Joel Iván Muñoz Quispe

Ferramenta CAD para extração de modelo de cobertura de saída por itens em verificação funcional

São Paulo
2011

Joel Iván Muñoz Quispe

Ferramenta CAD para extração de modelo de cobertura de saída por itens em verificação funcional

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica.

Área de concentração:
Microeletrônica

Orientador:
Prof. Dr. Livre-Docente
Wang Jiang Chau

São Paulo
2011

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 24 de novembro de 2011.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Muñoz Quispe, Joel Iván
Ferramenta CAD para extração de modelo de cobertura de saída por itens em verificação funcional / J.I. Muñoz Quispe. -- ed.rev. -- São Paulo, 2011.
164 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1. Microeletrônica 2. Circuitos integrados 3. Qualidade do projeto I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II. t.

Dedicatória

Esta dissertação é dedicada para as pessoas pelas quais eu sou o que sou e pelas quais eu luto dia a dia. Meus queridos pais Elisa e José Antonio e meus queridos irmãos José e Luis, meus grandes mestres em diferentes aspectos da vida.

Agradecimentos

Todo trabalho tem um começo e um método para chegar ao fim. Este caminho está cheio de obstáculos e é parte do crescimento tanto profissional como pessoal. Mas todo crescimento na vida precisa de um guia, um orientador. Neste caso, eu fico agradecido com o Prof. Dr. Wang Jiang Chau, primeiro por confiar em mim e me aceitar como seu orientando e segundo por me guiar no meus primeiros passos como cientista. Ao Prof. Dr. Marius Strum por ser também um guia, mas sobretudo ser como um pai e um amigo.

A minha família, quem sempre me guiou ainda na distância, somente separados pela cordilheira e pelo mato da selva. Mas isso não é obstáculo para nosso carinho. A meu tio José Julian por me ajudar com os aspectos do visto. Esta ajuda me permitiu começar uma nova etapa na minha vida, a qual está sendo fechada com este documento, obrigado por tudo.

Ao governo do Brasil e a CNPq pela bolsa de mestrado outorgada para mim para a realização desta pesquisa. Esta bolsa é simbolo de um país que procura crescer tecnologicamente.

Um agradecimento especial a meu amigo Mario Raffo, pela amizade, mostrada há 6 anos; a pessoa que me ajudou desde o começo e segue me ajudando. Ele se converteu num irmão e às vezes num pai. A minha "mãe" Izabella pela ajuda brindada para mim, obrigado por ser quem é e por desenhar um sorriso nas pessoas e, em algumas, deixar a mente mais "brilhante".

A meus amigos do GuE, Jorge Lucio, Jorge Benavides, Manuel, Heiner, David, Hector Villacorta e Erick Raygada, pelos bons momentos de camaradagem, assim como de colegas desta área de pesquisa, que enche nossos corações. Aos amigos do LME, Gustavo, Jorge, Carlos e Leonardo, pelo apoio e momentos de risada durante este processo de aprendizagem e ensino.

A meus chefes Andrés Farfán e Márcio Toma pela sua paciência e ajuda tanto no trabalho e o desenvolvimento desta dissertação.

Aos amigos que fui conhecendo na minha permanência na USP, Dante, Demóstenes, Dennis, El buen Kikin, Jorgito, Luchin, obrigado pela amizade e bons momentos no G106 disco bar CRUSP. À minha assistente social do CRUSP, Mara, por seu apoio quando eu mais precisava.

Agradeço a meus amigos da vida toda, Manuel, Tommy e Javier por essa amizade fiel que a gente tem desde criança.

“El único modo de hacer un gran trabajo es amar lo que haces. Si no lo has encontrado todavía, sigue buscando. No te acomodes. Como con todo lo que es propio del corazón, lo sabrás cuando lo encuentres.”

Steve Jobs- CEO da Apple Inc.

Resumo

Nos ambientes de desenvolvimento de sistemas integrados da atualidade, os requisitos dos sistemas devidos ao alto grau de funcionalidades incorporadas vêm-se incrementando, gerando uma alta complexidade nos projetos. Isto traz como consequência o aumento na quantidade de ciclos dentro do fluxo de projeto. Uma solução tem sido o uso de blocos IP para acelerar o desenvolvimento. Entretanto, para garantir um grau elevado de confiabilidade destes componentes, os processos de verificação devem comprovar que todas as propriedades do circuito estejam sendo cumpridas.

Uma das técnicas utilizadas para isto é verificação funcional por simulação, que procura explorar, através da injeção de vetores de teste, a maior porção possível de todo o espaço de estados do circuito. Quanto maior o número de estados possíveis, maior o número de vetores de testes que devem ser inseridos. Portanto, o número de vetores de teste deve ser reduzido de forma considerável, entretanto, por este fato, métricas para determinar a completeza do processo de verificação, definidas como modelos de cobertura, têm sido necessárias.

As métricas de cobertura são estabelecidas segundo as estratégias de observação do projeto sob verificação, DUV, sendo bastante comum na indústria a de caixa preta que tem como objetivo a estimulação das entradas e a observação dos eventos de saída do DUV. Neste caso, para determinar se o sistema cumpre com as especificações, o engenheiro de verificação, deve definir os eventos à saída que considera relevantes e as métricas para determinar a quantidade de vezes que devem ser observadas. Este tipo de modelagem é conhecido como cobertura por itens. A quantidade de itens e os eventos a serem observados podem ser definidos pelo conhecimento especialista, dos engenheiros de verificação ou, para simplificar esta tarefa, uma distribuição uniforme é adotada. Como estas formas de modelagem não abstraem todas as propriedades do circuito, o perfil da distribuição de valores dos eventos (parâmetros) escolhidos, em geral, não estão correlacionados com o perfil real verificado durante a execução dos testbenches , tendo como consequência o aumento dos tempos de simulação.

Para tratar do problema acima, o presente trabalho tem como objetivo geral o desenvolvimento de uma metodologia para obter um modelo de cobertura de saída que apresente um perfil de distribuição semelhante ao real e que, assim, assista o engenheiro de veri-

ficação na seleção dos pontos ou intervalos de saída de interesse, adicionado-os às decisões derivadas de seu conhecimento especialista. Pela metodologia utilizada, encontra-se a(s) equação(ões) que define(m) a(s) saída(s) do circuito sob verificação e, a partir destas, a distribuição probabilística por evento observável.

No centro da metodologia está a ferramenta PrOCov (Probabilistic Output Coverage), projetada com os objetivos acima. A metodologia e a ferramenta foram testadas com alguns exemplos de circuitos, modelos em alto nível do filtro FIR, do processador FFT e do filtro Elliptic, todos descritos em SystemC. Nos três casos testados, o PrOCov encontrou satisfatoriamente os respectivos perfis de saída. Estes foram comparados com os perfis obtidos por simulação, mostrando que uma excelente precisão pode ser obtida; apenas pequenas variações foram encontradas devidas a erros de aproximação. Também variações de precisão e tempo de simulação em função da resolução dos parâmetros de saída (eventos) foram analisadas nesta dissertação.

Palavras-chave: Verificação funcional. Cobertura. SystemC. CAD para verificação. Microeletrônica.

Abstract

In current integrated system development environments, the requirements for the design of multi-function systems have increased constantly. Consequently, the number of iterations in the design flow has also grown. A solution for this problem has been the use of IP-cores to speed up the hardware development. However, to guarantee high level of reliability for these components, the verification process has to be kept strict in other to prove if the all system properties have been satisfied.

The mainstream technique that has been used in the industry for the verification process is the dynamic functional verification. It aims to explore, by test vector injection, all the state space of the circuit. The higher the number of possible states, the higher the number of test vectors to be inserted. Therefore, the number of test vectors must be kept as low as possible. Due to that, completion and sufficiency metrics, identified as the coverage model, should be carefully defined.

The coverage metrics are established according the observation strategies of the design under verification, DUV, where the black box approach is very common in the industry, being aimed at the stimulation of the inputs and observing the events of the DUV output. To determine whether the system meets the specifications, the verification engineer must define the events (s)he considers relevant at the output and the metrics used to determine the amount of times that the results must be observed. This type of modeling is known as item coverage. The amount of items and events to be observed may be defined by the experience of the engineer, but in most cases, to simplify this task, a uniform distribution is adopted. Those forms of modeling do not abstract the functionality of the circuit, then, the probability distribution of the chosen events is uncorrelated to the real simulated distribution, when the testbenchs are implemented. Therefore, the resulting simulation time increases.

To solve the problem that is mentioned above, this work aims the development of a methodology to compute the output coverage, which should be similar to the real output value distribution and thus assist the engineer in the selection of the proper check points or output ranges of interest, by adding them to the decisions derived from his(her) knowledge. This methodology finds the equations that represent the outputs of the DUV and, from them, it computes the output probabilistic distribution.

At the core of this methodology is the PrOCov (Probabilistic Output Coverage) tool, which was developed with the goals above. Both methodology and tool were tested with three circuits described in high level language, the FIR filter, FFT processor and Elliptic filter, written in SystemC. In all three cases, PrOCov presented a satisfactorily output

distribution. Excellent precision could be achieved by the results, with only small variations found due to approximation errors. Also variations of accuracy and simulation time due to different resolutions of the output parameters (events) were analyzed in this dissertation.

Keywords: Functional Verification. Coverage. SystemC. CAD for verification. Microelectronics.

Sumário

1 Introdução **1**

1.1 Motivação 3

1.2 Objetivos 7

1.3 Organização da dissertação 8

2 Fundamentos teóricos **9**

2.1 Aspectos associados 9

2.2 Paradigmas dentro da verificação funcional 10

2.2.1 Metodologia de verificação baseada em asserções 11

2.2.2 Metodologia de verificação usando do modelo de referência 12

2.3 Conceitos básicos 13

2.3.1 Ambiente de simulação - Testbench 13

2.3.2 Geração de casos de teste 13

2.3.3 Cobertura 14

3 Trabalhos correlatos **21**

3.1 Jerinic e Muller, ano 2004 21

3.1.1 Objetivo 21

3.1.2 Conceitos 21

3.1.3 Exemplo 24

3.1.4 Trabalhos correlatos 24

3.1.5 Limitações da estratégia proposta 25

3.2 Trabalho de Mishra e Dutt , ano 2005 25

3.2.1 Objetivo 25
3.2.2 Metodologia 26
3.2.3 Resultados 27
3.2.4 Limitações da estratégia proposta 27
3.3 Trabalho de Fallah, Devadas e Keutzer, ano 1998 28
3.3.1 Objetivo 28
3.3.2 Metodologia 28
3.3.3 Resultados 29
3.3.4 Trabalhos correlatos 29
3.3.5 Limitações da estratégia proposta 30
3.4 Trabalho de Verma, Harris e Ramineni, ano 2007 30
3.4.1 Objetivo 30
3.4.2 Metodologia 30
3.4.3 Resultados 33
3.4.4 Limitações da estratégia proposta 33

4 Modelagem e cômputo da cobertura 34

4.1 O modelo de cobertura 34
4.1.1 A necessidade de cobertura de entrada e saída 34
4.1.2 Funções probabilísticas para antecipar perfil de saída 37
4.1.3 Representação intermédia - FSMD 38
4.2 Cômputo da distribuição de probabilidade de uma operação aritmética . . . 41
4.2.1 Distribuição de probabilidade da soma 42
4.2.2 Densidade de probabilidade da multiplicação 46
4.2.3 Considerações para o cômputo da PD 50
4.3 Cômputo da PD de uma equação algébrica 54
4.4 Cálculo da PD na presença de comandos de controle 56
4.4.1 Análise de comandos condicionais 56

