_q$ substituindo-se a variável x por β. Um elemento β é um zero de ordem m de um polinômio $p(x)$ se e só se $(x-\beta)^m$ dividir $p(x)$ e $(x-\beta)^{m+1}$ não dividi-lo. Além disso, um polinômio $p(x)$ de grau n possui no máximo n zeros.

Para qualquer polinômio mônico $p(x)$ em $\mathbb{F}_q$ de grau não nulo, o *anel de polinômios módulo* $p(x)$ é o conjunto de todos os polinômios com grau menor que $\deg p(x)$, associado à adição e multiplicação de polinômios módulo $p(x)$. Este anel é denotado por $\mathbb{F}_q[x]/p(x)$. Quando $p(x)$ for um polinômio primo de grau n, o anel $\mathbb{F}_q[x]/p(x)$ será um campo com q^n elementos, chamado *campo de Galois* e denotado por $\mathbb{F}_{q^n}$.

Um *elemento primitivo* α do campo $\mathbb{F}_q$ é tal que todo elemento não nulo de $\mathbb{F}_q$ pode ser expresso como potência de α. Os elementos primitivos são úteis na construção de campos, já que, conhecido um elemento primitivo, é possível construir-se todo o campo e sua tabela de multiplicação através das potências deste elemento (pode-se mostrar que todo campo de Galois possui um elemento primitivo).

Um *polinômio primitivo* $p(x)$ em $\mathbb{F}_q$ consiste num polinômio primo em $\mathbb{F}_q$, tal que o elemento representado por x é primitivo no campo extensão construído módulo $p(x)$. Existem polinômios primitivos de todos os graus em todos os campos de Galois. Além disso, mostra-se que um elemento primitivo de um campo é um zero de qualquer polinômio primitivo neste campo. Um campo $\mathbb{F}_q$ é dito *fechado algebricamente* se todo polinômio $f(x) \in \mathbb{F}_q[x]$ de grau maior ou igual a 1 tiver raízes em $\mathbb{F}_q$. Qualquer campo $\mathbb{F}_q$ possui uma extensão $\overline{\mathbb{F}_q}$ fechada algebricamente. Esta extensão $\overline{\mathbb{F}_q}$ é dita o *fecho algébrico* de $\mathbb{F}_q$.

Referências Bibliográficas

[1] Avizienis, A. Toward Systematic Design of Fault-Tolerant Systems. *IEEE Transactions on Computer*, v. 30, p. 51–58, Apr 1997.

[2] Ali, L.; Sidek, R.; Aris, I; Suparjo, B. S. e Ali, M. A. M. Challenges and directions for testing ic. *Integr. VLSI J.*, Elsevier Science Publishers B. V., v. 37, n. 1, p. 17–28, 2004. ISSN 0167-9260.

[3] Jalote, P. *Fault Tolerance in Distributed System*. Englewood Cliffs, New Jersey: Prentice Hall, Inc, 1994.

[4] Curry, R.J. Evaluation of economics of testing in a manufacturing environment. *In Proceedings of IEEE AUTOTESTCON*, p. 481 – 489, 2002.

[5] Bushnell, M. L. e Agrawal, V. D. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. [S.l.]: Kluwer Academic Publishers, 2001.

[6] Gonciari, P.; Al-Hashimi, B. e Nicolici, N. Test cost reduction through compression. *Electronics Systems and Software*, v. 1, p. 37–41, June/July 2003.

[7] Zhang, S.; Choi, M.; Park, N. e Lombardi, F.;. Probabilistic balancing of fault coverage and test cost in combined built-in self-test/automated test equipment testing environment. *Proceedings of 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2004)*, p. 48 – 56, October 2004.

[8] Ungar, L.Y. e Ambler, T. Economics of built-in self-test. *IEEE Design & Test of Computers*, v. 18, n. 5, p. 70–79, Sept-Oct 2001.

[9] Jas, A.; Krishna, C.V. e Touba, N.A. Weighted pseudorandom hybrid BIST. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, p. 1277 – 1283, Dec 2004.

[10] Kalligeros, E.; Kavousianos, X. e Nikolos, D. Multiphase BIST: a new reseeding technique for high test-data compression. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, p. 1429 – 1446, Oct 2004.

