UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANÁLISE DA PERFORMANCE DO ALGORITMO D

por

Edelweis Helena Ache Garcez Dornelles

Dissertação submetida como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Raul Fernando Weber
Orientador

Porto Alegre, julho de 93.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Dornelles, Edelweis Helena Ache Garcez

ANÁLISE DA PERFORMANCE DO ALGORITMO D / Edelweis Helena Ache Garcez Dornelles.—Porto Alegre: CPGCC da UFRGS, 93.
173 p.: il.

Dissertação (mestrado)—Universidade Federal do Rio Grande do Sul, Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, 93. Orientador: Weber, Raul Fernando

Dissertação: Sistemas Digitais, CAD para Sistemas Digitais
Algoritmo *D*, Geração de Testes, Testabilidade, Observabilidade, Controlabilidade

AGRADECIMENTOS

A minha filha, pela paciência de enfrentar longas tardes na UFRGS ou na escola de forma que eu pudesse terminar esta dissertação. Ao Leonardo, meu ex-marido, que me ajudou muito. Ao Assis, meu namorado, que me ajudou a revisar este texto dando excelentes sugestões que em muito enriqueceram esta dissertação.

Um agradecimento especial a minha mãe, que deixou a sua casa para vir ficar comigo nos momentos mais críticos para que eu pudesse me abstrair de uma série de problemas domésticos e me dedicar totalmente a este trabalho.

Ao meu orientador, Raul Fernando Weber, pelo apoio técnico e disponibilidade que nunca deixaram a desejar. Aos professores, colegas e funcionários que de alguma forma ajudaram na execução deste trabalho, em especial a minha banca e à professora Taisy Weber.

À CAPES pelo apoio financeiro e a todos que de alguma maneira contribuiram para que este trabalho fosse realizado.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ATPG	*Automatic Test Pattern Generation*
CAD	*Computer Aided Design*
CAMELOT	*Computer Aided MEasure for Logic Testability*
CC	*Combinacional Controllability*
CF	Cobertura de Falhas
CO	*Combinacional Observability*
CPU	*Central Processing Unit*
CR	*Controllability Reconvergence*
CTF	*Controllability Transfer Factor*
CY	*Controllability*
DTM	*Dynamic Testability Measure*
FD	Falhas Detectadas
FR	Falhas Redundantes
IC	*Integrated Circuit*
MR	*Maximum Reconvergence*
MOS	*Metal Oxide Semiconductor*
ME	Medidas Estruturais
MF	Medidas Funcionais
MP	Medidas Probabilísticas
MT	*Testability Measure*
MTFP	Medidas de Testabilidade Funcionais e Probabilísticas
NP	Não polinomial
OTF	*Observability Transfer Factor*
OY	*Observability*
PI	*Primary Input*
PO	*Primary Output*
PREDICT	*Probabilistic Estimation of Digital Circuit Testability*
ROM	*Read Only Memory*

SA	*Steam Assignments*
SC	*Sequential Controllability*
SCOAP	*Sandia Controllability and Observability Analysis Program*
SO	*Sequential Observability*
STF	*Static Testability Measure*
TF	Total de Falhas
TP	Técnica de Pré-implicação
ULA	Unidade Lógica e Aritmética
UUT	*Unit Under Test*
VHDL	*VHSIC Hardware Description Language*
VLSI	*Very Large Scale Integration*
XOR	*Exclusive OR*
XNOR	*Not Exclusive OR*

RESUMO

A geração de testes para circuitos combinacionais com fan-outs reconvergentes é um problema NP-completo. Com o rápido crescimento da complexidade dos circuitos fabricados, a geração de testes passou a ser um sério problema para a indústria de circuitos integrados. Muitos algoritmos de ATPG (*Automatic Test Pattern Generation*) baseados no algoritmo D, usam heurísticas para guiar o processo de tomada de decisão na propagação D e na justificação das constantes de forma a aumentar sua eficiência.

Existem heurísticas baseadas em medidas funcionais, estruturais e probabilísticas. Estas medidas são normalmente referidas como observabilidade e controlabilidade que fazem parte de um conceito mais geral, a testabilidade. As medidas que o algoritmo utiliza podem ser calculadas apenas uma vez, durante uma etapa de pré-processamento (medidas de testabilidade estáticas - STM's), ou dinamicamente, recalculando estas medidas durante o processamento sempre que elas forem necessárias (medidas de testabilidade dinâmicas – DTM's).

Para alguns circuitos, o uso de medidas dinâmicas ao invés de medidas estáticas diminui o número de *backtrackings* por vetor gerado. Apesar disto, o tempo total de CPU por vetor aumenta.

Assim, as DTM's só devem ser utilizadas quando as STM's não apresentam uma boa performance. Isto pode ser feito utilizando-se as medidas estáticas até um certo número de *backtrackings*. Se o padrão de teste não for encontrado, então medidas dinâmicas são utilizadas.

Entretanto, é necessário ainda buscar formas de melhorar o processo dinâmico, diminuindo o custo computacional. A proposta original do cálculo das DTM's apresenta algumas técnicas, baseadas em *selective tracing*, com o

objetivo de reduzir o custo computacional. Este trabalho analisa o uso combinado de heurísticas e propõe técnicas alternativas, na forma das heurísticas de recálculo parcial e recálculo de linhas não *free*, que visam minimizar o *overhead* do cálculo das DTM's.

É proposta ainda a técnica de Pré-implicação que transfere a complexidade do algoritmo para a memória. Isto é feito através de um pré-processamento que armazena informações necessárias para a geração de todos os vetores de teste. De outra forma estas informações teriam de ser calculadas na geração de cada um destes vetores.

A implementação do algoritmo D com as várias heurísticas permitiu a realização de um experimento prático. Isto possibilitou a análise quantitativa da performance do algoritmo D para vários tipos de circuitos e demonstrou a eficiência de uma das heurísticas propostas neste trabalho.

PALAVRAS-CHAVE: Algoritmo-D, Geração de Testes,Testabilidade, Controlabilidade, Observabilidade.

TITLE: "PERFORMANCE ANALYSIS OF *D*-ALGORITHM"

ABSTRACT

The test generation for combinacional circuits that contain reconvergence is a NP-complete problem. With the rapid increase in the complexity of the fabricated circuits, the generation of test patterns poses a serious problem to the IC industry. A number of existing ATPG algorithms based on the *D* algorithm use heuristics to guide the decision process in the *D*-propagation and justification to improve the efficiency.

The heuristics used by ATPG algorithm are based on structural, functional and probabilistics measures. These measures are commonly referred to as line controllability and observability and they are combined under the more general notion of testability. The measures used by ATPG algorithms can be computed only once, during a preprocessing stage (static testability measures - STM's) or can be calculated dinamically, updating the testability measures during the test generation process (dymanic testability measures - DTM's).

For some circuits, replacing STM's by DTM's decreases the average number of backtrackings per generated vector. Despite these decrease, the total CPU time per generated vector is greater when using DTM's instead of STM's.

So, DTM's only must be used if the STM's don't present a good performance. This can be done by STM's until a certain number of backtrackings. If a test pattern has still not been found, then DTM's are used.

Therefore, it is yet necessary to search for ways to improve the dynamic process and decrease the CPU time requirements. In the original approach some techniques for reducing the computational overhead of DTM's based on the well-know technique of selective path tracing are presented.

In this work, the combined use of heuristics are analised and alternative techniques – the heuristics of partial recalculus and not free lines recalculus – are proposed. These alternative techniques were developed in order to minimize the overhead ot the DTM's calculus.

It is yet proposed the pre-implication technique which transfers to memory the algorithm complexity. It includes a preprocessing stage which storages all necesary informations to the generation of all test vectors. So, these informations don't need be computed in the generation of each test vector.

The implementation of the D-Algorithm with diferent heuristics has possibilited a practical experiment. It was possible to analise the performance of the D-Algorithm on diferent circuit types and to demonstrate the efficiency of one of the proposed heuristics.

KEYWORDS: *D*-algorithm, Testability, Controllability, Observability, Test generation.

1 INTRODUÇÃO

A tarefa de gerar testes e/ou diagnosticar falhas em circuitos digitais ainda é uma tarefa que despende tempo excessivo, logo, é uma etapa da área de projeto automatizado de sistemas digitais com alto custo. Isto acontece principalmente porque existe um número muito grande de testes que podem ser aplicados ao dispositivo se utilizarmos todas as possibilidades indistintamente (teste exaustivo). Além disso, o número de testes cresce de forma exponencial em função do número de entradas do circuito. Tentando resolver este problema, várias soluções são buscadas no sentido de agilizar o processo de teste, garantindo a qualidade dos produtos. Sendo assim, existem vários algoritmos de geração automática de vetores de teste que visam encontrar o subconjunto ótimo de testes, onde os vetores consigam detectar o maior número possível de falhas, ou seja, tenham boa cobertura, com pouco processamento para o seu cálculo e que não seja muito grande para não imputar tempo excessivo na hora da aplicação. Estes algoritmos baseiam-se no fato de que alguns vetores não são capazes de detectar falhas, visto que os circuitos podem possuir características capazes de mascarar falhas, e de que outros vetores podem detectar mais de uma falha. Não há, pois, necessidade de utilizar-se todos os padrões possíveis no teste, basta apenas identificar os padrões capazes de detectar falhas.

Estes algoritmos, entretanto, são enumerativos e seu tempo de resolução é exponencial em função das entradas, por isso, o que se faz com o objetivo de acelerar a geração de testes é utilizar de heurísticas que auxiliem estes algoritmos a chegar mais rapidamente a seu objetivo.

Um algoritmo clássico da área de testes é o Algoritmo D [ROT 66], que é enfocado neste trabalho em suas diversas versões, munidas de algumas heurísticas. Este algoritmo foi desenvolvido em 1966, porém, é o algoritmo

mais largamente usado até os dias de hoje, porém com auxílio de medidas que melhorem o seu desempenho. Ele oferece uma boa cobertura de falhas, apesar de na sua forma original despender muito processamento até obter o padrão desejado. Isto acontece, principalmente, pelo fato do algoritmo originalmente não utilizar-se de nenhum critério na busca do padrão, fazendo-o por tentativa e erro.

Este algoritmo gera, originalmente, um teste para cada falha, porém notou-se posteriormente que alguns testes detectavam mais de uma falha e que era possível verificar isto durante a geração, fez-se pois alguns melhoramentos neste sentido. Observou-se também a necessidade de critérios que auxiliassem este algoritmo a escolher adequadamente os valores e as conexões a que estes seriam atribuídos. Isto é feito levando-se em consideração parâmetros como controlabilidade, observabilidade, estrutura e outros, diminuindo assim as chances de erro nas escolhas feitas durante o processamento do vetor, que fazem com que o algoritmo fique fazendo e desfazendo operações (*backtracking*) Estes *backtrackings* são a maior causa do excesso de processamento e levaram ao desenvolvimento de heurísticas que procuram minimizar este overhead na geração de testes.

O algoritmo D basicamente compõe-se de três rotinas básicas: a inserção da falha, a propagação da falha e a justificação das constantes, que é utilizada tanto na inserção quanto na propagação da falha.

A inserção da falha consiste na atribuição de valores às linhas de forma que se presencie um erro na linha-alvo, ou seja, a linha onde se quer ter certeza que não está falha. Para isto, atribui-se valores capazes de gerar uma diferença, isto é, que na linha-alvo o circuito bom e o circuito falho apresentem valores diferentes. Deste modo, se pode afirmar, de acordo com o valor presente, a existência ou não de falha. Esta diferença (D) é que deu nome ao algoritmo e que é a base da detecção da falha.

Como os circuitos são encapsulados, só produzir esta diferença em uma linha interna, a qual se quer testar, não é suficiente, por isso é necessário ainda, propagar esta diferença até um ponto observável (saída primária) para que se possa observá-la externamente no circuito. Isto é feito pela rotina de propagação de falhas que atribui valores às linhas de forma que o sinal seja observado.

A justificação das constantes consiste na colocação de valores nos pontos controláveis (entradas do circuito) de maneira que se tenha nas linhas internas o valor desejado. Todavia, a inserção destes valores nem sempre é possível. Por vezes, o valor é conflitante com alguns valores que já estavam presentes nas conexões do circuito. Neste caso, o algoritmo deve realizar o *backtracking*, isto é, desfazer as operações que geraram o conflito e tentar outro caminho para justificar os valores sem conflito. Este fazer e desfazer de operações é que consome tempo de processamento. Além disso existe a escolha do caminho para propagar que nem sempre é o ótimo. Existem situações em que o algoritmo vê-se frente a uma circunstância onde mais de um caminho pode propagar a falha. Neste caso, pode acontecer de um caminho levar diretamente a um ponto observável (saída primária) e o outro ainda ter que percorrer um longo caminho até que isto aconteça, estando sujeito a muitos conflitos neste trajeto. É interessante, pois, aliar-se ao procedimento uma medida capaz de verificar estas situações de forma a auxiliar o algoritmo durante as escolhas, objetivando diminuir a quantidade de tempo e processamento requeridos na busca do padrão de teste.

Este trabalho, então, analisa todas as heurísticas e técnicas presentes na literatura, que visem contornar estes problemas. Existem diversos enfoques em que podem basear-se estas heurísticas: funcionalidade das portas lógicas, distância entre pontos do circuito (em número de portas), estrutura das conexões do circuito (presença de fan-out reconvergentes), e probabilidade de sinal.

As heurísticas de distância foram umas das primeiras a serem utilizadas e auxiliavam o algoritmo a detectar qual caminho estava mais perto da saída primária, para utilizá-lo na propagação, ou mais perto da entrada primária, para utilizá-lo na justificação das contantes. Mais tarde verificou-se que só isto não era suficiente, pois se o caminho era por portas que necessitavam a colocação de muitos valores para a propagação ou justificação, era mais interessante escolher outro, apesar de necessitar passar o sinal por mais portas. Isto se dava devido a funcionalidade das portas e do número de entradas destas. Passou-se então a penalizar as portas para valores distintos, visto que determinadas funções como a AND ou OR, têm apenas um padrão de valores na entrada que gera um valor lógico específico, no caso 1 para a AND e 0 para a OR, enquanto que todos os outros padrões geram o valor contrário. Utilizou-se ainda a mesma medida de distância, porém com uma função que levasse em consideração este detalhe. Pensou-se também na relevância de fan-outs reconvergentes, pois verificou-se serem eles os maiores causadores dos conflitos, trazendo como consequência um número maior de *backtrackings*. Outro artífício utilizado de forma a melhorar o desempenho do algoritmo é o recálculo das medidas de auxílio da geração de testes de forma dinâmica. Com isto tenta-se manter a qualidade das medidas durante todo o processo de ATPG.

Este trabalho propõe a utilização conjugada de medidas capazes de coexistir. Analisa a possível queda de desempenho, partindo do princípio que a utilização de um grande número de heurísticas ao mesmo tempo pode acabar por impetrar maior processamento, devido aos muitos cálculos que são necessários. Com isto, pode acontecer do algoritmo com as heuríticas ter desempenho pior que o próprio algoritmo em sua forma original. Analisa ainda, a possibilidade do recálculo dinâmico ser feito de maneira mais racional e com os diversos tipos de medidas. Por fim, apresenta os resultados que ilustram o desempenho do algoritmo em diversos casos.

Este trabalho está assim dividido:

O **capítulo 2** coloca em termos gerais o conceito de falha e sua modelagem para adequação aos algoritmos de geração de testes.

O **capítulo 3** abrange o teste e sua aplicação na fabricação de circuitos VLSI. Apresenta ainda os tipos de vetores de teste, otimizações passíveis de serem feitas sobre estes, os passos do teste e a avaliação da qualidade do teste em termos de cobertura e complexidade.

O **capítulo 4** explana o procedimento algorítmico da geração de testes no algoritmo D, definindo as operações realizadas e principais rotinas. É avaliado, ainda, quais parâmetros são capazes de influenciar a geração de testes, ou seja, o problema encontrado ao se gerar estes testes. Determina-se, pois, a noção de testabilidade, que abrange os conceitos de controlabilidade e observabilidade.

O **capítulo 5** versa sobre as heurísticas que podem ser utilizadas no sentido de auxiliar os algoritmos de geração de testes a contornar os problemas antes expostos. Apresenta, então, medidas de testabilidade capazes de auxiliar o algoritmo D a escolher os caminhos durante a geração que não imputem excessivo processamento (diminuição do número de *backtrackings*). Expõe, além disso, como estas medidas podem ser calculadas dinamicamente e quais vantagens e desvantagens isto acarreta.

O **capítulo 6** sugere a utilização conjugada de algumas heurísticas, analisando o ganho ou perda de cada combinação. Propõe, ainda, uma otimização para a técnica de testabilidade dinâmica que tem por objetivo diminuir o processamento desta técnica.

O **capítulo 7** relata o experimento prático realizado e compara os resultados obtidos analisando quantitativamente a performance do algoritmo D com as diversas heurísticas e em vários tipos de circuitos.

Finalmente conclue-se o trabalho avaliando como estas otimizações e o uso conjugado das heurísticas, que foram fruto deste trabalho, podem contribuir para o aperfeiçoamento da área de testes.

2 A FALHA

Falha é a manifestação ou efeito de um defeito físico que pode ser ocasionado por problemas na fase de fabricação, como por exemplo, fios que ficam em contato acidentalmente, componentes mais lentos que o especificado, entre outros.

As falhas podem ser modeladas de modo a representar o comportamento lógico de uma falha física, deste modo o modelo consegue representar a realidade física em um modelo lógico capaz de ser processado no computador.

Idealmente, esperaria-se que o processo de fabricação produzisse apenas circuitos funcionalmente bons. Isto, entretanto, não acontece, visto que na prática o número de circuitos bons pode variar de 100 % até um ou nenhum circuito bom por *wafer*. Por este motivo a etapa de verificação das falhas é extremamente importante, pois dela depende a qualidade do produto.

Dependendo da aplicação, este processo pode ser crítico, exigindo a garantia de um mínimo de falhas. O processo de detecção destas falhas deve, portanto, ser cuidadoso e eficiente, já que isto pode denegrir a imagem do fabricante e seu produto.

2.1 Causas das Falhas em Circuitos Integrados

Sistemas Digitais podem falhar por várias razões: falhas de fabricação, falhas em componentes durante a vida do sistema por degradações provocadas pelo uso, falhas de projeto, entre outras.

Um erro inicial no sistema pode ser causado por um projeto incorreto ou por ineficiência do processo de teste. A manifestação do defeito pode apare-

cer de modo permanente ou temporário, e neste último pode ser causada por agentes externos – falha transiente – ou por uma degradação dos parâmetros – falha intermitente (ver tabela 2.1).

Falha	Causa da Falha
Componente Inicialmente com Erro	Projeto Incorreto Teste Incompleto
Componente com Falha durante a Operação	Falhas permanentes Falhas Temporárias – transientes Falhas Temporárias – intermitentes

Tabela 2.1: Falhas e suas causas

2.1.1 Problemas de Projeto

Um sistema pode falhar se foi projetado incorretamente. As causas mais freqüentes de erros de projeto são: especificações e documentações sem clareza, verificação de projeto incompleta, e, ainda, alterações de projeto incorretas ou imprecisas. Estas falhas ocasionarão grandes custos mais tarde.

Destas, a mais significante é a documentação imperfeita. Isto acontece porque não é utilizado um método formal para especificar as necessidades do sistema. Com isso as documentações são geralmente especificadas em inglês, o que significa que não se pode verificar formalmente se o projeto atende à especificação.

A documentação e o teste, juntos, têm a responsabilidade pelo maior custo na fabricação de muitos sistemas. Em contraposição a isto, é muito raro o projetista se preocupar com o teste e/ou a documentação. Por tradição o projetista não se preocupa com isto até o final do projeto. Porém esta atitude tem lentamente se modificado em relação ao teste, em virtude do alto custo e da necessidade premente, por causa da miniaturização dos sistemas, de se preparar o teste durante os estágios primários do projeto. Esta preocupação, contudo,

não se estende à documentação já que os custos são menos concentrados o que dificulta a percepção de sua importância.

Um trabalho promissor, desenvolvido no sentido de padronizar a documentação, é *Standard for Graphics Symbols for Logic Functions*, que já foi adotado pela IEEE (ANSI/IEEE Std. 91-1984 [KAM 85]).

Os aspectos lógicos de um projeto são tipicamente verificados através de simulação. Ocorre, porém, que a simulação feita não é exaustiva e o que acontece na maioria dos casos, é que os estímulos de entrada para a simulação são determinados pelo projetista. Este conjunto de estímulos, entretanto, pode não ser representativo de modo a mostrar os defeitos do projeto. No sentido de resolver este problema pode-se gerar estes estímulos automaticamente, através de alguma técnica de geração de estímulos para a simulação como proposto em [DOR 92]

2.1.2 Problemas na Fabricação

Um dos fatores mais preponderantes causadores de falhas em sistemas é a presença de circuitos com falhas de fabricação. Um circuito com falha não poderia ser utilizado, entretanto, o problema é garantir a abstenção de falhas. O teste perfeito é capaz de garantir isto, já o imperfeito é gerador potencial de futuras falhas. Isto será visto adiante no capítulo 3 sobre teste.

2.2 Modelagem de Falhas

A modelagem da falha consiste da representação precisa de uma falha física em um formato utilizável pela simulação e geração de testes, ou seja, esta representação envolve a definição abstrata ou lógica da falha,

de modo a representar o mesmo comportamento errôneo das falhas físicas modeladas. Deste modo, o procedimento de simulação, por exemplo, pode compor a realidade de um circuito com falha com o objetivo de verificar o comportamento desta falha no funcionamento do componente.

Na geração de testes, o modelo é utilizado para que, uma vez definido o ponto falho, possa-se gerar o teste capaz de observar a falha, delatando o circuito defeituoso.

Diferentes modelos são necessários para cada nível de abstração – elétrico, lógico, funcional – em que o circuito de interesse foi modelado. Por exemplo, no nível elétrico, as falhas que podem ocorrer são mudanças de voltagem, corrente ou resistência. Parâmetros elétricos, entretanto, não são significativos no nível lógico e as falhas são definidas em termos dos valores lógicos 0 e 1. Com isto abstrai-se a causa, modelando-se apenas o efeito lógico causado pela falha física.

Um modelo de falhas adequado deve ser coerente com o nível de abstração no qual vai ser usado, não deve ser complexo a ponto de imputar excessivo custo computacional e, principalmente, deve refletir o comportamento da falha física com a exatidão suficiente para a aplicação. Resumindo, um modelo de falhas deve ser preciso o suficiente para a aplicação e de fácil utilização.

Infelizmente, é difícil encontrar uma modelagem que atenda perfeitamente às necessidades satisfazendo estes quesitos. O que se tem são modelos de falhas F_a que podem representar adequadamente um grande número de falhas físicas F_p.

Apesar de não corresponder a realidade exata, o modelo pode vir a ser adequado. Por exemplo, na geração de testes, a tarefa básica é gerar testes para todas as falhas F_p conhecidas ou esperadas no circuito N_p. Usa-se então

um modelo do circuito N_a que modela N_p, e se os testes usados para detectar todas as falhas F_a detectam todas as falhas F_p, o modelo de falhas é adequado. Neste caso diz-se que F_a cobre F_p. A porcentagem de falhas F_p que podem ser detectadas por F_a chama-se cobertura de falhas.

Apesar de ser difícil precisar esta medida, ela serve para indicar a qualidade do modelo de falhas. Através dela consegue-se estipular o quanto o modelo foi adequado. Esta medida, contudo, não significa apenas isto, ela mede também a precisão do algoritmo de geração de testes, e até a imperfeição do projeto do circuito.

Várias características são usadas para classificar o modelo de falhas: a periodicidade destas falhas, o efeito no comportamento funcional do circuito, a velocidade de operação induzidas por estas falhas, entre outras. Quanto à periodicidade as falhas podem ser permanentes, transientes ou intermitentes.

Pode-se distinguir as falhas quanto ao número de falhas distintas que podem estar presentes simultaneamente. Isto influencia muito na modelagem porque deste parâmetro depende o tamanho da lista de falhas a ser considerada. O que se assume normalmente, até por uma questão de diminuição de complexidade, é que só pode ocorrer uma falha a cada tempo. Isto é bastante usado e sabe-se que a cobertura das falhas múltiplas oferecida por esta simplificação é bastante boa.

2.2.1 O modelo *Stuck-at*

Existem várias falhas físicas que se comportam de maneira que uma conexão pareça estar sempre no valor lógico 0 ou 1. A modelagem deste comportamento chama-se *stuck-at*, ou *grudado-em*.

É o modelo mais utilizado e não possui restrições quanto ao circuito, modelando falhas permanentes no nível lógico e com baixa complexidade computacional. Em um circuito de n linhas, o número de falhas *stuck-at* distintas é $2n$. Este número é pequeno se comparado aos $n(n-1)/2$ das falhas do tipo curto-circuito entre duas linhas (*bridging faults*) ou 3^{n-1} que é o número de falhas múltiplas.

O modelo *stuck-at* é o modelo utilizado no Algoritmo D [ROT 66], PODEM [GOE 81] e muitos outros algoritmos de geração de testes. A notação utilizada é $s-a-0$, ou seja, o sinal s *grudado-em*-0 (ver figura 2.1).

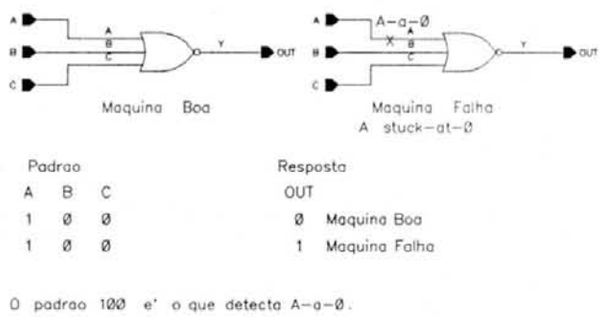


Figura 2.1: Falhas *stuck-at*

Uma questão natural é qual a cobertura física dada pelos testes capazes de detectar falhas *stuck-at* simples. Este problema pode ser resolvido com um minucioso exame das falhas físicas que ocorrem nas atuais tecnologias [BEH 82]. A cobertura exata é difícil de ser obtida mas evidências empíricas mostram que para circuitos combinacionais e sequenciais em geral, implementados com tecnologias MOS comuns ou bipolar, o modelo *stuck-at* oferece boa cobertura de falhas físicas permanentes.

A experiência mostra que o modelo *stuck-at* é viável em chips com mais de 2000 a 3000 portas, especialmente quando combinados com técnicas de projeto visando a testabilidade.

Este modelo tem a vantagem de ser de fácil compreensão e relativamente independente da tecnologia. A mesma base de dados pode ser, inclusive, utilizada para o projeto, simulação de falhas e diagnóstico. Este modelo apresenta ainda facilidade na expressão da cobertura de falhas em termos de porcentagem de falhas *stuck-at* cobertas pelo teste.

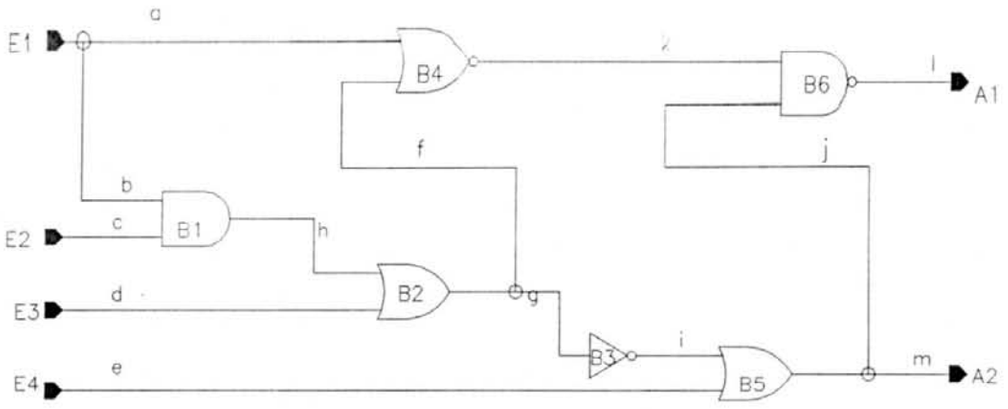


Figura 2.2: Exemplo de localizações de falhas *stuck-at*

Na figura 2.2, examina-se um circuito lógico com 4 entradas e 2 saídas primárias. As conexões são definidas de a até m, os sinais de entradas, de $E1$ a $E4$ e as saídas $A1$ e $A2$. O circuito contém 6 portas lógicas, $B1$ até $B6$ e compõe-se de 13 conexões, logo pode-se considerar 26 falhas do tipo *stuck-at*, 13 para *stuck-at*-0 e 13 para *stuck-at*-1. Considerando-se falhas m últiplas este número cresce para $3^{13} - 1 = 1.594.322$ falhas.

2.2.2 Extensões do Modelo *Stuck-at*

Existem determinadas falhas que transformam um circuito combinacional em sequencial e não podem ser modeladas pelo modelo *stuck-at* simples. Sabe-se, entretanto, que modificações no circuito podem alterá-lo de forma a ser suportado por este mesmo modelo. O problema é que estas modificações, no geral, são um tanto complexas, o que faz com que nem sempre este método seja operacional.

2.2.3 O Modelo *Stuck-at* para Falhas Múltiplas

O modelo *stuck-at* pode ser generalizado a ponto de modelar as múltiplas falhas que podem ocorrer em um mesmo momento. Isto causa um aumento bem considerável no número de falhas, na ordem de $3^n - 1$ para um circuito de n linhas, contra $2n$ do modelo *stuck-at* simples. Este número é muitas vezes impraticável, principalmente com a miniaturização que permite circuitos com um número cada vez maior de portas.

Sabe-se, por outro lado, que o conjunto T para todas as falhas *stuck-at* simples, usualmente cobre todas as falhas múltiplas. Quando isto não acontece é porque uma falha pode estar mascarando a outra. Este mascaramento mútuo, porém, é raramente encontrado em circuitos reais, o que justifica o uso mais pronunciado do modelo *stuck-at* simples.

2.2.4 Modelo Funcional

Modelos com objetivo de verificar a função básica realizada pelo circuito são chamados de Modelos Funcionais. Podem fazer parte desta classe, modelos que visam identificar falhas de função no próprio nível lógico, o que é o caso da modelagem de falhas do Algoritmo S [DOR 92], ou modelagens mais abstratas, que tentam trabalhar a nível comportamental contendo, porém, normalmente imprecisões e heurísticas.

Existem ainda os modelos funcionais específicos, que visam apenas certas classes de circuitos, por exemplo, RAM's ou ULA's.

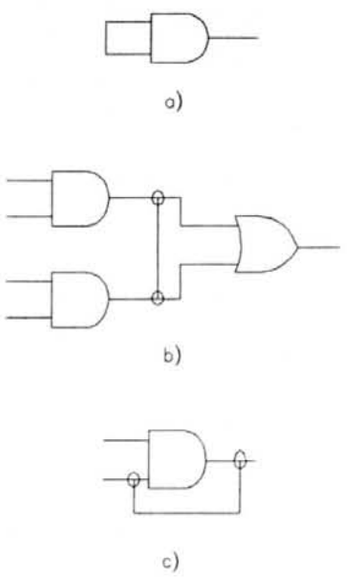


Figura 2.3: *Bridging-Faults*

2.2.5 *Bridging-Faults*

Uma falha do tipo *Bridging-Fault*, ou falha-ponte, ou simplesmente curto, são causadas quando dois fios ficam acidentalmente em *curto* (ver figura 2.3 a, b e c).

Um circuito com t linhas tem a possibilidade de ter t^2 falhas-ponte. Com o conhecimento da estrutura, entretanto, pode-se reduzir este número utilizando-se somente falhas potenciais.

As falhas deste modelo não são bem representadas pelo modelo *stuck-at*, por isso necessitam de um modelo próprio que modele o efeito causado pelas ligações extras. Em muitas tecnologias esta falha tem o efeito de causar um *Wired*-AND ou *Wired*-OR, entre os sinais. Isto pode ser visto na figura 2.4.

Uma consideração importante sobre as falhas-ponte é o fato de uma falha deste tipo poder transformar um circuito combinacional em um circuito sequencial pela introdução de um *feedback-path*.

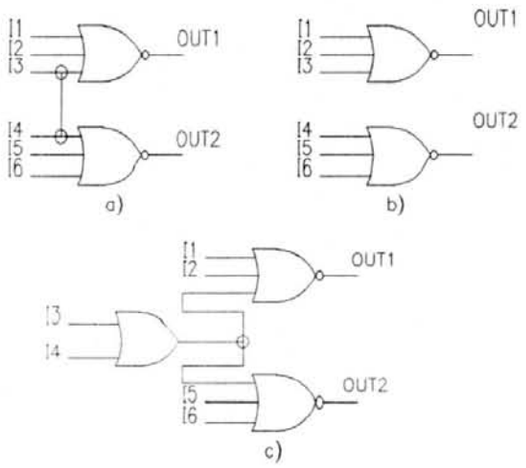


Figura 2.4: a) Circuito em curto b) Circuito Bom c) Modelagem da máquina-Falha

2.2.6 Falhas de Atraso

São atrasos no circuito que estão fora da especificação, por exemplo, uma porta lenta. Para detectá-la, os teste devem ser aplicados a intervalos próximos do tempo de chaveamento do circuito. Se o circuito não tem problemas de atraso demasiado, as respostas esperadas estarão sempre disponíveis na saída em tempo hábil. A modelagem é baseada em caminhos de atraso entre os pontos de teste.

2.2.7 Falhas em Transistores MOS (*stuck-open*)

Os modelos de falhas *stuck-at* e *bridging* analisam o circuito no nível lógico. Existem falhas físicas em transistores MOS que não são modeladas adequadamente por estes modelos. Uma falha *stuck-open* assume que a falha física é um transistor que nunca conduz corrente, ou seja, em termos de nível de chaves permanecem sempre abertos [WAD 78].

Um transistor que não conduz em uma célula MOS combinacional resulta na saída da célula permanentemente flutuando. Esta falha introduz memória na célula o que torna o teste deste tipo de falha extremamente difícil.

2.3 Lista de Falhas

O circuito lógico modelado apresenta, de acordo com o modelo e as conexões e portas, um número de locais passíveis de falha. À lista destes locais, chama-se Lista de Falhas.

Na figura 2.2, por exemplo, tem-se 13 locais ou conexões passíveis de falhas *stuck-at*, ou seja, 13 conexões que podem ter seu valor permanentemente *grudado* em determinado valor.

A lista de falhas para o circuito da figura 2.2, assumindo-se o modelo *stuck-at* é mostrada na tabela 2.2. Nesta tabela vê-se as 13 conexões e as falhas asssociadas a elas, sempre duas, ou seja, o sinal *stuck-at*-0 e o sinal *stuck-at*-1.

Conexão	Falhas
a	a-a-0 ; a-a-1
b	b-a-0 ; b-a-1
c	c-a-0 ; c-a-1
d	d-a-0 ; d-a-1
e	e-a-0 ; e-a-1
f	f-a-0 ; f-a-1
g	g-a-0 ; g-a-1
h	h-a-0 ; h-a-1
i	i-a-0 ; i-a-1
j	j-a-0 ; j-a-1
k	k-a-0 ; k-a-1
l	l-a-0 ; l-a-1
m	m-a-0 ; m-a-1

Tabela 2.2: Lista de Falhas de um circuito com treze conexões

3 O TESTE

Teste é o processo de aplicação de procedimentos sobre o objeto a ser verificado com o objetivo de comprovar suas propriedades. No que tange a teste de circuitos integrados, o objetivo pode ser verificar o projeto, visando encontrar falhas em sua concepção, ou sobre o dispositivo já fabricado, assumindo-se que o projeto encontra-se correto. Neste caso, o que se buscam são falhas de fabricação que possam comprometer o bom funcionamento do dispositivo.

O teste de projeto verifica a consistência entre a descrição e o projeto físico e é também chamado de verificação de projeto. A técnica mais utilizada nesta tarefa é a simulação do circuito em funcionamento, ou seja, o projeto é simulado utilizando-se como entrada os estímulos de simulação gerados por um procedimento algorítmico ou manualmente pelo projetista. Se respostas de acordo com o esperado são obtidas nas saídas, assume-se que o projeto está correto. Uma técnica comum são as chamadas técnicas de *toggle*, que tentam garantir que cada nodo do circuito recebeu os dois valores, 0 e 1.