4.4.2 Comandos de laços 68

5 Desenvolvimento da ferramenta 70

5.1 Representação intermediária dos FSMDs 70

5.1.1 Nó de designação 71

5.1.2 Nós condicionais 72

5.1.3 Nós de laços 73

5.2 Metodologia e algoritmos para a computação de distribuições de valores à saída 77

5.2.1 Extração de equação das saídas 78

5.2.2 Cômputo da distribuição de saída 91

6 Resultados 97

6.1 Implementação da metodologia de cômputo da distribuição de saída 97

6.2 Metodologia experimental 97

6.2.1 Definição dos experimentos 98

6.2.2 Cômputo da cobertura e geração dos modelos de cobertura 101

6.2.3 Simulação do modelo de referência e avaliação dos modelos de cobertura 102

6.2.4 Levantamento e análise de resultados 104

6.3 Resultados dos experimentos 104

6.3.1 Consistência do modelo gerado 105

6.3.2 Tempo de simulação 111

6.3.3 Variações de resolução no cálculo da PD de saída 118

7 Conclusões e trabalhos futuros 121

Apêndice A – Fundamentos de probabilidade 124

A.1 Conceito de probabilidade 124

A.2 Variável aleatória 125

A.3 Funções de variáveis aleatórias 125

Apêndice B – Algoritmos para operações de relação **127**

Apêndice C – Algoritmos utilizados no cômputo da distribuição de saída **129**

Apêndice D – Cômputo de todos os casos da PD da multiplicação **134**

D.1 Cômputo da integral 135

D.1.1 Caso $x \in [a, b]$, $0 \leq a$ 135

D.1.2 Caso $x \in [a, b]$, $b \leq 0$ 142

D.1.3 Caso $x \in [a, b]$, $a < 0$, $0 < b$ 148

D.2 Definição de equações gerais 152

D.3 Algoritmos propostos para o cômputo da PD da multiplicação 156

Lista de Figuras

1.1 Fluxo de projeto de ASICs 2

1.2 Espaço alcançado por conjunto de vetores de entrada 4

1.3 Progressão de verificação 4

1.4 Exemplo de distribuição desejada 6

2.1 Arquitetura da Ferramenta FoC 11

2.2 Modelo de Verificação Dinâmica por Simulação com Asserções 12

2.3 Modelo de Verificação Dinâmica por Simulação usando modelo de Referência 13

2.4 Ambiente de Simulação 13

2.5 Ambiente de Simulação Reativo 20

2.6 Progresso da Cobertura Funcional com e sem CDG 20

3.1 Espaço de Simulações Sobrepostos 22

3.2 Domínio de Parâmetros 23

3.3 Mascaramento de Falha no Circuito 28

3.4 Exemplo de Modelagem de Cobertura 31

3.5 Arquitetura de Extrator de Cobertura 31

3.6 Processo no Código Verilog 32

3.7 CDFG do Processo mostrado na Figura 3.6 32

3.8 Código de Cobertura 32

4.1 Exemplo de Modelamento de Cobertura 35

4.2 Progressão da cobertura das saídas para o exemplo mostrado na figura 4.1(b) 36

4.3 Distribuição de saída estabelecida para o circuito 36

4.4 Exemplo de progressão da Cobertura de Saída 37

4.5 Exemplo de Somador e Distribuição de possíveis valores 38

4.6 Representação em grafos (FSMD) de comandos 40

4.7 Representação específica em grafos dos comandos de controle na dissertação 41

4.8 Exemplo de Distribuição não Uniforme entre eventos 43

4.9 Exemplo da Densidade Probabilidade no caso da soma de uma variável aleatória e uma constante 44

4.10 Áreas de Integração para o computo da PD da soma 44

4.11 Padrão da Densidade de Probabilidade da soma de variáveis de distribuição uniforme 45

4.12 Exemplo da Densidade Probabilidade no caso da multiplicação de uma variável aleatória e uma constante 47

4.13 Áreas de Integração para o caso $L_x \in \mathbb{R}^- \wedge L_y \in \mathbb{R}^- \wedge H_x \in \mathbb{R}^+ \wedge H_y \in \mathbb{R}^+$ e $L_y H_x < L_x H_y$ 48

4.14 Exemplo da Densidade Probabilidade no caso de X^n, sendo $n = 2$ para o exemplo 50

4.15 Exemplo de densidade de probabilidade da Soma 51

4.16 Definição do Espaço de Eventos Possíveis para a saída V 53

4.17 PD's sobrepostos das possíveis combinações dos eventos de X e Y 54

4.18 PD da saída V resultante da soma das PD de todas combinações de evento das entradas X e Y 54

4.19 Diagrama de fluxo com comandos de execução 55

4.20 Exemplo de cálculo da PD da função $v = g(X_0, X_1, X_2, X_3, X_4, X_5) = 4X_0 - 2X_1^2 X_2 + X_3 X_4 X_5$ 57

4.21 Exemplo de condicional 57

4.22 Áreas consideradas no cômputo da probabilidade de caminhos em operações de relacionamento 59

4.23 Exemplo generalizado de um condicional de dois caminhos 60

4.24 Exemplo de FSMD com um condicional 61

4.25 Exemplo de cômputo da densidade de probabilidade da saída Z do modelo da figura 4.24 61

4.26 Dois condicionais em serie 62

4.27 Exemplo de Análise de nós condicionais 63

4.28 Caminhos possíveis do exemplo da figura 4.27 64

4.29 Grafo generalizado de nós condicionais encadeado 66

4.30 Exemplo com o comando de controle de laço 69

5.1 Sintaxe geral dos comandos de execução 71

5.2 Sintaxe geral do comando de controle condicional 72

5.3 Exemplo de *comando de controle condicional* numa FSMD 73

5.4 Sintaxe geral do comando de controle de laço 74

5.5 Exemplo da FSMD estendida do laço usando o formalismo da figura 5.4 . . . 74

5.6 Exemplo de FSMD com laço ***FOR*** 76

5.7 Exemplo de percurso numa FSMD 78

5.8 Conteúdo do Objeto da classe $VarOutClass$ para a equação 5.1 82

5.9 Exemplo de dois nós em FSMD código do programa 83

5.10 FSMD exemplo de armazenamento numa base de dados de uma variável . . 83

5.11 Base dados de Z, considerando esta uma variável interna 84

5.12 Base dados de Z, considerando esta uma variável de saída 84

5.13 Algoritmo para a seleção de caminho e sua probabilidade 86

5.14 Algoritmo para o cômputo da operação de relação '$X < c$' 87

5.15 Algoritmo para a Análise do Modelo e Cômputo da Distribuição de Saída . . 89

5.16 Algoritmo para o computo da Distribuição Probabilísticas de uma variável . 92

5.17 Exemplo de equação representada em grafos 94

5.18 Árvore de multiplicadores para gerar os componentes da equação 94

5.19 Exemplo de cômputo, em forma de grafos, da distribuição do componente . . 94

5.20 Algoritmo para calcular a PD Resultante de X^n 95

5.21 Algoritmo para calcular a PD de $v^{'}$ no conjunto definido por X^n 95

5.22 Árvore para somar os Componentes da Equação 96

5.23 Exemplo de equação representada em grafos 96

6.1 Fluxo de teste da metodologia 99

6.2 Cômputo do Perfil de Cobertura e Geração de Modelos de Cobertura 102

6.3 Testbench simplificado para validação da metodologia 103

6.4 Comparação entre o número de vetores de teste esperados (computado pela ferramenta com resolução de 20.000) e o número de vetores que foram contados para obter 100% de cobertura na saída do exemplo FIR. 106

6.5 Comparação entre o número de vetores de teste esperados (computado pela ferramenta com resolução de 20.000) e o número de vetores que foram contados para obter 100% de cobertura na saída 0 do exemplo FFT. 106

6.6 Comparação entre o número de vetores de teste esperados (computado pela ferramenta com resolução de 20.000) e o número de vetores que foram contados para obter 100% de cobertura na saída 0 do exemplo Elliptic. 107

6.7 Progressão da cobertura da saída 0 do exemplo FFT utilizando o modelo gerado com a metodologia proposta (com resolução de 20000) e o modelo manual (distribuição uniforme para os eventos selecionados) 115

6.8 Variação de tempo de cálculo em função da resolução 119

B.1 Algoritmo para o computo da operação de relação '$X \leq c$' 127

B.2 Algoritmo para o computo da operação de relação '$X > c$' 127

B.3 Algoritmo para o computo da operação de relação '$X = c$' 128

C.1 Algoritmo para Cômputo da PD dos componentes da equação 129

C.2 Algoritmo para calcular o PD da Multiplicação de duas variáveis independentes 130

C.3 Algoritmo para Cômputo da PD da equação 131

C.4 Algoritmo para calcular o PD da soma de duas variáveis independentes . . . 132

C.5 Algoritmo para o computo da Distribuição Probabilísticas das variáveis de saída 133

D.1 Espaço Amostral das variáveis aleatórias quando $x \in [a, b]$,$0 \leq a$ 135

D.2 Definição da área de Integração para o caso de $a = 0$ e $c = 0$ 136

D.3 Definição da área de Integração para o caso de $a = 0$ e $c < 0$ 136

D.4 Definição da área de Integração para o caso de $a = 0$ e $d = 0$ 137

D.5 Definição da área de Integração para o caso de $a = 0$ e $d < 0$ 137

D.6 Definição da área de Integração para o caso de $a = 0$, $c < 0$ e $0 < d$ 138

D.7 Definição do espaço de integração quando $0 < a$, $0 < c$ 139

D.8 Forma Geral das gráficas apresentadas na figura D.7 139

D.9 Área de Integração para o caso $0 < a$ e $d = 0$ 140

D.10 Definição do espaço de integração quando $0 < a$, $d < 0$ 140

D.11 Forma Geral da gráfica apresentada na figura D.10(b) 141

D.12 Definição do espaço de integração quando $0 < a$, $0 < c$, $0 < d$ 141

D.13 Espaço Amostral das variáveis aleatórias quando $\Omega_x \in \mathbb{R}$ e $\Omega_y \in \mathbb{R}^-$ 142

D.14 Área de Integração para o caso de $b = 0$ e $c = 0$ 143

D.15 Área de Integração para o caso de $b = 0$ e $0 < c$ 143

D.16 Área de Integração para o caso de $b = 0$ e $d = 0$ 144

D.17 Área de Integração para o caso de $b = 0$ e $d < 0$ 144

D.18 Área de Integração para o caso de $b = 0$ e $c < 0$ e $0 < d$ 145

D.19 Área de Integração para o caso de $b < 0$ e $c = 0$ 145

D.20 Definição do espaço de integração quando $b < 0$, $0 < c$ 146

D.21 Área de Integração para o caso de $b < 0$ e $d = 0$ 146

D.22 Definição do espaço de integração quando $b < 0$, $d < 0$ 147

D.23 Definição do espaço de integração quando $b < 0$, $c < 0$, $0 < d$ 147

D.24 Espaço Amostral das variáveis aleatórias quando $\Omega_x \in [a, b]$,$a < 0$, $0 < b$ . . . 148

D.25 Definição do espaço de integração quando $a < 0$, $0 < b$, $0 < c$ 148

D.26 Definição do espaço de integração quando $a < 0$, $0 < b$, $d < 0$ 149

D.27 Definição do espaço de integração quando $a < 0$, $0 < b$, $c < 0$, $0 < d$, $bd < ac$ 149

D.28 Espaço de Integração Geral para os casos apresentados na figura D.27 150

D.29 Definição do espaço de integração quando $a < 0$, $0 < b$, $c < 0$, $0 < d$, $ac < bd$ 150

D.30 Definição do espaço de integração quando $a < 0$, $0 < b$, $c < 0$, $0 < d$, $ac = bd$ 151

D.31 Espaço de Integração Geral para os casos apresentados nas figuras D.29 e D.30151

D.32 Áreas de Integração Generalizada para o Caso 1 152

D.33 Áreas de Integração Generalizada para os Casos 2 e 3 153

D.34 Áreas de Integração Generalizada para os Casos 4 e 5 154

D.35 Casos de Límites de Integração 155

D.36 Computa o valor de $\lim_{v \to v_0} v\ln(\left|\frac{vm}{w}\right|)$ 156

D.37 Computa o valor do PD baseado da equação D.20 156

D.38 Computa o valor do PD baseado da equação D.21 156

D.39 Computa o valor do PD baseado da equação D.16 157

D.40 Computa o valor do PD baseado da equação D.17 158

D.41 Computa o valor do PD baseado da equação D.17 159

D.42 Computa o valor do PD baseado da equação D.17 160

D.43 Computa o PD baseado da equação D.17 161

Lista de Tabelas

4.1 Condições no Gráfico da Figura 4.10 no caso da Soma 45

5.1 Membros da Classe ***ExeCmdClass*** 72

5.2 Membros da Classe ***CondCmdClass*** 73

5.3 Membros da classe ***ForCmdClass*** 77

5.4 Membros da classe ***EndForCmdClass*** 77

5.5 Membros da Estrutura ***CompStruct*** 79

5.6 Membros da Estrutura ***ElementStruct*** 80

5.7 Membros da Classe ***VarClass*** 80

5.8 Membros da Classe ***VarOutClass*** 80

6.1 Definição de cada saída dos modelos de referência a serem usadas 101

6.2 Resultados para obter 100% de cobertura na saída do FIR (metodologia proposta com resolução de 20.000) 108

6.3 Resultados para obter 100% de cobertura na saída 0 do exemplo FFT (metodologia proposta com resolução de 20.000) 109

6.4 Resultados para obter 100% na saída 0 do Exemplo Elliptic(metodologia proposta com resolução de 20.000) 110

6.5 Exemplo para computar o número de vetores requeridos na saída 0 do Elliptic para completar o 100% da cobertura 112

6.6 Número de vetores requeridos para obter 100% de cobertura 114

6.7 Resultados da Progressão de Cobertura na saída 0 do FFT(Métodos Manuais) 117

6.8 Número de Vetores usados para obter 100% de cobertura segundo a resolução usada 120

Lista de símbolos

ALU	Arithmetic Logic Unit
AMBA	Advanced Microcontroller Bus Architecture
ARM	Advanced RISC Machine
ASIC	Application-Specific Integrated Circuit
CDFG	Control-Data Flow Graph
CTL	Computation Tree Logic
DUV	Design Under Verification
FoC	Formal Checking
FSM	Finite State Machine
FSMD	Finite State Machine with Datapath,
GET_PRG	Generation Tool for PD based Random Stimuli Generation
HD	Hard Disk
HDL	Hardware Description Language
IP	Intellectual Property
ISA	Instruction Set Architecture
MDD	Multi-Decision Diagram
MPEG	Moving Pictures Experts Group
OCCOM	Observability-based Code Coverage Metrics
OPC-IP	Open Protocol Core IP
PD	Probability Distribution
PDG	Parameter Domain Graphs
PrOCov	Probabilistic Output Coverage
RCTL	Regular CTL
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SoC	System-on-Chip
VHDL	VHSIC Hardware Description Language

1 Introdução

A grande capacidade de integração da atualidade permite a construção de circuitos integrados com muitas funcionalidades. Alguns destes circuitos contêm sistemas completos, conhecidos como Sistemas-Sobre-Silício (do inglês, Systems-on-Chip, SoC). Manter a consistência em projeto de SoCs tem sido um grande desafio para os projetistas de sistemas pelo tamanho e complexidade envolvidos. Uma das estratégias mais importantes utilizadas para lidar com tal complexidade é o particionamento do sistema, ou seja, o projeto em blocos menores para que possam ser reutilizados em futuros desenvolvimentos, reduzindo-se, assim, os tempos de chegada ao mercado (Time To Market) requeridos pelas atuais exigências comercias (1). Para facilitar tal reuso, os projetistas estão focados na construção de grandes quantidades de elementos projetados exclusivamente para este fim, conhecidos como blocos de propriedade intelectual (intelectual property blocks, IPs ou IP cores) (2). Os IP cores caracterizam-se por apresentar uma forma padronizada de comunicação, como por exemplo: CORECONNECT (3), AMBA (4) e OPC-IP (5). Outro aspecto fundamental para o reuso é que os elementos garantam um alto grau de confiabilidade, ou seja, a concordância da funcionalidade implementada com aquela estabelecida na sua especificação. Para este objetivo é necessária a realização de etapas de verificação que permitam identificar e corrigir falhas introduzidas durante o projeto.