[11] Youhua Shi e Zhe Zhang. Multiple test set generation method for LFSR-based BIST. *Proceedings of the Design Automation Conference (ASP-DAC)*, p. 863 – 868, Jan 2003.

[12] Dufaza, C. Theoretical properties of LFSRs for built-in self-test. *INTEGRATION, the VLSI journal*, p. 17–35, 1998.

[13] Gupta, S.K., Pradhan, D.K. e Reddy, S.M. Zero Aliasing Compression. *Fault Tolerant Computing Symposium*, p. 254–263, 1990.

[14] Pradhan, D.K., Gupta, S.K. A New Framework for Designing and Analyzing BIST Techniques and Zero Aliasing Compression. *IEEE Transactions on Computers*, v. 40, n. 6, p. 743 –763, June 1991.

[15] Chakrabarty, K. e Murray, B. T. Optimal Zero-Aliasing Space Compaction of Test Responses. *IEEE Transactions on Computers*, v. 47, n. 11, November 1998.

[16] Abramovici, M; Breuer, M. A. e Friedmann, A. D. *Digital System Testing and Testable Design*. [S.l.]: IEEE Press/ W.H.Freeman, 1990.

[17] Kavousianos, X.; Bakalis, D.; Nikolos, D. e Tragoudas, S. A new built-in TPG method for circuits with random pattern resistant faults. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 21, n. 7, p. 859–866, Jul 2002.

[18] Hiraide, T.; Boateng, K. O.; Konishi, H.; Itaya, K.; Emori, M.; Yamanaka, H. e Mochiyama, T. BIST-aided scan test - a new method for test cost reduction. *Proceedings of 21st VLSI Test Symposium*, p. 359 – 364, April - May 2003.

[19] Brglez, F. e Fujiwara, H. A Neutral List of 10 Combinatorial Benchmark Circuits and a Target Translator in Fortran. *Proceedings of International Symposium on Circuit and Systems*, p. 663–698, June 1985.

[20] Brglez, F., Bryan, D. e Kozminski, K. Combinational profiles of sequential benchmark circuits. *In Proceedings of the IEEE International Symposium on Circuits and Systems*, p. 1929–1934, 1989.

[21] Malaiya, Y.K. e Yang, S. A Coverage Problem for Random Testing. *Proc. IEEE International Test Conference*, p. 237–245, Nov 1984.

[22] Koenemann, B., Mucha, J. e Zwiehoff, G. Built-in test for complex digital integrated circuits. *IEEE J. Solid-State Circuits*, p. 315–318, 1980.

[23] MacCluskey, E. J. Verification testing - a pseudoexhaustive test technique. *IEEE Transactions on Computers*, June 1984.

[24] Dislis, C.; Ambler, A.P.; Dear, I.D. e Dick, J.H. Economics in design and test. *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, p. 384 – 387, October 1993.

[25] Schulz, M., Trischler, E. e Sarfert, T. M. Socrates: a Highly Efficient Automatic Test Pattern Generation System. *IEEE Transation on Computer-Aided Design*, v. 7, n. 1, p. 126–137, January 1988.

[26] Hellebrand, S., Reeb, B., Tarnick S. e Wunderlich, H. J. Pattern Generation for a Deterministic BIST Scheme. *ACM/IEEE International Conference on CAD95, ICCAD-95*, San Jose, Ca, p. 1–7, November 1995.

[27] Pomeranz, I., Reddy, L. N. e Reddy, S. M. Compactest: a Method to Generate Compact test Sets for Combinational Circuits. *IEEE International Test Conference*, p. 194–203, 1991.

[28] Novak, O. e Nosek, J. On Using Deterministic Test Sets in BIST. *6th IEEE International On-Line Testing Workshop (IOLTW)*, p. 127–132, July 2000.

[29] Karkala, M., Touba, M. A. e Wunderlich, H. J. Special ATPG to correlate test patterns for low-overhead mixed-mode BIST. *7th. Asian Test Symposium*, Singapore, p. 492–499, December 1998.

[30] Vlasov, P.A. Cellular Automata, Tendencies and Results. *RNNS/IEEE Symposium on Neuroinformatics and Neurocomputers*, v. 2, p. 1227–1230, Outubro 1992.

[31] Assis, F. M. e Pedreira, C. E. An architecture for computing Zech's logarithms in GF(2^m). *IEEE Transactions on Computers*, v. 49, n. 5, p. 519–524, 2000.