O teste feito no circuito já pronto visa a comprovação do comportamento ou de parâmetros como consumo elétrico ou tempo de propagação de sinais. Este teste é feito para encontrar falhas de manufatura, utilizando um dos modelos disponíveis na literatura (ver capítulo 2 sobre falhas). Estes defeitos físicos são encontrados a partir da aplicação de padrões de teste (estimulação elétrica) e checagem do resultado esperado.

Situando-se as várias funções de teste nas etapas de fabricação de circuitos VLSI (ver figura 3.1) verifica-se que, partindo da especificação do dispositivo, parte-se para o projeto. Nesta hora já pode-se encontrar uma preocupação com a testabilidade e o futuro teste que vai ser aplicado no circuito. Neste momento pode-se utilizar técnicas de projeto visando a testabilidade, que é a adição de componentes adicionais que facilitem posteriormente

a etapa de teste e até a colocação de todo o teste dentro do projeto, ou seja, o auto-teste, que pode vir a ser acionado periodicamente após o circuito já estar em funcionamento de maneira a detectar eventuais erros acarretados por desgaste. A análise de testabilidade, neste caso, tem o objetivo de indicar as áreas do dispositivo onde deve ser incluída esta lógica adicional. Esta mesma análise, que utiliza as técnicas enfocadas neste trabalho, serve posteriormente para auxiliar o algoritmo de geração de testes.

Figura 3.1: Etapas de teste

Nesta etapa, após o uso das ferramentas de síntese lógica, por exemplo, pode-se ter o teste para encontrar erros de projeto (verificação formal e/ou simulação lógica).

O projeto estando pronto e já cristalizado, pode-se iniciar a geração dos testes que serão aplicados posteriormente ao circuito na etapa de pós-fabricação com o objetivo de encontrar erros de manufatura. Quando este conjunto de teste está pronto submete-se o projeto juntamente com os vetores

a uma simulação de falhas. O circuito é simulado como se estivesse falho. As respostas do circuito falho são comparadas com as do circuito bom. Com isto, consegue-se determinar quantas falhas são detectadas pelo conjunto de vetores gerados, isto é, determina-se a cobertura de falhas. Assim, pode-se determinar a acuracidade dos vetores que foram gerados e, eventualmente, mudá-los.

Finalmente, o projeto físico é realizado e passa-se para a etapa de fabricação. Quando os circuitos estão prontos são submetidos aos padrões gerados anteriormente e, com isto, pode-se determinar quais componentes encontram-se sem falhas e o contrário.

Neste trabalho enfoca-se diretamente a etapa de geração de testes e análise de testabilidade com o objetivo de acelerar esta geração auxiliando o algoritmo D, que foi o algoritmo escolhido para gerar os testes.

3.1 Os Vetores de Teste

O objetivo da geração de testes, como o próprio nome diz, é gerar um conjunto de vetores ou padrões de teste capazes de, quando aplicados ao circuito, detectar neste as falhas existentes. Este conjunto deve detectar o máximo de falhas sem imputar gasto excessivo de tempo nas etapas do teste onde ele é utilizado, que são a geração destes padrões e, posteriormente, a sua aplicação no dispositivo a ser testado.

3.2 Tipos de Vetores de Teste

Existem vários tipos de vetores ou padrões de teste, dependendo da modelagem utilizada e do critério de geração. O teste exaustivo é o que

utiliza todos os padrões possíveis. O pseudo-exaustivo baseia-se no modelo de dependência de falhas. O conjunto *stuck-at* são os vetores que detectam falhas do tipo *stuck-at* e os vetores para falhas funcionais são os gerados de especificações não estruturais do circuito [AKE 80].

3.2.1 Conjunto de Vetores Exaustivos

Um teste exaustivo para um circuito combinacional de n entradas é o conjunto de todos os 2^n vetores de entrada possíveis [BRE 76]. Uma das maiores dificuldades com estes conjuntos é que alguns defeitos físicos causam falhas não-combinacionais, como por exemplo, falhas-ponte ou *stuck-open* e que não são detectadas por um teste exaustivo.

Uma solução já foi proposta para o problema das *stuck-open*, que é a adição ao conjunto de vetores, de vetores específicos para este tipo de falha. Isto porém, aumenta o processamento a ser executado, o que consequentemente aumenta o custo.

Outra dificuldade com o teste exaustivo é que ele pode ser muito longo. O teste exaustivo requer 2^n padrões para as n-entradas do circuito. Assim, para circuitos com mais de 20 entradas o tamanho do teste torna-o proibitivo.

3.2.2 Conjunto de Vetores Pseudo-exaustivos

Para circuitos em que cada saída depende de um subconjunto de todas as entradas, o conjunto pseudo-exaustivo testa apenas as saídas exaustivamente e não o circuito inteiro. Esta técnica é baseada em um modelo de falhas, o modelo das restrições de dependência de falhas [McC 71] [MIL 87],

que assume que o conjunto de entradas de que depende a saída do circuito não aumenta na presença de falhas.

3.2.3 Conjunto de Vetores para Falhas *Stuck-at*

Existem técnicas que geram conjuntos de vetores menores que o exaustivo. As mais populares assumem que só falhas *stuck-at* podem ocorrer e requerem o diagrama lógico do circuito a ser testado.

Métodos que selecionam caminhos, como o Algoritmo D que será visto posteriormente, conseguem o conjunto mínimo que testa todas as falhas *stuck-at* simples não redundantes, mas temos que considerar o tempo de processamento dispendido no cálculo.

Apesar destes métodos considerarem apenas as falhas simples, estes conjuntos detectam grande parte das falhas múltiplas e outras falhas não-modeladas, como por exemplo, *bridging-faults* [MEI 74] [MIL 88].

3.2.4 Conjunto de Vetores de Teste Funcionais

Existem situações em que não se tem disponível o diagrama lógico do circuito, por este motivo, os algoritmos funcionais derivam os testes utilizando uma especificação funcional do circuito e fazendo algumas suposições sobre implementação, por exemplo, em um projeto, um conjunto de teste para falhas *stuck-at* em uma RAM pode ser gerado assumindo-se que o Decodificador de Endereço foi implementado com um decodificador tradicional, ou seja, um decodificador feito com portas AND e inversores, sem fan-outs reconvergentes [MIL 87].

3.2.5 Conjunto de Vetores de Teste Randômicos

O conjunto de vetores de teste gerados randomicamente pode ser obtido com baixo custo e só depende do número de entradas do circuito, sendo independente da topologia do circuito.

Estes testes são gerados por um processo randômico e o procedimento pode ser observado no fluxograma da figura 3.2. É possível utilizar-se vetores randômicos até uma certa cobertura e o algoritmo D para as falhas que não foram cobertas pelos testes randômicos [VIS 72].

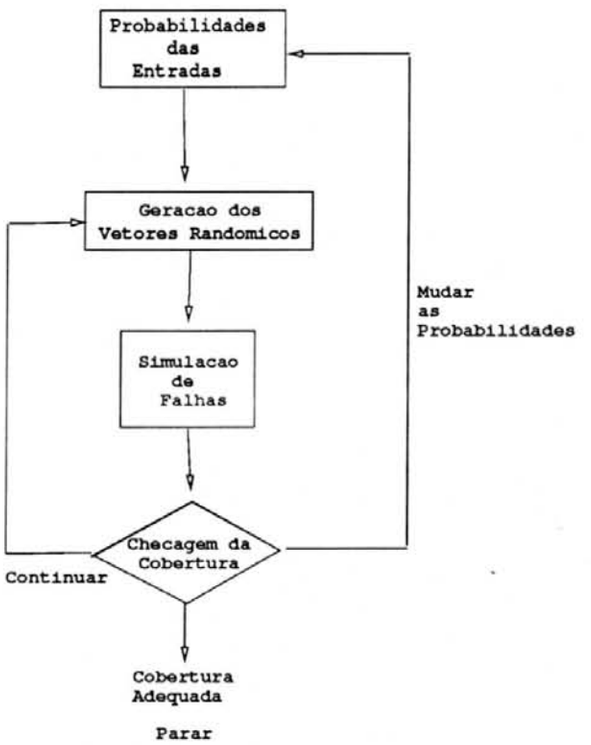


Figura 3.2: Fluxograma da geração de vetores randômicos

Na figura 3.2, a probabilidade das entradas significa a probabilidade com que os 0's e 1's serão gerados. Por exemplo, uma função com igual probabilidade de 0's e 1's geraria a metade de bits em 1 e a outra metade em 0's. Se a cobertura alcançada não for suficiente deve-se mudar as probabilidades e recomeçar a geração.

3.3 Compactação da Lista de Falhas e dos Vetores Gerados

A lista de falhas completa não necessita ser utilizada na geração de testes, visto que nem sempre estas falhas são relevantes. Como o número de testes ou vetores gerados é baseado nesta lista, pois é gerado um teste para cada falha, é importante verificar se algumas destas falhas podem ser desconsideradas. Este processo é chamado de compactação da lista de falhas.

Por vezes, pode-se determinar que alguns dos vetores gerados não são necessários pois detectam falhas que já estavam cobertas por outro vetor de teste. Este caso, entretanto, não otimiza a geração de testes, visto que os vetores são compactados após a geração, logo, o tempo de processamento gasto na geração já foi perdido. Apesar disso, consegue-se diminuir o tempo de aplicação dos testes, que também não é pequeno.

3.3.1 Colapso de Falhas

O colapso de falhas é a determinação das falhas que não necessitam geração de vetores distintos. Isto acontece porque várias falhas são cobertas pelo mesmo teste.

O colapso de falhas utiliza-se dos conceitos de falhas por equivalência ou por dominância, visto que algumas falhas podem gerar a mesma máquina-falha ou uma máquina-falha que domine a outra.

Seja um circuito combinacional C, que realiza a função f e as funções f_α na presença da falha α e f_β com a falha β.

As falhas α e β são ditas equivalentes quando $f_\alpha = f_\beta$. Logo, classe de equivalência é o conjunto de todas as falhas que realizam a mesma função.

Neste caso, nenhum teste será capaz de distingui-las já que o efeito final é o mesmo. Assim, na geração de testes é necessário considerar apenas uma falha de cada classe de equivalência. Isto diminui consideravelmente a lista de falhas, o que, consequentemente, diminui o número de testes que devem ser buscados, sendo considerado uma compactação da lista de falhas.

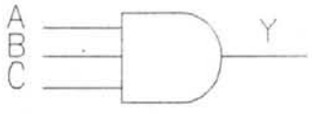


Figura 3.3: Colapso de falhas em uma porta AND

Na porta AND da figura 3.3, vê-se que A-a-0, B-a-0 e C-a-0 formam uma classe de equivalência, ou seja, todas estas falhas forçam a saída do circuito para 0. Generalizando pode-se dizer que para as portas AND, NAND, NOR e OR com n entradas, considera-se apenas $n+2$ falhas das $2(n+1)$ falhas possíveis. A redução de falhas por este princípio é dito colapso de falhas por equivalência.

Existe ainda a redução por dominância, porém cabe ressaltar que só pode ser utilizada com o propósito de detecção de falhas e não diagnóstico. Diz-se que a falha α domina a falha β se $vetor_\alpha \subset vetor_\beta$, onde vetor é o padrão de teste. Deste forma pode-se testar usando apenas o $vetor_\alpha$ porque o $vetor_\beta$ detecta as mesmas falhas. Isto pode ser visto como uma compactação dos vetores gerados.

3.3.2 Otimizações durante a Geração

É possível detectar, durante a geração que o vetor que está sendo gerado detecta, não apenas a falha-alvo, mas também outras falhas, as quais podem ser retiradas da lista de falhas. Isto otimiza os dois passos, visto que diminui o tempo de geração e também de aplicação. Isto será novamente

abordado com mais detalhes em capítulo posterior, durante a explanação dos procedimentos algorítmicos de geração de testes.

3.4 Dificuldades do teste

O teste dos circuitos é uma etapa que imputa uma série de custos, principalmente gerados pelo gasto de tempo que este processo consome. Por este motivo, busca-se otimizar o processo utilizando heurísticas para acelerar a geração, melhorando o desempenho dos algoritmos de ATPG.

As principais dificuldades do teste, que o tornam um processo caro e demorado e que impulsionam as pesquisas no sentido de buscar melhores procedimentos e soluções são os seguintes [ABA 83]:

- O número de falhas que devem ser consideradas é grande, pois os circuitos podem ter milhares de portas, elementos de memória e linhas de interconexão, todos individualmente sujeitos a diferentes tipos de falhas.
- Os elementos internos do circuito somente podem ser observados ou controlados através dos pinos de entradas e saída da pastilha. Como cada vez mais elementos são encapsulados numa pastilha, a tarefa de gerar um teste adequado se torna cada vez mais longa e difícil.
- Quando o teste envolve componentes já prontos, de outro fabricante, os detalhes de implementação não são conhecidos. O que se tem sobre o componente é o manual do usuário que detalha o conjunto de instruções e descreve a arquitetura do processador a nível de transferência entre registradores.
- No caso de geração de testes para circuitos sequenciais, como mais e mais flip-flops's estão sendo encapsulados nas pastilhas, novas

falhas [BRE 76] precisam ser consideradas e estas são difíceis de detectar e requerem grande número de vetores.

- A natureza dinâmica de muitos sistemas VLSI requer sistemas de teste de alta velocidade que possam testar os circuitos quando eles estão operando em sua velocidade máxima.

- A estrutura de barramento de alguns dispositivos VLSI torna o isolamento da falha difícil. Nestes dispositivos vários componentes compartilham o mesmo barramento.

- Além do problema do tempo dispendido na geração automática de vetores de teste, há ainda, o tempo gasto na aplicação destes testes e o uso da memória utilizada para armazenar os padrões de entrada e saída do chip.

- É difícil quantificar o aumento na complexidade do testador com o aumento do número de pinos e de dados de teste. Esta complexidade torna o testador mais caro para ser construído e requer mais horas de teste. Isto, conseqüentemente, aumenta o custo do teste.

3.5 Métodos de Teste

Há muitos métodos de teste para circuitos LSI, cada um com seu enfoque próprio, contudo, pode-se dividir os métodos em duas grandes categorias, o teste concorrente e o teste explícito (ver figura 3.4) [HAY 80] [REG 85].

O teste concorrente geralmente é implementado dentro do circuito e o maior uso deste tipo de teste é o diagnóstico. Uma vantagem deste método na manutenção do sistema é que não há necessidade de geração de padrões, já que os padrões usados na operação normal servem como vetores de teste, assim as falhas são detectadas instantâneamente dentro do chip. Este método também

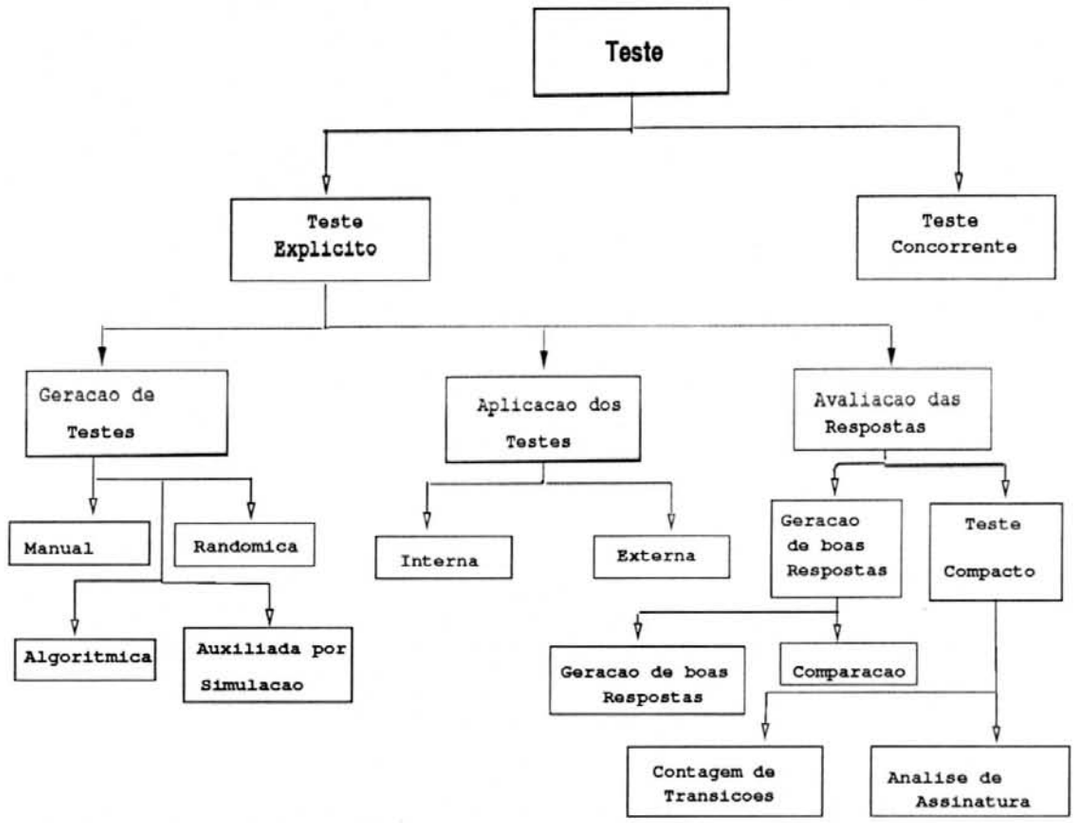


Figura 3.4: Métodos de teste

consegue detectar falhas transientes exatamente por esta sua propriedade de testar sob condições normais de uso. Entretanto, com o teste concorrente não é possível exercitar-se todos os elementos justamente porque utiliza padrões de operação normal, o que nem sempre excita todos os caminhos. Desta forma, podem existir locais do circuito onde existem falhas, porém, que não são detectadas. Outra desvantagem oferecida pelo método é a adição de pinos extras e hardware adicional, e a impossibilidade de verificação de erros de voltagem ou *timing* por não se poder testá-los sob condições especiais visto ser usada a operação normal para a testagem. Assim, o teste concorrente oferece uma menor cobertura de falhas que o teste explícito.

O teste explícito é o teste utilizado durante a manufatura. Neste caso, padrões especiais de vetores de entrada previamente gerados servem como testes e desta forma, a computação normal e o teste ocorrem em momentos diferentes. Como mostrado na figura 3.4, o teste explícito é separado em

três etapas: a geração de testes, a aplicação destes testes e a avaliação das respostas obtidas. Estes passos serão vistos com mais detalhes a seguir, posto que este trabalho trata exclusivamente do teste explícito, mais especificamente, a etapa de geração de testes.

Como o teste explícito é o teste enfocado neste trabalho, mais especificamente a etapa de geração de testes, estes passos serão vistos com mais detalhes a seguir.

Vale ressaltar que existem outras classificações para o teste, por exemplo, teste ativo e passivo [WEY 92].

3.6 Passos do Teste Explícito

O teste explícito pode ser visto contendo os seguintes passos: geração de padrões de teste, aplicação dos padrões no dispositivo a ser testado e avaliação das respostas obtidas no teste. Estes passos não são isolados, dependem um do outro. Por exemplo, se diminuirmos o número de vetores a serem usados para teste, conseguiremos diminuir o tempo de aplicação destes testes e, automaticamente, o tempo de avaliação das respostas.

3.6.1 A Geração dos Testes

As sequências que serão utilizadas para excitar o circuito durante o teste necessitam ser escolhidas e isto influencia diretamente na qualidade deste teste. Por exemplo, utilizando-se todas as combinações possíveis todas as falhas detectáveis serão também detectadas, por outro lado ao utilizar-se somente um padrão existe a possibilidade de detectar poucas falhas, ou até nenhuma.

Como já foi mencionado, a utilização do teste exaustivo é impraticável, logo é necessário a escolha dos padrões ou vetores que serão utilizados no teste. Isto pode ser feito manualmente, pelo projetista, porém, com trabalho excessivo e pouca acuracidade, ou seja, vetores de baixa detectabilidade.

Por isso, várias técnicas são buscadas com o objetivo de automatizar este processo. São as técnicas de geração automática de vetores de teste (ATPG), ou seja, técnicas algorítmicas que baseiam-se em descrições do circuito, sejam elas estruturais ou comportamentais, estejam em formato netlist ou em alguma linguagem de descrição de hardware, como por exemplo VHDL, e geram vetores capazes de detectar as falhas deste circuito descrito.

3.6.2 Aplicação dos Padrões

Como mostrado na figura 3.4, existem dois métodos que podem ser utilizados nesta etapa. O teste externo, que usa equipamentos especiais de teste para aplicar o teste externamente. E o teste interno que é a aplicação interna do teste forçando o dispositivo a executar o processo de *self-testing*. Isto equivale a acionar o teste concorrente citado anteriormente.

Obviamente, este segundo método só pode ser utilizado em sistemas capazes de executar programas, como microprocessadores, por exemplo. Testes externos dão mais controle sobre o processo de teste e habilitam o teste sob diferentes condições temporais e elétricas. Por outro lado, o teste interno é mais fácil de utilizar visto que não necessita de equipamento especial.

3.6.3 Avaliação das Respostas Obtidas

A avaliação das respostas obtidas tem por objetivo detectar respostas errôneas, que indicam a existência de uma ou mais falhas classificando o circuito como bom ou ruim (teste *go/no go*), ou isolar a falha se ela existe, localizando-a (localização da falha).

Como mostra a figura 3.4, a detecção de respostas errôneas pode ser feita de duas maneiras, gerando as respostas boas para posterior comparação, ou com o uso de técnicas de compactação.

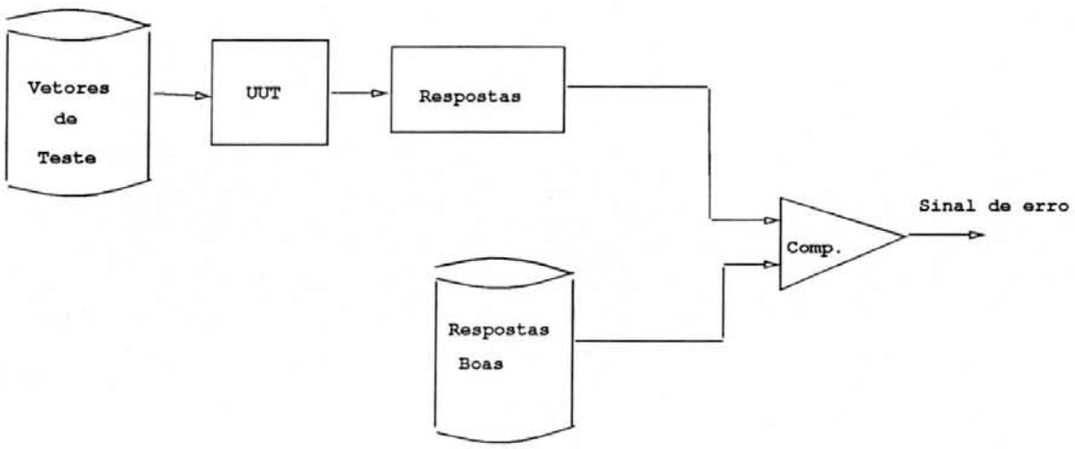


Figura 3.5: Comparação com respostas boas

A utilização da comparação com respostas boas pode ser feita manualmente pelo projetista, o que não é trivial ou com o uso de um simulador. A desvantagem da geração de boas respostas é que se modificarmos o circuito, o processo deve ser totalmente refeito e é necessária a alocação de grandes áreas de memória para armazenar as respostas. A velocidade do aplicador fica, ainda, restrita ao tempo de acesso desta memória que armazena as respostas boas. Na figura 3.5 vê-se o processo de teste utilizando um conjunto de respostas boas e comparando-as às respostas da UUT (*Unit Under Test*).

Outra maneira de aplicar comparando à boa resposta, é utilizando-se um circuito garantidamente sem falhas (*golden unit*). Os padrões de teste

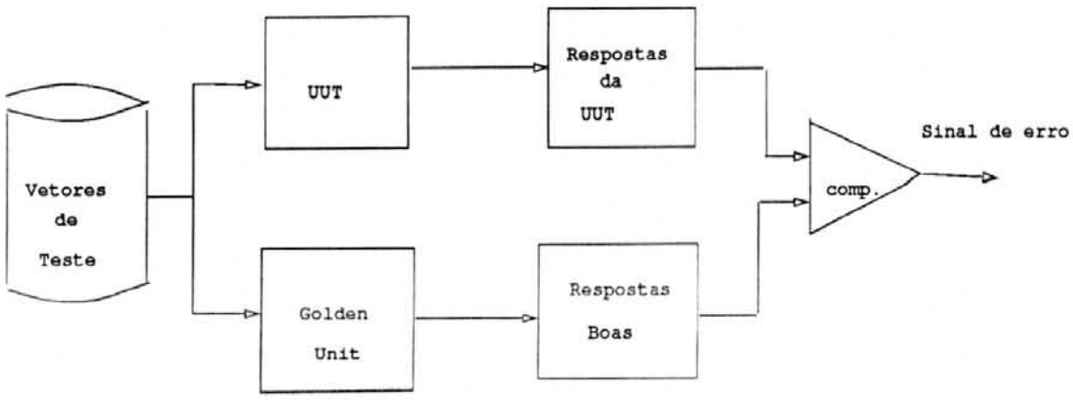


Figura 3.6: Comparação com o uso de *golden unit*

são aplicados simultaneamente no dispositivo que está sendo testado e na *golden unit* e, posteriormente, compara-se as respostas. A principal desvantagem é que cada testador necessita de uma cópia desta *golden unit* (ver figura 3.6) [ABA 83].

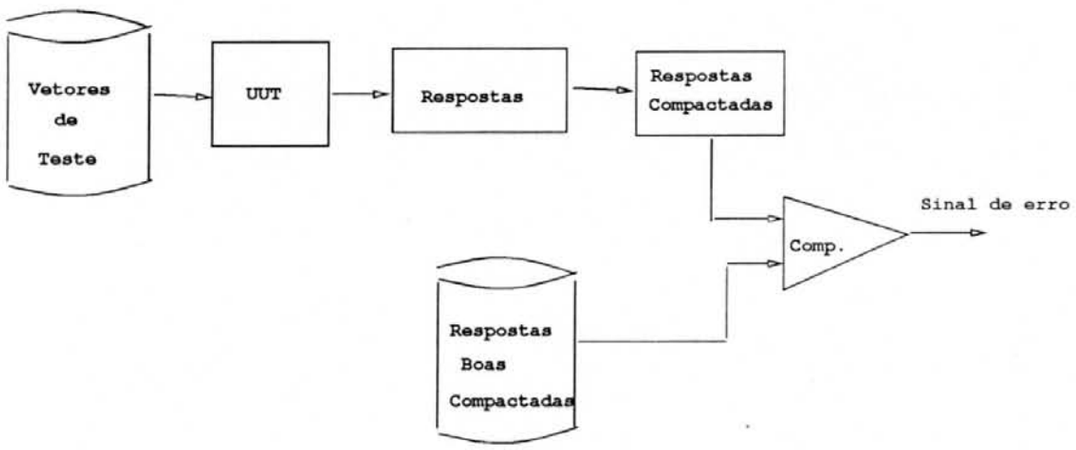


Figura 3.7: Compactação

As técnicas de compactação tentam resolver o problema de armazenamento das respostas diminuindo a necessidade de memória. Nesta técnica, as respostas boas são armazenadas compactadas e após o teste as respostas deste são também compactadas e só então é feita a comparação dos dois conjuntos já compactados (ver figura 3.7).

As técnicas de compactação mais empregadas comercialmente são contagem de transições (*transition counting* [HAY 76]) e análise de assinatura (*signature analysis* [WEY 92]).

A análise de assinatura oferece um mais alto nível de confiança na detecção de respostas falhas que a contagem de transições, porém esta requer um hardware mais simples e menos memória para armazenar as assinaturas boas. Como consequência, o método de contagem é também mais rápido [ABA 83].

3.7 Qualidade do Teste

A qualidade do teste é medida em termos de tamanho do teste e cobertura de falhas. O tamanho da sequência de teste determina o tempo do processo de testagem em um equipamento automático de teste. A cobertura de falhas é definida como a fração de falhas modeladas detectadas pela sequência de teste.

3.7.1 Cobertura de Falhas

A cobertura de falhas é a porcentagem de falhas cobertas por um vetor de teste e é, geralmente, calculada em relação ao conjunto de todas as falhas *stuck-at* simples, após ter sido feita a redução deste conjunto através do colapso de falhas. Para se ter o número de falhas detectadas usa-se um simulador de falhas, desta forma vai-se supondo a existência das falhas possíveis e computando-se quantas são detectadas pela aplicação do vetor de teste.

Matematicamente a fórmula para cálculo é extremamente simples:

$$CF = \frac{FD}{TF}$$

onde CF é a cobertura de falhas, FD as falhas detectadas e TF o total de falhas.

Muitos circuitos grandes, entretanto, possuem algumas falhas redundantes [FRI 67] e, por definição estas falhas não podem ser detectadas por nenhum teste (ver seção 4.3.2 sobre falhas não testáveis). A porcentagem destas falhas é pequena, usualmente menor que 5%. Para suportar esta redundância, pode-se modificar a fórmula de cálculo da cobertura:

$$CF = \frac{FD}{TF - FR}$$

onde CF é a cobertura de falhas, FD as falhas detectadas, TF o total de falhas e FR as falhas redundantes.

Há ainda uma fórmula alternativa, onde o número de falhas redundantes é adicionado ao número de falhas detectadas, ao invés de ser subtraído do total de falhas. O problema nestas fórmulas, contudo, é a estimativa do número de falhas redundantes, pois, às vezes o processamento necessário para identificar a redundância é por demais oneroso.

Supõe-se que a cobertura de falhas dada por padrões randômicos siga a lei exponencial [WIL 85]. Não há porém, uma concordância sobre a cobertura dada por vetores determinísticos.

Não existe um critério universal sobre a cobertura de falhas ideal, isto depende da aplicação, entretanto, normalmente são buscadas coberturas em torno de 90% a 100% de todas as falhas *stuck-at* simples. Por causa da alta

complexidade da simulação de falhas, muitas vezes determina-se a cobertura de falhas pelas fórmulas anteriormente apresentadas mas utilizando-se, ao invés de falhas detectadas entre o total de falhas, uma amostra randômica das falhas.

Pode-se também avaliar a qualidade do teste em relação ao custo do teste.

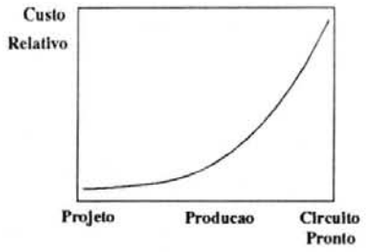


Figura 3.8: Custo relativo X etapa da fabricação do circuito

O custo do reprojeto que advém da detecção da falha cresce quanto mais aproxima-se a fase final da fabricação do circuito (ver figura 3.8). Por exemplo, um erro lógico detectado na fase de projeto pode custar ao engenheiro responsável apenas alguns dias de trabalho. Já um erro encontrado na avaliação do protótipo pode custar dias de alteração e até a confecção de um novo protótipo. O custo de se detectar um erro a nível de placa é dez vezes maior que a nível de chip. Por sua vez uma falha a nível de sistema custa dez vezes mais que a nível de placa.

No custo total da geração de testes pode-se divisar duas áreas de influência: o custo com a geração de testes e o custo com a simulação de falhas.

Os custos aqui relacionados referem-se à uso de CPU, memória, etc. O custo com a simulação de falhas muitas vezes predomina, principalmente em casos onde o circuito é muito grande e/ou sequencial. Assim, uma redução de custo na simulação de falhas influencia diretamente no custo da geração de testes.

Uma solução para isto seria computar a cobertura durante o curso normal da geração de testes com medidas probabilísticas [SET 90].

3.7.2 Complexidade do Teste

A teoria sobre os problemas NP-completos [KAR 72] é talvez o mais importante desenvolvimento teórico na pesquisa algorítmica da década de 70. É um fenômeno inexplicável que, para muitos dos problemas conhecidos e pesquisados, só existem como melhor solução um algoritmo com tempo não-polinomial.

A teoria dos problemas NP-completos não trouxe uma solução para estes problemas em tempo polinomial, entretanto sabe-se que um problema NP-completo tem a propriedade de ser resolvível em tempo polinomial se todos os outros problemas NP-Completo também puderem ser resolvidos em tempo polinomial. Logo, se uma solução for encontrada para um problema NP-completo, todos os problemas que se encaixam nesta categoria também estarão solucionados.

No geral, o problema do diagnóstico de falhas, ou seja, determinar se uma falha *stuck-at* simples pode ser detectada por um experimento de entrada/saída, e o problema da redundância, ou seja, determinar se todas as falhas do circuito são detectáveis, são NP-Completos.

Existem, porém, na área de testes, sub-problemas capazes de serem resolvidos em tempo polinomial, é o caso dos circuitos *unate* com 2 níveis [BRA 90]. Neste caso, os dois problemas, o do diagnóstico e o da redundância, são resolvíveis em tempo $O(m^3)$ e $O(m^2)$, respectivamente, sendo m o número de linhas do circuito [FUJ 82]. Isto é especialmente útil nas técnicas de projeto visando a testabilidade, principalmente, particionamento.

3.7.3 Tamanho do teste

Estimar o tamanho do conjunto de teste para detectar todas as falhas *stuck-at* é possível mas não trivial [KRI 84].

Neste caso advisa-se vários problemas, conhecidos e citados na literatura, como o número mínimo de testes capazes de detectar todas as falhas *stuck-at* presentes em um conjunto de linhas do circuito ou no circuito inteiro, ou ainda, um número aproximado. Todos estes problemas são NP-Completos.

4 PROCEDIMENTOS DE GERAÇÃO DE TESTES

A geração automática de vetores de teste é realizada por um computador logo, deve existir um procedimento algorítmico que execute a função de analisar o circuito e examinar sua topologia, descobrindo o vetor capaz de detectar cada falha, sem ter que utilizar-se do teste exaustivo.

Existem diversos enfoques em relação à implementação dos algoritmos, a enumeração, que é o caso do algoritmo examinado neste trabalho (Algoritmo *D*), técnicas de inteligência artificial, ou ainda, o exame enumerativo inverso, partindo das saídas primárias em direção às entradas.

4.1 O Algoritmo *D*

O algoritmo *D* [ROT 66] é o algoritmo mais clássico da área de geração automática de vetores de teste e baseia-se no fato de que alguns padrões passíveis de serem utilizados para o teste mascaram as falhas, ou seja, não geram resposta diferente da esperada nas saídas. Visando encontrar os padrões que apresentam uma resposta diferente da esperada, de modo que a falha possa ser detectada apenas pela observação dos pinos de saída, o algoritmo é realizado em duas etapas: inserção da falha e propagação da falha de forma a atingir um ponto observável.

A etapa de inserção visa ativar o sinal falho, ou seja, a resposta não esperada no sinal desejado deve se tornar visível. Por exemplo, se o objetivo é testar a saída de uma porta AND *stuck-at*-0 (*grudada-em*-0), o padrão a ser utilizado é o que assume todas as entradas da porta em 1. Assim, o valor esperado na saída é o valor lógico 1, mas caso a linha esteja realmente

grudada-em-0, a porta não vai funcionar como era previsto, criando-se assim uma diferença entre o circuito bom e o circuito com a falha (o sinal D) [ROT 66].

Só isto, entretanto, não é suficiente para garantir a detecção da falha. Se esta linha não for diretamente uma saída primária, outros sinais podem impedir que se observe alguma diferença nas saídas. Para que isto não aconteça o sinal é propagado até a saída. Isto é feito colocando-se nas outras linhas do circuito valores que permitam a passagem da diferença D de modo a ser observável na saída.

4.1.1 Conceitos Fundamentais para o Algoritmo D

No procedimento automático do Algoritmo D usam-se alguns conceitos e notações que são usuais em geração de testes. Estes conceitos são revisados a seguir de forma a auxiliar no entendimento de toda a área de ATPG.