Sabe-se que a etapa de verificação é aquela, dentro do fluxo de projeto, que mais recursos humanos e de tempo consome, sendo identificada como o principal gargalo, chegando a representar até 60% ou 70% de esforço do projeto. A Figura 1.1 mostra um fluxo típico de projeto de ASICs, onde o primeiro passo no processo é a análise da idéia do circuito, que gera um algoritmo. A partir desta análise pode-se também obter as especificações do projeto e o documento da especificação. Com tal documento o projetista faz uma tradução da especificação para um código em linguagem de descrição de hardware (do inglês, hardware description language, HDL) para realizar a síntese do circuito. Antes deste passo deve-se, através da tarefa de verificação funcional, confirmar que o circuito projetado cumpre funcionalmente com a especificação dada. Após a verificação, faz-se a síntese do circuito, e a simulação para se garantir todas as especificações de tempo. O último passo é a fabricação do circuito.

Dada a dificuldade associada à verificação, a indústria tem demonstrado uma preocupação generalizada para melhorar a sua eficiência. Para tratar deste problema, duas vertentes vêm sendo seguidas: a "Verificação Formal" e a "Verificação Dinâmica por Simulação". A primeira existe em duas categorias:

1. Verificação de modelos: analisa-se formalmente os problemas ou violações das especificações dadas pelo projetista ante o comportamento que o circuito deveria ter (6).

2. Verificação de equivalência: faz-se a comparação formal entre dois modelos, para se certificar matematicamente que o sistema descrito (circuito ou sistema de origem) na codificação HDL é equivalente ao sistema de saída da síntese do circuito (Figura 1.1) (6)[1].

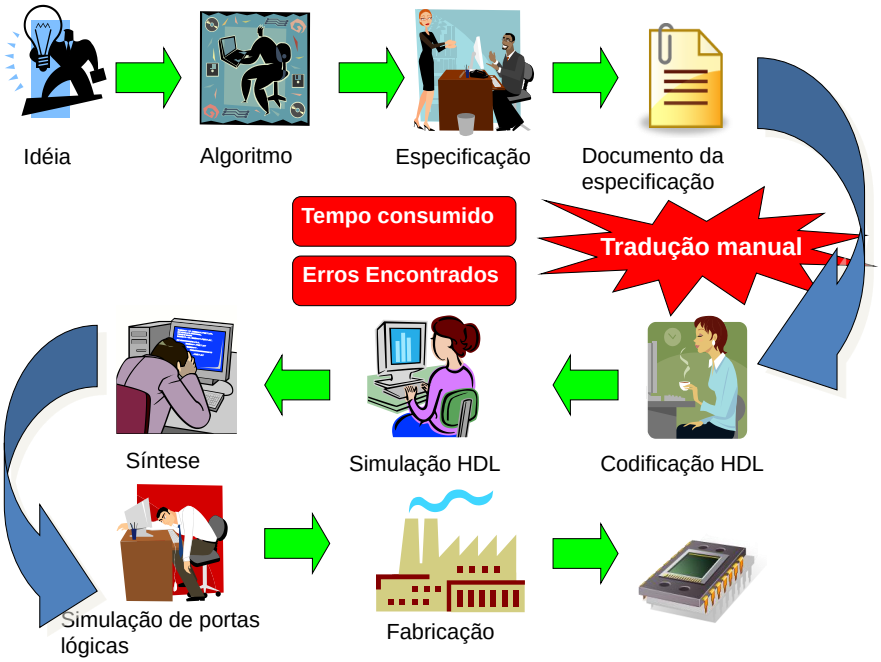


Figura 1.1: Fluxo de projeto de ASICs

Apesar da verificação formal poder provar a existência (ou inexistência) de erros de projeto, esta prática apresenta limitações quanto ao tamanho do projeto sob verificação (7), já que o número de estados possíveis de operação cresce exponencialmente com o número de elementos encontrados nos modelos, o que pode originar o fenômeno conhecido como "Explosão do Espaço de Estados" (do inglês State Space Explosion). Por isto, a verificação

[1]Esta definição corresponde à checagem de equivalência no nível lógico, que é a forma mais comum encontrada na literatura, entretanto, a rigor, a checagem de equivalência pode ser entre qualquer dois modelos de representação.

formal é limitada a blocos pequenos. Por outro lado, a verificação dinâmica por simulação não apresenta tal limitação, além de ter a vantagem de ser escalável. Esta dissertação terá como foco a verificação funcional por simulação, e por simplicidade, adotaremos para ela a denominação simplificada de verificação funcional.

Entendemos, então, que a verificação funcional objetiva conferir, por via de simulação, que a funcionalidade de um sistema, determinada na sua especificação, mantenha-se na sua implementação (6). Uma técnica muito comum baseia-se na idéia de que duas implementações de uma mesma especificação realizadas em níveis de abstração diferentes e por equipes diferentes apresentam erros diferentes (caso ocorram), os quais podem ser identificados a partir das diferenças de seus comportamentos. No caso dos IP cores, estas duas implementações podem corresponder a um modelo abstrato que contém unicamente características funcionais do sistema que está sendo projetado e a um modelo de implementação escrito num Nível de Transferência entre Registradores (do inglês Register Transfer Level, RTL). O primeiro desses modelos, conhecido como modelo de referência, é usualmente desenvolvido em uma linguagem de alto nível como C/C++, Java ou Matlab e é considerado "sem erros". O segundo é modelo do projeto sob verificação (do inglês, design under verification, DUV) descrito numa linguagem de descrição de hardware como VHDL, VERILOG ou SystemC. Estes modelos são comparados num ambiente de simulação denominado testbench (6), onde casos de teste são gerados e aplicados a ambos os modelos e onde as respostas são avaliadas.

1.1 Motivação

À exemplo da Verificação Formal, a Verificação Funcional apresenta o problema de alcançabilidade do espaço de estados do sistema (7); por exemplo, se um circuito contém 10 pinos de entrada e 100 registradores, e se o simulador gera 1000 casos de teste por segundo, então o tempo necessário para se verificar todos os possíveis estados do sistema pode ser aproximado em $(((2^{10+100})seg.)/(1000) = 1.3x10^{30}$ anos (8). No entanto, a principal vantagem da Verificação Funcional frente à Formal é que ela necessita de menor número de recursos computacionais, o que a faz permanecer como o método mais popular na indústria. Por outro lado, um dos maiores problemas encontrados é a geração adequada de estímulos, pois estes devem permitir observar o maior número de características funcionais do sistema sem precisar percorrer todos os estados possíveis do sistema. Tal problema é ilustrado na Figura 1.2, onde o conjunto I representa todos os vetores de teste de entrada, mas só um grupo deles é escolhido, e, na prática, irá permitir percorrer apenas um conjunto estados S' de um sistema com S estados possíveis ($S' \subseteq S$).

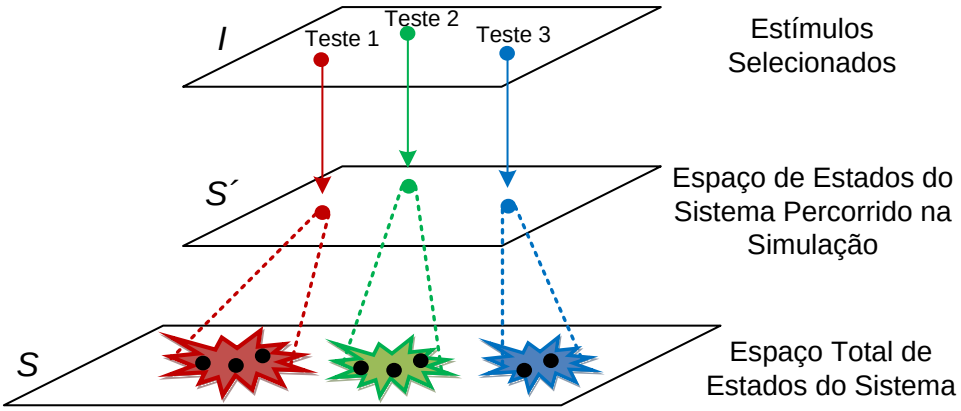


Figura 1.2: Espaço alcançado por conjunto de vetores de entrada

Nas metodologias atuais, a geração de estímulos aleatórios é largamente adotada nas etapas iniciais da verificação funcional, uma vez que um conjunto grande de estados pode ser alcançado rapidamente. Análises realizadas sobre resultados de tempo de simulação dos testbenches, adicionados aos tempos para a elaboração dos casos de teste, têm mostrado que a progressão (no tempo) da cobertura funcional é mais rápida para o caso de aplicação de estímulos aleatórios se comparada àquela dos estímulos dirigidos (6). Entenda-se aqui a cobertura como uma métrica que representa o nível de abrangência atingida pela simulação no espaço funcional do sistema.

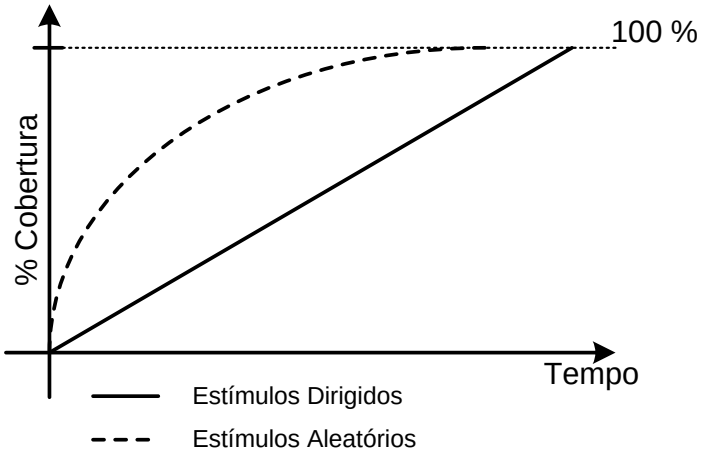


Figura 1.3: Progressão de verificação

De forma geral, a cobertura é definida como a métrica que determina quão bem estimulado foi o espaço de estados do DUV. As métricas de cobertura podem-se ser classificadas em dois tipos: estrutural e funcional. A primeira, para a qual existe a maior quantidade de ferramentas, baseia-se na observação de que componentes do código (linhas, desvios e estados, etc.) tenham sido exercitados. As técnicas de cobertura estrutural têm sido amplamente usadas e várias deficiências têm sido encontradas. A limitação principal encontrada nelas é sua pouca (ou nula) relação com a funcionalidade do sistema sob verificação, ou

seja, não se tem certeza o quanto da especificação funcional tenha sido verificada. Por esta razão, apesar de a cobertura estrutural ser sempre utilizada nos processos de verificação, a cobertura funcional tem sido mandatória, pois está diretamente relacionada aos aspectos funcionais que o engenheiro de verificação deseja observar. Por outro lado, a elaboração do modelo de cobertura funcional é difícil de se determinar, uma vez que não é claro como correlacionar aspectos funcionais descritos na especificação com parâmetros do DUV. Este problema se acentua para modelos de DUV de caixa preta, foco desta dissertação, onde apenas as entradas e saídas estão disponíveis.

Para auxiliar o engenheiro de verificação na obtenção do modelo de cobertura funcional, alguns estudos têm sido realizados onde são propostos modelos como o de Verma et al. (9), baseado na análise das dependências entre as variáveis, e de Fallah et al. (10), baseado em técnicas de observabilidade. O primeiro modelo gera, pelas dependências, condições que devem, ou não, ser observadas, como se fossem asserções. O segundo modelo procura através de colocação de *tags* em determinadas variáveis (a serem determinadas pelo engenheiro de verificação como as serem conferidas) que são então propagadas até as saídas evidenciando a observabilidade. As duas metodologias propostas atuam sobre a estrutura interna (código) do DUV o que caracteriza-os como pertencentes às estratégias de caixa-cinza (branca).