[32] Mastrovito, E. D. *VLSI Architecture for Computations in Galois Fields*. [S.l.]: Linkoping studies in science and technology. Dissertation, 1991.

[33] H.K. Lee e D.S. Ha. Atalanta: an efficient ATPG for combinational circuits. *Technical Report, 93-12, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia*, 1993.

[34] Karkala, M., Touba, N. A. e Wunderlich, H. J. Special ATPG to Correlate Test Patterns for Low-Overhead Mixed-Mode BIST. *7th. Asian Test Symposium*, v. 32, p. 462–469, December 1998.

[35] Souza, C. P., Freire, R. C. S. e Assis, F. M. Mixed-Mode BIST Based on LFSR Synthesized Using Berlekamp-Massey Algorithm and Aided by Genetic Optimization. *Digest of papers of the 6th IEEE Latin-American Test Workshop*, p. 337–341, Março 2005.

[36] Blahut, R. E. *Theory and Practice of Error Control Codes*. Owego, NY: Addison-Wesley Publishing Company, Inc., 1983.

[37] Massey, J. L. Shift-Register Synthesis and BCH Decoding. *IEEE Transactions on Information Theory*, v. 15, n. 1, p. 122–127, January 1969.

[38] Gustavson, F. G. Analysis of the Berlekamp-Massey linear feedback shift-register synthesis algorithm. *IBM J. Res. Dev.*, p. 204–212, 1976.

[39] Vardy, A. (editor). *Codes, Curves, and Signals, Common Threads in Communications*. Erewhon, NC: Kluwer Academic Publishers, 1998.

[40] Souza, C. P., Freire, R. C. S. e Assis, F. M. Usando o Algoritmo de Berlekamp-Massey em Testes de Circuitos Integrados. *Anais do XXII Simpósio Brasileiro de Telecomunicações (SBrT'05)*, p. 794–799, Setembro 2005.

[41] Whitley, D. A Genetic Algorithm Tutorial. *Statistics and Computing*, v. 4, p. 65–85, 1994.

[42] Larranaga, P. Kuijpers, C. M. H. Murga, R. H. e Dizdarevi, S. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review 13*, p. 129–170, 1999.

[43] H. K. Lee e D. S. Ha. An efficient forward fault simulation algorithm based on the parallel pattern single fault propagation. *In Proceedings of International Test Conference*, p. 946–955, October 1991.

[44] Huang, L. R. ; Jou, J.Y. e Kuo, S.Y. Gauss-elimination-based generation of multiple seed-polynomial pairs for LFSR. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 16, n. 9, p. 1015–1024, September 1997.

[45] Chiusano, S.; Prinetto, P. e Wunderlich, H. J. Non-intrusive BIST for systems-on-a-chip. *Proceedings of the International Test Conference*, p. 644– 651, 2000.

[46] Dasgupta, D. e Attoh-Okine, N. Immunity-Based System: a Survey. *IEEE International Conference on Computational Cybernetics and Simulation*, p. 369–374, 1997.

[47] Forrest, S., Perelson, A. S., Allen, L e Cherukuri, R. Self-Nonself Discrimination in a Computer. *Proceedings of IEEE Symposium on Research in Security and Privacy*, Oakland, CA, p. 202–212, May 1994.

[48] D'haeseleer, Forrest, S. e Helman, P. An Immunological Approach to Change Detection: Algorithms, Analysis, and Implication. *Proceedings of IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1996.

[49] Helman, P. e Forrest, S. An Efficient Algorithm for Generating Random Antibody Strings. *Technical Report No. CS94-7, Department of Computer Science*, University of New Mexico, 1994.

[50] Bradley, D.W.; Tyrrell, A.M. Immunotronics - Novel Finite-State-Machine Architectures with Built-In Self-Test Using Self-Nonself Differentiation. *IEEE Transactions on Evolutionary Computation*, v. 6, p. 227–238, Jun 2002.

[51] Souza, C. P., Freire, R. C. S. e Assis, F. M. Testing System-on-a-Chip using Artificial Immune System. *Proceedings of IEEE International SOC Conference*, p. 61–64, Setembro 2005.