4.1.1.1 Funções lógicas

Uma função lógica (booleana) f é uma função de n entradas e m saídas sobre o conjunto de valores $\{0, 1, X\}$ onde X representa *don't care*. Seja B o conjunto das entradas e Y o conjunto de valores das saídas, pode-se definir: o *ON-set* como o conjunto de valores de entradas $x_i (i = 1, 2, \ldots, n)$ pertencentes a B tal que $f(x_i) = 1$, isto é, a saída da função é o valor lógico 1; o *OFF-set*, o conjunto de valores de entradas x_i tal que $f(x_i) = 0$ e *don't care set*, o conjunto de valores de entrada x_i tal que $f(x_i) = X$.

Muitas representações de funções lógicas são possíveis. A mais utilizada é a forma tabular ou tabela-verdade. Por exemplo, uma função de 2 entradas e uma saída pode ser definida como mostrado na tabela 4.1.

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

Tabela 4.1: Exemplo de tabela-verdade

Para esta função, o *ON-set* é $\{[0,1],[1,0],[1,1]\}$ e o *OFF-set* $\{[0,0]\}$.

4.1.1.2 Cubo

Um cubo é uma representação posicional de uma função lógica que permite armazenar, de maneira compacta, os valores assumidos pela função. Seja p um termo associado com uma expressão algébrica de soma de produtos de uma função lógica de n entradas e m saídas. O cubo de p é especificado por um vetor $c = [x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m]$, onde $x_i = 0$ se x_i aparece complementado em p, $x_i = 1$ se x_i aparece não complementado em p, $x_i = X$ se x_i não aparece em p, $y_i = 0$ se p não está presente na representação algébrica da função e, $y_i = 1$ se p está presente na representação algébrica da função. Por exemplo, para a função:

$$A = B + C$$

Os cubos de representações de A e B se apresentam como na tabela 4.2.

B	C	A
0	0	0
1	X	1
X	1	1

Tabela 4.2: Cubos de representação

O cubo pode ainda ser usado para representar posicionalmente todos os pontos intermediários do circuito. Esta representação é utilizada no procedimento automático do algoritmo D. Neste enfoque, todas as linhas internas do circuito são numeradas e o cubo tem uma posição para cada uma destas linhas (ver figura 4.1)

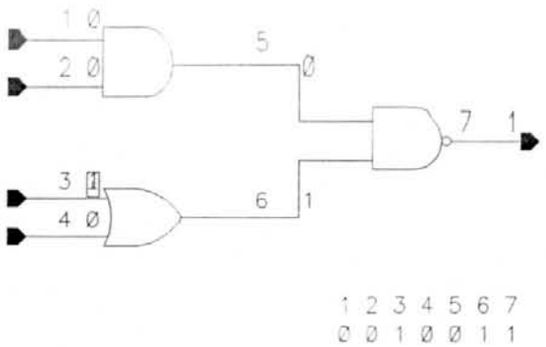


Figura 4.1: Cubo de representação para algoritmo D

4.1.1.3 Álgebra cúbica

A álgebra cúbica define um conjunto de operações que podem ser aplicadas sobre os cubos. Por exemplo, complemento, união e intersecção.

A intersecção entre dois cubos é uma operação que pode ser aplicada entre dois elementos destes cubos de forma a gerar um resultado que segue as seguintes regras:

$$0 \cap 0 = 0 \cap X = X \cap 0 = 0$$

$$1 \cap 1 = 1 \cap X = X \cap 1 = 1$$

$$X \cap X = X$$

$0 \cap 1 =$ intersecção nula ($\emptyset$).

Por exemplo:

$$XXX1 \cap 1X0X = 1X01$$

A intersecção envolvendo diferença segue a definição da tabela 4.3.

$\cap$	0	1	X	D	$\overline{D}$
D	θ	θ	D	μ	γ
$\overline{D}$	θ	θ	$\overline{D}$	γ	μ

Tabela 4.3: Tabela de intersecção dos cubos de propagação

Os símbolos μ e γ são utilizados para detectar se acontece a intersecção de $D \cap \overline{D}$, $\overline{D} \cap D$, $D \cap D$ e $\overline{D} \cap \overline{D}$, o que não é permitido. Porém, caso haja apenas γ sem nenhum μ, troca-se no segundo vetor todos os D's por $\overline{D}$'s e $\overline{D}$'s por D's. Após isto só existiram μ e então intersecciona-se normalmente fazendo $D \cap D = D$ e $\overline{D} \cap \overline{D} = \overline{D}$

4.1.1.4 Cubo primitivo de uma função lógica

Os implicantes primos da função f são todas as combinações de valores de entrada que produzem 1 lógico na saída. Raciocínio análogo se faz caso necessite-se a função $\overline{f}$, porém usando-se todos os valores que geram o valor lógico 0.

Os cubos primitivos de uma função lógica f são a representação em forma de cubo dos implicantes primos de f e $\overline{f}$.

		AB	AB	AB	AB
		00	01	11	10
C	0	0	1	0	0
C	1	1	1	1	1

Tabela 4.4: Mapa de Karnaugh

Por exemplo, seja uma função lógica qualquer dada pelo Mapa de Karnaugh da tabela 4.4.

A	B	C	f	função
X	X	1	1	C
0	1	X	1	$\overline{A}B$
1	X	0	0	$A\overline{C}$
X	0	0	0	$\overline{BC}$

Tabela 4.5: Cubos de representação dos implicantes primos

Os implicantes primos de f são C e $\overline{A}B$ (ou seja, $f = C + \overline{A}B$) e os implicantes de $\overline{f}$ são $A\overline{C}$ e $\overline{BC}$ (ou seja, $\overline{f} = A\overline{C} + \overline{BC}$). A lista de implicantes primos chama-se cobertura singular de f e pode-se representá-la utilizando a notação de cubos (ver tabela 4.5).

4.1.1.5 Cubo D primitivo de uma falha lógica

O cubo D primitivo de uma falha lógica é o cubo que ativa a falha, ou seja, se o objetivo fosse a detecção de uma falha do tipo *stuck-at*-1, de nada adiantaria submeter o circuito a um padrão de entradas que levasse a linha, a qual se testa, para o valor lógico 1, visto que desta maneira não se poderia notar o defeito. Ao se aplicar um conjunto de entradas que gere o valor 0 no sinal e, propagando-se este sinal até um ponto observável, é que o erro pode ser detectado.

Por exemplo, a ativação de uma falha do tipo *stuck-at*-0 em uma porta AND de duas entradas deve ser feita com o padrão apresentado na tabela 4.6, sendo A e B as entradas e C a saída.

A	B	C
1	1	D

Tabela 4.6: Cubo D primitivo para a porta AND

Este padrão é capaz de ativar a falha porque o circuito bom vai responder a este estímulo com o valor 1 na saída C e o circuito defeituoso, ou seja, com a saída permanentemente em 0, produzirá a diferença.

Os cubos D primitivos podem ser determinados a partir dos cubos primitivos de f (a função realizada pela circuito livre de falhas) e f_α (a função realizada pelo circuito falho) como segue: uma entrada produz um erro na saída (D ou $\overline{D}$) se ela está contida em um implicante primo de $f(\overline{f})$ e em um implicante primo de $\overline{f_\alpha}(f_\alpha)$. Para determiná-los então, intersecciona-se as entradas dos implicantes de f e $\overline{f}_\alpha$ encontrando-se as entradas que produzem a diferença (D), ou seja, 1 no circuito bom e 0 no circuito falho. Intersecciona-se também os implicantes primos de $\overline{f}$ e f_α encontrando-se as entradas que produzem a diferença complementada $\overline{D}$, ou seja, 0 no circuito bom e 1 no circuito falho. Para as portas lógicas AND, OR, NOR, NAND, XOR e XNOR, basta usar o conjunto de entradas que gera o valor lógico 1, como cubos de inserção da falha D e o conjunto de entradas que gera o valor lógico 0 como cubos de inserção da falha $\overline{D}$.

4.1.1.6 Cubo de propagação da falha

Os cubos de propagação são os cubos que permitem a observação da diferença na saída e são utilizados de forma a que o sinal alcance um ponto observável, que no teste de um circuito encapsulado são as saídas primárias.

Estes cubos devem satisfazer a seguinte premissa: os valores a serem colocadas nas outras linhas que não a que apresenta a diferença devem ser tais que não interfiram na observação deste sinal na saída da porta.

Para ilustrar este conceito pode-se observar a figura 4.2 que apresenta uma porta AND de três entradas. Todos os outros sinais que não apresentam a diferença devem assumir o valor 1, visto que é o valor que, nesta

função, não interfere na observação da diferença saída. Logo a diferença poderá atingir o ponto Y e com isto foi propagada.

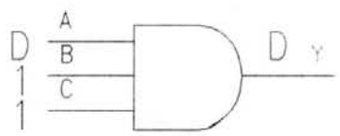


Figura 4.2: Porta AND de três entradas

Estes cubos são encontrados selecionando-se os implicantes primos de f (função realizada pelo circuito livre de falhas) que apresentem 1 na linha-alvo (ou seja, a linha que vai ser propagada), com os implicantes primos de $\overline{f}$ que apresentem 0 na referida linha. Depois, intersecciona-se estes cubos, colocando-se D em intersecções de $0 \bigcap 1$ e $\overline{D}$ em $1 \bigcap 0$. A intersecção é realizada novamente colocando-se D em intersecções de $1 \bigcap 0$ e $\overline{D}$ em $0 \bigcap 1$. Os cubos resultantes da intersecção constituem os cubos de propagação da falha.

4.1.2 Operações Fundamentais na Geração de Testes

As etapas já mencionadas de inserção da falha e propagação desta, bem como a justificação das constantes e o *backtracking* devem ser feitas por um procedimento automatizável, que seja facilmente implementado em uma linguagem de programação. Este é o enfoque com que elas serão analisadas a seguir.

4.1.2.1 Procedimento algorítmico de inserção da falha

A inserção da falha é feita assumindo-se o cubo de inserção da falha como cubo inicia, tendo-se o cuidado de organizar posicionalmente os sinais.

Na figura 4.3, vê-se o procedimento de forma gráfica. O cubo $[1\ X\ D]$, ou seja, 1 e X como entradas (sinais 3 e 5) e D como saída (sinal 6) da porta

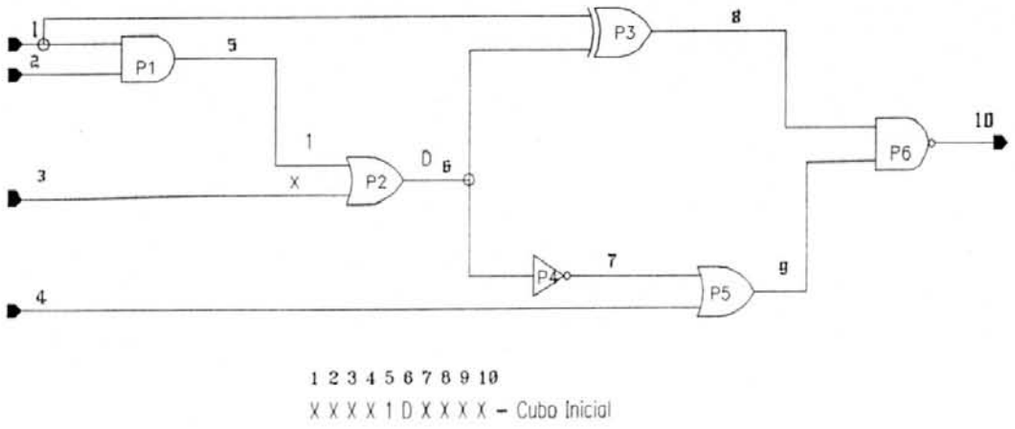


Figura 4.3: Cubo inicial

OR, é colocado posicionalmente no cubo inicial nas posições correspondentes, e as demais posições são preenchidas com *don't care*.

Após isto, o algoritmo deve verificar se alguma constante foi inserida. Neste caso deve-se ativar o processo que justifica os valores até as entradas primárias, explicado a seguir.

4.1.2.2 Procedimento algorítmico de justificação das constantes

Para a realização desta justificação são utilizados os cubos primitivos da função de cada porta envolvida. Para cada constante inserida deve-se verificar a posição do sinal no circuito verificando qual a função da porta que tem o valor constante como saída. Os cubos primitivos de f, caso a constante seja 1 ou de $\overline{f}$ no caso de 0, devem ser então interseccionados com o vetor corrente, ou seja, o vetor com todos os sinais e D's do momento.

Se nenhum vetor resultar numa intersecção com sucesso, o algoritmo deve voltar atrás e desfazer a inserção da constante, tentando substituir por alguma outra opção que tenha ficado empilhada (ver seção 4.1.2.4 sobre *backtracking*).

Na figura 4.4 pode-se verificar o cubo inicial, com o cubo de inserção da falha. Posto que a constante foi inserida na linha 5, que é a saída de uma

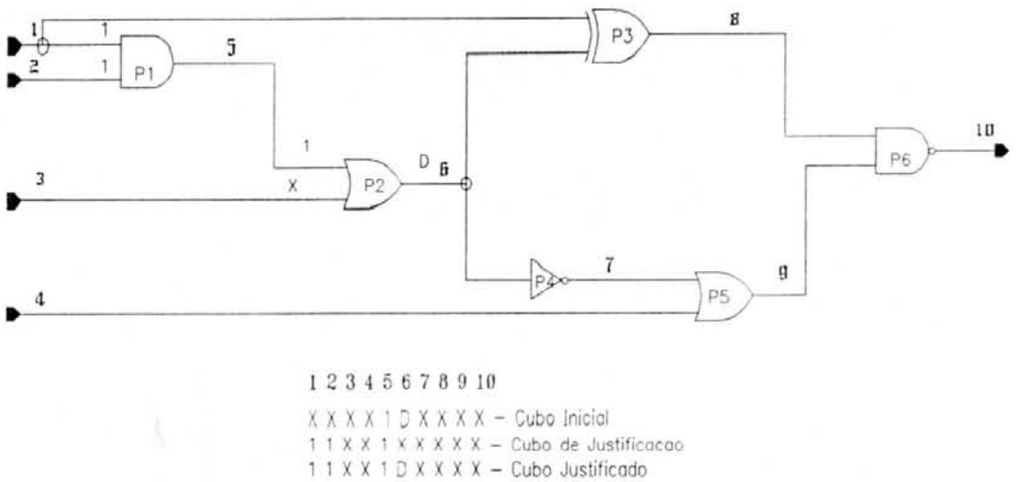


Figura 4.4: Justificação das constantes

porta AND, e a porta AND tem os seguintes cubos primitivos, [0 X] e [X 0] que geram 0 e [1 1] que gera 1, e a constante a ser justificada é o 1, na linha 5, faz-se a escolha do cubo [1 1], que é o que gera o valor lógico 1. Este cubo deve, então, ser interseccionado com o cubo corrente de modo a obter-se o cubo resultante.

4.1.2.3 Procedimento algorítmico de propagação da falha

A propagação da falha é feita pela intersecção do cubo corrente com o cubo de propagação, que é o cubo capaz de deixar o sinal diferença atingir a saída da porta. Por exemplo, na porta OR, o padrão capaz de propagar o sinal diferença é [D 0] que resulta D ou [$\overline{D}$ 0] que resulta $\overline{D}$ (ver figura 4.5). O padrão [D 1] não permite a passagem do sinal D, visto que sempre produz o valor lógico 1 na saída.

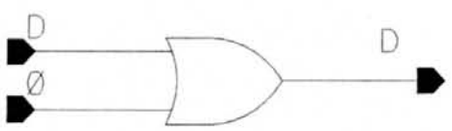


Figura 4.5: Propagação em uma porta OR

A propagação nas outras portas é feita de modo análogo.

O procedimento algorítmico é feito de seguinte forma: identifica-se o objetivo, ou seja, a linha onde o D está presente e que deve ter seu sinal propagado. É gerada uma lista de todos os caminhos, por onde este sinal pode ser sensibilizado, que são as portas onde este sinal está ligado, logo, os caminhos por onde o sinal pode alcançar a saída primária. Estas portas são intituladas a *fronteira D*.

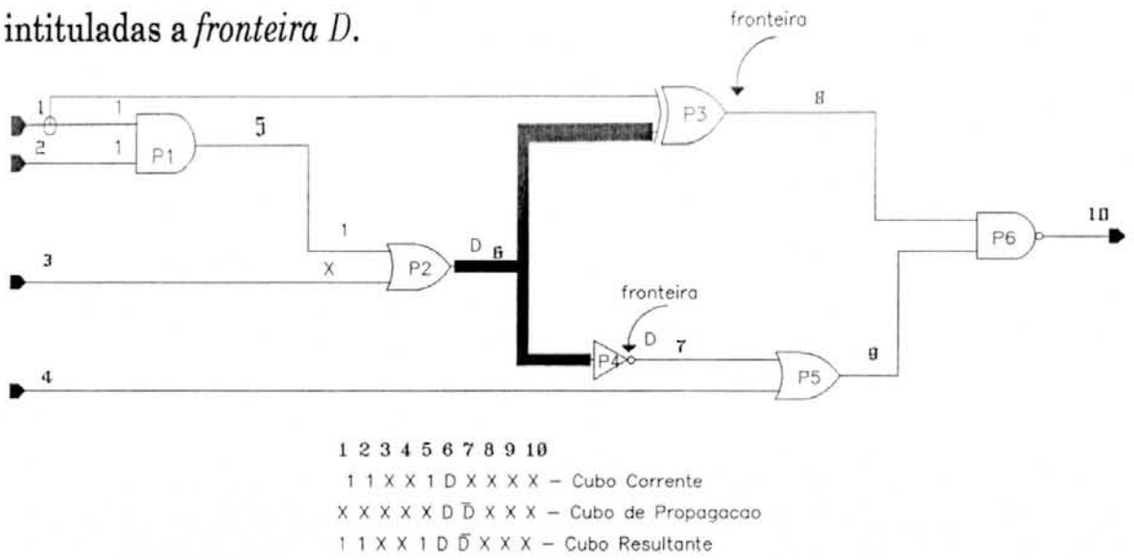


Figura 4.6: Fronteira D e propagação por P4

Na figura 4.6, a propagação do D pode ser feita pela porta P3 ou por P4, para alcançar a linha 10 (saída primária). A fronteira D é, portanto, P3 e P4. O algoritmo deve então escolher uma das opções da fronteira para propagar o sinal. As outras opções ficam empilhadas. O algoritmo original não utiliza nenhum critério para escolher entre estas opções. De acordo com a porta escolhida da fronteira seleciona-se os cubos de propagação da falha e é feita a intersecção.

Na propagação utiliza-se também a colocação de constantes de forma a permitir que o sinal seja observado. Estas constantes devem também ser justificadas, conforme o procedimento explicado anteriormente.

4.1.2.4 *Backtracking*

Por vezes o algoritmo escolhe um caminho que causa um conflito futuro. Quando isto acontece, o procedimento deve desfazer as operações de modo a tentar resolver tentando outros valores. Isto fica mais claro observando-se a figura 4.7.

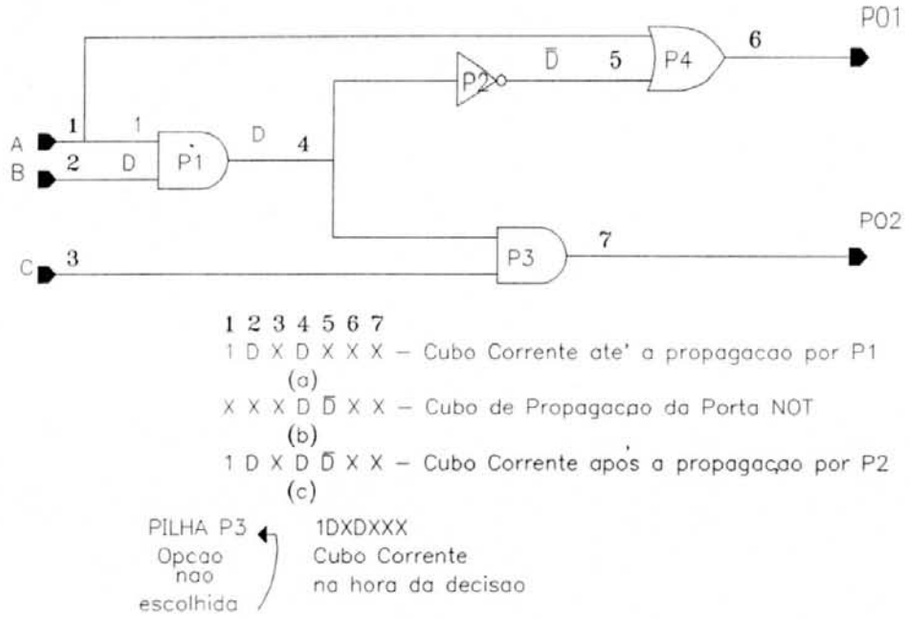


Figura 4.7: *Backtracking*

No circuito da figura 4.7, a diferença D já foi propagada até a linha 4, ou seja, a saída da porta AND P1. Para que alcance o ponto observável, ou seja, as saídas primárias, é necessário ainda propagar o D até uma delas.

Para isto existem dois caminhos, ou seja, a fronteira D se compõe de dois elementos P2 e P3. O algoritmo não usa nenhum critério, assim, uma das alternativas é empilhada junto com o cubo corrente de modo que se possa restaurar o momento corrente caso a tentativa malogre. A outra opção, logo, é a escolhida.

Supondo que o algoritmo escolha P2, P3 é colocada na pilha junto com o estado corrente naquele instante de forma a poder voltar caso a opção escolhida resulte em insucesso.

Uma alternativa ao uso da pilha, é desfazer todas as operações, *desinterseccionando* o vetor. Este procedimento despende mais processamento, enquanto que a alternativa anterior despende memória. O resultado final, porém, é o mesmo. Nesta explanação será usado o primeiro método.

Na propagação por P2 obtém-se, como cubo resultante, a intersecção do cubo corrente com o cubo de propagação da porta NOT (ver figura 4.7). Como a intersecção resultou em sucesso, o cubo corrente após a propagação por P2 é empilhado (ver tabela 4.7) para que tente-se voltar o *backtracking* mantendo sempre o cubo mais recente.

P3	1	D	X	D	$\overline{D}$	X	X
P3	1	D	X	D	X	X	X

Tabela 4.7: Pilha do caminho não utilizado armazenado

Depois da propagação por P2, P4 passa a ser a única porta na fronteira D. Assim, o cubo corrente ($1DXD\overline{D}XX$) é interseccionado com o cubo de propagação da porta OR ($0XXX\overline{DD}X$, ver figura 4.8), porém a intersecção não obtém sucesso, já que o valor necessário para a propagação na linha 1 é 0, e o valor 1 já está presente, o que configura um conflito.

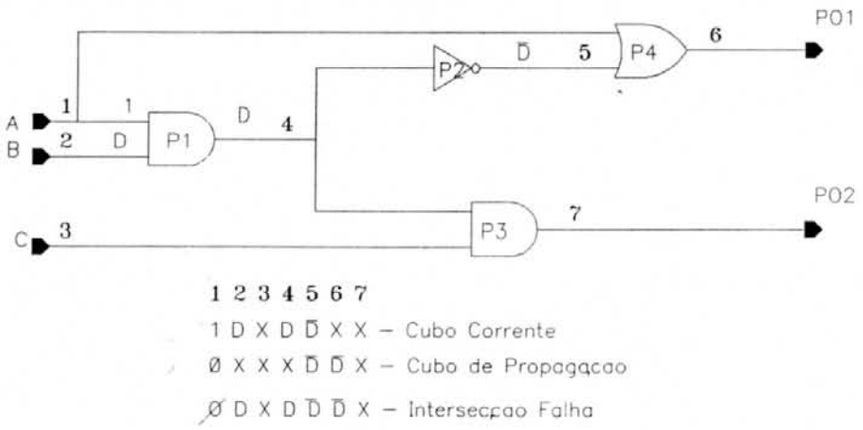


Figura 4.8: Conflito

Por este motivo, a operação de *backtracking* deve ser acionada. O algoritmo verifica se alguma opção ficou empilhada. Em caso afirmativo, e é

o caso do nosso exemplo, a opção que ficou reservada (P3) é então utilizada da seguinte maneira: o cubo empilhado passa a ser o cubo corrente e a porta P3 passa a ser a fronteira D. Note-se que o cubo que vai ser utilizado como corrente é $1DXD\overline{D}XX$ porque usa-se o cubo que mais recentemente havia sido interseccionado com sucesso.

A propagação, então é feita pela intersecção do cubo corrente com o cubo de propagação da porta P3, $[D\ 1]$, resultando D. O cubo final é mostrado na figura 4.9.

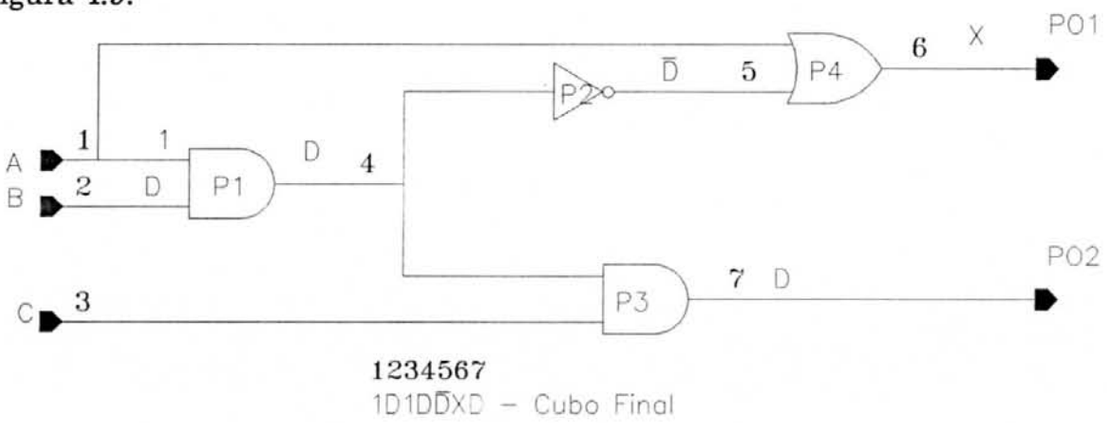


Figura 4.9: Cubo final

É importante ressaltar que, caso a opção desempilhada não pudesse ter sido interseccionada com sucesso, o outra opção que havia sido empilhada anteriormente poderia ser utilizada, e assim sucessivamente.

4.2 Otimizações do Algoritmo D

Durante a geração, é possível saber que o vetor que está sendo gerado já detecta outras falhas além da falha-alvo. Isto é muito útil porque, a partir desta constatação, pode-se tirar a falha também detectada da lista de falhas, o que diminui o tempo dispendido na geração de testes, e também o número de testes gerados, logo diminui também o tempo de aplicação dos

padrões. Esta otimização, entretanto, serve apenas para testes que não objetivam diagnóstico.

4.2.1 Caminho D

O caminho D é o caminho de propagação utilizado pelo algoritmo para levar o valor até a saída, ou seja, são todas as linhas que, durante a geração receberão um D (ou $\overline{D}$).

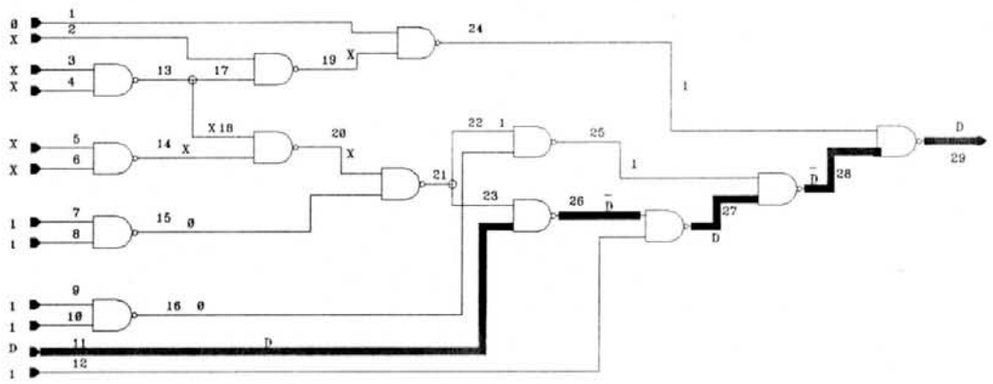


Figura 4.10: Caminho D

Na figura 4.10, o caminho das linhas 11, 26, 27, 28 e 29, que receberam um D na geração do padrão que detecta a falha 11-a-0 (linha 11 com *stuck-at*-0).

4.2.2 Reconvergência

Um fan-out é dito reconvergente quando, após um número n de conexões e portas, os caminhos gerados da bifurcação do fan-out reconvergem, isto é, se encontram novamente como entradas da mesma porta lógica.

4.2.3 Otimização para Circuito de uma só Saída

Teorema 1 - *Para circuitos com uma só saída, um conjunto de teste que detecta todas as falhas simples nas entradas primárias e caminhos reconvergentes, detecta todas as falhas simples do circuito* [ROT 82].

Uma importante conclusão deste teorema é que a lista de falhas pode ser derivada do circuito pela listagem de apenas as entradas primárias e posições de fan-outs reconvergentes o que diminui drasticamente a lista de falhas.

Na figura 4.10, a lista de falhas seria a lista das entradas primárias, de 1 a 12, e as falhas nas linhas 17, 18, 22 e 23, que são pontos de divergência que reconvergem posteriormente.

Com isto consegue-se uma boa redução, entretanto, seria necessário, ainda, a verificação do colapso de falhas, citado anteriormente, de modo a reduzir ainda mais a lista de falhas. Neste caso, por exemplo, as falhas 3-a-0 e 4-a-0 são equivalentes, logo uma poderia ser descartada.

4.2.4 Otimização para Circuitos com Mais de uma Saída

Há, basicamente, dois métodos a serem utilizados em circuitos com mais de uma saída. O primeiro é a lógica segmentada, que consiste em tratar-se uma saída de cada vez, o que diminui a complexidade. Entretanto, tem como desvantagem o tempo gasto em falhas já detectadas pelo outro caminho e o tempo de compactação dos testes em vista do grande número de testes gerados. O segundo método gera testes para as múltiplas saídas, porém o algoritmo é bem mais complexo.

Teorema 2 - *Para um circuito com múltiplas saídas, um teste que detecta todas as falhas nas entradas primárias e fan-outs, reconvergentes ou não, detecta todas as falhas simples do circuito.* [ROT 82]

4.3 Parâmetros Capazes de Influenciar o Teste

A geração de testes é feita mediante a observação de uma descrição do circuito, logo, a maneira como este circuito se comporta ou está estruturado, pode facilitar ou dificultar o teste. Vários parâmetros, tais como, estruturais, lógicos e até temporais, podem influenciar o teste.

4.3.1 Estrutura

A estrutura do circuito é a disposição das portas e conexões, a existência de fan-ins ou fan-outs, o número de saídas, de entradas, etc..

Na geração de testes, a presença dos fan-outs, ou seja, conexões que se bifurcam gerando múltiplos caminhos (ver figura 4.11), é um fator muito relevante, ou até, essencial para a geração de conflitos, e, conseqüentemente, para o aumento de número de *backtrackings*.

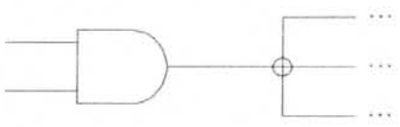


Figura 4.11: Fan-out

Um fan-out reconvergente é gerador em potencial de conflitos, exceto se o valor colocado na linha reconvergente for, casualmente, o valor necessário para a propagação do sinal.

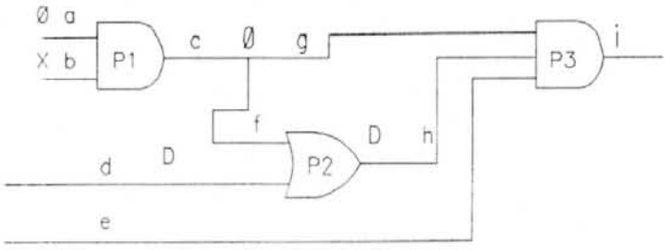


Figura 4.12: Circuito reconvergente com conflito

No circuito da figura 4.12, por exemplo, para propagar o sinal D na linha d foi necessário colocar 0 na linha c, o que implica em 0 e X em a e b, respectivamente. O fan-out da linha c é reconvergente, ou seja, seu caminho $c-g-i$ até a saída primária se encontra novamente com outro caminho $c-f-h-i$ na porta P3. Com isto, o valor 0 colocado na linha c gerou um conflito na propagação pela porta P3, pois era necessário o valor lógico 1 nas linhas g e e para que a diferença pudesse ser observada.

4.3.2 Falhas Não Testáveis

Falhas não testáveis, são falhas que não podem ser detectadas por um procedimento normal de teste com um aparelho de testagem funcional. Estas falhas são também conhecidas como falhas indetectáveis.

Existem quatro mecanismos que podem causar uma falha indetectável:

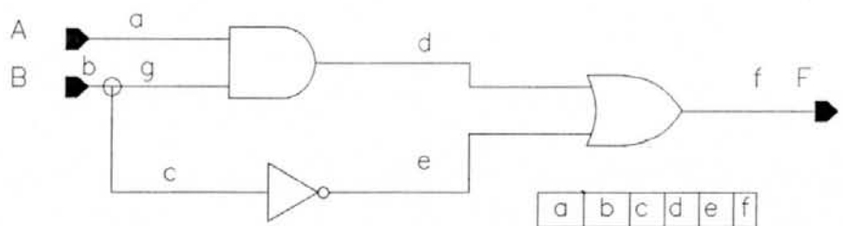
1) **Redundância.** A redundância presente nos circuitos é um dado muito relevante na geração de testes. Por este motivo faz-se necessária a apreensão deste conceito para o futuro entendimento de sua influência nas técnicas de ATPG.

Uma linha de um circuito combinacional C é dita ser irredundante quando, ao alterar-se o valor lógico, a função de C é alterada.

Se a linha l é irredundante, então existe um teste apropriado para detectar as duas falhas: l-a-0 e l-a-1. Se existe apenas um teste para detectar uma das falhas l-a-0 ou l-a-1, então o sinal l é parcialmente redundante. A linha totalmente redundante não tem teste nem para l-a-0, nem para l-a-1. O circuito C é dito irredundante quando cada linha dele é irredundante.

No caso de uma linha não possuir esta propriedade, o circuito é dito redundante e, neste caso, pode-se deletar a referida linha sem prejuízo para o funcionamento da função lógica. Pode-se afirmar ainda que, na presença de uma linha redundante tem-se uma falha indetectável.

Por exemplo, no circuito da figura 4.12 vê-se claramente que a linha e é irredundante. Montando-se a tabela-verdadade fica claro que a alteração do sinal e é capaz de gerar uma mudança na função.



a	b	c	d	e	f
Ø	Ø	Ø	Ø	1	1
Ø	1	1	Ø	Ø	Ø
1	Ø	Ø	Ø	1	1
1	1	1	1	Ø	1

Figura 4.13: Circuito Redundante

Na figura 4.13, entretanto, a linha g é redundante, visto que nenhuma alteração no seu valor é capaz de causar alteração na função. Note-se porém que o fato do circuito ser reconvergente não significa sua redundância.

Analisando-se superficialmente, a impressão dada é que a presença de uma falha indetectável em uma linha redundante é insignificante, já que o circuito funciona perfeitamente a despeito da falha. Contudo, a presença da redundância não só torna o processo da geração de testes mais difícil, como também invalida alguns padrões no caso do uso de vetores randômicos.

2) **Dependências Internas de Sinais.** Pode acontecer que, por causa da estrutura das conexões, o circuito tenha pontos em que nunca pode ocorrer determinado padrão. Na figura 4.14 não é possível a existência de um sinal lógico 1 nas duas entradas da porta OR. Nesta situação, porém, uma falha que muda a função da porta só para entradas que nunca podem ocorrer não tem efeito no funcionamento do circuito. No exemplo, uma falha deste tipo é a mudança da porta OR por uma porta XOR. Esta falha não pode ser detectada por um teste booleano.