Na atualidade, uma das formas principais utilizadas na indústria é a cobertura funcional por itens, especialmente interessantes para DUVs em caixa-preta. Esta metodologia consiste na contagem de observações realizadas de um conjunto de eventos considerados relevantes e de especial interesse. Eventos são ocorrências em parâmetros do sistema. Denomina-se parâmetro uma entrada, uma saída, um valor registrado internamente ou uma declaração que determinam ou mudam o modo de operação do sistema integrado. A partir dos parâmetros, pode-se definir o espaço de combinações de parâmetros como o espaço de verificação. A cobertura pode se referir a parâmetros de entrada, de saída, ou internas, sendo as duas primeiras denominadas de modelos de cobertura de entrada e de saída, respectivamente.

Um modelo de cobertura de entrada bastante simples é definir uma distribuição uniforme para os itens de eventos a serem observados. Alternativamente, pode-se determinar o modelo de entrada pela modelagem do espaço de possíveis combinações de entrada aplicáveis a um determinado sistema integrado. Neste processo, estabelece-se "aspectos" definidos como comportamentos ou eventos de especial interesse e têm maior prioridade dentro do plano de verificação e, portanto, ao se projetar a cobertura, um maior número de vetores do total a ser observado deveria ser dedicado àqueles. Para ilustrar esta possibilidade, considere-se um exemplo com o espaço de eventos conformado por $X_0, \ldots, X_4$, mas segundo aspectos estabelecidos pelo engenheiro de verificação, o modelo de cobertura tem

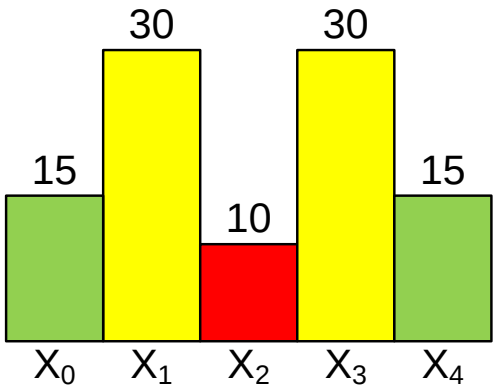


Figura 1.4: Exemplo de distribuição desejada

a distribuição mostrada na figura 1.4, onde se observa que os eventos X_1 e X_3 requerem maior número de vetores como indicativo de que tenha sido verificado apropriadamente.

Uma alternativa para a geração de modelos de cobertura de entrada, assim como de estímulos de teste, foi proposta por Castro et al. em (11), a partir da formalização do espaço de verificação em domínios de parâmetros, conceito introduzido em (12). O espaço de combinações de parâmetros de entrada é definido por dependências entre eles e é modelado como um conjunto de domínios, onde cada domínio é uma combinação válida de valores de parâmetros. O espaço é representado em forma de um grafo, e um algoritmo gera o modelo de cobertura por itens.

Se por um lado pode-se obter diretamente o perfil dos parâmetros de entrada, que pode conformar o modelo de cobertura de entrada através das dependências entre os parâmetros e do perfil de distribuição dos estímulos, por outro, é bastante complexa a geração do modelo de cobertura de saída por itens, como já mencionado anteriormente. Até o momento, não se tem na área de verificação modelos de cobertura de saída por itens de maior abrangência, como os de (9) para parâmetros internos. Atualmente, na indústria, os modelos de cobertura de saída por itens são estabelecidos usando-se modelos simplistas, como a de distribuição uniforme, ou baseados na experiência do engenheiro de verificação, que os torna extremamente pessoais. Uma das decorrências destas estratégias manuais é que a distribuição real observada na saída fica descorrelacionada do modelo comportamental do DUV. Ao se realizar a geração de vetores de teste de forma aleatória, a progressão da cobertura de saída não é linear uma vez que existem eventos com maior probabilidade de ocorrência e que conseguem atingir rapidamente o número de itens especificados para si no modelo de cobertura utilizado. Outros eventos, ao contrário, podem mostrar dificuldade para atingir os seus objetivos de cobertura, implicando em grande aumento do tempo de verificação, tornando-o ineficiente.

Como forma de evitar ineficiências, pode-se aproximar o modelo manual de cober-

tura de saída por itens à distribuição real de saída do DUV. Infelizmente, não existe um método adequado de se conhecer tal distribuição. Enquanto a distribuição de valores dos parâmetros de entrada está relacionada ao perfil dos vetores de teste aplicados, o qual é definido pelo próprio engenheiro de verificação, a distribuição à saída depende da relação funcional entrada-saída que, em geral, é bastante complexa. Adicionado a isto, ao existir uma relação entrada-saída, o número de itens que se deseja observar por cada evento de saída pode depender das restrições da entrada, além da distribuição destas, tornando a distribuição da saída mais imprevisível.

1.2 Objetivos

O presente trabalho tem como objetivo geral o desenvolvimento de uma metodologia para obter um modelo de cobertura que permita uma maior abrangência, que pode servir ao engenheiro de verificação na seleção dos pontos ou intervalos de saída de interesse, adicionado às decisões derivadas de seu conhecimento especialista. Estes pontos serão obtidos através de distribuição de entrada em um perfil definido e entregue pelo projetista.

Este projeto prevê o atendimento dos seguintes objetivos específicos:

1. A partir do análise do problema, estabelecer uma metodologia conceitual para sistematizar o processo de verificação funcional baseada em cobertura de saída por itens.

2. Estabelecer uma metodologia para estabelecer a relação entrada-saída dentro de um formato conveniente para a geração de modelo de cobertura de saída.

3. Programar uma ferramenta que, a partir da descrição do espaço de parâmetros de entrada, permita definir o espaço de saída correspondente do DUV, o qual também deve auxiliar na definição dos estímulos a serem gerados para se observar os aspectos funcionais desejados. Para isto, empregar-se-á a linguagem C++.

4. Programar uma ferramenta que a partir da descrição do espaço de verificação crie automaticamente o modelo de cobertura na linguagem SystemC.

5. Validar a metodologia para a geração de modelo de cobertura de saída, comparando-o com o perfil dos resultados de saída obtido por simulação.

6. Agregar toda a metodologia e compará-la ao método de estímulos aleatórios simples.

1.3 Organização da dissertação

Além do presente capítulo introdutório, este documento está organizado em mais seis capítulos:

- O capítulo 2 apresenta os conceitos fundamentais da verificação funcional e da análise de cobertura, apresentando também os conceitos em torno das técnicas de Verificação Dirigida por Cobertura.
- O capítulo 3 apresenta os trabalhos publicados na literatura técnica sobre métricas de cobertura para avaliação de sistemas.
- O capítulo 4 apresenta os fundamentos teóricos e análises realizadas para se determinar o método de cômputo da cobertura de saída.
- O capítulo 5 apresenta os algoritmos e procedimentos de cálculo derivados dos fundamentos teóricos de cômputo da cobertura de saída gerados no capítulo 4.
- O capítulo 6 apresenta os resultados experimentais da ferramenta desenvolvida com a metodologia proposta.
- O capítulo 7 apresenta as conclusões e trabalhos futuros que podem ser gerados a partir do trabalho realizado nesta dissertação.

2 Fundamentos teóricos

O propósito deste capítulo é apresentar os conceitos teóricos em torno da Verificação Funcional, com um foco maior nos aspectos de cobertura. Para um melhor entendimento do tema de estudo, primeiramente se faz uma introdução aos temas associados à Verificação Funcional (entenda-se Verificação Dinâmica por Simulação). A seguir explicam-se os paradigmas com que se trabalha atualmente. E por último, são tratados os conceitos básicos como o ambiente de simulação, os tipos de geração de estímulos e as métricas de cobertura, além de como estas podem ser aproveitadas para acelerar o processo de verificação.

2.1 Aspectos associados

A verificação funcional por simulação compartilha conceitos de outras áreas da eletrônica como a da confiabilidade de sistemas. Nela, um determinado sistema é analisado para determinar se este não diverge do comportamento coerente com a especificação durante um certo tempo. Em sistemas de software as principais divergências de comportamento são devidas a defeitos no projeto, já que os componentes destes sistemas não apresentam desgaste, fato que torna o seu desenvolvimento semelhante ao dos sistemas projetados com linguagens de descrição de hardware (do inglês hardware description languages, HDL). Na teoria de confiabilidade de sistemas existem três definições importantes (7):

1. Falha (Fault): A falha é o defeito que se encontra dentro do sistema e que pode gerar um comportamento diferente do desejado. No caso de um projeto de hardware em HDL, ela pode ocorrer por uma interpretação errada da especificação ou um engano no momento de escrever o código.

2. Avaria (Failure): A avaria é falha do circuito que foi estimulada e que é detectável.

3. Erro (Error): O erro é a manifestação de uma falha na saída (ou pontos observáveis) do sistema.

Segundo esta lógica, num circuito pode existir uma falha latente e, uma vez estimulada esta falha, uma avaria pode ser produzida. No caso que esta possa ser observada na saída do circuito, então se produz um erro. Caso a avaria seja mascarada então o erro não é observado. Estas ideias são utilizadas também na área de Teste Físico de circuitos (13). Em ambos os casos a seleção de vetores de teste deve permitir observar na saída a existência (ou não existência) de uma falha no sistema. A partir disto, pode-se inferir os seguintes conceitos:

1. Controlabilidade: Refere-se à habilidade de se estimular uma zona específica do circuito para ativar uma falha (se houver) (14) (15).

2. Observabilidade: Refere-se à habilidade de observar os efeitos do estímulo aplicado ao circuito, o que permite identificar seus erros no processo de simulação (14) (15).

A facilidade de detecção de falhas dentro do sistema depende do grau de observabilidade, podendo a verificação ser classificado em três classes (7):

1. Caixa Preta: neste caso só é possível observar as entradas e saídas do sistema. Uma vantagem de se usar a técnica de caixa preta é que qualquer mudança interna da estrutura do projeto sob verificação (do ingles, design under verification, DUV), durante o desenvolvimento do projeto, não tem impacto no código do testbench usado na verificação. Por outro lado, já que só se tem controlabilidade sobre as entradas e observabilidade nas saídas, as condições e os sinais internos ficam inalcançáveis (7).

2. Caixa Branca: permite observar todos os sinais internos do DUV, o que permite o acompanhamento próximo do comportamento do sistema. A caixa branca tem a desvantagem de sua forte relação com a estrutura interna do sistema, e, no caso de haver alterações em partes do código do DUV, exigiria-se fazer alterações no testbench (7).

3. Caixa Cinza: este método é uma combinação dos métodos de caixa preta e branca. É o modelo típico usado nos ambientes de simulação, principalmente porque permite certo nível de predição do sistema (7).

2.2 Paradigmas dentro da verificação funcional

A partir dos pontos apresentados na seção 2.1, descrevemos aqui uma explicação mais detalhada do processo de verificação, com base nos paradigmas atuais seguidos na indústria.

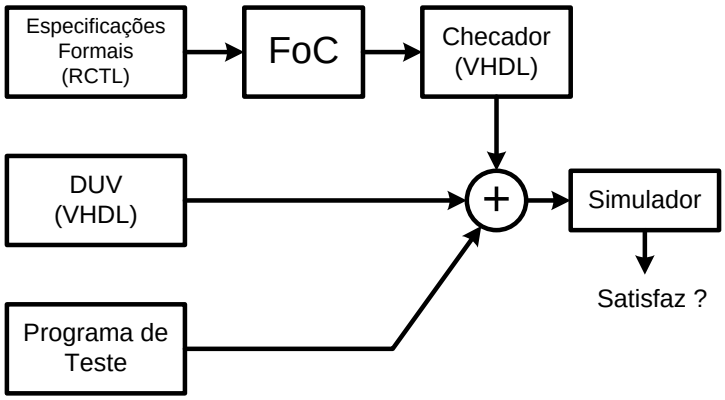


Figura 2.1: Arquitetura da Ferramenta FoC

2.2.1 Metodologia de verificação baseada em asserções

Esta subsecção detalha o uso das asserções na verificação funcional. Uma asserção é uma declaração incorporada ao código do DUV que tem como objetivo a observação de uma propriedade que deve ser verificada. Esta declaração, diferentemente do código do projeto, não contribui em nenhum elemento de hardware no circuito que está sendo projetado, se encarregando principalmente na comprovação da consistência entre o circuito desejado e o circuito real.

A metodologia de verificação baseada em asserções pode ser considerada como a união entre a verificação por métodos formais e a dinâmica por simulação, uma vez que pode empregar métodos formais para a determinação de aspectos que devem ser observados durante a simulação (14), e cuja conformidade perante as especificações pode ser avalia da matematicamente (mais detalhes na seção 2.3).

Um exemplo de ferramenta baseada em asserções é FoC (Checador Formal, do inglês Formal Checker), desenvolvida na IBM (16), na qual o engenheiro de verificação ingressa com as especificações, usando uma linguagem RCTL (do inglês Regular Computation Tree Logic) (17) - extensão da linguagem CTL (18), como indicado na Figura 2.1 . Com esta informação, a ferramenta faz a análise e transforma a descrição de propriedades em RCTL para asserções descritas na linguagem VHDL. As asserções são armazenadas no checador que se encarrega de observar que as propriedades descritas sejam cumpridas.