[52] Souza, C. P., Freire, R. C. S. e Assis, F. M. Testing System-On-a-Chip using Artificial Immune System. *Anais do VII Congresso Brasileiro de Redes Neurais*, p. 205.1–205.3, Outubro 2005.

[53] Souza, C. P., Freire, R. C. S. e Assis, F. M. A BIST scheme Based on Self-Nonself Discrimination of Immune System. *Proceedings of IEEE Workshop on Machine Learning for Signal Processing*, p. 765–774, Outubro 2004.

[54] Bhattacharya, B.B., Dmitriev, A. e Goessel, M. Zero-aliasing space compression using a single periodic output and its application to testing of embedded cores. IEEE Computer Society, Washington, DC, USA, p. 382–388, 2000.

[55] Costa Branco, P. J., Dente, J. A., e Vilela Mendes, R. Using Immunology Principles for Fault Detection. *IEEE Transation on Industrial Electronics*, v. 30, n. 2, p. 302–375, 2003.

[56] Martins, J. F., Costa Branco, P. J., Pires, A. J. e Dente, J. A.,. Fault Detection Using Immune-Based Systems and Formal Language algorithms. *Proceedings of the* 39^{th} *IEEE Conference on Decision and Control*, Sydney, Australia, December 2000.

[57] Goldberg, D. E. e Lingle, Jr. R. Alleles, loci and the tsp. *Proceedings of The First International Conference on Genetic Algorithms and Their Application*, Hillsdale, New Jersey, p. 154–159, 1985.

[58] Banzhaf, W. The molecular travelling salesman. *Biological Cybernetics 64*, p. 7–14, 1990.

[59] Souza, C. P., Freire, R. C. S. e Assis, F. M. A Test Response Analyzer Based on Berlekamp-Massey Algorithm for BIST. *Procedings of 5th IEEE Latin-American Test Workshop*, p. 88–93, Março 2004.

Índice Remissivo

At-speed testing, 37

Algoritmo de Berlekamp-Massey, 46
    Modificado, 54
Algoritmo de seleção negativa, 78
    Critérios de casamento, 81
    Fase de Geração, 78
    Fase de monitoramento de erro, 79
Algoritmo genético, 47
    Cruzamento parcialmente mapeado, 64
    Indivíduo, 64
    Mutação por troca, 64
    Representação por percurso, 64
algoritmo genético, 63
Análise de assinatura
    Aliasing, 25, 88
Analisador imunológico, 88
ATALANTA, 67

BIST, 24

Campos finitos, 108
    Adição módulo 2, 108
    Anéis, 110
    Campo binário, 108
    Campos, 110
    Campos de Galois, 108
    Campos finitos baseados em anéis de inteiros, 111
    Campos finitos baseados em anéis de polinômios, 112
    Grupos, 109
    Multiplicação módulo 2, 108
Circuito integrado autotestável, 24
    Vantagens, 26
Circuito sob teste, 24, 86
Cobertura de Falhas, 26
Compactação de hipercubos de testes, 57
    Consistência, 57
    Suporte, 57
Critério de casamento
    r-contínuos, 83
    r-Hamming, 83, 84
Custo de teste, 23

Discriminação próprio/não-próprio, 77

Equipamento de teste automático, 23

Falhas *stuck-at*, 30, 33, 62
Falhas de difícil detecção, 45
FSIA, 91
FSIM, 67

Gerador de testes, 24, 86

Baseados em fixação de bits, 37
Baseados em reinicialização de LFSR, 37, 71
Determinísticos, 25, 36, 42, 45
Exaustivos, 34
Mistos, 36, 45
Ponderados, 37
Pseudo-aleatórios, 25, 35, 38, 45

Hipercubos de teste, 42, 53

Identificação de falhas de difícil detecção, 68
ISCAS85, 67, 91
ISCAS89, 67

Kylix, 67

LFSR, 38
Polinômio de realimentação, 38
LFSRMAKER, 67

Operadores genéticos, 105
Operador de cruzamento parcialmente mapeado (PMX), 105
Operador de mutação por troca (EM), 107

Sobreárea de *hardware*, 26
Cálculo da sobreárea de *hardware*, 70
Método da equivalência de portas, 70

Tempo de teste, 27
Testes, 23
Determinísticos
Compactação, 43
Pseudo-aleatórios, 25, 35–38, 45

Verificação de sobreposição, 59