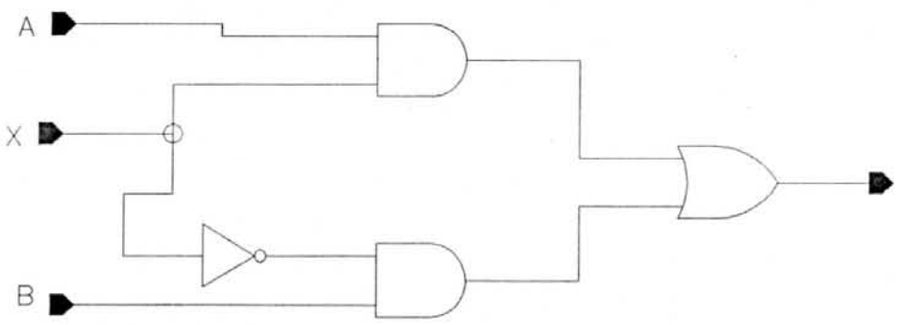


Figura 4.14: Dependência do Sinal

3) **Controle de Hazards.** Se parte do circuito está presente com o objetivo de eliminar *hazards*, falhas que desabilitem este controle não serão detectadas visto que não afetam a operação estática do circuito.

Na figura 4.15, a porta AND A3 só assume o valor lógico 1 quando as entradas A e B têm 1. Neste caso, entretanto, uma das saídas das outras portas AND, $A1$ ou $A2$, vai ser 1, o que não permite a propagação da falha. A porta AND A3 está presente apenas para prevenir um *hazard* estático, logo,

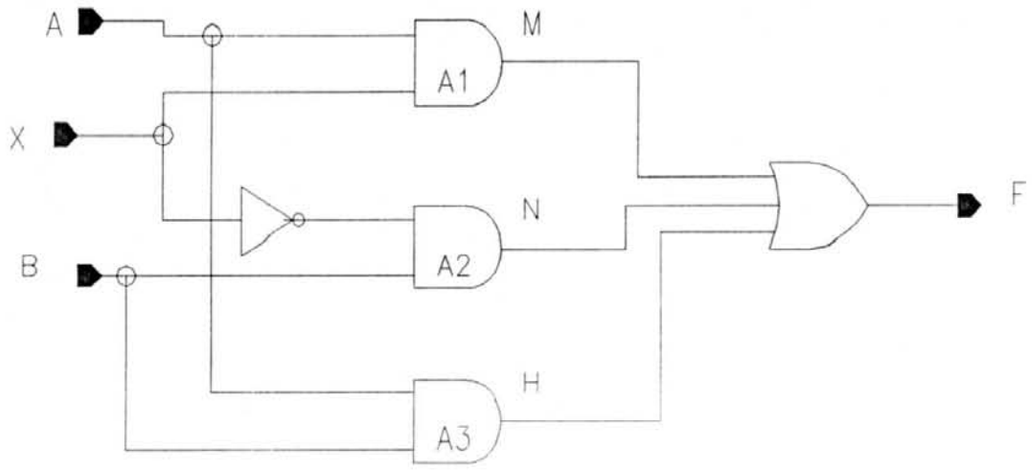


Figura 4.15: Controle de *hazards*

uma falha nesta porta não pode ser detectada por um teste booleano. É uma porta redundante, não faz nenhuma diferença na função lógica do circuito.

4) **Controle de Erros.** Por vezes o circuito é adicionado de partes que somente servem para detectar erros enquanto o sistema está em uso. Estes circuitos também têm sempre partes indetectáveis.

4.4 O Problema das Falhas *Difíceis* na Geração de Testes

Uma grande parcela das falhas dos circuitos combinacionais [GOE 81] são de fácil detecção por padrões randômicos, isto é, muitos padrões detectam esta falha e os algoritmos de geração de testes conseguem rapidamente encontrar um destes padrões, sem necessitar, inclusive, de heurísticas adicionais.

Estas falhas são chamadas de *randomicamente fáceis / algoritmicamente fáceis* (RF/AF) de detectar. Como estas falhas tem facilmente um vetor de teste, elas não são consideradas um problema para o teste.

Existem falhas que têm baixa probabilidade de detecção por padrões randômicos, são as chamadas *randomicamente difíceis* (RD) de detectar. Estas falhas necessitam de um algoritmo dotado de alguma heurística para serem detectadas, por exemplo, PODEM ou outra, mas, como existem heurísticas capazes de permitir ao algoritmo a fácil detecção destas falhas, elas são consideradas *randomicamente difíceis / algoritmicamente fáceis* (RD/AF) de detectar.

Muitos circuitos geralmente grandes, porém, têm falhas difíceis de detectar com padrões randômicos e não tem algoritmo capaz de gerar facilmente vetores de teste, apesar das heurísticas, dentro de um razoável número de *backtrackings*, ou seja, 1000 [IVA 88]. Estas falhas são denominadas *randomicamente difíceis / algoritmicamente difíceis* (RD/AD) de detectar.

Muitas vezes estas falhas estão presentes em circuitos redundantes, mas não necessariamente. Em alguns casos a falha é indetectável, ou seja, não existe um padrão de teste para detectar esta falha que possa ser encontrado rapidamente por um algoritmo de geração automática de vetores de teste.

Em geral o número de falhas RD/AD é bem pequeno, entretanto este número pode ser crítico para aumentar, por exemplo, a cobertura de falhas de 90% para 99%. Por isto a motivação deste trabalho em encontrar maneiras de melhorar a performance do Algoritmo *D*.

O problema encontrado é detectar quais falhas são RF/AF, RD, RD/AF ou RD/AD. Com este objetivo buscou-se medidas capazes de quantificar a probabilidade de detecção das várias falhas. Estas medidas constituem as medidas de testabilidade, que serão expostas a seguir.

4.5 Testabilidade

A análise da testabilidade do circuito é importante não só para a geração de testes mas também para auxiliar o projetista na construção do projeto de forma a resolver previamente áreas difíceis de testar.

A medida de testabilidade visa quantificar a propriedade da testabilidade em um circuito e pode ser vista a partir de dois outros conceitos: observabilidade e controlabilidade.

4.5.1 Controlabilidade

A controlabilidade é a medida da capacidade de controle do valor (0 ou 1) em uma determinada localização do circuito. É a quantificação da facilidade pertinente ao nodo, de receber um determinado valor.

Existem medidas de controlabilidade que distinguem valor, ou seja, calculam a controlabilidade para o valor 1 e 0, distintamente (por exemplo, SCOAP [GOL 80]) ou que calculam um valor único (por exemplo, CAMELOT [BEN 80]).

Como um exemplo de valor mais ou menos controlável tem-se a figura 4.16. Neste caso, vê-se claramente que a obtenção de um valor 0 na linha X é muito mais controlável. Isto acontece porque em uma porta AND é muito mais fácil a obtenção de um valor 0, pois basta atribuir 0 a uma das linhas para que a saída assuma este valor. Já o valor 1 só é conseguido com a colocação de todas as entradas em 1. Isto é especialmente importante quando têm-se vários níveis envolvidos, o que dificulta a obtenção destes valores.

Assim a controlabilidade pode ser vista por dois ângulos, o valor mais controlável de acordo com a função da porta, por exemplo 0 na AND e

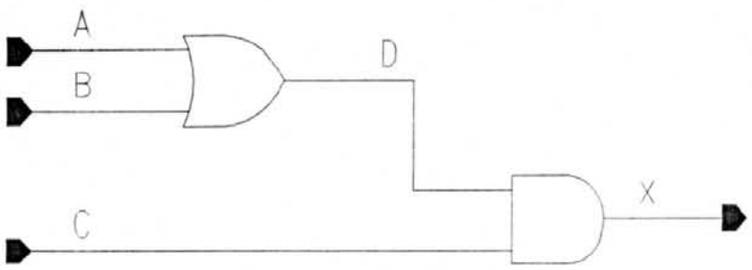


Figura 4.16: Controlabilidade

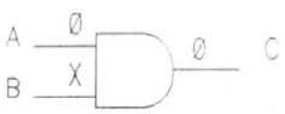


Figura 4.17: Valor não observável

NAND, 1 na OR e NOR, ou ainda, a distância, ou melhor, o número de sinais que são necessários para que determinado valor possa estar presente na linha.

Este último enfoque também pode ser verificado na figura 4.16, onde claramente se divisa a facilidade muito maior da atribuição do valor 0 para a linha B, que é uma entrada primária, logo, o mais controlável possível. A atribuição do valor 0 na linha D implicaria na colocação de 0 em A e B, ou seja, mais dois sinais, que eventualmente já podem até conter valores conflitantes com o valor necessitado.

4.5.2 Observabilidade

A observabilidade é a medida da facilidade de obtenção de determinado valor na saída de uma porta lógica.

Nem todos os valores lógicos (0 ou 1) permitem a observação, por exemplo, numa porta AND, o valor lógico 0 não permite a observação dos outros valores (ver figura 4.17).

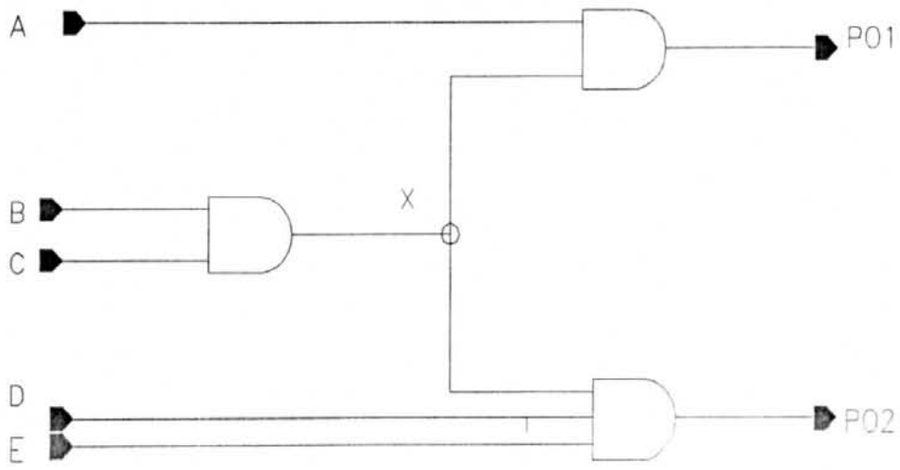


Figura 4.18: Observabilidade

Outro fator importante na observabilidade é que, quanto mais valores forem necessários para a propagação, menos observável é o sinal. Na figura 4.18, o valor de X pode ser melhor observado na saída PO1, onde basta setar A para 1, que na saída PO2, visto que para propagar o sinal seria necessário que D e E assumissem o valor 1. Neste caso, isto pode não parecer deveras relevante, já que estas linhas são entradas primárias. Porém, normalmente, os sinais têm uma lógica anterior, que necessita ser totalmente justificada.

4.5.3 Fragilidade da Observabilidade e Controlabilidade

Apesar de muitas das heurísticas que serão vistas basearem-se na controlabilidade e observabilidade dos sinais, nem sempre uma boa observabilidade e controlabilidade garantem uma boa testabilidade.

As medidas existentes na literatura apresentam as seguintes limitações [SAV 83]:

- Falham na presença de fan-outs reconvergentes;
- As medidas de testabilidade são definidas baseadas na boa controlabilidade e boa observabilidade e isto nem sempre é verdade;

- Não são medidas precisas e assim, por vezes fazem o projetista adicionar hardware, para facilitar o teste, em lugar errado.

5 HEURíSTICAS DE ACELERAÇÃO DA GERAÇÃO DE TESTES

Visando acelerar a execução dos algoritmos de geração de testes, é possível fazermos uso de heurísticas que auxiliem o processamento das respostas, otimizando partes do algoritmo que consomem muito tempo. Uma dessas tarefas é a consistência ou justificação das constantes. Como visto anteriormente, o Algoritmo D realiza, durante a operação de propagação e inserção da falha, a colocação de valores que são necessários para que os sinais atinjam a saída. Estas constantes, entretanto, necessitam de uma operação de consistência para a garantia de que não conflitem com outros sinais já presentes nas linhas. Quando isto acontece, o algoritmo necessita voltar atrás em todas as suas decisões e escolher um outro caminho. Esta operação é chamada de *backtracking* e despende uma quantia considerável de processamento, visto que pode ser por muitas vezes, escolhido um caminho inconsistente obrigando o algoritmo a desfazer várias operações até encontrar a solução.

Logo, a eficiência do processo de inserção de constantes está diretamente associada com a habilidade de reduzir o número de *backtrackings* necessários. Uma busca ideal do caminho não realiza nenhum *backtracking*, ou seja, escolhe o caminho exato onde todas as constantes podem ser inseridas sem conflito com os valores já presentes nas linhas e sem prejudicar futuras inserções necessárias nas futuras propagações do sinal D. O pior caso é a busca que precisa esgotar todos os caminhos possíveis só encontrando a solução na última tentativa.

Quanto mais constantes inseridas, mais chances de conflitos com outros valores já presentes nas linhas e quanto maior o número de divergências de dados (mais de um fan-out), maior a possibilidade de conflitos com inserções futuras.

Baseadas nestas idéias, muitas funções foram idealizadas para guiar o processo de decisão na busca do caminho a ser escolhido. Estas funções são baseadas em valores calculados heuristicamente e influenciam de maneira muito significativa na performance do gerador, já que podem determinar o número de *backtrackings* necessários.

Dividiu-se, neste trabalho, as heurísticas em classes, de modo a organizá-las de acordo com o parâmetro que pretendem explorar. Chamou-se *Heurísticas Funcionais* as heurísticas baseadas na função das portas, explorando dominância ou probabilidade de obtenção de determinados valores lógicos. Às *Heurísticas Estruturais* atribuiu-se as técnicas que observam a estrutura das conexões, tratando o problema da reconvergência. Chamou-se, ainda, de *Técnicas Probabilísticas*, as técnicas que penalizam a probabilidade de obtenção de determinado valor e *Técnica de Pré-implicação* à Técnica de pré-cálculo de todos os cubos. Agregou-se ainda algumas técnicas menos específicas e menos eficientes, que devem ser usadas apenas como critério de desempate, sob o nome de *Técnicas Diversas*.

5.1 A Utilização de Medidas de Testabilidade

O algoritmo PODEM [FUJ 83] foi o pioneiro na utilização de medidas de testabilidade no auxílio à escolha de caminhos no sentido de diminuir o número de *backtrackings*. O PODEM é o algoritmo D com heurísticas funcionais embutidas de forma a auxiliar as decisões. O critério utilizado é a controlabilidade e observabilidade dos caminhos calculados a partir de fatores funcionais (dominância ou probabilidade).

Na geração de testes, o algoritmo cria uma árvore de decisão onde há mais de uma escolha disponível em alguns nodos de decisão. Se a escolha é arbitrária, como no algoritmo D na sua forma original, pode ser necessária

durante a execução, que o algoritmo retorne e tente outro caminho. Esta operação é chamada *backtracking* e, como visto anteriormente, é o fator que diminui a performance da geração de testes.

A heurística adotada pelo PODEM na operação de escolha de constantes é a seguinte:

- Se o objetivo corrente é tal que pode ser obtido setando-se o sinal em 0 ou 1, escolhe-se o estado mais controlável, por exemplo, 0 para uma porta AND/NAND e 1 para OR/NOR.
- Se o objetivo corrente não pode ser obtido desta forma, ou seja, só pode ser obtido setando-se todas as entradas da porta para um estado não controlável, então escolhe-se primeiro a entrada mais difícil, ou seja, com menor controlabilidade.

Deste modo, consegue-se eliminar primeiro as alternativas mais fadadas ao insucesso, o que diminui a chance de se realizar trabalho desnecessário.

Para esta heurística podem ser utilizados as medidas de controlabilidade/observabilidade funcionais citadas posteriormente.

Na operação de propagação, o algoritmo gera a fronteira D, que contém os possíveis caminhos de propagação do sinal. Para decidir entre as escolhas, o PODEM adota heurísticas usando medidas de observabilidade, ou seja, escolhe a porta que está mais próxima à saída primária.

Para reduzir o número de *backtrackings* é importante a detecção, tão cedo quanto possível, de prováveis conflitos.

Uma heurística que o PODEM faz, neste sentido, é determinar os sinais que têm uma única opção de justificação, ou implicação, e atribuí-los todos de uma vez. Assim, diminui-se as chances de atribuições conflitantes.

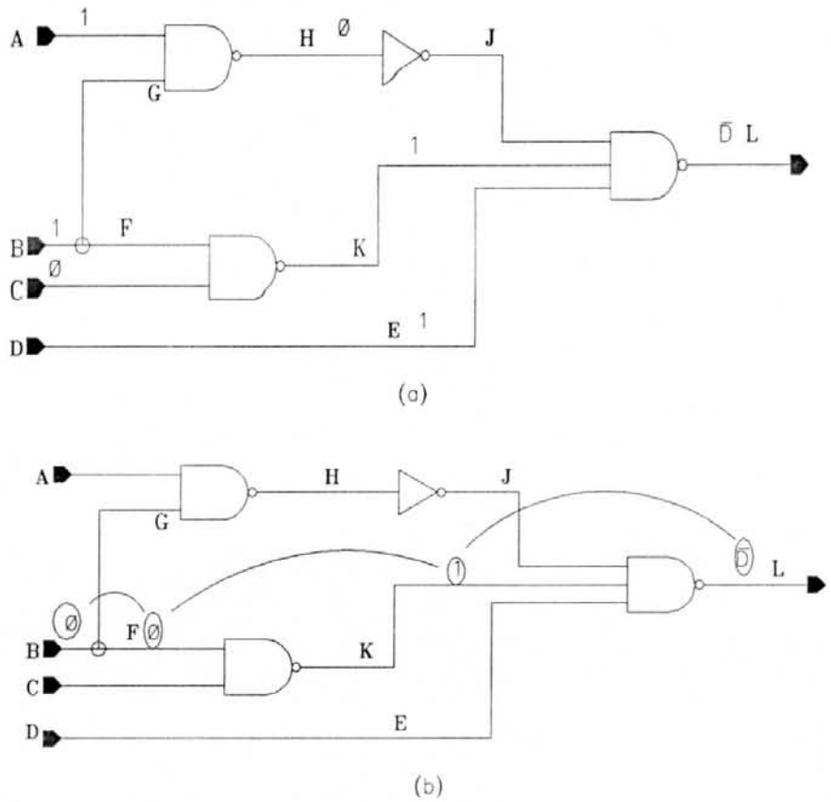


Figura 5.1: Justificação da linha L

Por exemplo, na figura 5.1(a), para a falha L-a-1, atribui-se $\overline{D}$ à linha L. Só existe uma implicação possível, neste caso, que é $J = 1$, $K = 1$ e $E = 1$. O fato de atribuir-se as três linhas e depois implicar ajuda a evitar conflitos. Isto fica claro observando-se a figura 5.1(b). A implicação de K antes de atribuir-se o valor lógico 1 às outras linhas poderia levar o algoritmo a implicar $K = 1$ por caminhos conflitantes. O fato de se atribuir 1 às outras linhas e, em seguida, $H = 0$, já que também este é um valor sem escolha, e após ainda, $A = 1$ e $B = 1$, que também são os únicos valores possíveis para esta implicação, evitaria o *backtracking*. Ainda para evitar conflitos futuros, o PODEM se utiliza de uma técnica de *lookahead* chamada *X-path-check* e que consiste em sempre olhar os caminhos da fronteira D e escolher o que tenha só linhas X, se possível.

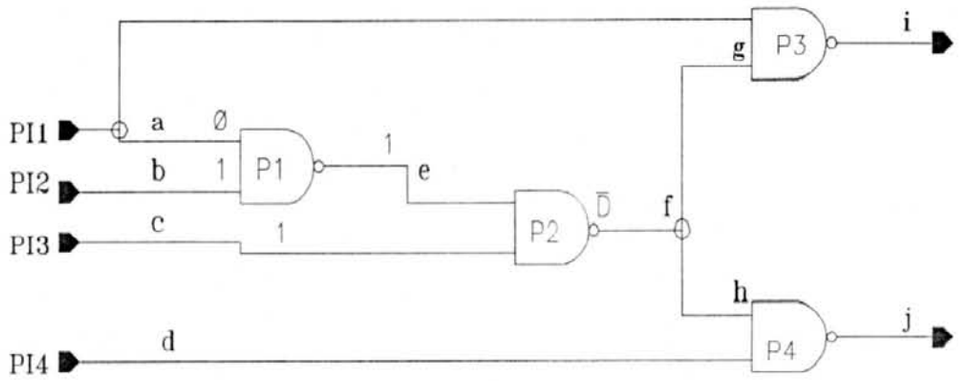


Figura 5.2: *X-Path-Check.*

No exemplo da figura 5.2, a fronteira D, num determinado momento é P3 e P4. A implicação já está feita, ou seja, para obter-se $\overline{D}$ em f foi necessário colocar e e c em 1 e a em 0. Com o uso do *X-path-check*, o algoritmo escolheria a porta P4 para propagar e não a P3, que já tem a linha a envolvida, com um sinal atribuído.

As técnicas de penalização atribuem valores às linhas ou aos nodos de forma a auxiliar o procedimento de geração de testes. Isto pode ser feito quanto à estrutura ou quanto ao funcionamento das portas lógicas envolvidas, seguindo os critérios de observabilidade e controlabilidade. As técnicas de penalização podem ser agrupadas em funcionais, probabilísticas e estruturais.

5.2 Técnicas Funcionais

As técnicas funcionais calculam as penalizações que são atribuídas aos circuitos com relação à função executada, explorando a dominância presente em muitas portas, por exemplo, a facilidade de se gerar um 0 em uma porta AND de duas entradas, visto que 3 das 4 combinações possíveis das entradas produzem 0 na saída. Baseadas nestes conceitos, que no fundo envolvem probabilidades, estas heurísticas foram desenvolvidas.

5.2.1 SCOAP

SCOAP (*Sandia Controllability/Observability Analysis Program*) [GOL 80] é um programa de análise de testabilidade baseado no cálculo de controlabilidade e observabilidade dos nodos do circuito, que neste caso baseia-se em penalizações acumulativas que são atribuidas aos nodos.

O SCOAP calcula seis funções para cálculo de observabilidade e controlabilidade dos nodos internos de um circuito. São elas:

- **controlabilidade sequencial para o valor lógico 1** (SC^1)
- controlabilidade sequencial para 0 (SC^0)
- **controlabilidade combinacional para 1** (CC^1)
- **controlabilidade combinacional para 0** (CC^0)
- observabilidade sequencial (SO) e
- observabilidade combinacional (CO).

A controlabilidade combinacional para 0 e para 1 de um nodo combinacional N, CC^0 e CC^1, está relacionada com o número mínimo de atribuições a nodos combinacionais, necessários para a obtenção do valor requerido.

O cálculo desta medida é feito com a observação do comportamento lógico da porta. Como exemplo, demonstra-se o cálculo para a porta AND. As demais tem procedimento similar.

Considere uma porta AND de duas entradas (A e B) e uma saída (C). A obtenção do valor da controlabilidade para 1, desta porta é feita mediante o seguinte cálculo:

$$CC^1(C) = CC^1(A) + CC^1(B) + \mathbf{1}$$

A obtenção do valor 0, segue a seguinte regra:

$$CC^0(C) = min[CC^0(A), CC^0(B)] + 1$$

A observabilidade é calculada com a seguinte equação:

$$CO(A) = CO(C) + CC^1(B) + 1$$

Pode-se verificar que o cálculo da porta AND para controlabilidade de 1 é feito pela soma das controlabilidades de 1 de todas as entradas, já que é necessário setar todas as entradas em 1 nesta porta, de forma a obter-se o valor lógico 1 na saída. Soma-se ainda o 1 da penalização por ter-se atravessado uma porta.

Já a controlabilidade para o valor 0 só leva em consideração a menor controlabilidade de 0, porque um 0 em qualquer uma das linhas é suficiente para levar a saída para 0. Também aqui é somado 1 da penalização.

O cálculo da observabilidade é feito das saídas primárias para as entradas, por isso penaliza-se cada entrada da porta com a observabilidade de sua saída, mais as controlabilidades dos valores das outras entradas capaz de permitir a observância da saída. No caso de uma porta AND, por exemplo, isto implicaria na controlabilidade do valor lógico 1 nas outras entradas o que permite a observância do sinal.

O cálculo das controlabilidades do circuito é feito da seguinte maneira:

- se o nodo I é uma PI (*Primary Input*):

$$CC^0(I) = CC^1(I) = 1$$

- **se o nodo I é uma porta AND (ver figura 5.3):**

$$CC^1(I) = \sum_{i=0}^{n} X_i + 1$$

$$CC^0(I) = \min_i(CC^0(X_i)) + 1$$

onde X_i são os valores de controlabilidade das entradas da porta AND e n são todas as entradas da porta.

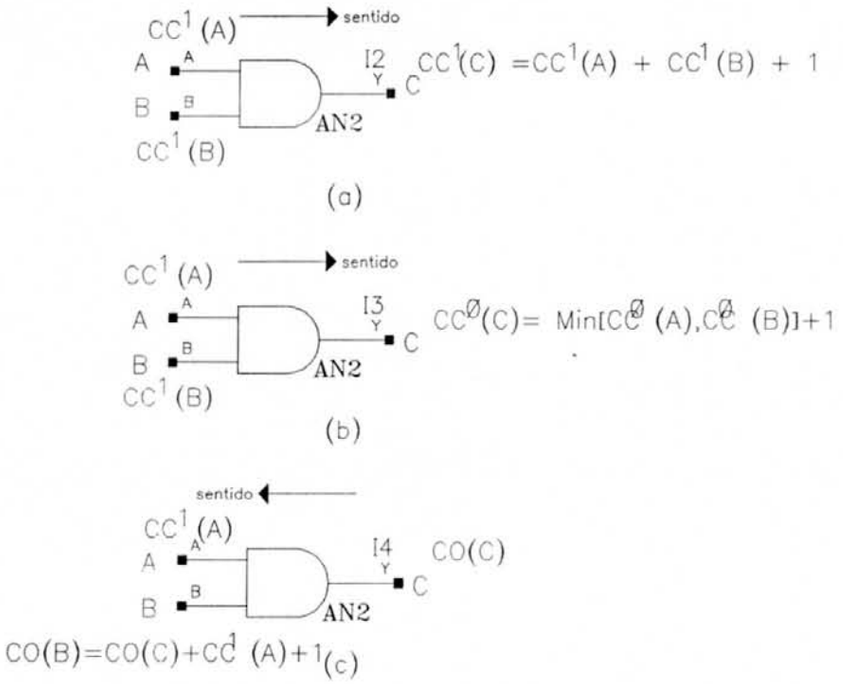


Figura 5.3: Cálculos do SCOAP em uma porta AND

Como explanou-se anteriomente, o cálculo das outras portas é similar, seguindo o mesmo raciocínio dependendo da função lógica realizada pela porta em questão.

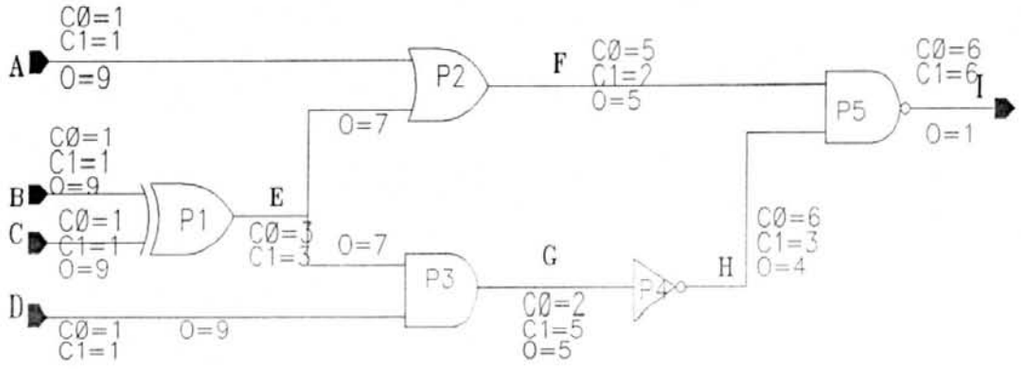


Figura 5.4: Exemplo de cálculos no SCOAP

Como um exemplo pode-se observar a figura 5.4, onde o cálculo da controlabilidade é feito com as regras detalhadas anteriormente, por exemplo, a porta P1 é uma porta XOR, logo a controlabilidade de 1 é calculada somando-se as controlabilidades das combinações que geram 1, no caso $B = 0$ e $C = 1$ ou $B = 1$ e $C = 0$ e escolhendo-se a menor. Assim, $CC^1(E) = CC^1(B) + CC^0(C) + 1 = 3$ ou $CC^1(E) = CC^0(B) + CC^1(C) + 1 = 3$ e como as duas combinações geram o mesmo resultado, é indiferente a combinação que vai ser escolhida. O cálculo de $CC^0(E)$ segue o mesmo raciocínio, assim como todos os cálculos do circuito.

A observabilidade é calculada de acordo com as seguintes regras:

- se o nodo I é uma PO (*Primary Output*):

$$CO(I) = 1$$

- se o nodo I é uma AND com entradas A e B e saída C

$$CO(A) = CO(C) + CC^1(B) + 1$$

$$CO(B) = CO(C) + CC^1(A) + 1$$

O procedimento para as outras portas lógicas é semelhante.

Na figura 5.4, a observabilidade da PO é 1, já $O(F)$ foi calculada da seguinte maneira:

$$O(F) = O(PO) + CC^1(H) + 1 = 5$$

O resto do circuito foi avaliado por este mesmo princípio.

5.2.2 CAMELOT

Outro procedimento disponível para medir a observabilidade e controlabilidade dos circuitos é o CAMELOT (*Computer-Aided MEasure for Logic Testability*) [BEN 80]. Este procedimento não identifica duas medidas diferenciadas para a controlabilidade e observabilidade. O seu valor é medido em termos da dificuldade de transferência de valores pela porta, ao invés de identificá-la como controlável para valores distintos. Nesta medida, conforme se percorrem portas lógicas pelo caminho, penaliza-se estes caminho com uma degradação que é calculada em função da dominância da porta.

No geral, a controlabilidade é calculada pela seguinte fórmula:

$$CY_{output} = CTF \times f(CY_{inputs})$$

onde CTF é o fator de transferência de controlabilidade e f é uma função das entradas do dispositivo que afetam a saída do nodo e $0 \leq CY \leq 1$, sendo que 0 significa *o menos controlável possível* e 1, o contrário, isto é, *totalmente controlável.* Como um exemplo, uma linha que está permanentemente em um valor constante não é controlável, logo $CY = 0$. Já uma entrada primária tem por definição $CY = 1$.

O fator de transferência reflete a degradação que a porta imprime na controlabilidade da saída do nodo.

Uma das maneiras de se calcular o fator de transferência é:

$$CTF = 1 - \left| \frac{N(0) - N(1)}{N(0) + N(1)} \right|$$

onde $N(0)$ é o número total de combinações de entradas que produzem 0 e $N(1)$, das combinações que produzem 1.

Como o enfoque do trabalho são os circuitos combinacionais, derivase as portas lógicas NOT e AND com 2 e 3 entradas para exemplificar o cálculo. O cálculo das portas restantes é similar.

A	B	Y
Ø	Ø	Ø
Ø	1	Ø
1	Ø	Ø
1	1	1

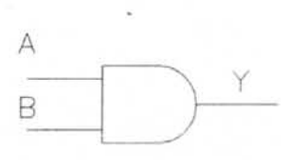


Figura 5.5: Porta AND com duas entradas

Na figura 5.5 vê-se uma porta AND de duas entradas, que tem $N(0) = 3$, ou seja, três das combinações da tabela-verdade produzem 0 e $N(1) = 1$, isto é, uma combinação capaz de gerar 1. O cálculo é feito da seguinte maneira:

$$CTF = 1 - \left|\frac{3-1}{3+1}\right| = 1 - \frac{2}{4} = \frac{1}{2} = 0.5$$

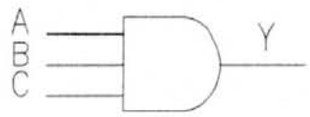


Figura 5.6: Porta AND com três entradas

O cálculo para a porta AND de três entradas da figura 5.6 é similar, apenas com a diferença no $N(0)$, que com três entradas passa a ter o valor 7, e $N(1)$ continua com 1, ou seja, só existe uma combinação entre as oito possíveis capaz de gerar 1. Logo,

$$CTF = 1 - \left|\frac{7-1}{7+1}\right| = 1 - \frac{6}{8} = \frac{1}{4} = 0.25$$

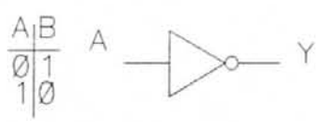


Figura 5.7: Porta NOT

Em uma porta NOT (ver figura 5.7), o procedimento é feito também pela observação da tabela-verdade, onde verifica-se que tem-se uma situação que gera 0 e uma que gera 1, logo $N(0) = N(1) = 1$. Assim,

$$CTF = 1 - \left|\frac{1-1}{1+1}\right| = 1$$

A função f combina as entradas que afetam a saída e tem a expressão geral da seguinte forma:

$$f(CY_{inputs}) = \frac{\sum CY inputs}{n}$$

onde n é o número de entradas. Para exemplificar o cálculo da controlabilidade examine-se a saída da porta P1 da figura 5.8.

$$CY = CTF \times f(CY_{inputs})$$

$$CTF = 1 - \left|\frac{3-1}{3+1}\right| = 1 - \frac{2}{4} = 0,5$$

$$f(CY_{inputs}) = \frac{1+1}{2} = 1$$

onde os valores 1's no numerador são as controlabilidades das entradas de P1. logo,

$$CY = 0,5 \times 1 = 0,5$$

A observabilidade também é calculada com a utilização da uma medida de fator de transferência de observabilidade (OTF). O cálculo da observabilidade é realizado da seguinte maneira:

$$OY_{(I-O)} = OY_{(O)} \times OTF_{(I-O)} \times g(CY_{entradas-suportadas})$$

onde OY_{I-O} é a observabilidade da entrada I com saída O, OY_O é a observabilidade da saída O; OTF_{I-O} é o fator de transferência de observabilidade para o nodo com entrada I e saída O e g é a função que suporta o caminho sensível, ou seja, as entradas que permitem a passagem.

O fator de transferência da observabilidade reflete a possibilidade de se combinar os valores de forma a propagar o sinal. A função g, por outro lado, mede as possibilidades que existem do conjunto propagador ser satisfatório, logo, o produto $OTF \times g$ reflete a possibilidade de uma porta propagar, isto é, do sinal ser observável.

$$OTF_{(I-O)} = \frac{N(SP : I-O)}{N(SP : I-O) + N(IP : I-O)}$$

onde $N(SP : I-O)$ é o número de combinações que permite a passagem do sinal I e $N(IP : I-O)$ é o número das combinações que bloqueiam.

Em uma porta AND de três entradas (ver figura 5.5), o cálculo do OTF seria:

$$OTF_{(A-Y)} = \frac{1}{3+1} = \frac{1}{4} = 0.25$$

A função g combina a controlabilidade CY de todas as outras entradas da porta que auxiliam a propagação do sinal.

$$g(CY_{entradas\_suportadas}) = \sum_{i=1}^{n} CY_{todas\_as\_entradas\_menos\_a\_que\_se\_calcula}$$

Para exemplificar considere-se a observabilidades da PI1, na saída da porta P5 (ver figura 5.8). O cálculo é feito da seguinte maneira:

$$OY_{(PI1-saida\_de\_P5)} = OY_{(saida\_de\_P5)} \times OTF_{(P5)} \times g(CY_{entradas})$$

$$OY(saida\_de\_P5) = 1$$

$$OTF_{(P5)} = \frac{1}{1+1} = \frac{1}{2} = 0,5$$

Como a porta P5 tem apenas outra entrada (saída da porta P3), além da entrada primária PI1, o valor de g é a controlabilidade desta outra entrada de P5, ou seja,

$$g(CY_{entradas\_suportadas}) = 0,375$$

Assim,

$$OY_{(PI1-saida\_de\_P5} = 1 \times 0,5 \times 0,375 = 0,1875$$

O cálculo das observabilidades através do circuito é feito com a penalização na direção das saídas para as entradas, seguindo as regras dadas anteriormente.

Como um exemplo de todo um circuito calculado, têm-se o circuito da figura 5.8, onde todas as linhas foram atribuídas com valores de observabilidade e controlabilidade pelo método do CAMELOT.