Um modelo generalizado deste método é o uso das propriedades estruturais do DUV, a partir dos quais se geram os aspectos de cobertura a serem levados em conta juntamente com as asserções. Uma arquitetura possível é mostrada na Figura 2.2, que para gerar eficientemente as asserções utiliza os dados da especificação, além das propriedades do DUV. Estas permitem fazer uma análise e refinar as métricas ou pontos específicos a serem considerados no processo da verificação.

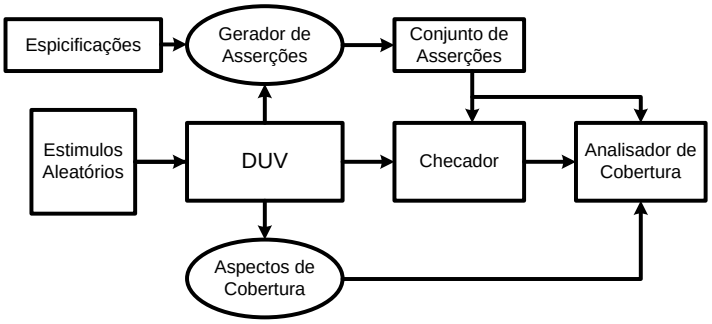


Figura 2.2: Modelo de Verificação Dinâmica por Simulação com Asserções

Na literatura mostra-se que este método de verificação tem uma eficiência elevada na detecção de erros em processadores, como: DEC Alpha 21164, com a detecção de 34% dos erros presentes (19), DEC Alpha 21264, com 25% dos erros presentes detectados (19) e Cyrix M3, com 25% (20). Estes exemplos revelam que os métodos formais, aplicados no processo de verificação dinâmica por simulação, têm muitas vantagens, uma vez que juntam as qualidades de ambas técnicas: satisfabilidade (pelos métodos formais) e evitar o problema da explosão de espaço de estados (pelo uso de verificação dinâmica por simulação).

A técnica de verificação por asserções apresenta duas dificuldades para o engenheiro de verificação: 1) desenvolver um verificador de complexidade uma vez que deve embutir todas as propriedades do circuito a serem observadas; 2) saber se o conjunto de asserções utilizadas corresponde a uma grande cobertura das especificações do sistema considerado.

2.2.2 Metodologia de verificação usando do modelo de referência

Diferentemente do anterior, a simulação baseada em Modelo de Referência utiliza-se de uma representação abstrata contendo todos os aspectos funcionais que o DUV deve ter, baseado na especificação. Isto permite o projeto de um verificador simples, já que a única tarefa a ser realizada é observar se os valores de saída do DUV e do modelo de referência coincidem; caso isto não ocorra, o DUV terá apresentado um erro e o projeto deve ser modificado. O checador é mais simples que no caso por asserções, já que as propriedades do sistemas estão representadas no modelo de referência. Na Figura 2.3, a arquitetura generalizada para este método é mostrada. Observa-se nela que o gerador de estímulos entrega os vetores de teste em forma de dados ou pacotes tanto para o DUV como para o modelo de referência. Para que estes dados sejam aceitos ou entregues pelo DUV, drivers e monitores, que convertem dados em sinais RTL (e vice-versa) são utilizados. A resposta de ambos blocos é comparada no checador, no caso de ser válida esta passa por analisador de cobertura.

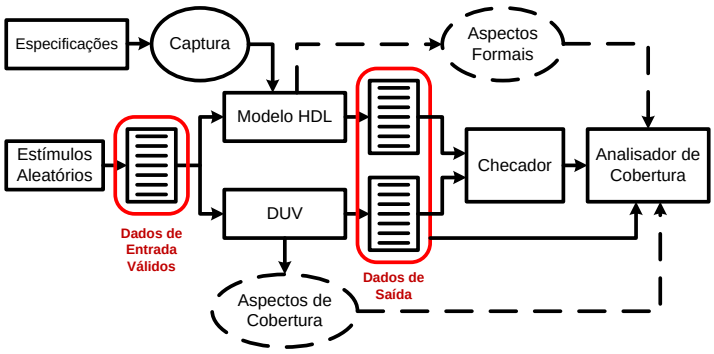


Figura 2.3: Modelo de Verificação Dinâmica por Simulação usando modelo de Referência

2.3 Conceitos básicos

2.3.1 Ambiente de simulação - Testbench

No processo de verificação, o ambiente de simulação (ou Testbench) deve modelar o ambiente de operação do DUV implicando assim na simulação de "todas" as possíveis condições em que o circuito deve trabalhar e que permitam avaliá-lo. Na Figura 2.4, os componentes do ambiente de simulação são mostrados: o gerador de estímulos (Gerador de Casos de teste), a autochecagem (Checador) e o modelo de referência. Segundo Wile et al. (7), realizar uma verificação requer que dois aspectos sejam muito bem considerados: a controlabilidade nos geradores de casos de teste, e a observabilidade na autochecagem.

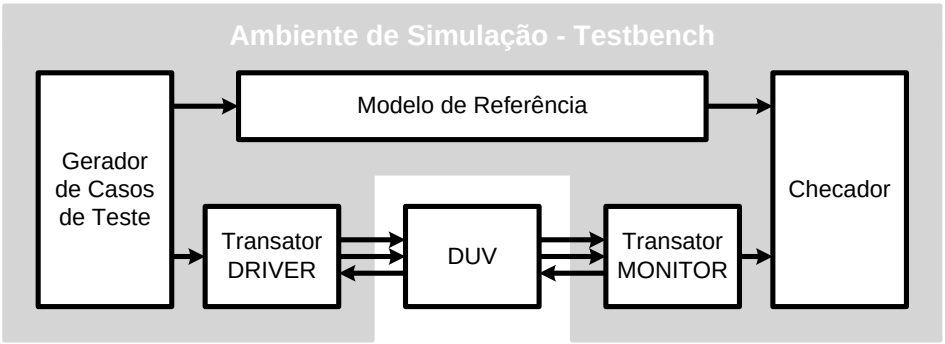


Figura 2.4: Ambiente de Simulação

2.3.2 Geração de casos de teste

A geração de casos de teste consiste na criação de estímulos válidos segundo a especificação. Existem quatro estratégias para gerar os casos de teste:

1. Dirigidas: Este é utilizado principalmente quando o engenheiro de verificação identifica casos individuais de teste de acordo com sua interpretação e experiência. Estes

testes objetivam ser representativos do funcionamento de um circuito e muitas vezes o conjunto de estímulos é recomendado na especificação do IP, apesar de sua limitação pela falta de escalabilidade (6).

2. Aleatórias: Permite a geração automática de casos de teste para suas variáveis de entrada (6). Para um melhor desempenho no momento da verificação procura-se usar restrições apropriadas para não gerar estados ilegais do sistema (12). As ferramentas existentes para a construção de testbenches contêm componentes de geração de estímulos que usam geradores de estímulos pseudoaleatórios, o qual adota tabelas com distribuições de probabilidade para a geração de decisões sobre os estímulos de entrada (7).

3. Casos Extremos: Pode-se considerar como um caso de estímulo dirigido, mas com a diferença que nestes casos tenta-se alcançar situações não-usuais; são estados que devem ser verificados com especial atenção, onde se pode encontrar um erro, porém após longos testes (os autores de (7) relatam centenas de trilhões de ciclos de teste). Outros métodos de estimulação (aleatórios, diretos) podem fornecer certo nível de eficiência, escalabilidade e reusabilidade, mas dificilmente conseguem envolver o nível de cobertura necessário para atingir erros relacionados a casos extremos dos valores dos estímulos de entrada (21).

4. Casos reais: os casos reais de teste são aqueles compostos, unicamente, por estímulos que representam dados reais de operação do sistema, como por exemplo vídeos utilizados em decodificadores MPEG. A relevância deste tipo de estimulação reside na necessidade de se monitorar, e verificar, o funcionamento do modelo do sistema, sob condições reais de operação.

2.3.3 Cobertura

Devido à complexidade dos sistemas atuais, métodos exaustivos de teste tornam-se inviáveis para encontrar as falhas dentro do circuito. Define-se, então, métricas de cobertura que permitem avaliar a abrangência e efetividade da verificação para um alto grau de confiabilidade. Estas métricas de cobertura podem ser classificadas em Funcional ou Estrutural. Para cada tipo, foram propostas estratégias de aceleração do processo da verificação. Tantos os tipos de cobertura como as estratégias são apresentados a seguir.

2.3.3.1 Tipos de cobertura

1. Cobertura Estrutural

A cobertura estrutural baseia-se nas métricas de cobertura dos sistemas de software, no qual analisam-se os elementos ou componentes do código que descreve os detalhes do DUV (8) (21). Dentro deste tipo de cobertura pode-se monitorar diferentes aspectos do código durante o processo de verificação, resultando em diferentes modelos de cobertura. Os principais são:

(a) Cobertura de linha: esta métrica tem como objetivo quantificar o número de vezes que cada uma das linhas do código HDL tenha sido exercitada. Com este método de cobertura, pode-se descobrir quais partes do código HDL não tenham sido devidamente ativadas durante o processo de verificação.

(b) Cobertura por ramos: esta métrica tem como objetivo quantificar o número de vezes em que as condições de controle (if, case, while, etc.) tenham sido exercitadas. Neste tipo de cobertura, deve-se considerar todas as opções possíveis que existem para se obter uma resposta.

(c) Cobertura por TOGGLE: esta métrica tem como objetivo quantificar o número de vezes em que se gera uma transição (de "1 para 0" ou "0 para 1") em cada um dos bits de um sinal. Este método de cobertura é útil para se verificar a adequada atividade de um sinal no sistema, com o qual se pode-se testar e comprovar o correto funcionamento da lógica de controle ou ter a certeza que os sinais entre dois módulos estão trabalhando corretamente.

(d) Cobertura por Condições: Esta cobertura tem como objetivo uma análise mais ampla que a cobertura por ramos oferece. Esta cobertura permite a medição nas estruturas condicionais aninhadas, testando todas as possibilidades de execução; por isso é considerada como uma extensão da cobertura por ramos, incluindo as condições adicionais existentes.

(e) Cobertura de caminhos: Este tipo de cobertura quantifica o número de caminhos que tenham sido percorridos no código HDL. Devido ao grande número de caminhos existentes na descrição de um sistema, em geral, é inviável testá-los todos.

(f) Cobertura de Máquinas de Estados Finitos: Este tipo de cobertura apresenta diferentes formas de medição, sendo a mais simples a contagem de quantas vezes cada um dos estados da Máquina de Estados Finita (do inglês Finite State Machine, FSM) foi percorrido. Outra das métricas é identificar o número de vezes que um estado transita a outro estado vizinho dentro da FSM. A terceira métrica de Cobertura, conhecida como Cobertura por Transição, identifica a visita sequencial de vários estados formando conjuntos diferentes comprimentos, e conta

a quantidade de vezes, dentro do processo de verificação, tal sequência tenha sido percorrida.

As diferentes métricas de cobertura estrutural apresentadas têm sido exaustivamente utilizadas nos processos de verificação pela facilidade e por comodidade no seu uso. Confirmação disto é a observação de que cada uma das ferramentas de verificação dinâmica, existentes no mercado, possuem diferentes aplicativos de cálculo de cobertura baseados nestas métricas. Através da experiência no uso destas, suas limitações foram também reconhecidas - as métricas não estão diretamente correlacionadas aos aspectos funcionais com que foi projetado cada um dos circuitos; portanto, no processo de verificação podem existir muitos caminhos e estados não visíveis pelo código. Por esta razão, a cobertura funcional tem sido preferida, apesar de apresentar uma complexidade maior.

2. Cobertura Funcional

   A Cobertura Funcional consiste de métricas modeladas ou selecionadas pelo engenheiro de verificação para capturar funcionalidades do sistema, podendo ser derivadas da implementação e/ou dos requerimentos funcionais do dispositivo. Para isto faz se uma extração dos atributos do sistema que representam eventos do DUV que serão observados durante a verificação. Os atributos podem ser em forma de propriedades ou eventos amostrados.

   Na Cobertura Funcional por Propriedades, estas expressam aspectos funcionais que devem ser cumpridas no DUV. Estas propriedades têm duas origens: por especificação e as derivadas da implementação. Por exemplo, o projeto 'A' precisa cumprir uma tarefa 'X' baseado numa série de regras (propriedades por especificação), mas, por outro lado, no momento de se projetar o circuito para esta tarefa o engenheiro especifica sequências de cômputo (determinadas por FSMs) e/ou sinais estabelecidas por ele para se obter a solução (propriedades de implementação).

   A forma que as propriedades podem ser verificadas é usando-se asserções. Este método permite unir a metodologia de verificação por métodos formais e a verificação dinâmica por simulação (14). A proximidade entre elas permite que as asserções tenham as mesmas propriedades definidas formalmente (8) (20), como abaixo:

   (a) Segurança (Safety): Uma propriedade de segurança é uma condição que nunca deveria ocorrer em um tempo limitado, idealmente NUNCA deveria acontecer. Um exemplo da propriedade de segurança é "Num registro, não mais de um bit deve mudar de valor por ciclo, num controle codificado em código Gray".