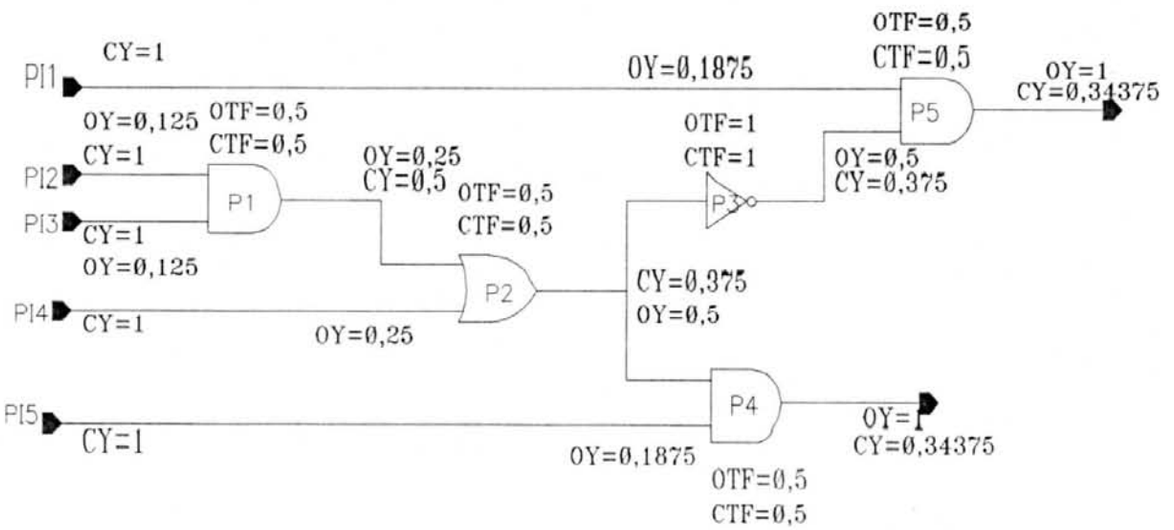


Figura 5.8: Exemplo de cálculos com o CAMELOT

5.3 Técnicas Probabilísticas

Uma maneira de penalizar os fan-outs é em termos da probabilidade de obtenção de determinado valor na conexão quando da utilização de um vetor randômico. Várias técnicas exploram esta visão para guiar algoritmos de geração de testes.

A probabilidade de detecção ($P_d(w/i)$) é a probabilidade que um vetor selecionado randomicamente consiga gerar a diferença em sua saída, o que o distinguiria de um circuito bom para a falha na linha w *stuck-at-i*.

A probabilidade de sinal ($P_s(w)$), é a probabilidade que a linha w tem de assumir o valor 1.

Existe uma relação entre estas medidas, dado que a probabilidade de detecção pode ser vista em termos de probabilidade de sinal. Se existe um valor em uma determinada linha que deve ser propagado por uma porta AND, a probabilidade do sinal 1 nas outras entradas da porta tem influência direta na probabilidade de detecção desta falha.

Pode-se afirmar que a probabilidade de detecção de w *stuck-at-i* ($P_d(w/i)$) é maior que a probabilidade de sinal P_s de todos os sinais necessários para a propagação.

De uma maneira geral, estes cálculos são feitos pelas seguintes regras:

- Entrada Primária: Probabilidade igual a $1/2$.
- Porta NOT: Complemento da probabilidade de entrada (1− Probabilidade de entrada) (ver figura 5.9).
- Porta AND: Multiplicação das probabilidades de todas as entradas (figura 5.9).
- Porta OR: Multiplicação dos complementos das probabilidades de todas as entradas (ver figura 5.9).

As outras portas (NAND e NOR) podem ser vistas como a porta AND e OR, respectivamente, aliadas à porta NOT.

Usando um exemplo simples pode-se ilustrar os cálculos.

Considerando-se o circuito da figura 5.10, vê-se nas linhas os valores binários que representam o resultado de uma simulação exaustiva. Baseando-se nos resultados desta simulação, as probabilidades de sinal podem ser facilmente calculadas.

A probabilidade de e foi calculada de acordo com o dito anteriormente, ou seja, pela multiplicação das probabilidades do valor lógico 1 das entradas da porta AND de onde o sinal deriva, estas entradas são a e b (ambas $1/2$). Observando-se os valores calculados na simulação pode-se facilmente constatar que isto se verifica e que a probabilidade é de apenas $1/4$ dos valores serem 1.

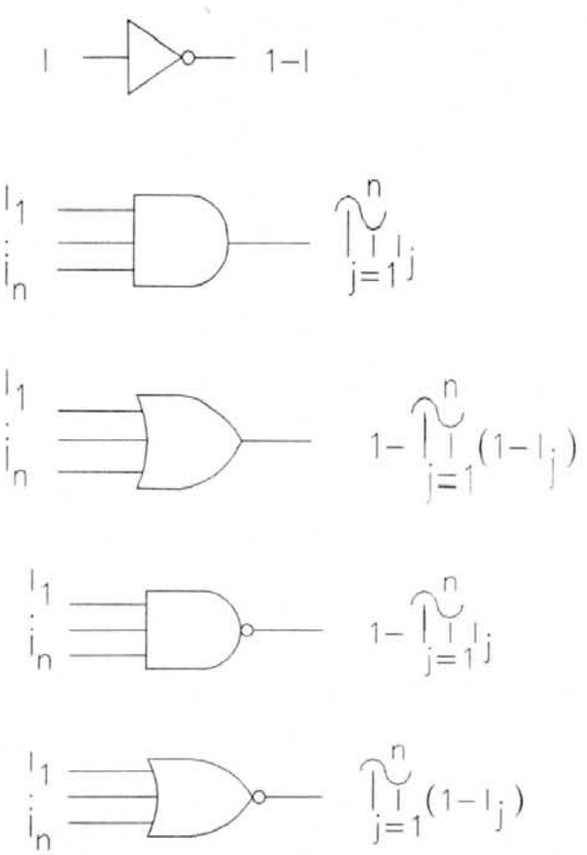


Figura 5.9: Cálculo das probabilidades

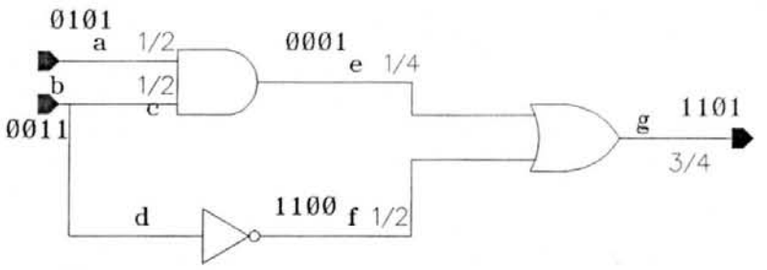


Figura 5.10: Exemplo de cálculos probabilísticos

A probabilidade pode ser utilizada como medida de controlabilidade. Assim, a controlabilidade de f é 1 menos a probabilidade do valor 0 na entrada b, isto é, 1/2. Um 1 na saída de uma porta OR exige que ambas as suas entradas não sejam simultaneamente 0. Como as probabilidades do valor 0 em suas entradas e e f são 3/4 e 1/2, sua controlabilidade para o valor 1 é calculada por:

$$1 - \frac{3}{4} \times \frac{1}{2} = 1 - \frac{3}{8} = \frac{5}{8}$$

Isto, entretanto, não condiz com a simulação exaustiva, que apresenta 3/4 dos seus valores em 1.

A razão do erro é a presença do fan-out reconvergente no circuito que faz com que os sinais e e f sejam correlatos. Sua probabilidade neste caso não pode ser multiplicada. Este problema será tratado na próxima seção.

As medidas probabilísticas sem fan-outs reconvergentes são facilmente calculáveis usando as fórmulas de propagação apresentadas na figura 5.9.

5.3.1 O algoritmo *Cutting*

O algoritmo *Cutting* [SAV 84] consiste em transformar o circuito combinacional em uma árvore cortando os fan-outs reconvergentes e inserindo nos pontos de corte intervalos probabilísticos fechados ou seja, o intervalo entre a mínima probabilidade 0 e a probabilidade máxima 1 incluindo 0 e 1 (ou seja, $[0,1]$). Com isto tenta-se aproximar, com o máximo de precisão, o intervalo de probabilidade onde o valor exato (probabilístico) se encontra.

A vantagem deste algoritmo é a redução da complexidade, tanto do tempo como do espaço, por conta da redução das conexões. A principal desvantagem é que não se tem um valor, mas sim um intervalo no qual está contido o valor exato não conhecido.

O algoritmo consiste basicamente do cálculo dos valores probabilísticos através das fórmulas e, no caso de fan-out reconvergente, é atribuído o intervalo fechado $[0,1]$ aos pontos de corte, exceto um, que assume a probabilidade do antecessor. Estes valores devem, então, ser propagados através das fórmulas de conjunto da figura 5.11, similares às fórmulas de valores simples.

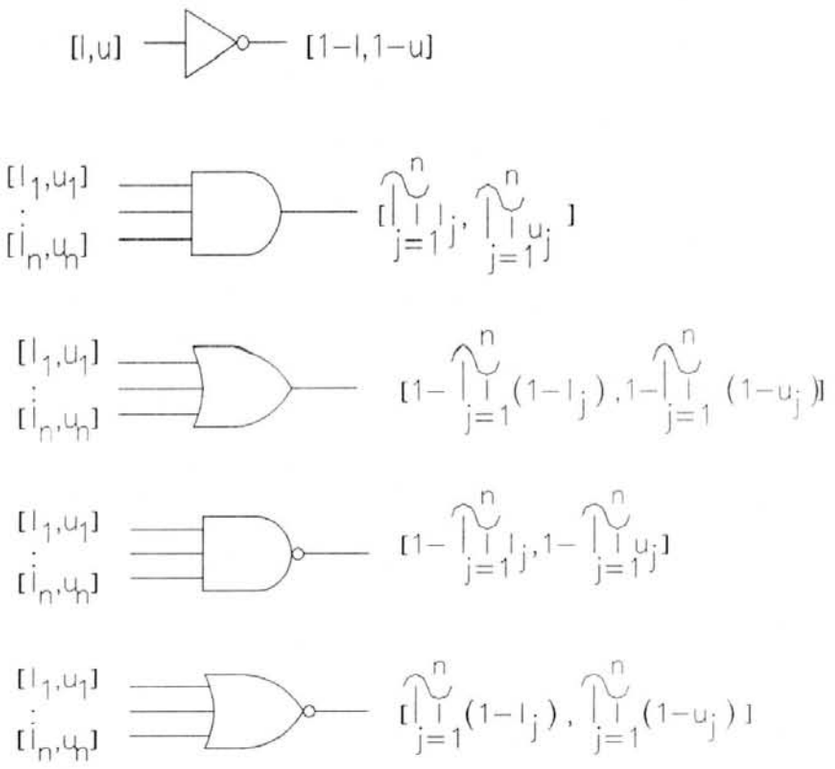


Figura 5.11: Cálculo das probabilidades em intervalos de valores

Na figura 5.12, vê-se um exemplo dos cálculos das linhas livres, ou seja, sinais independentes. Os pontos $w1$ e $w2$ é que devem ter seus valores calculados a partir de cortes por serem sinais influenciados por uma reconvergência enterior.

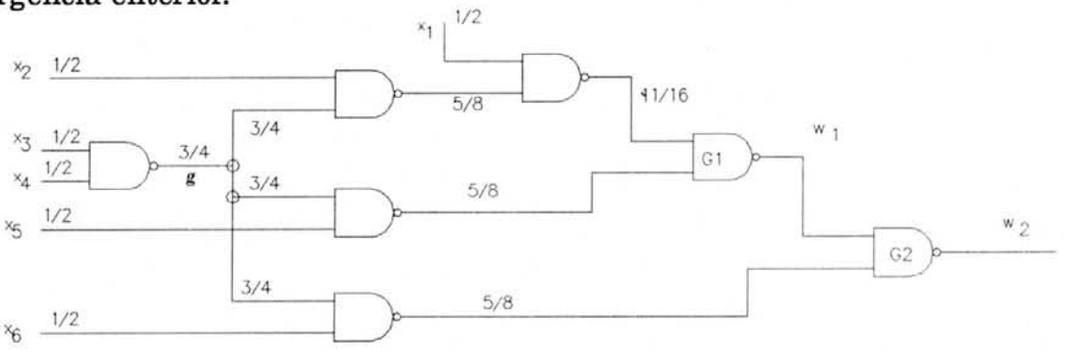


Figura 5.12: Exemplo de cálculos das linhas sem restrições

Na figura 5.14, o fan-out g recebe a restrição do algoritmo, para isto, coloca-se dois dos desvios em $[0,1]$. Com isto, são propagados os intervalos probabilísticos e têm-se um valor para $w1$.

Por exemplo, na figura 5.13, h foi calculado como apresentado ao lado da linha.

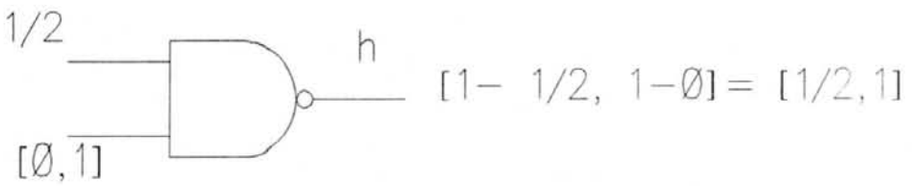


Figura 5.13: Exemplo de cálculos de intervalos

Cortar o fan-out representa substituir o ON-set desta linha por dois conjuntos, o conjunto vazio e o conjunto universo.

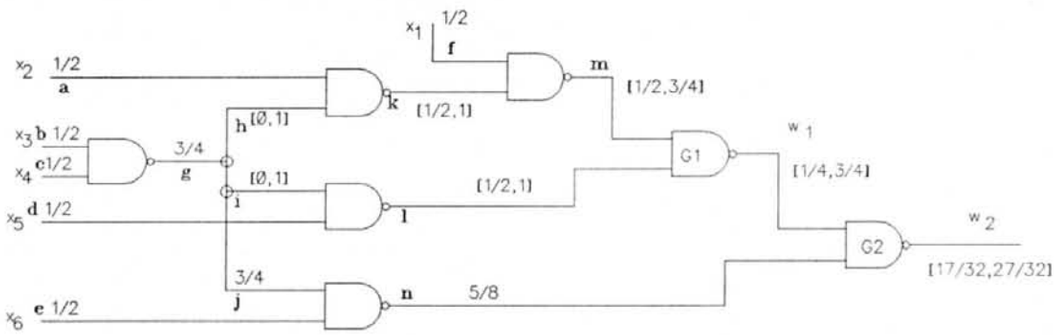


Figura 5.14: Probabilidades de sinal das linhas ignorando dois dos fan-outs

Na figura 5.15, vê-se os valores já totalmente calculados.

Restrição no cálculo

A paridade de um caminho é definida em termos do número de inversões que acontecem neste caminho. Por exemplo na figura 5.14, o caminho entre o ponto de divergência g e $w1$, por exemplo, que envolve as conexões g, h, k e m é um caminho com paridade par, já que envolve um número par de inversões.

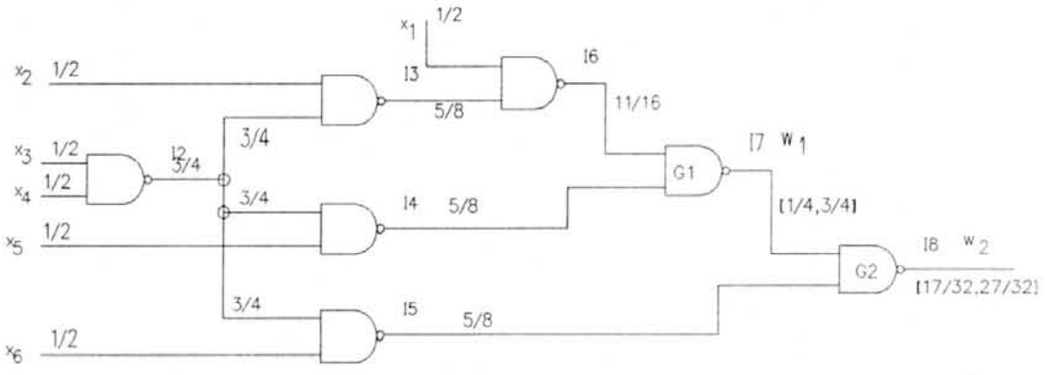


Figura 5.15: Circuito com todos os valores probabilísticos calculados

A paridade da reconvergência é determinada examinando-se a paridade dos caminhos que reconvergem. Diz-se que dois caminhos reconvergem com igual paridade quando cada um dos caminhos reconvergentes apresenta a mesma paridade.

Em alguns casos é possível atribuir os intervalos $[0, g]$ ou $[g, 1]$, ao invés de $[0, 1]$, onde g é a probabilidade de sinal do sinal divergente. Se g é um intervalo $[l, u]$, então um dos intervalos, $[0, u]$ ou $[l, 1]$ deve ser utilizado. Isto aproveita a paridade inversa apresentada por alguns fan-outs reconvergentes.

Para ilustrar o uso da tabela de restrições (ver tabela 5.1) pode-se examinar o circuito da figura 5.16 onde verifica-se que os dois caminhos que divergem de w e reconvergem em h, o fazem com paridades diferentes. O tipo de porta da reconvergência é NOR ($G6$) e os caminhos reconvergentes são $\{G4 - G2, G5 - G3\}$. Se o caminho escolhido para o corte fosse $\{G4 - G2\}$ que apresenta paridade par entre w e h, de acordo com a tabela 5.1, o intervalo a ser atribuiído seria $[\frac{3}{4}, 1]$. O restante dos cálculos pode ser visto na figura 5.16.

O procedimento realizado utilizando-se a restrição é o seguinte:

Passo 1: Todas as entradas recebem probabilidades $1/2$ ($P_i = 1/2$) e calcula-se a probabilidade de todas as linhas até a chegada em um ponto de divergência de dados.

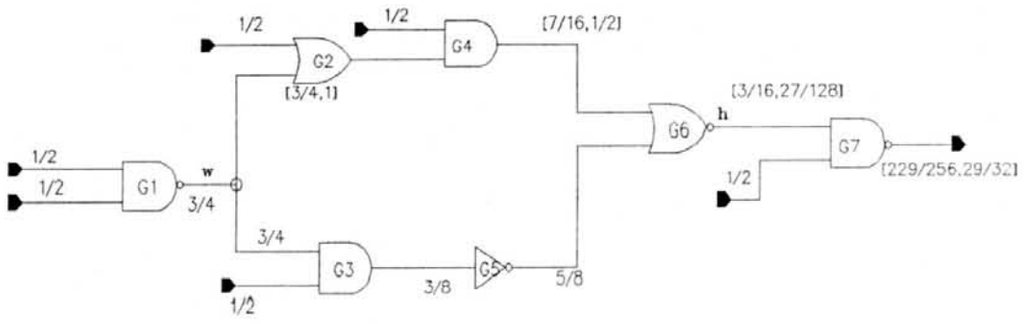


Figura 5.16: Circuito exemplo para o cálculo de restrições

Paridade do caminho	Tipo de porta na reconvergência		Paridade da reconvergência
Escolhido para o Corte	AND/NOR	OR/NAND	
Par	[g,1]	[0,g]	igual
Impar	[0,g]	[g,1]	igual
Par	[0,g]	[g,1]	diferente
Impar	[g,1]	[0,g]	diferente

Tabela 5.1: Tabela de restrições

Passo 2: Da entrada para as saídas, corta-se os fan-outs reconvergentes atribuíndo para eles um intervalo restrito de probabilidade. Se todos os caminhos que esta linha participa e suas portas de reconvergência têm o mesmo intervalo de acordo com a tabela, então assuma este intervalo, caso contrário, ou seja, em um caminho de reconvergência, ele participa de um caminho que oferece pela tabela $[0, g]$ e por outro que apresenta $[g, 1]$, utiliza-se o intervalo fechado $[0, 1]$.

Passo 3: Os intervalos na árvore resultante devem ser propagados.

Utilizando-se como exemplo o circuito da figura 5.17, e supondo que a linha que vai ser cortada seja $c2$, vê-se que esta linha participa de dois caminhos reconvergentes, $\{G8 - G11 - G12, G9 - G5\}$ com $G12$ como porta reconvergente, e $\{G8 - G11 - G12 - G13, G10 - G13\}$ com $G13$ como porta de reconvergência. No primeiro caminho $c2$ segue um caminho ímpar ($\{G8 - G11 - G12\}$) e os caminhos têm paridades diferentes. Já no segundo

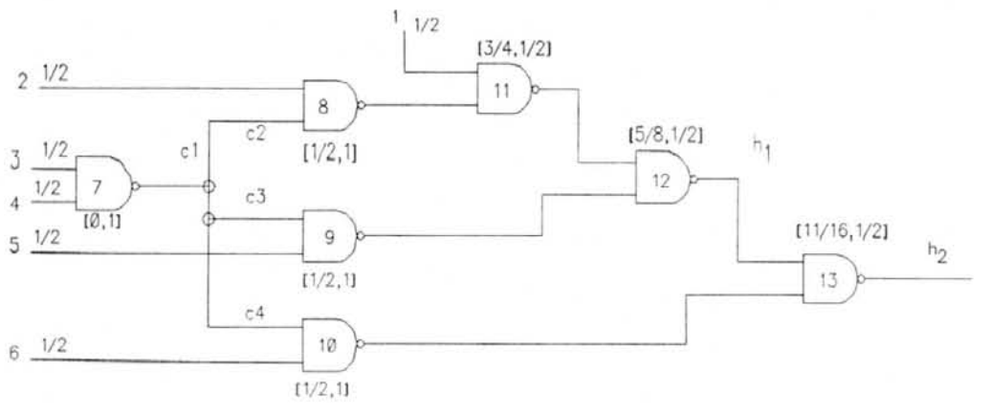


Figura 5.17: Exemplo do uso do corte utilizando as restrições

caminho, $c2$ segue por um caminho par ($\{G8-G11-G12-G13\}$) e os caminhos têm igual paridade. Em ambos os casos, porém, o intervalo a ser utilizado é $[0,g]$, onde g é a probabilidade de $c1$. Neste caso atribui-se o intervalo $[0,\frac{3}{4}]$ à $c2$. Supondo que as faixas encontradas na tabela para $c2$ fossem distintas, teria que ser utilizado $[0,1]$.

É possível a otimização dos intervalos alterando-se os pontos de corte, ou seja, se um caminho resulta em intervalos diferentes na tabela o que vai ocasionar a atribuição de $[0,1]$. Deve-se levar em consideração que outro corte poderia conseguir um intervalo menor, logo, deve-se escolher este outro ponto de corte. Na figura 5.17, se tivesse sido escolhido $c4$ antes de $c2$, ao invés do contrário, teria sido necessário a atribuição do intervalo completo $[0,1]$ para $c4$.

Se o algoritmo for executado $k \geq 1$ vezes e se as probabilidades computadas são $[l_i, u_i]$ para $i = 0, \ldots, k$, escolhe-se como faixa final $[l, u]$ onde:

$$l = \max_i (l_i), i = 1, 2, \ldots, k$$

$$u = \min_i (u_i), i = 1, 2, \ldots, k$$

Os intervalos de probabilidade encontrados contém a probabilidade exata. Neste caso, como não é proposto nenhum método para encontrar um número exato, usando-se somente estes cálculos estes valores de intervalos é que serviriam como o valor de testabilidade que auxiliaria o algoritmo D. O algoritmo PREDICT, visto adiante, propõe uma maneira de se encontrar o valor exato dentro do intervalo.

5.3.2 COP

A controlabilidade no COP [BRG 84] é calculada da maneira citada anteriormente, porém cabe ressaltar as fórmulas de cálculo da cardinalidade, que é usada como medida de controlabilidade, e observabilidade. Para isto utiliza-se o exemplo da figura 5.18 e a tabela 5.2. A linha da tabela nomeada 1-co indica o número de valores lógicos 1 controláveis na entrada, por exemplo, como a, b e c são entradas primárias tem-se metade das entradas no valor lógico 1, ou seja, quatro das entradas são 1-co; 0-co é o número de valores lógicos 0 controláveis; 1-ob é o número de valores lógicos 1 observáveis e representa o número de valores 1 na tabela não marcados por * nem por **; 0-ob é o número de valores lógicos 0 observáveis e representa os valores 0 sem marcas; e o símbolo # representa a cardinalidade, ou seja, o número de valores lógicos 1's.

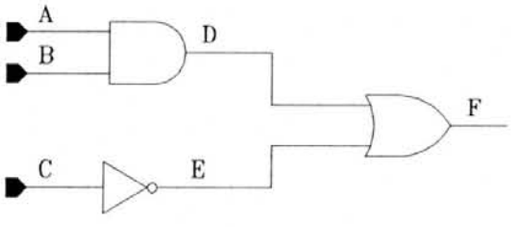


Figura 5.18: Circuito exemplo

A cardinalidade (#) do conjunto de valores 1 presentes na saída de uma porta AND é calculada da seguinte maneira:

	A	B	C	D	E	F
1	0 *	0 *	0	0 **	1	1
2	0 **	0 **	1	0	0	0
3	0 *	1 *	0	0 **	1	1
4	0	1 **	1	0	0	0
5	1 *	0 *	0	0 **	1	1
6	1 **	0	1	0	0	0
7	1 *	1 *	0 *	1 **	1 **	1
8	1	1	1 *	1	0 **	1
1-co	4	4	4	2	4	5
0-co	4	4	4	6	4	3
1-ob	1	1	3	1	3	5
0-ob	1	1	3	3	3	3
t-ob	2	2	6	4	6	8
#	4	4	4	2	4	5

Tabela 5.2: Tabela de restrições para exemplo

$$\#D = \frac{(\#A \times \#B)}{\#U}$$

onde A e B são as entradas, D a saída (ver figura 5.18), $\#U$ é a cardinalidade do conjunto universo, ou seja, todas as entradas possíveis, no caso, 8. No caso do exemplo:

$$\#D = \frac{4 \times 4}{8} = 2$$

Este resultado pode ser conferido na tabela 5.2. A cardinalidade do conjunto de valores 0 é encontrado da seguinte maneira:

$$\#D' = \#U - \#D = 8 - 2 = 6$$

A cardinalidade do conjunto da saída de um inversor é apenas trocada. Na figura 5.18,

$$\#E = \#C'$$

Quanto à observabilidade, marcou-se na tabela 5.2 com ** quando existem restrições topológicas de observabilidade, por exemplo, não se observa o valor de D quando E é 1, logo, em todas as posições da tabela onde E é 1, D está marcado com dois asteriscos. Marcou-se com * a restrição indireta, ou seja, não adianta A ser observável em D se D não é observável em F, logo estes casos também são levados em conta.

A cardinalidade destes conjuntos observáveis pode ser encontrada por cálculos. Por exemplo, a cardinalidade de D em F ($\#(F/D)$):

$$\#(F/D) = \frac{\#U \times (\#U - \#E)}{\#U} = \frac{8 \times (8-4)}{8} = 4$$

ou

$$\#(F/D) = \frac{\#U \times \#E'}{\#U} = \frac{8 \times 4}{8} = 4$$

Genericamente,

$$\frac{\#X \times \#Y}{\#U}$$

onde, X é o conjunto dos valores a serem observados, Y os valores que permitem a observação e U o conjunto universo.

Da mesma forma para padrões de A observáveis em F ($\#(F/A)$):

$$\#(F/A) = \frac{\#(F/D) \times \#B}{\#U} = 2$$

ou seja, dois valores são observáveis em F. Na tabela 5.2, o valor lógico 0 da linha 4 e o valor lógico 1 da linha 8.

Também matematicamente, pode-se calcular apenas o número de 1's observáveis em A ($\#(F/a)$), com a seguinte fórmula:

$$\#(F/a) = \frac{\#a \times \#(F/A)}{\#U} = 1$$

5.3.3 PREDICT – *Probabilistic Estimation of Digital Circuit Testability*

O PREDICT [SET 85] utiliza os intervalos calculados pelo procedimento citado anteriormente como CUTTING e a partir destes valores calcula um valor único. A diferença no cálculo dos intervalos é que o algoritmo do PREDICT atribui o intervalo fechado $[0,1]$ a todos os desvios do fan-out reconvergente ao mesmo tempo.

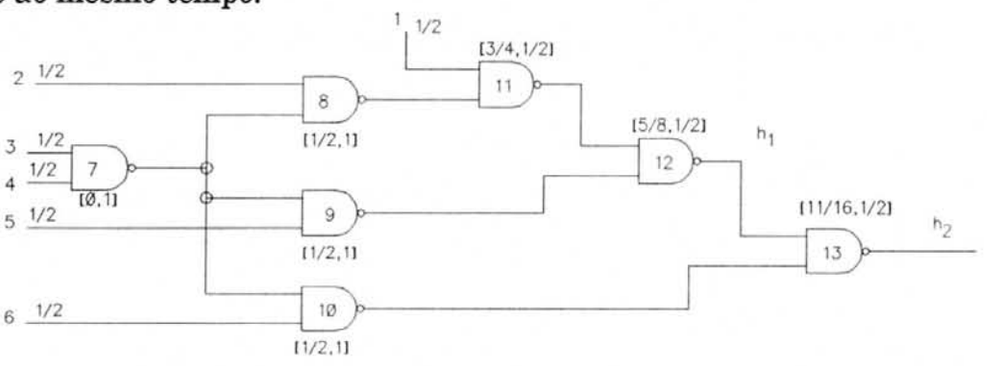


Figura 5.19: Cálculo dos intervalos no PREDICT

Na figura 5.19 vê-se o mesmo circuito apresentado no CUTTING com os valores dos intervalos calculados. Os cálculos foram feitos da mesma forma que no CUTTING exceto que todos os cálculos são feitos de uma vez com o intervalo [1, 0], como explicado anteriormente.

De posse dos intervalos de valores, a controlabilidade única é calculada:

$$C\mathbf{1}(X) = \sum_{allA} C\mathbf{1}(X|A)Prob(A)$$

onde $C\mathbf{1}(X|A)$ é a controlabilidade condicional quando A assume o valor lógico (0 ou 1) e $Prob(A)$ é a probabilidade de A assumir o valor.

A procedure computacional pode ser vista como:

- Identificar os nodos reconvergentes;
- Determinar os caminhos reconvergentes (sub-rede, ou seja, redes formadas pelos caminhos reconvergentes);
- Calcular as controlabilidades.

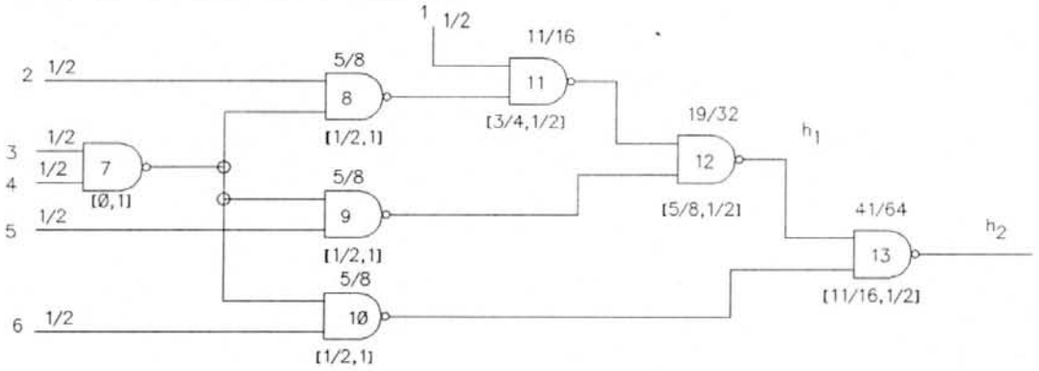


Figura 5.20: Valores únicos calculados

Na figura 5.20 vê-se os valores calculados. Tem-se embaixo dos nodos os intervalos calculados assumindo-se para o fan-out da porta 7 o intervalo

fechado $[1, 0]$. O valor que encontra-se sobre os nodos é o valor único que foi calculado pesando os valores dos intervalos com a probabilidade do valor 0 e 1 da porta. Assim,

$$C1(11) = \frac{3}{4} \times \frac{3}{4} + \frac{1}{4} \times \frac{1}{2} = \frac{9}{16} + \frac{1}{8} = \frac{9+2}{16} = \frac{11}{16}$$

$$C1(13) = \frac{3}{4} \times \frac{11}{16} + \frac{1}{4} \times \frac{1}{2} = \frac{33}{64} + \frac{1}{8} = \frac{33+8}{64} = \frac{41}{64}$$

como 11 e 13 são portas NAND, tem-se $\frac{1}{4}$ de probabilidade de valores 0 e $\frac{3}{4}$ de valores 1.

Heurísticas para diminuir a complexidade

Pode-se utilizar duas heurísticas para diminuir a complexidade do algoritmo apesar de sacrificar um pouco a acuracidade dos resultados computados. Ambas as heurísticas são baseadas na idéia de que sob algumas circunstâncias sinais correlatos podem ser tratados de forma independente sem muita perda de precisão. Nos dois casos é utilizado um valor limiar.

- **Heurística 1**

  A distância utilizada aqui é em relação ao número de conexões entre os nodos. O valor limiar T é escolhido e então usa-se os nodos que apresentam a distância T do nodo-alvo. Calcula-se, então, os valores da controlabilidade como se fossem valores independentes.

  Na figura 5.21 vê-se que, com um valor limiar de 1, usaria-se para o cálculo da $C1$ do nodo 12 apenas os nodos de uma distância de uma conexão, ou seja, 11 e 9. Neste caso, a $C1$ seria calculada como se os valores fossem independentes, isto é,

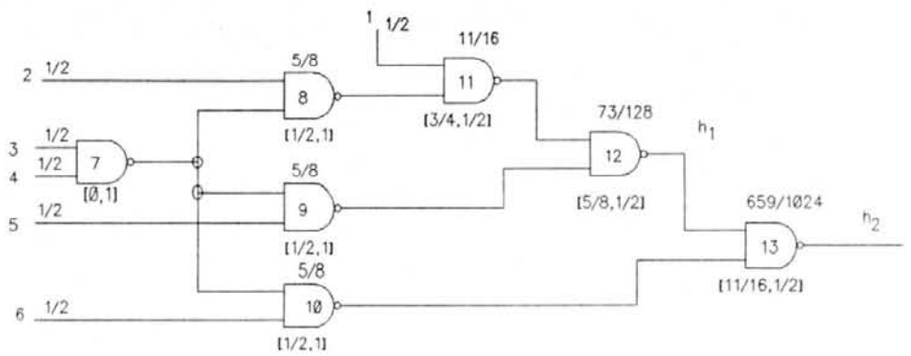


Figura 5.21: Cálculo utilizando a primeira heurística

$$C1(12) = 1 - \frac{11}{16} \times \frac{5}{8} = 1 - \frac{55}{128} = \frac{73}{128}$$

O cálculo é feito normalmente como o cálculo probabilístico apresentado anteriormente.

Na tabela 5.3 pode-se verificar o cálculo e o erro de cálculo com os diferentes valores-limiar.

	Controlabilidade		Erro	
Valor-limiar	C1(12)	C1(13)	C1(12)	C1(13)
1	73/128	659/1024	-3/128	3/1024
2	73/128	659/1024	-3/128	3/1024
3	19/32	173/256	0	9/256
4	19/32	41/64	0	0

Tabela 5.3: Resultados dos cálculos com a primeira heurística

- **Heurística 2**

Na heurística 2 o procedimento é o mesmo diferindo apenas no conceito de distância e seu cálculo. Nesta heurística a distância é definida como:

$$d(V,W) = \frac{|L(V) \triangle L(W)|}{|L(V) \cap L(W)|}$$

onde $|x|$ significa a cardinalidade (número de elementos) de x; $x \triangle y$, a diferença simétrica entre x e y e $x \cap y$, a intersecção dos conjuntos x e y. A intersecção de x e y é o conjunto dos elementos que estão presentes tanto em x quanto em y. A diferença simétrica é o conjunto complementar da interseção em relação ao universo. ou seja, o que pertence só a x, só a y ou só ao universo.

Intuitivamente esta medida da distância pode ser entendida da seguinte forma: No caso extremo, quando os cones de influência de V e W não têm PI's em comum, os sinais V e W são totalmente independentes. Quando o cone de influência tem uma ou mais PI's em comum, quanto maior a sobreposição, ou seja, PI's em comum, maior é o denominador, isto é, a cardinalidade da interseção, menor será o valor da medida. Para uma dada sobreposição, entretanto, quanto maior o número de PI's que influencia só V ou só W (mas não ambos), menor é a correlação entre as PI's comuns às duas portas. Isto justifica a diferença simétrica no denominador.

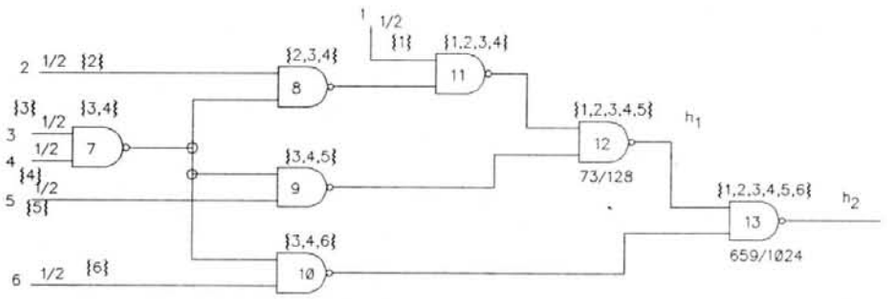


Figura 5.22: Cálculo utilizando a segunda heurística

Por exemplo, na figura 5.22, têm-se entre chaves as PI's que influenciam a porta. Para ilustrar os cálculos da distância por esta nova fórmula, suponha-se o cálculo da distância entre o nodo 9 e o nodo 11.

$$d(9,11) = \frac{|\{3,4,5\} \triangle \{1,2,3,4\}|}{|\{3,4,5\} \cap \{1,2,3,4\}|} = \frac{|\{1,2,6\}|}{|\{3,4\}|} = \frac{3}{2} = 1.5 \cong 1$$

Logo, a distância entre 9 e 11 é 1.5, ou truncando, 1. Como um segundo exemplo, a distância entre o nodo 7 e o nodo 11 ficaria:

$$d(7,11) = \frac{|\{3,4\} \triangle \{1,2,3,4\}|}{|\{3,4\} \cap \{1,2,3,4\}|} = \frac{|\{1,2,5,6\}|}{|\{3,4\}|} = \frac{4}{2} = 2$$

Logo, se o valor limiar estipulado fosse 1, o algoritmo iria parar nos nodos 9 e 11, caso o nodo a ser calculado fosse 12.

Na tabela 5.4 vê-se os resultados com a utilização desta heurística com vários valores-limiar.

	Controlabilidade		Erro	
Valor-limiar	C1(12)	C1(13)	C1(12)	C1(13)
1	73/128	659/1024	-3/128	3/1024
2	19/32	161/256	0	-3/256
4	19/32	41/64	0	0

Tabela 5.4: Resultados dos cálculos com a segunda heurística

5.4 Técnicas Estruturais

As técnicas estruturais são as que penalizam ou delimitam áreas no circuito de forma a amenizar o problema dos conflitos e, conseqüentemente, diminuir o número de *backtrackings*. Estas técnicas prestam especial atenção à disposição dos fan-outs e reconvergência.

5.4.1 FAN

O FAN [FUJ 83] é um refinamento do PODEM, ou seja, se utiliza de todas as heurísticas do PODEM além das suas heurísticas estruturais.

Um conceito utilizado por este algoritmo é o de linhas *bound*, linhas *free* e *head line*. Quando uma conexão L faz parte de um caminho que vem de um ponto de fan-out (divergência), a linha L é dita *bound*. Na figura 5.23, as linhas em destaque.

A linha L que antecede as conexões *bound* chama-se *head line* (ver figura 5.23). As conexões que não são *bound* nem *head lines* são ditas *free*. Após a classificação das conexões, as seguintes afirmações podem ser feitas:

Um subcircuito composto só de linhas *free* não é passível de conflito, ou seja, a justificação acontece garantidamente sem *backtrackings*.

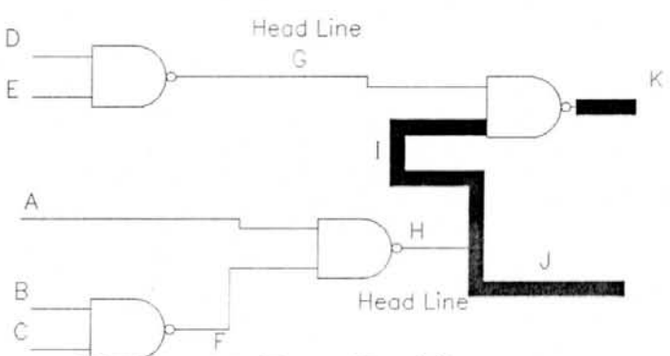


Figura 5.23: *Free* e *Head lines*

Logo, o conflito está intimamente ligado com a presença de fan-outs reconvergentes. A solução encontrada pelo FAN de modo a tentar amenizar o problema envolve a proposição:

Parar o *backtrack* na *head line* e só realizar a justificação do seu valor ao final da geração. Por este motivo, estas linhas são também chamadas de *backtrace-stop lines*

Isto porque sabe-se que as linhas de níveis anteriores à *head line* não podem causar conflito. Portanto é melhor não fazer a justificação do sinal, visto que não se sabe se o valor inserido na *head line* não vai gerar conflito. Caso isto aconteça, o *backtracking* fica muito simplificado, uma vez que basta ao algoritmo retornar até o ponto da *head line*, sem ter que desfazer todo o

trabalho de consistência ou justificação. No caso de não haver conflito, ao final da geração a justificação é feita.

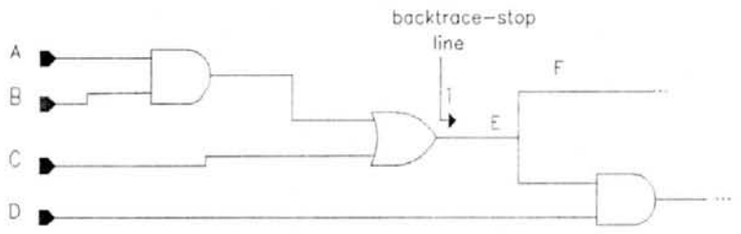


Figura 5.24: Exemplo de *backtrace-stop line* no FAN

Na figura 5.24, a justificação do valor 1 na linha E não seria feita no momento da sua atribuição. Isto porque se acontece um conflito futuro com a linha F, todo o trabalho de atribuir valores às linhas A ,B e C estaria perdido. Estas linhas não são passíveis de conflito, logo pode-se fazer a justificação ao final de toda a geração, quando se tem certeza do valor da linha E.

O *backtrace* múltiplo também é sugerido, visto ser mais eficiente que o *backtrace* simples, onde um só objetivo é definido para uma justificação de um 0 ou 1.

O *backtrace* múltiplo pode ser implementado de muitas formas, uma delas é definindo o objetivo por uma tripla da forma:

$$(S, n_0(S), n_1(S))$$

onde S é uma linha-objetivo, $n_0(S)$ é o número de vezes que o valor lógico 0 é necessário em S e o $n_1(S)$ é o número de vezes que o valor 1 é requerido.

O cálculo de $n_0(S)$ e $n_1(S)$ será explanado adiante. O *backtrace* múltiplo inicia com um conjunto inicial de objetivos e, conforme o andamento,

este conjunto vai sendo alterado e passa a ser o conjunto de objetivos corrente. Um conjunto de objetivos conseguidos nas *head lines* é chamado de conjunto de objetivos *head* e um conjunto de objetivos nos pontos de fan-out é um conjunto de objetivos de fan-out.

Um objetivo inicial para setar 0 na linha S é

$$(S, n_0(S), n_1(S)) = (S, 1, 0)$$

Um objetivo inicial para setar 1 na linha S é

$$(S, n_0(S), n_1(S)) = (S, 0, 1)$$

O próximo objetivo no *backtrace* múltiplo é calculado, como no PODEM, buscando-se sempre as entradas mais facilmente controláveis. Formalmente:

- **Porta AND**

  Seja X a entrada mais facilmente controlável para assumir o valor 0. Então

  $$n_0(X) = n_0(Y), n_1(X) = n_1(Y)$$

  e para as outras entradas X_i:

  $$n_0(X_i) = 0, n_1(X_i) = n_1(Y)$$

  onde Y é a saída da porta AND.

- **Porta OR**

  Seja X a entrada mais facilmente controlável para o valor lógico 1. Então

$$n_0(X) = n_0(Y), n_1(X) = n_1(Y)$$

  e para as outras entradas X_i:

$$n_0(X_i) = n_0(Y), n_1(X_i) = 0$$

  onde Y é a saída da porta OR.

- **Porta NAND**

  Seja X a entrada que pode ser mais facilmente controlada para o valor lógico 0. Então

$$n_0(X) = n_1(Y), n_1(X) = n_0(Y)$$

  e para as outras entradas X_i:

$$n_0(X_i) = 0, n_1(X_i) = n_0(Y)$$

  onde Y é a saída da porta NAND.

- **Porta NOR**

  Seja X a entrada que pode ser mais facilmente controlada para o valor lógico 1. Então

$$n_0(X) = n_1(Y), n_1(X) = n_0(Y)$$

e para as outras entradas X_i:

$$n_0(X_i) = n_1(Y), n_1(X_i) = 0$$

onde Y é a saída da porta NOR.

- **Porta NOT**

$$n_0(X) = n_1(Y), n_1(X) = n_0(Y)$$

- **Ponto de fan-out**

$$n_0(X) = \sum_{i=1}^{k} n_0(X_i), n_1(X) = \sum_{i=1}^{k} n_1(X_i)$$

O fluxograma da figura 5.25 mostra o procedimento do *backtrace* múltiplo. Cada objetivo que chega ao ponto de fan-out congela seu *backtracking* até que todos terminem ou também cheguem ao mesmo fan-out, ou seja, até que a lista de objetivos correntes fique vazia. Após é tomado o ponto de fan-out como objetivo final no processo de *backtracking*. Detecta-se o conflito quando acontece de, no ponto de fan-out, chegar tanto n_0 quanto n_1 diferentes de 0. Neste caso, assume-se 0 se $n_0 > n_1$ ou 1, caso contrário, e é necessária a implicação.

5.4.2 FAST

O FAST [ABR 86] é um algoritmo que se utiliza de duas heurísticas para auxiliar a geração dos testes. A primeira delas é uma variante do *fanin-weight* [PUT 71], onde a medida de controlabilidade tenta refletir o potencial de conflito que a linha oferece, baseando-se nos seguintes cálculos, onde $C(L)$ indica a controlabilidade da linha L:

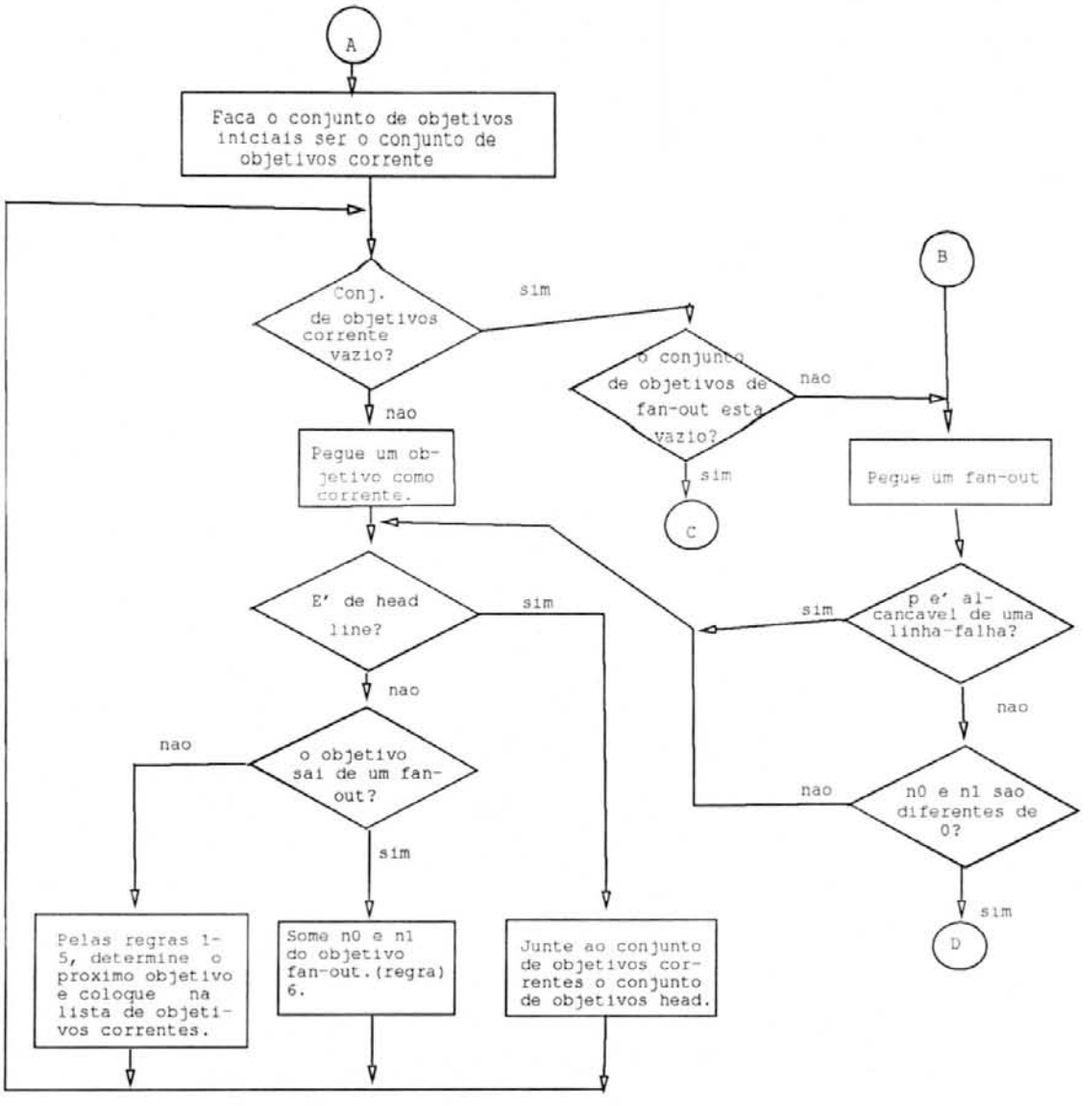


Figura 5.25: Fluxograma do *backtrace* múltiplo

- se L é uma PI, $C(L) = 0$;
- se L é a saída de uma porta

$$C(L) = \sum C(i)$$

onde o somátorio é sobre todas as entradas i da porta.

- se L é um dos caminhos do fan-out i cujo número de caminhos é f_i

$$C(L) = C(i) + f_i - 1$$

Nesta heurística, a controlabilidade de L só é 0 se e somente se ela é uma *free line*. Se $C(L) = 0$ e de L deriva uma linha K com $C(K) \neq 0$, L é uma *head line*, logo, pode-se utilizar esta medida para identificar as *head lines*.

Esta medida, entretanto, é estritamente topológica, e foi modificada no FAST de modo a pesar também a função lógica, explorando a dominância presente em certas funções. Para isto, usa-se duas funções de controlabilidade, $C0$ e $C1$, que significa uma medida de controlabilidade para conflito na inserção do valor 0 ou 1, respectivamente.

A segunda heurística utilizada pelo FAST penaliza só os fan-outs reconvergentes pois são eles que causam os conflitos. O problema nesta heurística é determinar os fan-outs reconvergentes, mas o FAST faz isto armazenando na estrutura do circuito, junto aos fan-outs, as saídas capazes de serem alcançadas daquele ponto de fan-out.

O cálculo modificado de forma a embutir também penalidades de acordo com a facilidade ou não de obtenção de determinado valor baseando-se no funcionamento lógico da porta é o seguinte:

- se L é uma PI, $C0(L) = C1(L) = 0$;
- se L é a saída de uma porta AND

$$C0(L) = \min_i(C0(i))$$

$$C1(L) = \sum_i C1(i)$$

onde i são todas as entradas da porta

- se L é um dos caminhos do fan-out i

$$C1(L) = C1(i) + f_i - 1$$

$$C0(L) = C0(i) + f_i - 1$$

O raciocínio para as outras portas é similar.

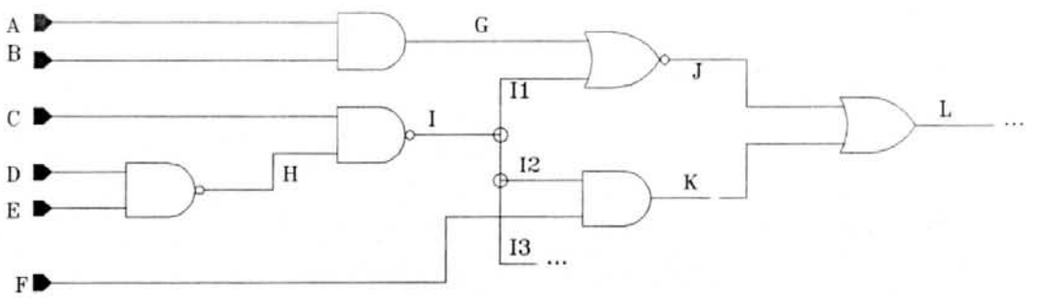


Figura 5.26: Circuito exemplo

linha	Primeira medida	Segunda Medida	
	C	C0	C1
A	0	0	0
B	0	0	0
C	0	0	0
D	0	0	0
E	0	0	0
F	0	0	0
G	0	0	0
H	0	0	0
I	0	0	0
I1	2	2	2
I2	2	2	2
I3	2	2	2
J	2	0	2
K	2	0	2
L	4	0	2

Tabela 5.5: Cálculos pelos dois métodos

Na tabela 5.5 que refere-se à figura 5.26 pode-se verificar que, enquanto a primeira medida identifica F, G e I como *head lines*, a segunda identifica L como *backtrace-stop line* só para o valor 0 e F, G e I como *backtrace-stop lines* para o valor 1. Facilmente percebe-se que L pode ser justificado com valor 0 sem que se precise atribuir valores lógicos a linha I.

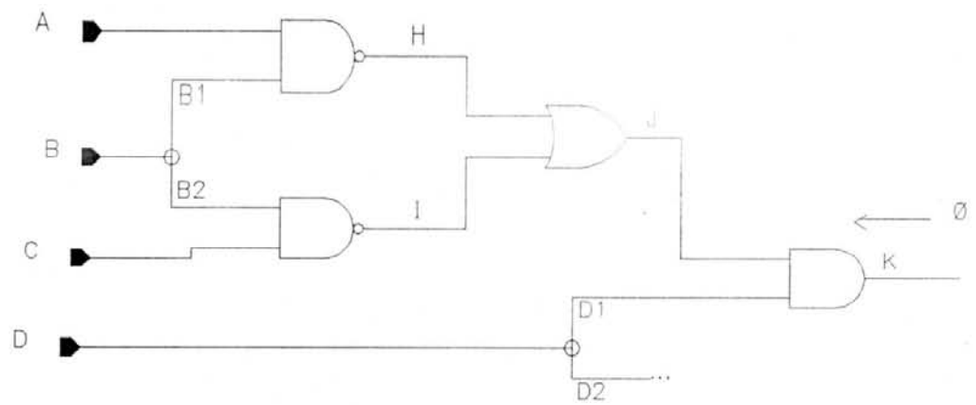


Figura 5.27: Exemplo do cálculo do FAST

linha	C0	C1
A	0	0
B	0	0
B1	1	1
B2	1	1
C	0	0
D	0	0
D1	1	1
D2	1	1
H	1	0
I	1	0
J	2	0
K	1	1

Tabela 5.6: Cálculos pelo FAST

Na tabela 5.6 que se refere ao circuito da figura 5.27 vê-se os valores de controlabilidade para os valores 1 e 0, calculadas com as heurísticas do FAST. As linhas A, B, C e D têm seus valores em 0 em vista da sua condição de entradas primárias, já $B1$, $B2$, $D1$ e $D2$, têm seus valores acrescidos de 1

porque são desvios de fan-outs que possuem mais um desvio além do próprio que está sendo calculado. H e I têm seu valor de $C0$ calculado em 1 porque, para setar estas linhas em 0 é necessário setar as duas entradas, A e $B1$ ou C e $B2$, para 1, logo, é preciso somar as $C1$'s destas entradas. O cálculo de $C1$ é similar, a diferença é que para esta porta é necessário setar apenas uma das entradas para 0, portanto usa-se a menor $C0$, ou seja, da linha com o 0 mais fácil de se conseguir, assumindo-se $C1(H) = C0(A)$ e $C1(I) = C0(C)$. O resto dos cálculos é feito de modo análogo.

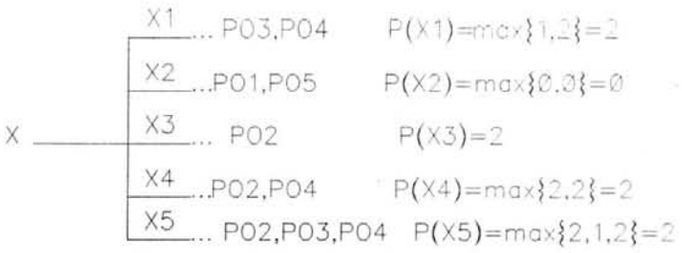


Figura 5.28: Identificação das reconvergências

Os fan-outs que tem a mesma PO como saída são reconvergentes. A penalização, neste caso, não é calculada simplesmente somando-se ao seu valor o número de outros caminhos de fan-out mas sim a penalidade assume o valor do número máximo de reconvergências. Por exemplo, na figura 5.28 o caminho $X1$ leva a duas saídas, $PO3$ e $PO4$. Na saída $PO3$ ele reconverge com o outro caminho $X5$, logo, para este caminho a penalidade é 1. Na $PO4$ ele reconverge com os caminhos $X4$ e $X5$, logo, para este caminho a penalidade é 2. Como deve ser assumido o maior valor, a penalidade para $X1$ é 2, ou seja o maior número de reconvergências. Note-se que caminhos não reconvergentes não são penalizados.

Na figura 5.29 observa-se a vantagem desta medida calculada com a segunda heurística. Identifica-se T como *backtrace-stop line* só para o valor 0 e ainda, que $T = 1$ deve ser justificado através de $S = 1$, porque esta atribuição não causa conflitos, visto que seu fan-out não é reconvergente. Com o uso da primeira medida não consegue-se identificar que $Q2$ e $P1$ não causam conflito.

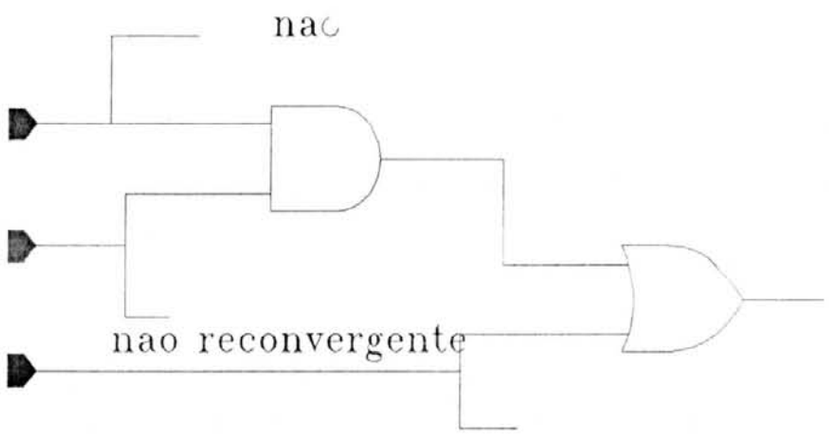


Figura 5.29: Exemplo de ganhos com a segunda heurística

linha	Primeira medida		Segunda Medida	
	C0	C1	C0	C1
S	1	2	0	0
P	0	0	0	0
P1	1	1	0	0
P2	1	1	0	0
Q	0	0	0	0
Q1	1	1	0	0
Q2	1	1	0	0
R	0	0	0	0
R1	1	1	1	1
R2	1	1	1	1
T	2	1	1	0

Tabela 5.7: Comparação entre as heurísticas

O FAST também usa valores de observabilidade calculados da seguinte maneira formal e que também é utilizada por outros algoritmos de ATPG:

- se L é uma PO, $O(L) = 0$;
- se L é a entrada de uma AND com a saída K

$$O(L) = O(K) + C\mathbf{1}(K) - C\mathbf{1}(L)$$

onde $C1(K) - C1(L)$ é o custo de setar todas as outras entradas para o valor capaz de propagar, no caso da AND, 1. O cálculo para as outras portas é semelhante:

- se L é um fan-out

$$O(L) = \min_i \{O(i)\}$$

onde i são todos os caminhos de L.

Outra técnica utilizada no FAST de modo a gerar um bom conjunto de testes é a compactação dinâmica [ABR 86].

5.4.3 GLOBAL

Os métodos estruturais abordados anteriormente faziam alusão aos fan-outs reconvergentes como potenciais de conflito. Estes algoritmos, entretanto, não levavam em consideração a estrutura global do circuito.

A função custo do GLOBAL leva em consideração a topologia total do circuito, analisando a estrutura global dos fan-outs [ABR 90]. É uma extensão do FAST, levando em conta que o potencial para conflito causados pelos fan-outs reconvergentes nem sempre é o mesmo, dependendo da estrutura total do circuito.

Seu cálculo de observabilidade é feito com as mesmas regras utilizadas pelo SCOAP. Os valores, entretanto, são distintos já que a controlabilidade é calculada de modo distinto.

O problema encontrado no FAST e que tenta ser solucionado pelo GLOBAL é o fato de que por vezes os fan-outs não prejudicam a inserção de

constantes, e isto não pode ser detectado sem uma visão global da topologia do circuito. O FAST penaliza todo e qualquer fan-out reconvergente, o GLOBAL, por sua vez, penaliza mas com restrições.

O algoritmo GLOBAL baseia-se também no princípio da identificação das *backtrace-stop lines*, porém com uma análise do tipo de reconvergência. Este algoritmo consegue identificar os fan-outs reconvergentes capazes de gerar conflito, já que existem fan-outs que, apesar de reconvergentes, podem ser justificados sem conflito.

Por exemplo, na figura 5.30, suponha-se que seja necessário justificar $K = 0$. A função custo do GLOBAL deve guiar o *backtracking* para J visto que, apesar da reconvergência presente neste caminho, não há possibilidade de conflito. Entretanto, o FAST deve selecionar $D1 = 0$ porque $C0(D1) < C0(J)$. Como o fan-out D é reconvergente isto pode causar um conflito mais tarde na colocação de outros valores. O cálculo do GLOBAL é capaz de identificar globalmente a situação.

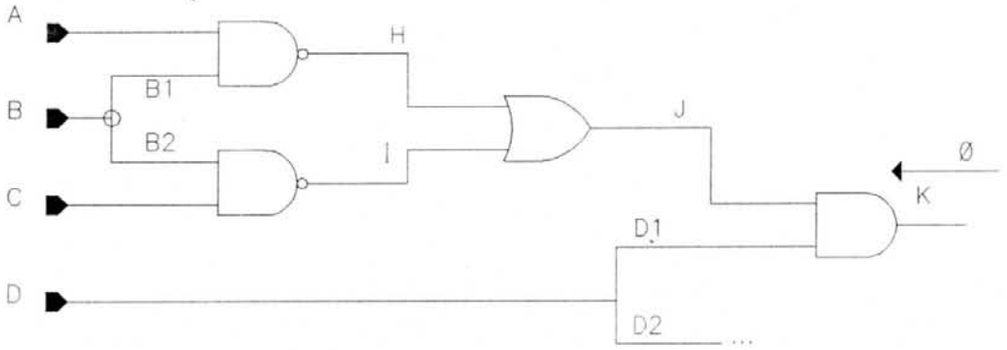


Figura 5.30: Fan-out reconvergente sem possibilidade de conflito

O GLOBAL utiliza a computação das penalidades de acordo com as regras do FAST, apenas não penalizando os fan-outs que reconvergem com paridades diferentes (conceito introduzido na seção 5.3.1 sobre o algoritmo *Cutting*). O valor imputado pelo GLOBAL ao fan-out f é o número máximo de outros desvios que reconvergem para f com paridades diferentes, ou seja, que o número de inversões do sinal em cada caminho é diferente.

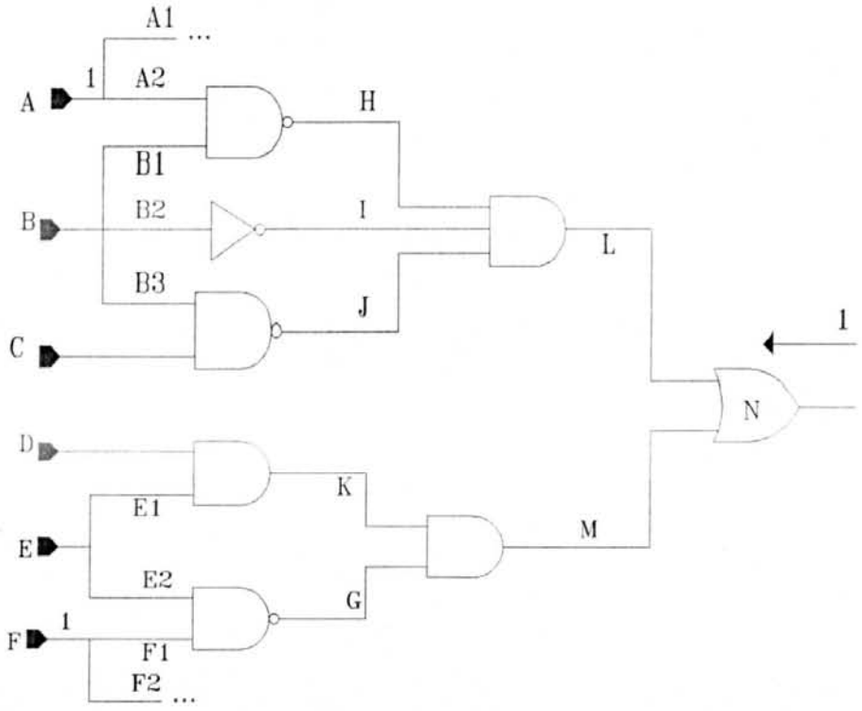


Figura 5.31: Identificação das paridades

Isto pode ser visto na figura 5.31. O fan-out B, por exemplo, não é penalizado no GLOBAL, visto que seus três caminhos reconvergem para a porta L com a mesma paridade 1. Assim,

$$C\mathbf{1}(B) = C\mathbf{1}(B\mathbf{1}) = C\mathbf{1}(B\mathbf{2}) = C\mathbf{1}(B\mathbf{3}) = 0$$

Já no fan-out E, isto não acontece. A paridade do caminho $\{E\mathbf{1}, K, M\}$ tem paridade 0 e $\{E\mathbf{2}, G, M\}$ paridade 1, de modo que $C\mathbf{1}(E) = 0$ e $C\mathbf{1}(E\mathbf{1}) = C\mathbf{1}(E\mathbf{2}) = \mathbf{1}$.

Procedimento algorítmico do GLOBAL

De forma a conseguir identificar os fan-outs que não geram conflito, o GLOBAL armazena algumas informações adicionais em seus cálculos que auxiliam esta identificação.

Para cada linha L e valor lógico v (0 ou 1), mantém-se um conjunto de atribuições necessárias ($SA(L,v)$) da forma $SA(L,v) = \{(S_i = v_i), (S_j = v_j), \ldots\}$ onde $(S_i = v_i)$ são as atribuições que são necessárias para que aquele valor se apresente naquela linha e S_i é um desvio que reconverge com paridade distinta.

As regras para o cálculo de SA são:

- se L é uma PI

$$SA(L,v) = \{(L = v)\}$$

se L for um fan-out que reconverge com paridade distinta, e

$$SA(L,v) = \emptyset$$

caso contrário.

- se L é a saída de um inversor com entrada i

$$SA(L,v) = SA(i,\overline{v})$$

- se L é a saída de uma porta AND

$$SA(L,0) = SA(i,0)$$

onde i é a entrada mais controlável para 0.

$$SA(L,\mathbf{1}) = \bigcup_i SA(i,\mathbf{1})$$

onde i representa todas as entradas da porta.

Quando se realiza esta união deve-se verificar se não ocorre um conflito, ou seja, o mesmo sinal recebendo valores distintos. Isto caracteriza o conflito e nestes casos deve ser acrescentado à $C1(L)$ uma constante de ajuste K (sugere-se 100 [ABR 90]).

As regras para as outras portas são similares

- se L é um fan-out com desvio s

$$SA(L,v) = SA(s,v)$$

a menos que $SA(s,v) = \emptyset$ e s seja um fan-out com paridade distinta. Neste caso, $(s = v)$ é adicionado à $SA(L,v)$ como seu único elemento. Isto minimiza o número de elementos no conjunto SA.

$$SA(L,v) = \{(s = v)\}$$

se $SA(s,v) = \emptyset$ e s é um fan-out que reconverge com paridade diferente, e

$$SA(L,v) = SA(s,v)$$

caso contrário.

GLOBAL identifica também as atribuições que podem ser justificadas sem conflito. Para fazer isto, GLOBAL calcula para cada linha L e valor

v uma conexão de controlabilidade reconvergente $CR(L, v)$, que é a saída da porta mais distante para a qual as entradas em $SA(L, v)$ reconvergem.

As conexões CR juntamente com o conjunto SA servem para detectar *backtrace-stop lines*. Por exemplo na figura 5.30 o fan-out B só é utilizado para justificar $H = 0$ ou $I = 0$, visto que para $H = 1$ ou $I = 1$ pode-se utilizar as PI's A e C, respectivamente. Consequentemente B só é utilizado para justificar $J = 0$. Desde que a conexão reconvergente de B é J, $CR(B, 0) = J$.

Para qualquer atribuição $(b = v)$ se o nível de entrada de $CR(L, v)$ é menor ou igual ao nível de L e no conjunto de atribuições não aparece conflito, ou seja , o mesmo sinal com atribuições diferentes, então L é identificado como *backtrace-stop line* para o valor v e o seu custo é colocado para 0.

Por exemplo na figura 5.30 $CR(J, 0) = J$. Isto significa que qualquer caminho utilizado para controlar $J = 0$ vai reconvergir para J. Como não há conflitos no conjunto de atribuições SA, J é identificado como *backtrace-stop line* para o valor 0.

As portas CR são determinadas numa busca das PI's para as PO's, juntamente com o cálculo dos valores de atribuição e controlabilidade.

Para cada linha L e valor lógico (0 ou 1) v, $CR(L, v)$ é calculado:

- se L é uma PI

$$CR(L, v) = L$$

- se L é a saída de um inversor com entrada i

$$CR(L, v) = CR(i, \overline{v})$$

- se L é a entrada de uma porta AND

$$CR(L,0) = CR(i,0)$$

onde i é a entrada com a mínima $C0$.

$$CR(L,1) = \max_i\{CR(i,1)\}$$

onde i abrange todas as entradas e max indica a seleção da porta de mais alto nível.

As regras para as outras portas são análogas.

- se L é um desvio de fan-out s, seu CR é a porta reconvergente de mais alto nível, isto é, o sinal mais distante para onde reconverge o fan-out.

$$CR(L,v) = \max\{CR(s,v), MR(L)\}$$

onde MR significa *Maximum Reconvergence Gate*, ou seja a porta com a máxima reconvergência.