(b) Liveness: A propriedade especifica que certo evento deve acontecer, em certo instante de tempo. A diferença com o caso anterior, é que não tem tempo fixo onde deve acontecer o evento esperado. Um exemplo da propriedade de Liveness é "Depois que o sinal reset é desativado, o ciclo de captura de instrução é iniciada".

Outra forma de classificar as asserções é pelo seu propósito, entre os quais se encontram:

(a) De Checagem (Checking): O seu propósito é capturar os requisitos de qualquer especificação ou abstração, com o propósito de detectar erros nos dados ou violações nas propriedades de tempo.

(b) De Cobertura (Coverage): Busca gerar um relatório da quantidade de vezes que um evento ocorre.

Além das classificações mencionadas anteriormente, as asserções podem ser classificadas como:

(a) Concorrentes (Declarative): É a avaliação de um evento que ocorre em todo momento.

(b) Seqüenciais (Procedural): É a avaliação de um evento, que está dentro de um Processo que é executado sequencialmente (exemplo o PROCESS no VHDL ou ALWAYS no Verilog).

Apesar das vantagens que a cobertura por asserções proporciona, como o bom desempenho (20) (22) (19), eles sofrem da complexidade para a sua obtenção correta, já que muitas das propriedades são extraídas manualmente do documento de verificação. Ademais, assim como quando utilizadas em métodos formais, as asserções sofrem da dificuldade de se saber qual é o número de propriedades que o circuito precisa verificar para um alto grau confiabilidade.

Existe, por outro lado, a Verificação Funcional por eventos amostrados (parâmetros ou variáveis) na simulação. Dentro deste tipo de cobertura funcional, pode-se considerar:

(a) Cobertura por itens: consiste de um conjunto de eventos relevantes e de especial interesse que se pretende observar à entrada, na parte interna, ou à saída do DUV. Tais eventos devem ser definidos em função dos valores, ou conjuntos de valores, que variáveis e/ou parâmetros do sistema ou de configuração podem adquirir na simulação, e do número de itens correspondentes a cada um dos eventos definidos, ou seja, a quantidade de vezes que o evento deve ocorrer para

se considerar satisfeito. No exemplo de quatro eventos abaixo,

$$\begin{aligned}
&\textit{event 1} \rightarrow \textit{value range}: \textit{(0x000000,0x3FFFFF)}, \textit{1000 items}\\
&\textit{event 2} \rightarrow \textit{value range}: \textit{(0x400000,0x7FFFFF)}, \textit{1000 items}\\
&\textit{event 3} \rightarrow \textit{value range}: \textit{(0x800000,0xBFFFFF)}, \textit{1000 items}\\
&\textit{event 4} \rightarrow \textit{value range}: \textit{(0xC00000,0xFFFFFF)}, \textit{1000 items}
\end{aligned}$$

deve-se medir a cobertura funcional de um endereço de 32 bits; o número total de eventos de cobertura corresponderia a aproximadamente 4 bilhões de valores. Neste caso, o melhor seria estabelecer eventos de cobertura funcional representados por conjuntos de valores de verdadeira importância, como o agrupamento em quatro conjuntos.

Existe, entretanto, um problema relacionado ao tamanho numérico de cada um desses eventos de cobertura, já que a faixa de valores de cada evento é ainda ampla demais para se estabelecer níveis exaustivos de cobertura. A forma de resolver este problema é determinar um número suficientemente representativo de itens a serem monitorados em cada faixa. No caso do exemplo dado, são definidos 1000 itens para cada conjunto.

(b) Cobertura Cruzada: Esta métrica é utilizada de forma semelhante ao da cobertura por itens, porém com mais de um item simultaneamente; por esta razão, o número de possíveis valores cresce fatorialmente com o de itens cruzados. Voltando ao exemplo anterior, onde o espaço de valores de um parâmetro foi dividido em quatro eventos de cobertura, é possível que cada um dos espaços de valores do endereço de 32 bits estivesse relacionado a valores específicos de outro(s) parâmetros(s). Considerando que um destes supostos parâmetros receba o nome "número_endereco", e que pode receber os valores 0, 1, 2, ou 3, seria possível então definir, por exemplo, os seguintes eventos de cobertura cruzada:

$$\begin{aligned}
&\textit{(numero_endereco =0)} \wedge \textit{(endereco >=00000001)} \wedge \textit{(endereco <=7FFFFFFF)}\\
&\textit{(numero_endereco =1)} \wedge \textit{(endereco >=80000000)} \wedge \textit{(endereco <=8FFFFFFF)}\\
&\textit{(numero_endereco =2)} \wedge \textit{(endereco >=A0000000)} \wedge \textit{(endereco <=CFFFFFFF)}\\
&\textit{(numero_endereco =3)} \wedge \textit{(endereco >=D0000000)} \wedge \textit{(endereco <=FFFFFFFF)}
\end{aligned}$$

Desta forma, cada evento de cobertura seria contabilizado somente no caso que, para cada evento, se cumprirem as três condições especificadas. Claramente, este é um dos motivos pelos quais, estímulos gerados aleatoriamente, devem ser coerentes em relação às dependências existentes entre eles.

2.3.3.2 Verificação dirigida por cobertura

Baseado nos modelos de cobertura apresentados anteriormente, realiza-se um plano de objetivos no processo de verificação, a partir do qual indica-se cada um dos pontos a serem observados. Uma vez geradas as métricas a serem usadas, procede-se a geração de estímulos aleatórios, ou no caso que se precise, geração de casos dirigidos.

Um fenômeno típico no processo de verificação à base de geração aleatória de casos de teste, é que estes se tornam ineficientes com a progressão da simulação pelo fato de o gerador de casos de teste poder estar gerando estímulos redundantes que não ajudam a acelerar o processo. Redundância deve ser entendida aqui como casos de teste que exploram zonas funcionais do DUV que já foram fortemente testadas, impedindo que outros aspectos funcionais não sejam (ou sejam levemente) testados. Para suprir esta desvantagem da geração dos casos de teste aleatório, podem-se usar técnicas específicas como: Geração Baseada em Restrições (do inglês Constraint-Based Generation) e Geração Dirigida por Cobertura (do inglês Coverage-Directed Generation, CDG), os quais são apresentados a seguir.

A geração baseada por restrições é uma metodologia que se caracteriza por definir grupos de restrições para calcular os valores dos estímulos e opera numa configuração de laço aberto, o qual corresponde ao ambiente de simulação mostrado anteriormente na Figura 2.4. Dentro deste método, pode-se classificar dois tipos de restrições:

1. Restrições Funcionais: estão baseadas nas especificações ou requisitos do projeto. Neste tipo de restrição têm-se em conta as capacidades do dispositivo, as características da arquitetura, meios de comunicação, etc.

2. Restrições de Verificação: tem como objetivo reduzir o espaço de estímulos válidos a um subgrupo útil para mostrar os erros do dispositivo. Este subgrupo é caracterizado por condições limitadas na entrada e propriedades temporais necessárias para ativar algumas limitadas condições internas do dispositivo.

A geração dirigida por cobertura é aquela na qual o engenheiro infere as restrições necessárias para gerar os vetores de teste, o CDG infere as restrições dependentes da cobertura especificada e os espaços ainda não explorados; para tal deve-se descobrir correlações entre os estímulos inseridos e espaços não cobertos (23).

Este Método usa uma configuração de laço fechado no ambiente de simulação, também conhecido como modelo reativo, como mostrado na Figura 2.5.

Os métodos dirigidos por cobertura usam os conceitos de observabilidade e controlabil-

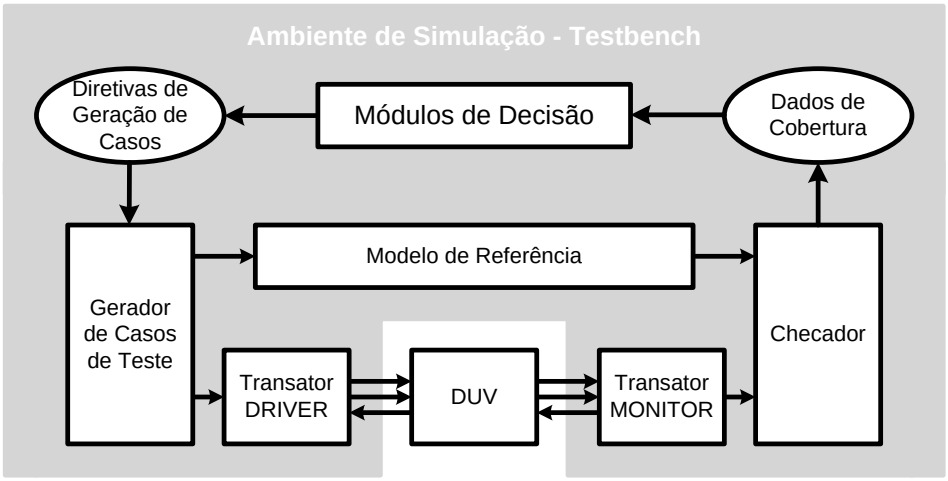


Figura 2.5: Ambiente de Simulação Reativo

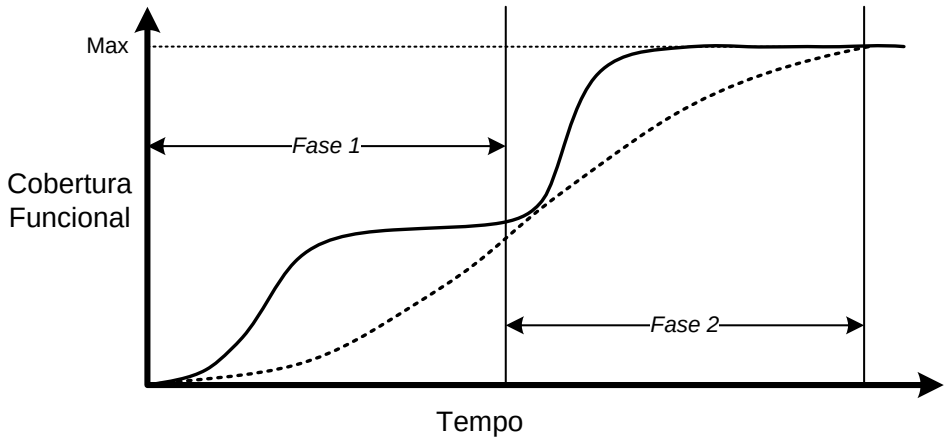


Figura 2.6: Progresso da Cobertura Funcional com e sem CDG

idade para monitorar os valores da saída e gerar a relação destes com os estímulos gerados. Com estes dados uma ferramenta associada muda os pesos probabilísticos dos vetores (Controlabilidade) para alcançar 100% de cobertura com um menor número de casos de teste (24), como mostrado na Figura 2.6(8) (Linha pontilhada, verificação sem CDG, e linha inteira, verificação com CDG). Apesar de estas técnicas conseguirem diminuir significativamente o tempo de execução dos testbenches, elas se caracterizam por necessitar de um extenso e complexo trabalho da equipe de verificação para determinar modelos e métricas de cobertura representativas de cada aplicação. Dificuldade adicional se dá por as métricas serem estabelecidas subjetivamente a partir do conhecimento específico dos sistemas sob verificação e da experiência por parte dos membros da equipe de verificação.

3 Trabalhos correlatos

Neste capítulo busca-se apresentar artigos relacionados a aspectos de cobertura, particularmente, metodologias que tratam da geração automatizada de modelos de cobertura funcional. Não é um trabalho compreensivo, porém é bastante representativo de como o meio acadêmico tem focado a questão. Dada a complexidade do problema, não se percebe nenhuma solução com cobertura mais robusta, evidenciando a necessidade de soluções mais práticas para o processo industrial de verificação.

3.1 Jerinic e Muller, ano 2004

3.1.1 Objetivo

Jerinic desenvolveu em (12) uma metodologia de seleção do espaço dos possíveis casos de teste com o objetivo específico de otimizar a geração de estímulos (corner cases). O conjunto total de possíveis valores dos parâmetros de entrada de simulação é reduzido através da definição de dependências entre tais parâmetros, conhecidas a partir da especificação funcional do DUV. A seguir, primeiramente, os conceitos em torno do tópico são apresentados, depois os trabalhos correlatos a tais conceitos e, por último, as limitações da metodologia aplicada.