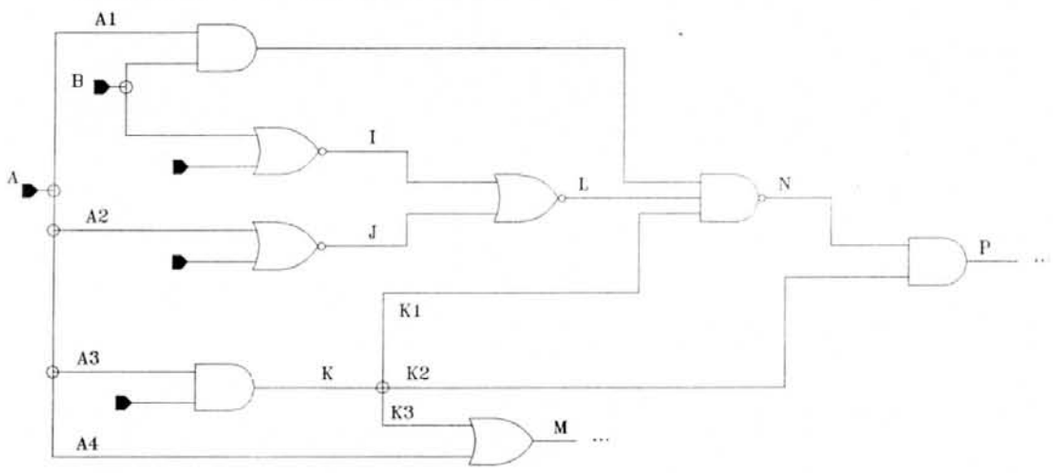


Figura 5.32: Circuito reconvergente para ilustração de portas MR

f	p(f)	MR(f)
A1	1	P
A2	1	P
A3	2	P
A4	0	M
B1	0	N
B2	0	N
K1	1	P
K2	1	P

Tabela 5.8: Penalidades e portas MR

Para ilustrar este conceito de porta MR, pode-se observar a figura 5.32 e a tabela 5.8.

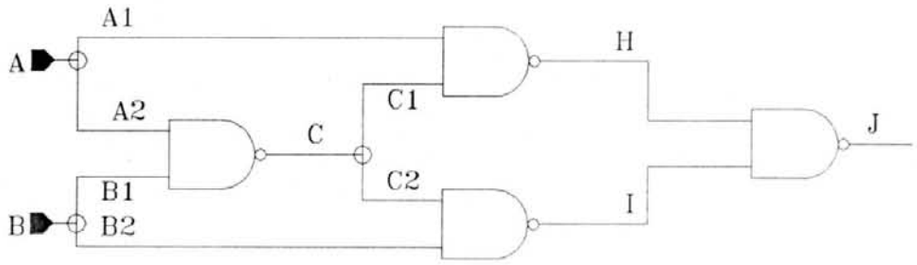


Figura 5.33: Circuito exemplo do GLOBAL

Na figura 5.33 e tabela 5.9 verifica-se todos os cálculos feitos utilizando-se estas regras. Note-se que os fan-outs A e B são penalizados visto que as paridades de seus caminhos não são iguais, já o fan-out C não é penalizado pelo motivo oposto. Percebe-se ainda neste exemplo, que o CR utilizado depois das PI's é o J, pois é a porta reconvergente mais distante (maior nível) e que, quando da chegada na própria porta J, que é *uma backtrace stop line*, como não se verificou conflito no conjunto SA, a controlabilidade foi determinada como sendo 0, tanto para 1 quanto para 0.

atribuições	SA	CR	FAST	GLOBAL
A=0	A=0	A	0	0
A=1	A=1	A	0	0
B=0	B=1	B	0	0
B=1	B=1	B	0	0
A1=0	A=0	J	1	1
A1=1	A=1	J	1	1
A2=0	A=0	J	1	1
A2=1	A=1	J	1	1
B1=0	B=0	J	1	1
B1=1	B=1	J	1	1
B2=0	B=0	J	1	1
B2=1	B=0	J	1	1
C=0	A=1,B=1	J	2	2
C=1	A=0	J	1	1
C1=0	A=1,B=1	J	3	2
C1=1	A=0	J	2	1
C2=0	A=1,B=1	J	3	2
C2=1	A=0	J	2	1
H=0	A=1,A=0	J	3	102
H=1	A=0	J	1	1
I=0	A=0,B=1	J	3	2
I=1	B=0	J	1	1
J=0	A=0,B=0	J	2	0
J=1	A=0,B=1	J	3	0

Tabela 5.9: Cálculos com o GLOBAL

5.5 Técnica de Pré-implicação

Uma técnica viável para reduzir a complexidade dos algoritmos de ATPG e que é proposta pela primeira vez neste trabalho, é a de pré-implicar todas as justificações de forma a acelerar o *backtracking*. Neste caso não haveria a necessidade do cálculo do vetor de justificação no momento da geração do teste. O circuito já teria armazenado junto a cada porta todos os vetores que justificariam os valores 0 e 1.

Observando a figura 5.34 verifica-se que para justificar 1 na linha g, basta escolher o único vetor disponível para 1, $XXX11X1XXX$. Caso tivesse

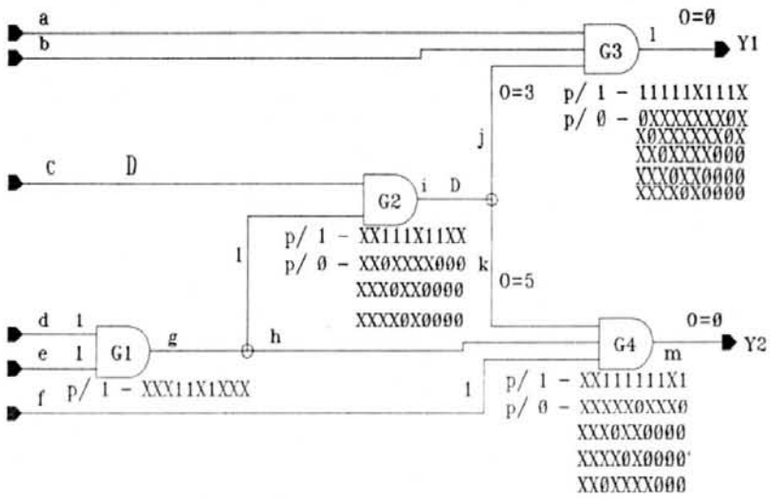


Figura 5.34: Circuito com cubos pré-implicados

mais de um vetor seria escolhido qualquer um, caso este vetor, mais tarde, não satisfizesse a propagação causando conflito, bastaria voltar a este ponto e escolher outro vetor, desconsiderando a escolha anterior. Suponha-se que é necessário um valor 0 na linha i, basta escolher um entre os quatro possíveis armazenados junto com a porta lógica G2.

O cálculo das pré-implicações é bem simples e é importante observar que, uma vez calculado, serve para a geração de todos os testes, não necessitando recálculo. Sua grande desvantagem é a quantidade de memória gasta para armazenar os vetores tornando o método proibitivo dependendo do tamanho do circuito.

Uma heurística que pode ser utilizada na técnica de pré-implicação, quando da chegada em um ponto de fan-out, é observar se as linhas que estão envolvidas na porta já têm valores atribuídos e se estes auxiliam ou prejudicam a propagação. Por exemplo no circuito da figura 5.35, ao se ter D na saída da porta G6, com a posse do vetor resultante $XXXXX111XXXX1XXXDXXX$, facilmente se poderia inferir, na comparação com os dois vetores de propagação disponíveis que a utilização do primeiro padrão ($XXXXXXXXXXXX1XX$-$XD1XD$) é melhor porque envolve apenas mais uma atribuição, já o segundo padrão que também pode ser utilizado ($XXXXXXXXXXXXX11DXDX$), a colocação de mais dois valores.

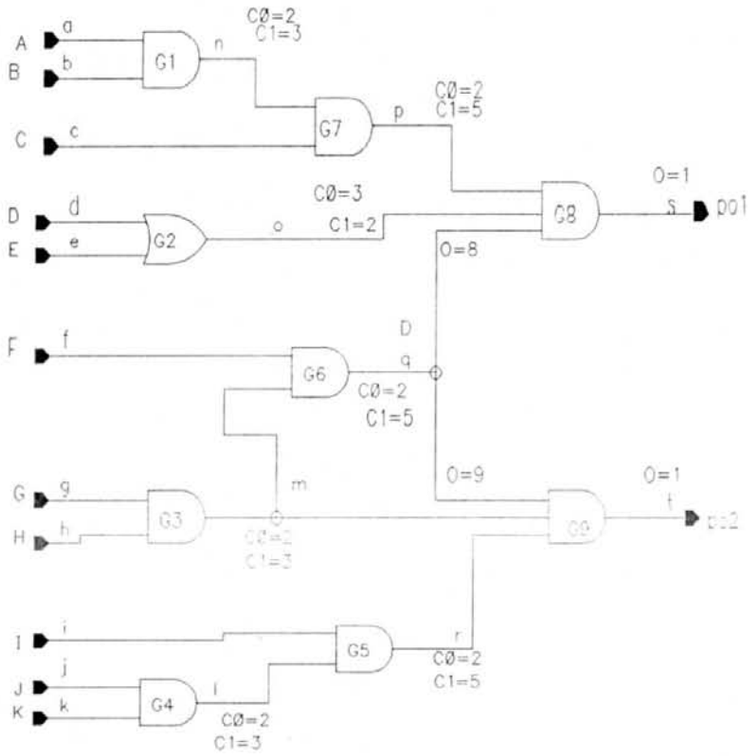


Figura 5.35: Circuito de exemplo da heurística

Só isto, entretanto, não é suficiente, já que poderia acontecer de, por exemplo, o vetor que ter duas atribuições, tem atribuições que são diretas às PI's e o outro, com apenas uma atribuição, que na verdade é a saída de um grande subcircuito. Nesta técnica não parece ser diretamente importante este fator já que não há a necessidade da implicação, porém, quanto mais valores inseridos, mais chance de conflitos futuros acontecem. Para detectar isto, basta interseccionar os dois vetores e verificar qual intersecção envolve menor inserção de constantes.

A tabela 5.10 mostra o procedimento de escolha para o exemplo da figura 5.35.

Cubo Corrente	Comentário
XXXXX111XXXX1XXXDXXX	Cubo inicial
XXXXXXXXXXXX1XXXD1XD	Primeira alternativa para propagar
XXXXX111XXXX1XXXD1XD	Resultado da intersecção
	Colocação de apenas um valor na linha r
XXXXXXXX1111XXXXX1XX	Cubo de justificação de $r = 1$
XXXXX11111111XXXD1XD	Resultado final
	Coloca-se 4 constantes na justificação
XXXXX111XXXX1XXXDXXX	Cubo inicial
XXXXXXXXXXXXX11DXXD	Segunda alternativa para propagar
XXXXX111XXXX1X11DXXD	Resultado da intersecção
	Colocação de dois valores nas linhas o e p
XXX11XXXXXXXXX1XXXXX	Cubo de justificação de $o = 1$
111XXXXXXXXXX1X1XXXX	Cubo de justificação de $p = 1$
11111XXXXXXXX111DXXD	Resultado Final

Tabela 5.10: Procedimento de escolha

5.6 Técnicas Diversas

Por vezes ao utilizarmos as técnicas descritas anteriormente pode acontecer de encontrarmos controlabilidades e observabilidades idênticas.

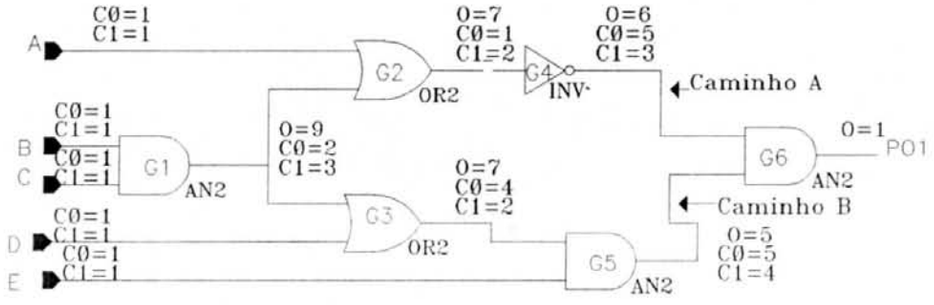


Figura 5.36: Técnicas diversas como desempate

Por exemplo, na figura 5.36 vemos que a observabilidade do ponto de divergência do sinal de saída da porta G1 é a mesma para os dois caminhos, porém pode ser mais interessante escolher o caminho A, já que se faz, na propagação por ele, uma atribuição a menos a uma PI. Estas pequenas

heurísticas podem ser utilizadas se necessário. Exemplos destas heurísticas podem ser, escolher caminhos com menos XOR's, aproveitando-se de sua simetria, ou seja, a inexistência de dominância, ou com mais NOT's, que neste caso não seriam penalizados.

5.7 Momento do Cálculo

O cálculo das medidas de testabilidade pode ser feito de duas maneiras. De modo estático, ou seja, as medidas são calculadas num primeiro momento e são utilizadas durante todo o processo de ATPG, sem nenhuma alteração. Ou de modo dinâmico, isto é, as medidas são recalculadas sempre que valores são inseridos nas linhas, já que estas inserções normalmente modificam a controlabilidade que foi calculada inicialmente.

5.7.1 Medidas estáticas

As medidas estáticas são medidas que são calculadas uma vez, num estágio de pré-processamento e levando em consideração apenas o momento em que as linhas não tem nenhum valor atribuído, ou seja, só tem valores X (*don't care*). O problema destas medidas é que, com o decorrer do processo de geração, esta realidade pode ir se modificando e, neste caso, estas medidas não são capazes de identificar a situação, por vezes guiando a geração por caminhos mais difíceis que, antes da atribuição, pareciam ser mais facilmente controláveis ou observáveis.

As medidas estáticas, porém, têm a grande vantagem de não imputarem grande custo computacional, uma vez que são calculadas apenas uma única vez.

5.7.2 Medidas dinâmicas

As medidas dinâmicas são medidas que tentam garantir a precisão dos valores durante todo o processo de ATPG. Estas medidas são recalculadas sempre que alguma constante é inserida, ou, tentando não repetir tantas vezes este recálculo, o que gasta excessivo processamento, recalculadas sempre depois que um número fixo de constantes são inseridas.

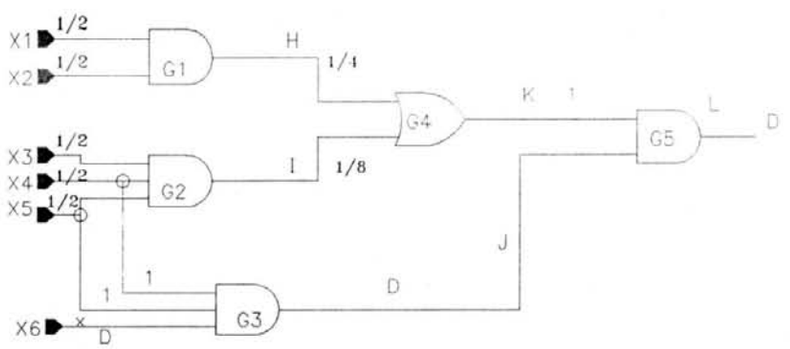


Figura 5.37: Vantagem das medidas de testabilidade dinâmicas para controlabilidade

Para esclarecer o conceito, observa-se a figura 5.37, onde assume-se que a linha $X6$ está falha. A falha deve ser propagada por G3 e G5 até a saída L. Para isto as linhas $X4$ e $X5$ devem assumir o valor lógico 1, para propagar G3 e K também deve receber 1 para propagar por G5. A justificação de $K = 1$ pode ser feita de duas maneiras. Colocando-se $H = 1$ ou $I = 1$. No caso da utilização de medidas estáticas, a escolha que seria feita, seria a linha $H = 1$, porque, antes da inserção dos valores constantes, era, realmente mais fácil a obtenção de um 1 nesta linha, já que isto implicava em colocar-se apenas $X1$ e $X2$ em 1 e que colocar $I = 1$, implicava em colocar $X3$, $X4$ e $X5$ em 1. Ocorre, porém, que agora, depois da atribuição das contantes, a setagem de $I = 1$ necessita de apenas uma atribuição, a de $X3$ igual a 1. Assim, utilizando-se a técnica dinâmica, o recálculo seria feito, atribuindo controlabilidade menor à linha I que a linha H, ou seja, 1/2 e 1/4, respectivamente, assumindo-se 0 o valor mais controlável e 1 o menos controlável. Desta forma, a escolha

seria mais precisa. A principal desvantagem disto, porém, é que nem sempre o recálculo é utilizado e o processamento da técnica é bastante expressivo.

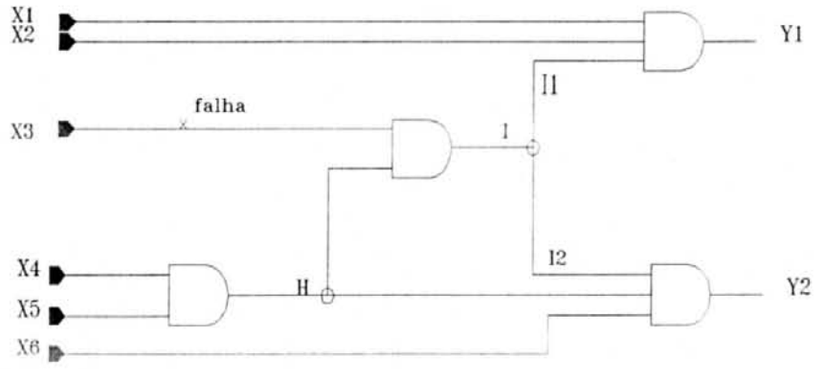


Figura 5.38: Vantagem das medidas de testabilidade dinâmicas para observabilidade

O mesmo conceito analisado na figura em termos de controlabilidade é utilizado para a observabilidade, já que a presença de constantes altera, da mesma forma, a facilidade de se observar determinado sinal. Isto pode ser visto na figura 5.38. Supõe-se uma falha na entrada primária $X3$. Neste momento se observarmos estaticamente a observabilidade, ou seja, assumindo o circuito sem sinais atribuídos, pode-se afirmar que, para a propagação de um sinal que estivesse na linha I, o caminho $I1$ é mais observável porque para propagar um sinal por ele basta setar $X1$ e $X2$ para 1, enquanto que para propagar por $I2$ é necessário $X4$, $X5$ e $X6$ em 1, e ainda utiliza-se um caminho com divergência de dados (linha H).

Continuando o exemplo, para propagar-se a falha em $X3$ para o ponto I é necessário atribuir $H = 1$. A partir deste momento, o caminho por $I1$ deixou de ser mais observável, visto que para propagar por $I2$ basta $X6 = 1$.

6 A UTILIZAÇÃO CONJUGADA DAS HEURíSTICAS

Utilizar as medidas de testabilidade estruturais, aliadas às Medidas de Testabilidade Funcionais e Probabilísticas (MTFP), é útil de forma a se conseguir medidas mais precisas, que reflitam a controlabilidade e observabilidade do circuito com uma visão mais ampla, levando em conta mais de um aspecto.

Algumas medidas apresentadas já têm este conceito em sua essência. É o caso do FAST, que alia à sua visão estrutural, que procura identificar as *backtrace-stop lines*, a visão funcional, visto que utiliza cálculos diferenciados a partir da função da porta. Um aspecto, entretanto, não consegue ser enfocado nesta medida: o da distância. Este algoritmo não penaliza as portas pelas quais vai passando enquanto não encontra um fan-out, assim, todas as porções do circuito que não apresentam possibilidade de conflito ficam com controlabilidades iguais a zero. Com isto, não é possível a identificação, quando da justificação, do caminho mais longo nestes locais.

	Medidas Funcionais	Medidas Estruturais	Medidas Probabilísticas	Técnica de Pré-implicação	Técnicas Diversas
MF		X		X	X
ME	X		X	X	X
MP		X		X	X
TP	X	X	X		X
TD	X	X	X	X	

Tabela 6.1: Conjugações Permitidas

Na tabela 6.1 pode-se perceber quais as técnicas que podem coexistir. Segue-se uma análise detalhada destas combinações.

6.1 Uso do FAN com MTFP

Esta combinação pode ser feita já que falamos de uma medida estrutural combinada com MTFP. O problema que se apresenta nesta combinação, porém, é que as MTFP's não contêm nenhuma informação que permita identificar as *backtrace-stop lines* para que se consiga delimitar até onde o *backtracking* deve ser efetuado. Por este motivo, seriam necessários os dois cálculos, tanto o cálculo capaz de identificar as *backtrace-stop lines* quanto o cálculo da MTFP. Haveriam ganhos em relação ao uso das MTFP utilizadas sozinhas, uma vez que se otimizaria a execução dos *backtrackings*, porém, teria-se o cálculo de identificação das *backtrace-stop lines* a mais, neste caso.

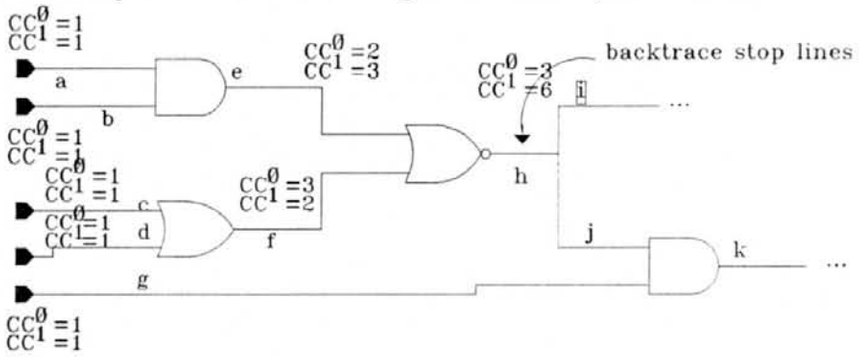


Figura 6.1: Conjugação do FAN e SCOAP

Na figura 6.1 vê-se o circuito com o SCOAP calculado, onde se verifica que a medida de controle do SCOAP não tem nenhum valor que identique a posição da *backtrace stop line*, porém, nestes cálculos do SCOAP pode-se acrescentar rotinas que identifiquem estes pontos. Os valores calculados servem no final, na etapa de justificação dos pontos que ficam entre as PI's e as *backtrace stop lines*, para identificar o menor número de atribuições ou o menor caminho.

No circuito da figura 6.1, por exemplo, se for necessário justificar no final da propagação um valor lógico 0 na linha h, que é a *backtrace stop line*, as medidas do SCOAP guiariam a justificação pela linha e, onde é necessário apenas a atribuição a uma PI.

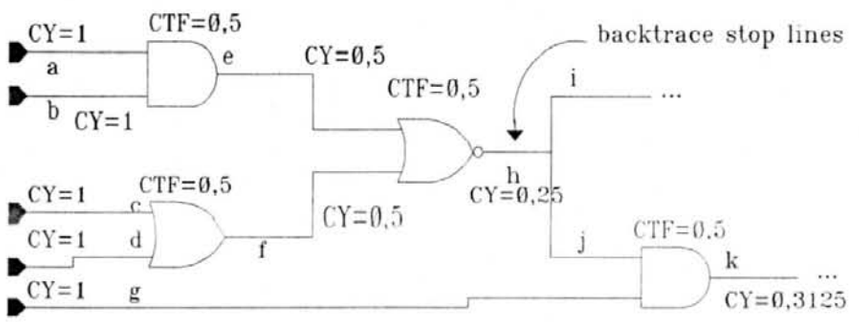


Figura 6.2: Conjugação do FAN e CAMELOT

Na figura 6.2 verifica-se que com o CAMELOT acontece a mesma coisa. A diferença encontra-se no fato do CAMELOT avaliar somente as distâncias. Com esta medida não é possível identificar-se que a linha e acarreta um menor número de atribuições.

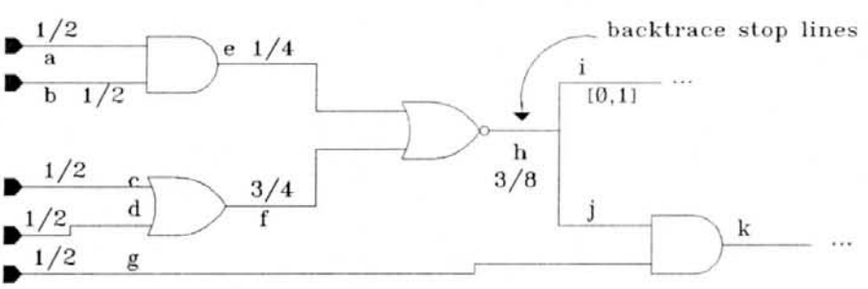


Figura 6.3: Conjugação do FAN e medidas probabilísticas

Na figura 6.3 apresenta-se um circuito com os valores probabilísticos calculados. Estas medidas em si não trazem indicações de *backtrace stop lines*, porém, como estas medidas precisam da determinação das posições dos fan-outs para fazer os cálculos, devido a relação existente entre os sinais, a determinação destas informações será imprescindível de qualquer forma. Tanto o PREDICT como o COP e o CUTTING utilizam esta determinação.

6.2 Uso do FAST com MTFP

O FAST caracteriza-se por não penalizar os caminhos até encontrar as *backtrace-stop lines*, com isto esta medida é capaz de facilmente identificar estes pontos. O problema é que no momento da justificação final, ou seja, que a propagação terminou e já se sabe que não haverão alterações e nem conflitos com as constantes inseridas, este algoritmo não tem valores capazes de auxiliar o processo de justificação.

Com este objetivo poderia ser usado qualquer MTFP e, neste caso, não seria necessário calcular os valores para todas as linhas do circuito, e sim, para esta região que os necessita, que é a área contida entre as entradas primárias e as *backtrace-stop lines*.

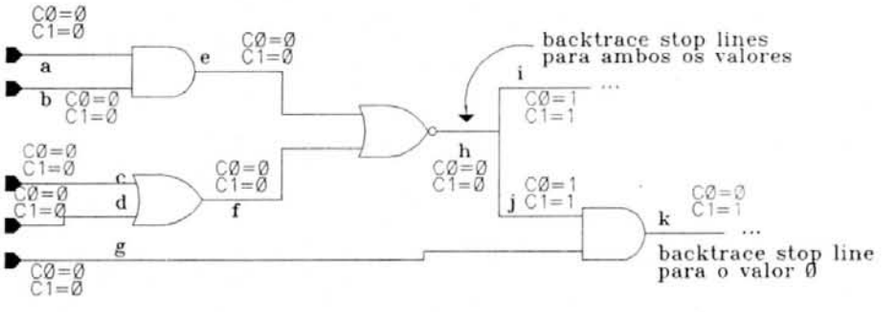


Figura 6.4: Cálculos no FAST

Na figura 6.4 verifica-se que as medidas do FAST não penalizam as conexões até encontrarem um fan-out (preferencialmente, com a possibilidade de identificá-lo como reconvergente). Desta forma, pode-se propagar sem se preocupar com esta porção do circuito. Por exemplo, para um valor na linha h, as linhas a, b, c, d, e e f podem ser justificadas somente no final. Estas linhas, porém, não têm valores calculados que auxiliam a justificação final. Para isto, após toda a propagação, quando o valor de h não vai mais ser alterado, poderia-se utilizar uma MTFP, como por exemplo SCOAP para auxiliar a justificação. Neste caso só seriam necessários os cálculos para as linhas a, b, c, d, e e f.

6.3 Uso do GLOBAL com MTFP

O GLOBAL [ABR 90] consegue identificar que existem fan-outs que, apesar de reconvergentes, não causam conflito e por isso não devem ser penalizados. Porém, como o FAST, este algoritmo não carrega valores capazes de auxiliar a justificação final. Por isto, da mesma forma que no FAST, as MTFP podem ser utilizadas com este fim.

6.4 Uso da Técnica de Pré-implicação com medidas de observabilidade

Na técnica de pré-implicação não há necessidade de medidas de controlabilidade, visto não existir a etapa de justificação de valores. Entretanto as medidas de observabilidade ainda são necessárias para guiar a propagação. Assim, qualquer das medidas pode ser utilizada com este propósito.

6.5 Uso da Técnica de Pré-implicação com o conceito de *Backtrace Stop Line*

De forma a diminuir o espaço ocupado pela técnica de pré-implicação, pode-se utilizar o conceito de *backtrace stop lines* e pré-calcular apenas para estas porções. Com isto consegue-se balancear entre as desvantagens dos dois métodos. Esta técnica é aconselhada para circuitos com estas porções bem balanceadas. Para a identificação das *backtrace stop lines*, o algoritmo mais indicado é o GLOBAL por não penalizar os fan-outs reconvergentes com igual paridade, que é o fator de diminuição dos cubos pré-implicados.

6.6 Uso do Cálculo Dinâmico nos diversos Algoritmos

É apresentada aqui uma visão geral dos vários algoritmos e combinações, quando da utilização dos recálculos dinâmicos.

6.6.1 MTFP calculadas dinamicamente

Todas as MTFP podem ter seus valores recalculados dinamicamente. O principal problema disto, é que despende-se tempo e processamento excessivos.

6.6.2 FAST conjugado com MTFP e cálculo dinâmico

O problema dos cálculos dinâmicos é o tempo de processamento que é necessário para os recálculos. Duas soluções são propostas neste trabalho. Uma solução é o recálculo de apenas partes do circuito, e é denominada de de heurística do *Recálculo Parcial*. A outra solução baseia-se na ativação do recálculo apenas para as linhas não *free*, e é denominada *Recálculo somente em linha não free*.

6.6.2.1 Recálculo parcial

O Recálculo Parcial pode ser feito com a utilização das medidas do FAST, que permitem a identificação das *backtrace-stop lines* juntamente com o recálculo dinâmico. Desta forma, pode-se recalcular valores somente a partir das *backtrace stop lines*, considerando-as como entradas virtuais do circuito. Estes recálculos, porém, devem ser feitos com uma MTFP, já que o FAST não

permite a identificação das distâncias, ou seja, os caminhos mais longos e com mais atribuições.

Para a justificação final, são suficientes valores estáticos, calculados apenas uma vez, já que como não há possibilidade de conflito, não existe necessidade de recálculo.

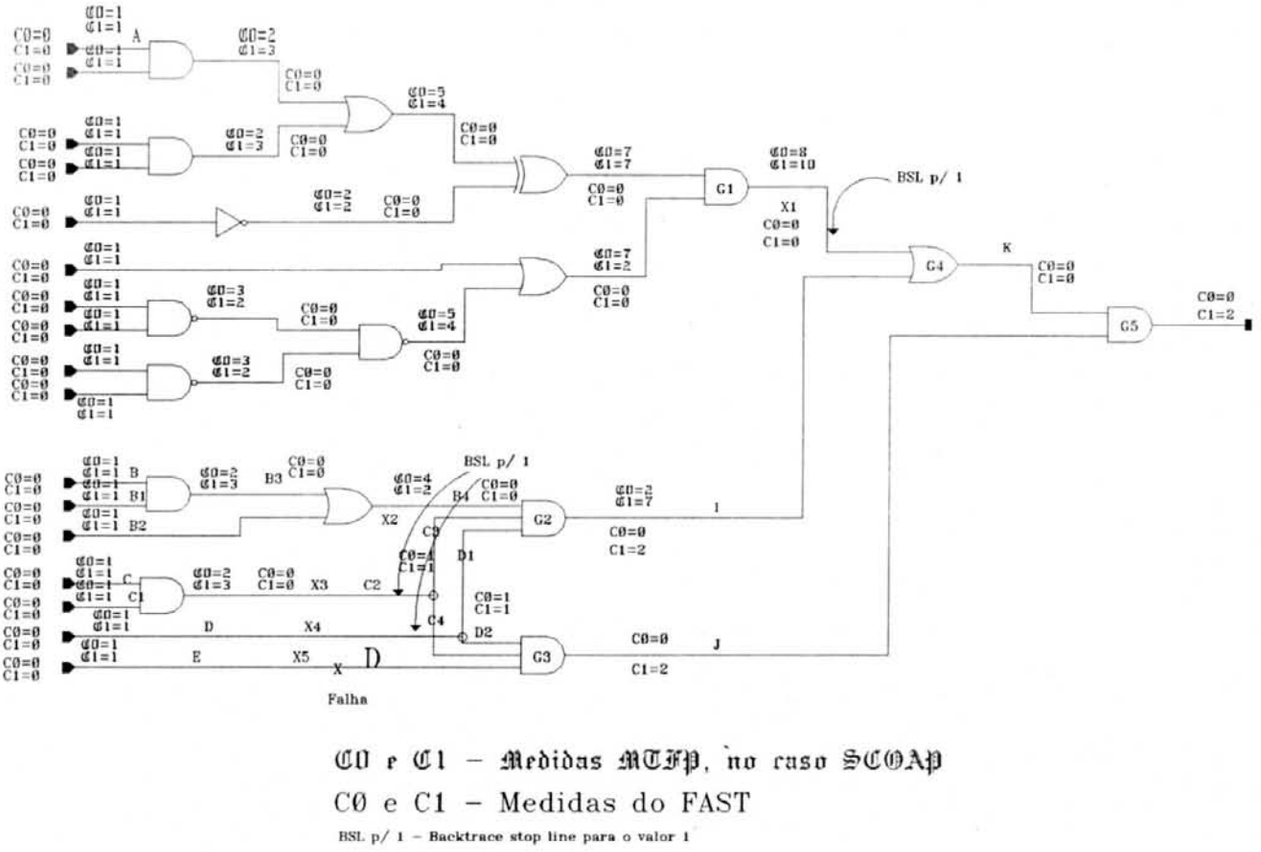


Figura 6.5: Medidas do FAST dinâmicas e SCOAP estáticas – primeiro momento

Para ilustrar esta primeira heurística utiliza-se o circuito da figura 6.5. Chama-se $X1, X2, X3$ e $X5$ as entradas virtuais e $A, A1, B, B1, \ldots$ as entradas reais. As medidas foram calculadas por duas medidas de testabilidade, FAST para identificar as *backtrace stop lines* e SCOAP. Note-se que os

cálculos do SCOAP foram feitos somente na área entre as entradas reais e as entradas virtuais.

A situação analisada é a de uma falha na linha $X5$ que necessita ser propagada. Para isto atribui-se às linhas $C4$ e $D2$ o valor lógico 1. Isto altera as controlabilidades já calculadas, entretanto, neste caso, recalcula-se dinamicamente as combinações apenas na porção compreendida entre as entradas virtuais e as entradas primárias, visto que as alterações mudam apenas de maneira relativa a controlabilidade das porções anteriores as entradas virtuais. Com isto os cálculos ficam como apresentados na figura 6.6.

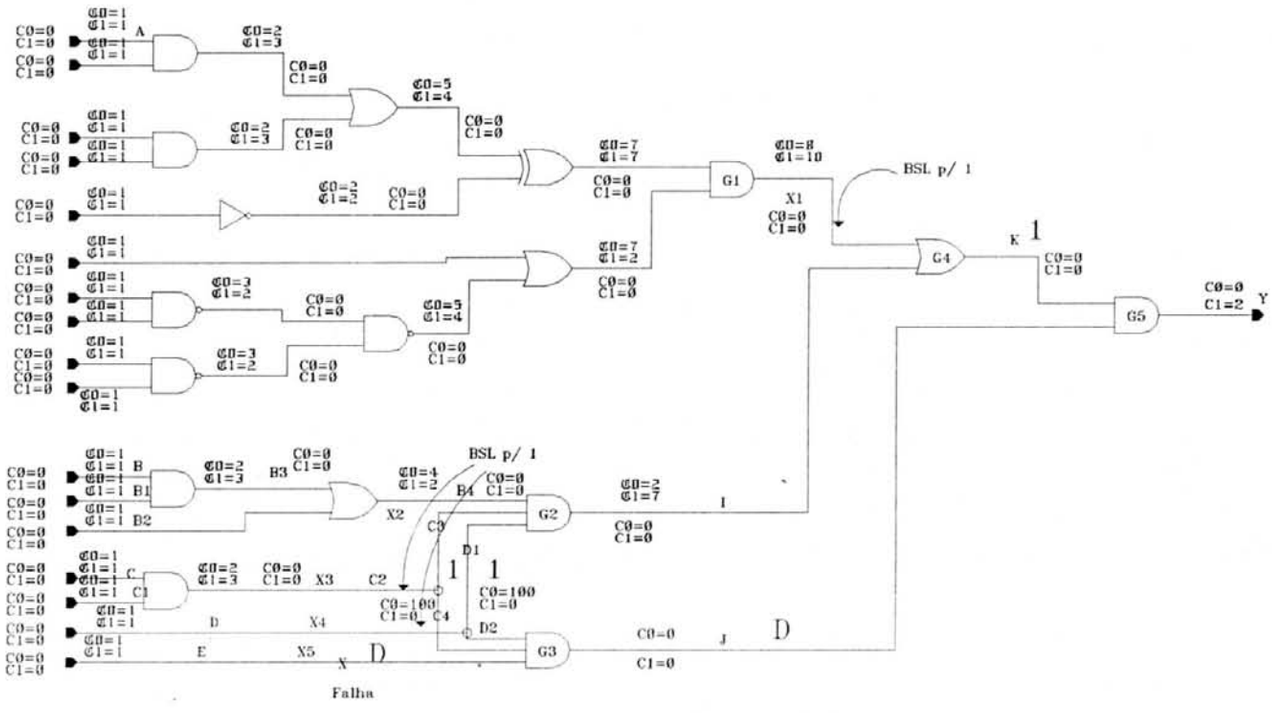


Figura 6.6: Medidas do FAST dinâmicas e SCOAP estáticas - segundo momento

Assim, um D aparece na linha J e é atribuído à linha K o valor 1 para que a diferença seja propagada para a saída Y. Com isto existe a necessidade de justificar-se o 1 da linha K. Neste momento utiliza-se as medidas dinâmicas para a escolha dos caminhos. E agora não mais o caminho I está indicado como menos controlável, visto isto não ser mais verdade. Como a controlabilidade recalculada dinamicamente indica que os caminhos não têm distinção das entradas virtuais para frente, utiliza-se as medidas estáticas do SCOAP para o desempate, o que nos leva à setagem de I para 1 ($C1(I) = 7$ e $C1(X1) = 10$). Como a propagação terminou pode-se fazer agora a justificação da porção entre as entradas reais e as entradas virtuais. Nesta justificação os valores da MTFP, no caso SCOAP, servem para auxiliar na escolha do menor caminho.