3.1.2 Conceitos

3.1.2.1 Espaços de parâmetros

Os parâmetros de entrada de um sistema conformam um espaço, ou área, cujo tamanho é dado pelo conjunto de todos os valores que os parâmetros podem adquirir. Então, um espaço P é constituído pelos parâmetros de verificação $p_1, p_2, \ldots, p_n$, onde cada parâmetro pode adquirir R valores; isto é mostrado na Equação 3.1, e o seu tamanho é definido pela Equação 3.2.

$$P = p_1 * p_2 * \cdots * p_n \tag{3.1}$$

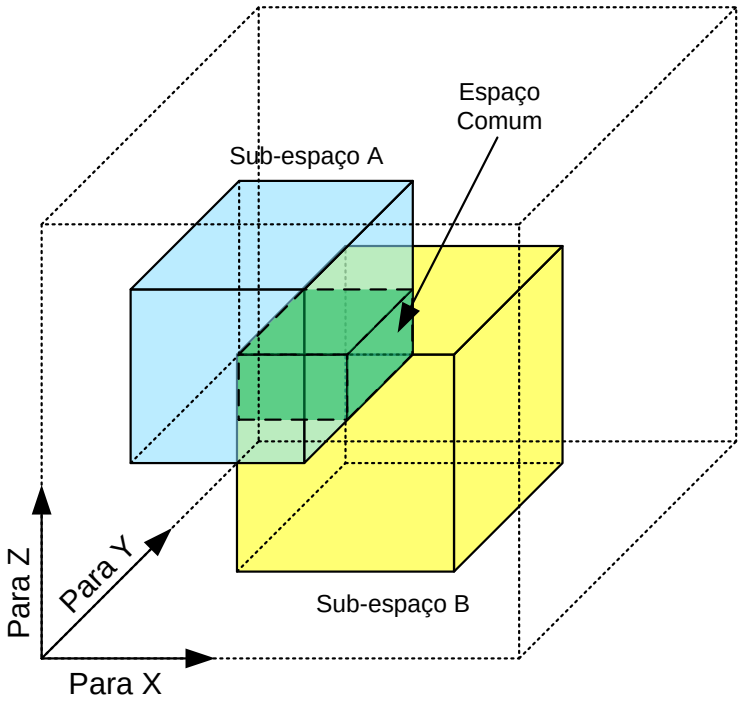


Figura 3.1: Espaço de Simulações Sobrepostos

$$S_{PARAM} = P = \prod R_i \tag{3.2}$$

Ao planejar a geração de estímulos em uma situação real, ao interpretar a especificação, o engenheiro de verificação pode dividir o espaço de parâmetros em subespaços, limitando a abrangência dos valores de alguns deles. O problema que ocorre na geração aleatória dos valores dos parâmetros é que subespaços de simulação sobrepostos, na Figura 3.1, a definição de diferentes faixas de valores para os parâmetros ParaX, ParaY e ParaZ criam dois subespaços o que se traduz em redundância de valores de teste aplicados. Esta é uma situação bastante comum quando a definição dos estímulos é uma atividade manual, que pode propiciar também a inclusão de valores inválidos da especificação na simulação.

Para prevenir que valores redundantes, ou inválidos, sejam gerados, e assim reduzir o tempo necessário para realizar a verificação, duas condições devem ser mantidas:

- Combinações inválidas de parâmetros devem ser excluídas do espaço de simulação.
- Somente devem ser usados subespaços que não se sobreponham.

3.1.2.2 Modelo de dependência

Para reduzir a quantidade de valores que os parâmetros podem adquirir, é necessário deduzir todas as relações de dependência existentes entre estes parâmetros. Tais dependências

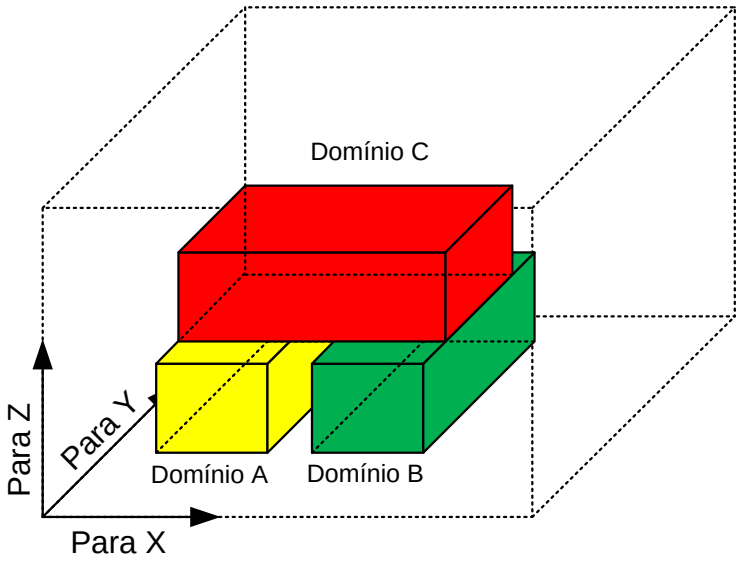


Figura 3.2: Domínio de Parâmetros

devem ser extraídas manualmente pelo responsável de verificação a partir das especificações e utilizadas para separar intervalos de valores de parâmetros em uma série de agrupamentos sem sobreposição, denominados de domínios de parâmetros (do inglês, *Parameter Domains, PDs*), como mostrado na Figura 3.2. Pela figura, três subespaços são definidos para a simulação de tal forma a que nenhum valor redundante seja gerado. Para a representação de forma gráfica de PDs, utilizam-se os Grafos de Domínios de Parâmetros (do inglês, *Parameter Domain Graphs, PDGs*). Um PDG é uma árvore, extensão dos diagramas de decisão múltipla (MDDs) (25) (12), onde cada conjunto de ramos e nós, da raiz à célula folha, é utilizado para armazenar um PD.

Existem dois tipos de relações de dependência:

Conflito: estabelece que um parâmetro X não tem efeito sobre o comportamento do DUV caso outro parâmetro Y assuma um determinado valor ou conjunto de valores, ou seja, X tem um CONFLITO com o valor de Y.

Requisito: indica que um parâmetro X influirá no comportamento do DUV apenas se outro parâmetro Y assumir um determinado valor ou conjunto de valores, ou seja, X tem um REQUISITO do valor de Y.

Baseado nos princípios acima, Jerinic desenvolveu ParaGraph, ferramenta e biblioteca para a criação e manipulação de PDs (12). Ela utiliza os requisitos e conflitos entre sinais, extraídos das especificações e definidos em um arquivo, para gerar os intervalos válidos de trabalho (26) (25).

3.1.3 Exemplo

Como exemplo de formação de PDs, consideremos que seja necessário reduzir o espaço de simulação de um IP que contém os parâmetros (com todos seus possíveis valores entre colchetes):

- paridade [par, ímpar, off]
- posição [1.. 10]
- comprimento [1.. 10]

O parâmetro paridade define o tipo de paridade (par ou ímpar) a ser usado na simulação, ou, então, se a geração de paridade está desligada (off). O parâmetro posição serve para escolher a posição do bit de paridade dentro do pacote enviado, enquanto o parâmetro comprimento regula o número de bits utilizados para formar o pacote de informação. Então, a cardinalidade do espaço de parâmetros é dada segundo 3.2:

$$|P| = \prod_{i=1}^{n} R = R_{paridade} * R_{posicao} * R_{comprimento} = 3 * 10 * 10 = 300 \tag{3.3}$$

Entretanto, pode-se observar que existem conflitos entre os parâmetros posição e paridade, que podem ser aproveitados para reduzir o espaço de parâmetros de simulação. Deste modo, criam-se neste exemplo três domínios de parâmetros:

- paridade [par], posição [1..10], comprimento [1..10]
- paridade [ímpar], posição [1..10], comprimento [1..10]
- paridade [off], comprimento [1..10]

A partir dos domínios gerados o número total de casos de teste, baseados na equação 3.4, é:

$$\begin{aligned} |P| &= D_{paridade \rightarrow impar} * D_{paridade \rightarrow par} * D_{paridade \rightarrow off} \\ &= 1 * 10 * 10 + 1 * 10 * 10 + 1 * 10 = 210 \end{aligned} \tag{3.4}$$

3.1.4 Trabalhos correlatos

A partir do arcabouço conceitual dos PDs, ferramentas de suporte foram desenvolvidas, entre elas a GET_PRG (do inglês, Generation Tool for Parameter-Domain based Random

Stimuli Generation), apresentada em (27) (28). Esta tem como objetivo gerar automaticamente, a partir do PDG, o gerador de estímulos com as restrições que selecionam o espaço representado pelo PDG.

Baseados na ferramenta GET_PRG, os autores mostram a eficiência do uso dos PDGs no processo de verificação dinâmica por simulação para um módulo de um circuito adaptador de rede Bluetooth e de um decodificador MPEG-4. Resultados foram obtidos para execução do testbench com 200.000 casos de teste, comparando-os a resultados de testebench com geração de casos de teste aleatório tradicional O número de casos de teste inválidos foram detectados e excluídos do caso aleatório tradicional, podendo-se verificar que, na maioria dos casos, este número está perto dos 50%, com exceção do MPEG4-CBP com 25% de casos inválidos. Desta forma, usando-se PDGs, o número de testes válidos por tempo de simulação é maior que no caso geração aleatória tradicional. Portanto o tempo de simulação para alcançar uma certa meta de cobertura é menor na metodologia baseado na metodologia com PDGs em relação ao método tradicional de geração aleatória, o que foi evidenciado experimentalmente

3.1.5 Limitações da estratégia proposta

O método de PDGs apresenta ótimos resultados na computação das restrições, mas, no caso de não existirem dependências entre os parâmetros de trabalho, o número de casos de teste não diminui.

Como limitações gerais do método, o PDG só inclui restrições nas entradas, portanto, se o engenheiro de verificação desejar verificar um grupo de estados na saída, ele não teria como predizer ou indicar o valor ou intervalo de valores de interesse.

3.2 Trabalho de Mishra e Dutt , ano 2005

3.2.1 Objetivo

O trabalho de Mishra e Dutt (29), proposto em 2005, refere-se a um método para avaliação de processadores segmentados (pipelined). Para isto, propõe-se o desenvolvimento de modelos de falhas para definir uma cobertura da arquitetura requerendo como dado de entrada o grafo do modelo da arquitetura. O grafo contém os caminhos por onde, dependendo da instrução, passam os dados assim como os elementos de processamento respectivos. Por isso, baseado na arquitetura RISC apresentada por Hennessy em (30), pode-se observar no gráfico as etapas de Fetch, Decode, Execute (esta composta por ALU, AddrCalc e o

Co-Processador), etc. A partir destes conhecimentos, a ferramenta gera os casos de teste que permitem percorrer as propriedades estabelecidas no grafo. O artigo de Mishra (29) apresenta os aspectos que devem ser observadas ao momento de gerar o Grafo do Modelo do DUV, além dos modelos de falhas estudados. Na realidade, o trabalho pode ser entendido como uma estratégia não só de obtenção de um modelo de cobertura de falhas, como também de orientação à geração de estímulos baseada neste mesmo modelo de falhas desenvolvido.

3.2.2 Metodologia

Para gerar o modelo de cobertura das falhas, primeiramente é inserido o grafo do modelo que captura as propriedades seguintes:

- Estruturais: considerando-se os elementos que compõem o processador (ALUs, Unidades de Armazenamento); além disso, inclui as etapas, dentro do processador, por onde os dados e/ou instruções percorrem.
- Comportamentais: são capturados, usualmente, pela Arquitetura por Conjunto de Instruções (do Inglês, Instruction Set Architecture, ISA). Nesta etapa de captura, determina-se cada campo das instruções que especifica sua semântica de execução, além dos caminhos que devem percorrer dentro da micro-arquitetura.

Mishra e Dutt na sua metodologia propõe quatro modelos de falhas para gerar cada um dos casos de teste para cada modelo, os quais são apresentados a seguir:

- Modelo de Falha de Registradores: confere se o valor no registrador foi escrito ou lido corretamente. Na presença de falha, o valor lido não retornará ao valor previamente escrito. A falha pode ser por um erro na leitura, escrita ou erro na decodificação do endereço.
- Modelo de Falha por Execução de Operação: este verifica se a saída do circuito diverge da resposta esperada. Pode ter como origem um erro na decodificação da operação, erro nos sinais de controle ou computação da operação.
- Modelo de Falhas de caminhos: uma operação op_i tem caminho de execução dentro do pipeline ep_{op_i}. Pode-se gerar um erro no caminho se um nó aceitar entradas válidas e produz uma saída incorreta, tanto para os casos de transferência de dados ou instruções.

- Modelo de falhas na execução do pipeline: o caso anterior só considera uma operação no tempo, mas no caso de arquitetura com pipeline pode-se gerar erros já que executa múltiplas operações.

Para cada um dos modelos apresentados há algoritmo que se encarrega de gerar os casos de teste para estimular cada uma das regiões especificadas em cada modelo (29).

3.2.3 Resultados

Para avaliar a proposta, empregam-se os processadores DLX (30) e LEON2 (baseado na arquitetura RISC do processador SPARC V8) (31). Nestes testes, a metodologia proposta por Mishra mostrou uma elevada eficiência comparada aos métodos de geração de teste aleatório e com restrições, utilizando-se como meta de cobertura, métricas do código descritivo do DUV. Resultados mostraram que a execução do testbench a partir de um gerador de estímulos acelera o processo de verificação consideravelmente; por exemplo, os autores mostram que, o número de casos de teste necessários para obter 100% de cobertura de falhas usando geração de casos de teste randômico é 30 vezes o número de casos requeridos com a proposta de Mishra (29).

Um caso similar acontece com a geração de casos com restrições para o qual, o número de casos de teste randômicos necessários é 5.68 vezes o número de casos verificados usando a metodologia de Mishra no caso da leitura ou escrita de registradores do processador DLX. Os autores mostram que o fenômeno é similar para testes realizados com o processador LEON2, mas com diferentes percentagens.

Por outro lado, ao usar cobertura de linhas em conjunto com a técnica proposta pelos autores, na maioria dos casos, esta alcançou valores menores ao que 100%, significando que existem zonas de processamento dentro de cada etapa do hardware que não foram exercitadas e que, portanto, podem levar a erros inesperados ou, então, pode existir código redundante, o qual nunca poderá ser alcançado.