Deve-se observar que, por vezes, a porção das entradas virtuais se altera depois do recálculo, já que sua posição é determinada em função dos valores de controlabilidade calculados. Por este motivo, após cada recálculo, é necessário realizar a verificação das entradas virtuais.

Um outro exemplo similar pode ser visto nas figuras 6.7 e 6.8, ilustrando o uso de mesmo raciocínio para a observabilidade.

Na figura 6.7 pode-se verificar as observabilidades de $I1$ e $I2$ antes da atribuição de valores às linhas, ou seja, 0 e 1, respectivamente. Na figura 6.8 vê-se os valores após o recálculo parcial, ou seja, 0 e 0 nas duas linhas. Porém, é visível que a observabilidade não é a mesma, isto aconteceu porque as partes *free* não foram levadas em consideração. Para resolver este problema basta somar a controlabilidade da parte estática que já havia sido calculada. No caso exemplo, escolhe-se $I2$, visto que a parte estática penalizava em 6. E $I1$ não foi escolhida porque sua parte estática imputava atribuição a uma árvore de linhas *free* com controlabilidade igual a 11.

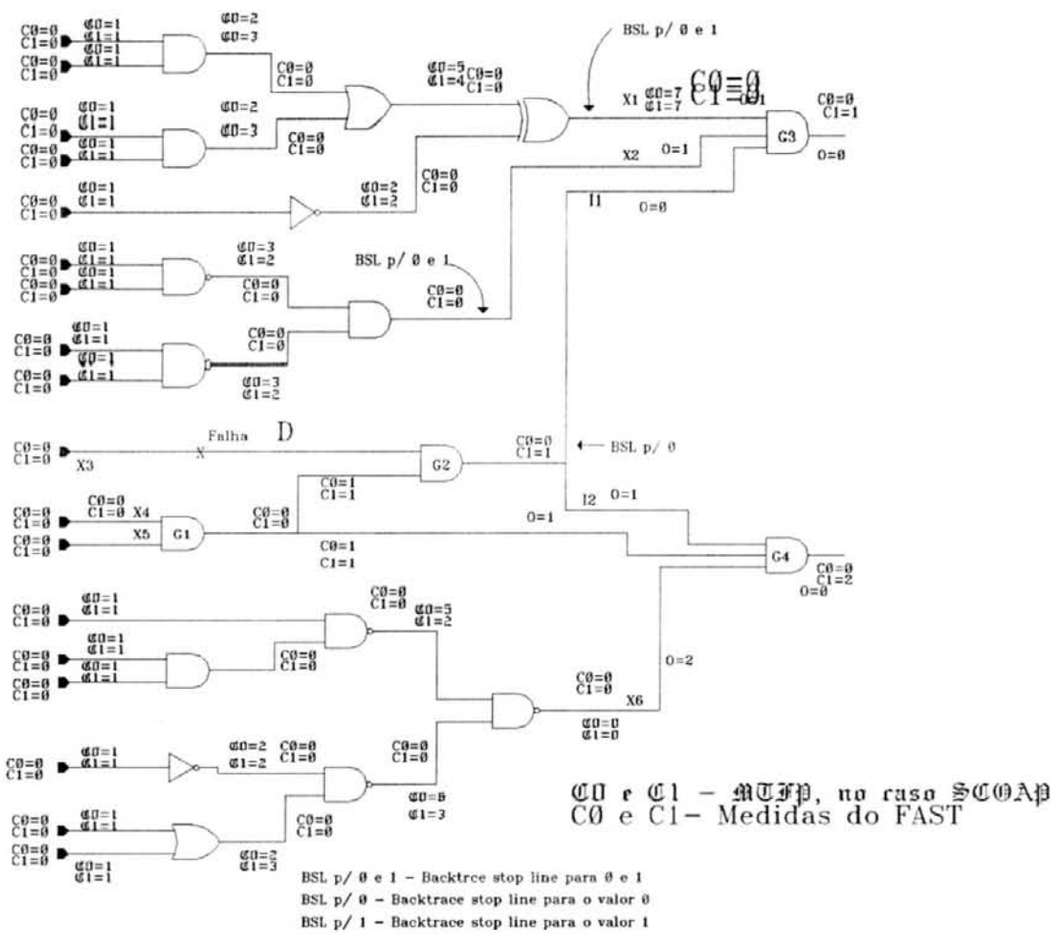


Figura 6.7: Medidas do FAST dinâmicas e SCOAP estáticas: Observabilidade – primeiro momento

6.6.2.2 Recálculo somente em linhas não *free*

A segunda heurística foi motivada pela desconfiança de que talvez a identificação das entradas virtuais e sua re-identificação a cada recálculo na heurística anterior tivesse alto custo computacional.

A idéia proposta é que o recálculo seja realizado em todas as conexões do circuito como no algoritmo dinâmico original. A diferença está, porém, no fato de que o recálculo somente será acionado quando da atribuição de um valor em uma conexão não *free*. Isto é justificado pelo fato de constantes nas linhas *free* não serem capazes de influenciar outros sinais. Esta heurística tem

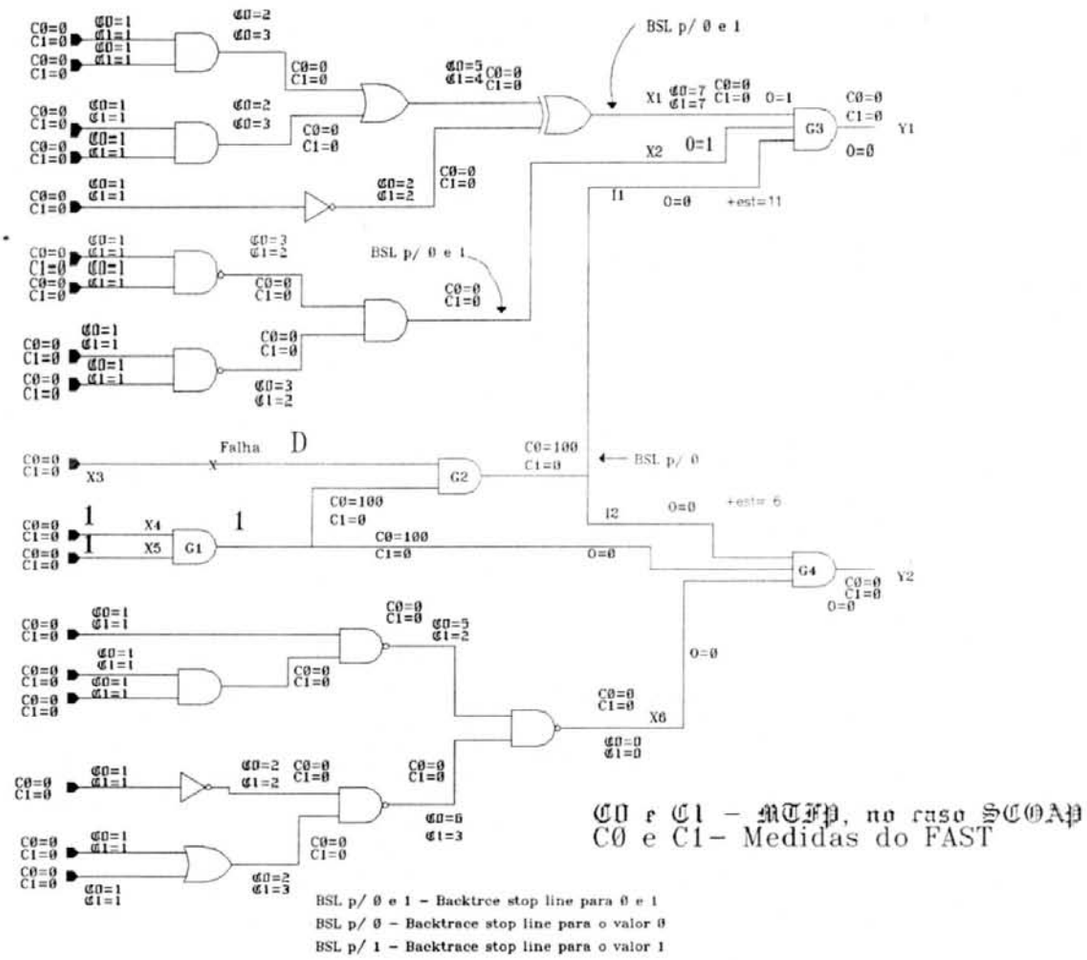


Figura 6.8: Medidas do FAST dinâmicas e SCOAP estáticas: Observabilidade – segundo momento

a vantagem de não necessitar identificar *backtrace-stop lines*, armazená-las e nem ficar identificando, a cada conexão, se ela deve ou não ser recalculada.

Os resultados das duas heurísticas serão apresentados no próximo capítulo.

6.6.3 GLOBAL conjugado com MTFP e cálculo dinâmico

Como o GLOBAL também serve para identificar as *backtrace-stop lines* também é possível esta combinação da mesma forma que a anterior com a única diferença que as medidas do GLOBAL não penalizam os fan-outs reconvergentes, ou seja, suas medidas são mais úteis que as do FAST, e o tempo de cálculo, de acordo com o próprio autor, mais ou menos o mesmo [ABR 90].

7 RESULTADOS DO EXPERIMENTO PRÁTICO

De forma a analisar com exemplos práticos as diversas heurísticas, o algoritmo *D* foi implementado em linguagem C em uma *SUN Sparc Station 2*, com 28 Mips e 4.2 MFlops. As opções suportadas pelo programa são:

- 1. Algoritmo *D* original (nenhuma heurística) – SH
- 2. Algoritmo *D* com as medidas do SCOAP – SC
- 3. Algoritmo *D* com as medidas do FAST – F
- 4. Algoritmo *D* com medidas do SCOAP dinâmico – SCD
- 5. Algoritmo *D* com medidas do FAST dinâmico – FD
- 6. Algoritmo *D* com heurística de Recálculo Parcial – RP
- 7. Algoritmo *D* com heurística de Recálculo em linhas não *free* usando o FAST – RLNFF
- 8. Algoritmo *D* com heurística de Recálculo em linhas não *free* usando o SCOAP – RLNFS

Como chegou-se a conclusão de que o uso de medidas dinâmicas só deveria ser acionado quando da perda de precisão por parte das medidas estáticas, foi implementada também uma opção de acionar o recálculo somente após um número fixo de *backtrackings*. Como os exemplos utilizados são, em geral, pequenos, o número de *backtrackings* utilizados foi 3 e 5, ou seja, recalcular após três ou cinco *backtrackings*. Esta opção pode ser combinada com qualquer das opções citadas acima que usam recálculos dinâmicos (de 4 a 8).

Os resultados nos diversos casos serão apresentados a seguir. Como o número de combinações possíveis era muito grande, optou-se por comparar as diversas opções parcialmente.

7.1 Circuitos Utilizados como Exemplos

Para que se pudesse analisar o comportamento do algoritmo nos diversos casos, tentou-se abranger exemplos de circuitos combinacionais comerciais bem diversos. A descrição dos circuitos utilizados pode ser vista na tabela 7.1.

Circuito	Descrição	Entradas	Saídas	# Portas	# Fan-out
MUX	Multiplexador 4X2	12	2	20	22
SOMADOR	Somador de 4 bits	9	5	35	31
CARRY	Gerador de Carry lookahead				
DECODER	Decodificador de BCD-to-decimal	4	10	31	23
COMP	Comparador Relacional	11	3	31	45
RANDOM LOGIC	Circuito Combinacional	17	1	17	2
C432	Decodificador de Prioridade	36	7	160	-
C499	ECAT	41	32	202	-
c1355	ECAT	60	26	383	-

Tabela 7.1: Circuitos usados como exemplo

Muitos dos circuitos utilizados são pequenos para que assim se possa ter maior controle da topologia. Os exemplos maiores (C432,C499 e C1355) são conhecidos circuitos que fazem parte do *benchmark* do ISCAS'85.

7.2 Comparação dos Resultados

7.2.1 Algoritmo D Original e as Medidas Estáticas

Foram gerados vetores para os diversos circuitos com o algoritmo D original e com o uso do PODEM, utilizando as medidas de testabilidade do SCOAP ou do FAST.

Na tabela 7.2, vê-se os resultados destas opções. Os valores apresentados são valores médios, ou seja, tempo médio de geração de um vetor(em segundos).

CIRCUITO	Tempo Médio de Geração de um Vetor(s)		
	Sem Heurística	SCOAP	FAST
MUX	0.013	0.012578	0.013126
SOMADOR	0.083545	0.060972	0.083147
CARRY	0.0584	0.057928	0.02455
DECODER	0.0147	0.015103	0.014467
COMPARADOR	0.133	0.155199	0.051837
RANDOM	0.038432	0.017370	0.00944
C432	2.100272	1.021499	1.377134
C499	3.049666	2.022052	5.252379
c1355	9.647	9.791266	9.448247

Tabela 7.2: Comparação entre Algoritmo D e PODEM

Examinando estes valores vê-se que nem sempre o uso das medidas de testabilidade trazem vantagens, por exemplo, no MUX ou DECODER. Esta semelhança nos valores do uso do SCOAP ou FAST com os do algoritmo original (sem heurística) pode dar-se pela quase inexistência de escolhas, ou seja, as três versões geram o padrão seguindo o mesmo caminho, que é o único caminho possível. Pode acontecer, ainda, da medida guiar pelo melhor caminho e o algoritmo original ter utilizado, por acaso, este mesmo caminho.

Há casos, entretanto, em que os ganhos são bastante significativos com a utilização das medidas de testabilidade. É o caso de C432, do COMP e CARRY. O C432 é um caso típico de ganho com qualquer heurística. Provavelmente este circuito leva o algoritmo D a tomar várias decisões no decorrer do processo e, nestes casos, ter um parâmetro auxiliando é imprescindível. O mesmo não acontece com os outros dois circuitos citados, COMP e CARRY, onde as medidas obtidas com o uso do FAST são melhores. Isto se dá devido ao alto número de reconvergências presente neste circuito e que não consegue ser detectadas pelo SCOAP.

É possível, ainda, que o uso de uma medida de testabilidade possa piorar o desempenho do algoritmo. Isto pode ser verificado no exemplo do circuito C499, onde o uso das medidas do FAST guia o processo por caminhos com mais conflitos o que acarreta uma diminuição do desempenho na geração do vetor. Para confirmar a má escolha induzida pela medida basta verificar-se o número de *backtrackings* para uma das conexões do circuito C499 (ver tabela 7.3).

	Número de *Backtrackings*		
Conexão	Sem Heurística	SCOAP	FAST
123	8	1	16

Tabela 7.3: Número de *backtrackings* na conexão 123

O algoritmo original realiza 8 *backtrackings*, ou seja, por 8 vezes o caminho escolhido resultou em conflito e o programa teve que desfazer operações já feitas. Com o uso das medidas de testabilidade do SCOAP este número foi reduzido para 1. Com o FAST o valor subiu para 16, o que comprova a indução do FAST a uma ou várias decisões errôneas no decorrer do processo de ATPG.

7.2.2 Medidas Estáticas e Dinâmicas

Como já mencionou-se anteriormente, as medidas dinâmicas consomem geralmente mais tempo e processamento. Isto confirma-se com a análise da tabela 7.4.

CIRCUITO	Tempo Médio de Geração de um Vetor(s)			
	SCOAP	SCOAP DIN	FAST	FAST DIN
MUX	0.012578	0.058133	0.013126	0.020914
SOMADOR	0.060972	0.335925	0.083147	0.114784
CARRY	0.057928	0.321109	0.02455	0.278772
DECODER	0.015103	0.080838	0.014467	0.018929
COMP	0.155199	0.638515	0.051837	0.07929
RANDOM	0.017370	0.039812	0.00944	0.015472
C432	1.021499	56.134951	1.377134	13.117656
C499	2.022052	20.309386	5.252379	3.552516
c1355	9.647	9.791266	9.448247	9.54286

Tabela 7.4: Comparação entre Medidas Estáticas e Medidas Dinâmicas

Como o recálculo oferece medidas mais precisas durante todo o processo de ATPG, pode acontecer do vetor ser encontrado mais rápido e a performance aumentar, apesar de não ser usual. Isto pode ser visto no circuito C499, onde o tempo médio do FAST Dinâmico é menor que o do FAST estático. Isto se dá porque, devido ao tipo de circuito, as medidas ficavam sem efeito muito rapidamente, ocasionando vários *backtrackings*. Isto deve-se ao fato do circuito possuir muitas linhas não *free*, ou seja, possui muitos sinais dependentes. Observando-se novamente a conexão 123 do circuito C499 (ver tabela 7.5), vê-se que o número de *backtrackings* diminui de 16 para 0, o que justifica o decréscimo de tempo. Este tipo de exemplo, porém , é raro. No geral as medidas dinâmicas demoram mais para serem calculadas. Nos exemplos apresentados o aumento médio, no caso do SCOAP, fica em torno de um tempo de 6 a 7 vezes maior, com exceções. No FAST a proporção é menor, entre 2 e 3 vezes.

	Número de *Backtrackings*	
Conexão	FAST	FAST DINÂMICO
123	16	0

Tabela 7.5: Número de *backtrackings* na conexão 123 com Cálculo Estático e Dinâmico

Devido a este acréscimo é que afirma-se que as medidas dinâmicas só devem ser utilizadas a partir de uma número fixo de *backtrackings*, ou seja, a partir da constatação de que as medidas disponíveis já não estão precisas, necessitando pois serem recalculadas.

7.2.3 Medidas Dinâmicas com e sem Recálculo Parcial

CIRCUITO	Tempo Médio de Geração de um Vetor(s)	
	FAST DIN	FAST DIN (Recálculo Parcial)
MUX	0.020914	0.035498
SOMADOR	0.114784	0.128444
CARRY	0.278772	0.190201
DECODER	0.018929	0.022595
COMP	0.07929	0.124176
RANDOM	0.015472	0.020446
C432	13.117656	16.743190
C499	3.552516	24.458193

Tabela 7.6: Comparação entre Medidas Dinâmicas com e sem heurística

A idéia de recalcular apenas porções do circuito demonstrou, na prática, não ser eficiente como pareceu teoricamente (ver tabela 7.6). Isto deve-se ao fato já mencionado anteriormente quando explanou-se a heurística, que é a necessidade da reidentificação das *entradas virtuais* a cada recálculo, o que consome excessivo processamento. Outra conclusão que advém do experimento prático é a de que circuitos que apresentam divergências de dados jé em suas entradas são os que têm a sua performance degradada. Isto de dá pelo fato de não haver, nestes circuitos, porção a ser descartada, porém

a identificação e reidentificação das *entradas virtuais* é feita, o que consome tempo.

O desempenho desta heurística poderia ser melhorado se houvesse uma distinção, por parte do programa, de fan-outs reconvergentes ou não. Por questões de simplicidade, o FAST foi implementado penalizando todos os fan-outs, sem restrições.

7.2.4 Medidas Dinâmicas sem e com Recálculo só em linhas não *free*

CIRCUITO	Tempo Médio de Geração de um Vetor(s)			
	FAST DIN	FAST DIN (+ heurística)	SCOAP DIN	SCOAP DIN (+ heurística)
MUX	0.020914	0.018135	0.058133	0.051566
SOMADOR	0.114784	0.111402	0.335925	0.121706
CARRY	0.278772	0.109360	0.321109	0.299446
DECODER	0.018929	0.019919	0.080838	0.127069
COMP	0.07929	0.088598	0.638515	0.186018
RANDOM	0.015472	0.009048	0.039812	0.019708
C432	13.117656	12.752155	56.134951	2.020908
C499	3.552516	3.59644	20.309386	3.648439

Tabela 7.7: Comparação entre Medidas Dinâmicas com ou sem Heurística

Observando os resultados da tabela 7.7, vê-se que esta heurística de acionar o recálculo apenas em linhas não *free* aumenta a performance do cálculo dinâmico. Isto só não acontece quando o número de linhas não free é bem pequeno. Neste caso, a heurística deixa de surtir efeito, mas como ela não imputa nenhum processamento adicional não causa nenhuma alteração significativa em termos de desempenho.

7.2.5 Medidas Dinâmicas com Recálculo Parcial acionadas sempre ou somente após um número fixo de *backtrackings*

CIRCUITO	Tempo Médio de Geração de um Vetor(s)		
	Recálculo Parcial Acionado sempre	Recálculo Parcial 3 backtrackings	Recálculo Parcial 5 backtrackings
MUX	0.035498	0.013266	0.013576
SOMADOR	0.128444	0.078818	0.079915
CARRY	0.190201	0.050331	0.048503
DECODER	0.022595	0.021223	0.018967
COMP	0.124176	0.047631	0.045537
C432	16.743190	2.263837	3.019302
C499	24.458193	4.796848	6.427392

Tabela 7.8: Comparação entre Recálculo Parcial acionado sempre e acionado após um número fixo de *backtrackings*

Na tabela 7.8 vê-se que, principalmente para circuitos grandes, onde há um número significativo de *backtrackings* é que a performance aumenta, fixando-se um número de *backtrackings* para acionar o recálculo. Este número, porém, não pode ser muito grande a ponto de permitir uma degradação excessiva das medidas, o que aumenta o número de *backtrackings*. É o caso do circuito C499, onde, com o recálculo acionado somente após cinco *backtrackings* terem ocorrido, a performance diminui. Isto acontece porque as medidas, por demorarem muito a serem recalculadas, chegam a perder a precisão. Em especial, o circuito C499, por suas características de dependência entre sinais, é extremamente suscetível a perda de precisão no decorrer do processo, por isto neste circuito a situação pode ser vista com facilidade. Isto se comprova pelo fato de, anteriormente, este circuito ter sido o único a conseguir ganho de performance com o uso das medidas dinâmicas.

Nos circuitos pequenos, onde o número de *backtrackings* para a geração de cada vetor fica perto do número de *backtrackings* fixado, por exemplo no DECODER, este parâmetro não causa alterações no desempenho, Nos circuitos em que o número de *backtrackings* é bem pequeno (menor que o

número fixo), a performance tende a se aproximar ao desempenho do algoritmo utilizando medidas estáticas. Isto é conseqüencia do fato do Recálculo parcial nem chegar a ser acionado durante todo o processo.

7.2.6 Medidas Dinâmicas acionadas sempre e somente após um número fixo de *backtrackings*

O experimento foi realizado também conjugando-se todas as outras opções dinâmicas com a opção de acionar o recálculo dinâmico somente após um número fixo de *backtrackings*. Estes resultados serão omitidos aqui devido ao fato dos valores seguirem a mesma tendência apresentada nos exemplos anteriores, o que leva às mesmas conclusões.

8 CONCLUSÃO

O processo de geração de testes despende tempo e processamento excessivos, visto ser um problema NP-completo. A utilização de heurísticas que acelerem os algoritmos no processo de busca são imprescindíveis. Cada heurística enfoca, porém, um ponto distinto (funcional, probabilístico ou estrutural) o que não supre todas as necessidades da busca do caminho ótimo dentre todos os possíveis. Neste trabalho sugeriu-se a utilização conjugada destas heurísticas como uma solução para o problema do enfoque específico característico de cada uma delas, avaliando-se a coexistência destas otimizações e os ganhos que podem ser obtidos. Sugeriu-se ainda uma técnica inédita, a pré-implicação dos cubos.

Foi analisado o comportamento que o algoritmo assume quando da presença das várias heurísticas conjugadas e propôs-se, a partir disto, duas novas heurísticas, a do Recálculo Parcial e a de Recálculo das Linhas não *free*. Estas heurísticas conjugam o cálculo dinâmico com o conceito de *backtrace stop line*.

Com a experimentação prática pôde-se verificar que o algoritmo com medidas estáticas apresenta normalmente ganhos quanto ao algoritmo original. As medidas estáticas, porém, perdem precisão no decorrer do processo de ATPG. Logo, a idéia de recalcular os valores de testabilidade dinamicamente traz benefícios, conseguindo garantir a qualidade das medidas durante todo o processo de geração de testes. A desvantagem é o consumo de processamento e, consequentemente, de tempo que isto acarreta. Com o objetivo de apresentar uma melhora no desempenho deste recálculo dinâmico as duas heurísticas foram propostas. Com base nos resultados verificou-se que uma das heurísticas acabava por imputar mais processamento para a sua execução

do que o algoritmo original. A segunda heurística, entretanto, apresentou uma melhora significativa de performance, comprovando sua eficiência.

BIBLIOGRAFIA

[ABA 83] ABADIR, M. S.; REGHBATI, H. K. LSI testing techniques. **IEEE Micro**, New York, v. 3, n. 1, p. 34-51, Feb. 1983.

[ABR 86] ABRAMOVICI, M.; KULIKOWSKI, J. J.; MENON, P. R. et al. SMART and FAST: test generation for VLSI scan design circuits. IEEE Design & Test of Computers, New York, v. 3, n. 4, p. 43-65, Aug. 1986.

[ABR 90] ABRAMOVICI, M. et al. Global cost function for test generation. In: INTERNATIONAL TEST CONFERENCE, 21., Sept. 10-14, 1990, Washington. **Proceedings...** New York:IEEE, 1990. p.35-43.

[AKE 80] AKERS, S. H. Test generation techniques. **Computer**, New York, v. 13, n. 3, p. 9-14, Mar. 1980.

[BEH 82] BEH, C.C. et al. Do stuck fault models reflect manufacturing defects? In: INTERNATIONAL TEST CONFERENCE, 13., Oct., 1982. **Proceedings...** Washington:IEEE, 1982. p.35-42.

[BEN 80] BENNETS, R.G.; MAUNDER C.M.; ROBINSON, G.D. CAMELOT - A Computer-Aided Measure for LOgic Testability. In: IEEE INTERNATIONAL CONFERENCE ON CIRCUITS AND COMPUTERS, Oct., 1980, PortChester/NY. **Proceedings...** Washington:IEEE, 1980. p.1162-1165.

[BEN 84] BENDING M.J. HITEST: A knowlegde-based test generation system. **IEEE Design & Test**, New York, v. 1, n. 2, p. 83-92, Feb. 1984.

[BRA 82] BRAHME, D. S.; ABRAHAM, J. A. Functional testing of microprocessing. **IEEE Transactions on Computers**, New York, v. 33, n. 6, p. 475-485, June 1982.

[BRA 90] BRAYTON, R. K. et al. **Logic minimization algorithms for VLSI synthesis**. Boston:Kluwer Academic Publishers, 1990. 193p.

[BRE 76] BREUER, M.; FRIEDMAN, A. D. **Diagnosis and reliable design of digital systems**. Woodland Hills:Computer Science Press, 1976. 308p.

[BRE 85] BREUER M.A.; ZHU X. A knowledge-based system for selecting a test methodology for a PLA. In: DESIGN AUTOMATION CONFERENCE, 22., June 23-26, 1985, Las Vegas. **Proceedings...** Washington:ACM/IEEE, 1985. 838 p., p. 259-265.

[BRG 84] BRGLEZ, F. On testability analysis of combinational networks. In: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 1984. **Proceedings...** New York:ACM/IEEE, 1984. p. 221-225.

[DIA 75] DIAS, F.J.O. Fault masking in combinatorial logic circuits. **IEEE Transactions on Computers**, New York, v. 24, n. 5, p. 476-482, May 1975.

[DOR 92] DORNELLES, E.H.A.G.; WEBER, R.F. Geração de estímulos para simulação baseada em geração de testes. In: SIMPÓSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 7., 29 set - 2 out. 1992, Rio de Janeiro/RJ. **Anais...** Rio de Janeiro:SBMICRO/SBC, 1992. p. 94-106.

[FRI 67] FRIEDMAN, A. D. Fault detection in redundant circuits. **IEEE Transactions Electronic Computers**, New York, v. 16, n. 2, p. 99-100, Feb. 1967.

[FUJ 82] FUJIWARA, H.; TOIDA, S. The Complexity of fault detection problem for combinational logic circuits. **IEEE Transactions on Computers**, New York, v. 31, n. 6, p. 555-559, June 1982.

[FUJ 83] FUJIWARA, H.; SHIMONO, T. On the acceleration of test generation algorithms. **IEEE Transactions on Computers**, New York, v. 32, n. 12, p. 1137-1144, Dec. 1983.

[GOE 81] GOEL, P. A implicit enumeration algorithm to generate tests for combinational logic circuits. **IEEE Transactions on Computers**, New York, v. 30, n. 3, p. 215-222, Mar. 1981.

[GOL 80] GOLDSTEIN, L.H.; THIPGEN, E.L. SCOAP - Sandia Controllability/Observability Analysis Program. In: DESIGN AUTOMATION CONFERENCE, 17., June 23-25, 1980, Minneapolis. **Proceedings...** New York:ACM/IEEE, 1980. 642p., p. 190-196.

[HAY 76] HAYES, P. Transition count testing of combinational logic circuits. **IEEE Transactions on Computers**, New York, v. 25, n.6, p. 613-620, June 1976.

[HAY 80] HAYES, J. P.; McCLUSKEY, E. J.- Testability consideration in microprocessors-based design. **Computer**, New York, v. 13, n. 3, p. 17-26, Mar. 1980.

[HUG 87] HUGHES, J. L. A.; McCLUSKEY, E. J.- Multiple stuck-at fault coverage of single stuck-at fault test sets. In: INTERNATIONAL TEST CONFERENCE 18., Sept. 10-14, 1987, Washington DC. **Proceedings...** New York:IEEE, 1987. p.849-855.

[IVA 88] IVANOV, A.; AGARWAL, V.K. Dynamic testability measures for ATPG. **IEEE Transactions on Computer-Aided Design**, New York, v. 7, n. 5, p. 598-608, May 1988.

[KAM 85] KAMPEL, I. J. **A Practical introduction to new logic symbols**. Boston:Butterworths, 1985. 124p.

[KAR 72] KARP, R. M. Reducibility among combinatorial problems. In: MILLER, R. E.; THATCHER, J. W. (eds.) **Complexity of computer computations**, New York:Plenum Pub. Corp., 1972. p. 85-104.

[KRI 84] KRISHNAMURTHY B.; AKERS S.B. On the complexity of estimating the size of a test set. IEEE Transactions on Computers, New York, v. 33, n. 8, p. 750-753, Aug. 1984.

[McC 71] McCLUSKEY, E. J.-; CLEGG, F. W. Fault equivalence in combination logic networks. **IEEE Transactions on Computers**, New York, v. 20, n. 11, p. 1286-1293, Nov. 1971.

[MEI 74] MEI, T. F. S. Bridging and stuck-at faults. **IEEE Transactions on Computers**, New York, v. 23, n. 11, p. 720-726, July 1974.

[MIL 87] MILLER, D. M. **Developments in integrated circuit testing**. London:Academic Press, 1987. 440p.

[MIL 88] MILLMAN, S. D.; McCLUSKEY, E. J.- Detecting bridging faults with stuck-at test sets. In: INTERNATIONAL TEST CONFERENCE, 19., 1988. **Proceedings...** New York:IEEE, 1988. p.773-778.

[MUE 81] MUEHLDORF, E.; SAVKAR, A. D. LSI: logic testing - an overview. **IEEE Transactions on Computers**, New York, v. 30, n. 1, p. 1-17, Jan. 1981.

[MUR 84] MURTHY,B.K.; AKERS, S.B. On the complexity of estimating the size of a test set. **IEEE Transaction on Computers**, New York, v. 33, n. 8, p. 750-753, Aug. 1984.

[PUT 71] PUTZOLU, G.R.; Roth J.P. A heuristic algorithm for the testing of asynchronos circuits. **IEEE Transactions on Computers**, New York, v. 20, n. 6, p. 639-647, June 1971.

[REG 85] REGHBATI, H. K. **VLSI: Testing & validation techniques**, Amsterdam:IEEE Computer Society Press, 1985.

[ROT 66] ROTH, J.P. Diagnosis of automata failures: a calculus and a method. **IBM Journal of Research Development**, New York, v. 10, n. 7, p. 278-291, July 1966.

[ROT 82] ROTH, J.P.; SAVIR, J. Testing for, and distinguishing between failures. In: INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT COMPUTING 12., June, 1982. **Proceedings...** New York:IEEE, 1982. p. 165-172.

[RUS 89] RUSSEL, G.; SAYERS I.L. **Advanced simulation and test methodologies for VLSI design**. London:Van Nostrand Reinhold, 1989.

[SAV 83] SAVIR, J. Good controllability and observability do not guarantee good testability. **IEEE Transactions on Computers**, New York, v. 32, n. 12, p. 1198-1200, Dec. 1983.

[SAV 84] SAVIR,J.; DITLOW, G.S.; BARDELL, P.H. Random pattern testability. **IEEE Transactions on Computers**, New York, v. 33, n. 1, p. 79-90, Jan. 1984.

[SET 85] SETH, S.C. et al. PREDICT: probabilistic estimation of digital circuits testability. In: INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT COMPUTING, 15., June 19-21, 1985, Michigan. **Proceedings...** New York:IEEE, 1985. p. 19-21.

[SET 90] SETH, S. C. et al. A statistical theory of digital circuits testability. **IEEE Transactions on Computers**, New York, v. 39, n. 4, p. 582-586, Apr. 1990.

[SMI 85] SMITH, G. L. Models for delay faults. In: INTERNATIONAL TEST CONFERENCE 16., Nov., 1985, Philadelphia/PA. **Proceedings...** New York:IEEE, 1985. p.342-349.

[THA 73] THAYSE A.; DAVIO M. Boolean differential calculus and its applications to switching theory. **IEEE Transactions on Computers**, New York, v. 22, n. 2, p. 409-420, Feb. 1973.

[THA 80] THATTE, S. M.; ABRAHAM, J. A. Test generation for microprocessors. **IEEE Transactions on Computers**, New York, v. 29, n.6, p. 429-441, June 1980.

[VIS 72] VISHWANI, D. A.; AGRAWAL, P. An Automatic test generation system for Illiac IV logic boards. **IEEE Transactions on Computers**, New York, v. 21, n. 9, p. 1015-1017, Sept. 1972.

[WAD 78] WADSACK, R. L. Fault modelling and logic simulation of CMOS and MOS integrated circuits. **Bell System Technical Journal**, New York, n. 57, p. 1449-1473, 1978.

[WEY 92] WEYERER, M.; GOLDEMUND, G. **Testability of eletronic circuits**. Englewood Cliffs:Prentice Hall, 1992. 232p.

[WIL 85] WILLIAMS, T.W. Test lenght in a self-testing environment. **IEEE Design & Test of Computers**, New York, v. 2, n. 2, p. 59-63, Apr. 1985.

Informática
UFRGS

Análise da Performance do Algoritmo D.

Dissertação apresentada aos Senhores:

Prof. Dr. Antônio Otávio Fernandes (UFMG)

O Examinador enviou parecer por escrito.

Prof. Dr. Flávio Rech Wagner

Profa. Dra. Ingrid Eleonora Schreiber Jansch Pôrto

Prof. Dr. Raul Fernando Weber

Vista e permitida a impressão.
Porto Alegre, 02 / 03 / 94 .

Prof. Dr. Raul Fernando Weber,
Orientador.

Prof. Dr. Ricardo A. da L. Reis,
Coordenador do Curso de Pós-Graduação
em Ciência da Computação.