3.2.4 Limitações da estratégia proposta

Os resultados mostram uma eficiência elevada na geração de casos de teste para alcançar a cobertura dos chegar a cobrir os pontos desejados, mas esta eficiência depende do modelo empregado na entrada do sistema. Uma das maiores vantagens das métricas em modelo de falhas, é que estas são mais fáceis de definir e usar. Por outro lado, a simplicidade na medição requer restrições adicionais, como assumir uma falha simples ou assumir que as falhas não são mascaradas. Isto ficou patente em alguns experimentos apresentados pelos

autores, onde as métricas de cobertura por código mostraram que há espaços de código que não são devidamente estimulado e testados.

3.3 Trabalho de Fallah, Devadas e Keutzer, ano 1998

3.3.1 Objetivo

No trabalho apresentado por Fallah, Devadas e Keutzer em (10), uma nova métrica de cobertura é proposta, focando o conceito de observabilidade, baseada na qual, uma ferramenta denominada OCCOM também foi desenvolvida para avaliar se eventos em determinada variável foram devidademente observados. Métricas por cobertura estrutural não se preocupam normalmente em mostrá-las, já que o objetivo delas é simplesmente ativar os elementos do código selecionados, como as linhas (só indicariam a passagem por determinada linha de código). O conceito de observabilidade é utilizado na proposta para determinar se os efeitos de uma avaria podem ser observados na saída. Os autores não apresentam critérios ou técnicas para a seleção de variáveis a serem observados deixando esta tarefa para os usuários.

3.3.2 Metodologia

A estratégia adotada emprega modelos de falhas, semelhantes aos métodos de teste físicos de circuitos, onde procura-se encontrar um estímulo que possa evidenciar um erro na saída. Por exemplo, a Figura 3.3 mostra que o cone de caminho vermelho, o qual indica uma avaria, não é visível na saída porque o seu valor gerado é mascarado pelo caminho azul.

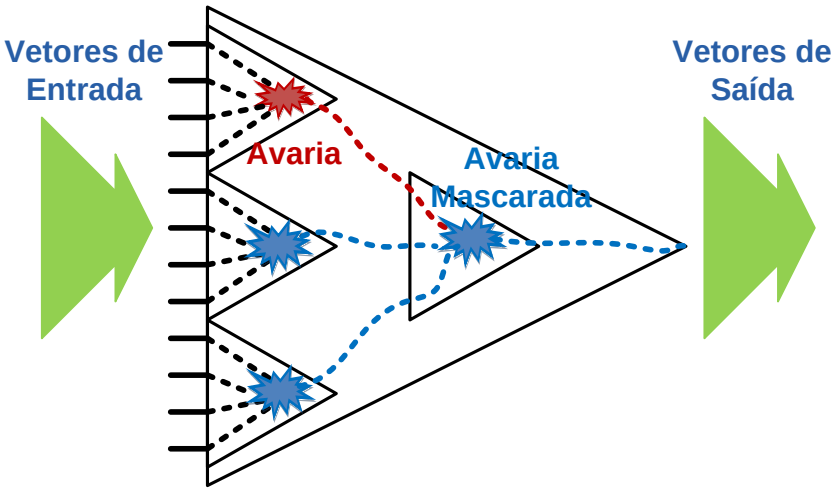


Figura 3.3: Mascaramento de Falha no Circuito

No caso do teste físico de circuito, para se determinar as falhas, tem-se propostas

diferentes técnicas, entre elas a que se conhece como D-Calculus (32). A mesma filosofia de trabalho é empregada para OCCOM, desenvolvida pelos autores (10) (33), mas neste caso a ferramenta insere um *tag* (ou marca) em variáveis a serem observadas durante o processo de simulação; tal *tag* é representado por $+\Delta$ para valores positivos e $-\Delta$ para valores negativos. A diferença do caso do teste físico de circuitos, é que o erro deve propagar-se em cada um dos fios do circuito, enquanto, neste caso, a marca deve propagar-se a cada uma das operações do código HDL. Para esta propagação, propõem-se usar os métodos de cálculo correspondentes tanto para os cálculos aritméticos e Booleanos. Este tipo de cobertura tem semelhança com a cobertura por caminhos, mencionada no capítulo 2, mas com a diferença que, neste caso, procuram-se condições em que os possíveis erros não sejam mascarados no processo de verificação, sem precisar então exercitar todos os caminhos.

3.3.3 Resultados

Em (10), dois testes são realizados:

- Teste da Métrica de Cobertura: faz-se uma comparação entre a cobertura por linha e a cobertura medida com OCCOM. Resultados mostraram, para diferentes circuitos, cada um com conjunto fixo de estímulos, um nível elevado de cobertura por linha, enquanto o modelo de cobertura OCCOM obtém uma menor cobertura, ou seja, a cobertura proposta é mais difícil de se cumprir. Tal fato indica que as métricas estruturais são mais facilmente alcançadas mas não deixam de fora a verificação de várias fontes possíveis de erro.

- Teste do Desempenho da Ferramenta: compara-se o processo de simulação tradicional com a simulação baseada em OCCOM e com a simulação especializada, proposta por Devadas em (15). Na proposta de Fallah, em (10), um simulador baseado em modelo de falhas foi desenvolvido, o qual é mais lento. De outro lado o tempo de processamento do OCCOM é também elevado em comparação aos métodos de simulação tradicionais baseados em cobertura estrutural.

3.3.4 Trabalhos correlatos

A partir dos conceitos gerados, Tasiran em (33), introduziu a metodologia de reajuste da distribuição probabilística. Neste caso, os *tags* permitem conhecer os casos de teste cujo valor foi mascarado e que podem ser fonte de erro; a partir deste conhecimento pode-se fazer uma exploração mais profunda dentro do espaço de estados e potencialmente descobrir os possíveis erros do sistema, além de acelerar o processo de verificação, evitando percorrer

estados que tenham sido bem testados.

3.3.5 Limitações da estratégia proposta

A limitação desta métrica é que ela não consiste de uma condição suficiente para determinar se um erro foi coberto ou não. Por exemplo, se dois *tags* forem gerados (um positivo e outro negativo) e eles se anularem, então gera-se um *tag* neutro Δ', porém este não se propaga corretamente. Portanto, o *tag* não seria observável na saída e OCCOM (10) poderia indicar que o erro não tenha sido coberto. Outro fator importante, é que o processamento com o *tag* gera um consumo extra de recursos computacionais, o que torna a simulação ainda mais complexa, sendo, por isto, que o método não é amplamente usado na indústria. Também, como gerador de modelo de cobertura, o método apresentado é incompleto uma vez que fica por conta do engenheiro de verificação, de acordo com a sua experiência, selecionar as variáveis a serem observadas.

3.4 Trabalho de Verma, Harris e Ramineni, ano 2007

3.4.1 Objetivo

O trabalho de Verma (9) tem como objetivo apresentar uma metodologia para a geração de um modelo de cobertura pela análise estática do código HDL do DUV. Isto permite obter os sinais e as dependências entre cada um deles, já que estes não são descritas no documento de especificação. A partir desta análise pode-se gerar os grupos de cobertura que determinam as propriedades que devem ser cumpridas durante o processo de verificação. Na prática, gera-se um conjunto de conjunto de condições em forma de asserções que devem ser respeitadas durante a simulação.

3.4.2 Metodologia

Para ilustrar a metodologia proposta, o exemplo da Figura 3.4 apresenta um circuito descrito por código Verilog, o qual consiste de três sinais Booleanos (a, b, wr_en). A especificação requer: "se o sinal 'a' é '1' então o sinal 'b' deve ser '1"', o que é expresso na equação 3.5 através da linguagem CTL (29).

$$AG(a \rightarrow b) \tag{3.5}$$

```
01: always @(*)
02:     begin
03:     if(a)
04:         if(wr_en)
05:             b = 1;
06:     end
```

Figura 3.4: Exemplo de Modelagem de Cobertura

Se cada um dos sinais tem o domínio em $0, 1$, então tem-se as seguintes combinações:

$$(a, b) = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$$

Mas considerando o requisito comentado anteriormente, só se podem considerar válidos os valores $\{(0, 0), (0, 1), (1, 1)\}$, uma vez que deve ser cumprida a condição "se o valor de 'a' é '1', então 'b' deve ser '1'. Portanto, se durante a simulação não ocorrer este evento, uma mensagem de erro deve ser gerada. Com estes dados, pode-se gerar dois grupos para este processo:

- Pontos Bons: Conformado pelas condições válidas que devem ser cumpridas durante a simulação.
- Pontos Ruins: Condições que nunca devem ocorrer durante a simulação.

Para a extração dos pontos a serem cobertos uma ferramenta, cuja arquitetura se mostra na Figura 3.5, foi desenvolvida. Sua primeira etapa é um parser para HDL, o qual se encarrega de gerar uma representação intermediária contendo os sinais e outros parâmetros do HDL. A segunda etapa é a geração do Grafo de Fluxo de Dados e Controle (do inglês, Control and Data Flow Graph, CDFG) do circuito; , por exemplo, para o código Verilog mostrado na Figura 3.6, gera-se o CDFG correspondente na Figura 3.7.

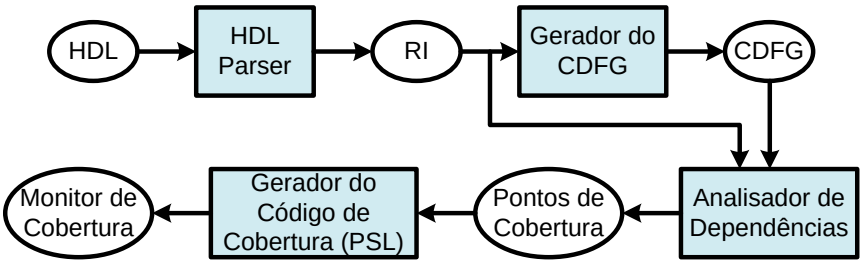


Figura 3.5: Arquitetura de Extrator de Cobertura

Com o CDFG, a ferramenta faz uma análise de dependências entre cada um dos sinais, baseado no algoritmo apresentado em (9). Este obtém os grupos conformados pelos pontos que são considerados bons ou ruins. Com esta informação gera-se um código de cobertura

```
always@(*)
    begin
    b = 1;
    if (a)
        if (wr_en)
            b = 2;
    b = 3;
    end
```

Figura 3.6: Processo no Código Verilog

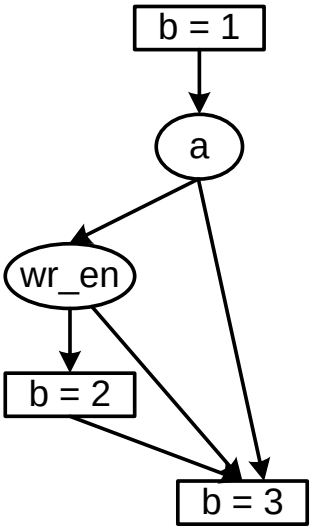


Figura 3.7: CDFG do Processo mostrado na Figura 3.6

em linguagem PSL, como mostrado na Figura 3.8. Este conjunto de condições pode ser interpretado, de fato, como asserções que devem ser respeitadas.

```
coverage_group example {
    sample_event = @ (posedge CLOCK);
    sample a, wr_en, q_{b,p1}, q_{b,p2};
    cross func_cov (a, wr_en, q_{b,p1}, q_{b,p2}){
        state cvg_1 ((a == 1)&&(wr_en == 1)&&(q_{b,p1}==1));
        state cvg_2 ((a == 1)&&(wr_en == 0)&&(q_{b,p2}==1));
        state cvg_1 ((a == 0)&&(wr_en == 0)&&(q_{b,p2}==1));
        bad_state fault_1 ((a == 1)&&(wr_en == 1)&&(q_{b,p1}==0));
        bad_state fault_2 ((a == 1)&&(wr_en == 0)&&(q_{b,p2}==0));
        bad_state fault_3 ((a == 0)&&(wr_en == 0)&&(q_{b,p2}==0));
          }
    }
```

Figura 3.8: Código de Cobertura

3.4.3 Resultados

Para mostrar a efetividade do método proposto, Verma et al. fizeram uma avaliação da cobertura funcional para cada um dos 20 benchmarks do processador DLX com falhas injetadas. Para o método proposto por Verma, a ferramenta gerou um total de 7887 pontos de cobertura (5349 pontos bons e 2538 pontos ruins), o qual representa $<< 1\%$ do total de pontos do espaço. Tais pontos foram usados para a comparação do avanço da Cobertura de Erros (erros observados). Os testes realizados mostraram uma proximidade entre o número de vetores injetados e o número de erros detectados, devido à boa qualidade do modelo gerado.

3.4.4 Limitações da estratégia proposta

Este tipo de método torna-se mais demorada quanto maior for o número de sinais do sistema porque cria um maior número de pontos de cobertura a serem cobertos, levando a um maior custo computacional.

Por outro lado, deve-se ter em conta que a representação com CDFG é uma representação limitada, o qual só representa um processo simples e dependente do código HDL original, sendo que ainda não está bem resolvido o tratamento de processos gerados por múltiplos CDFGs.

4 Modelagem e cômputo da cobertura

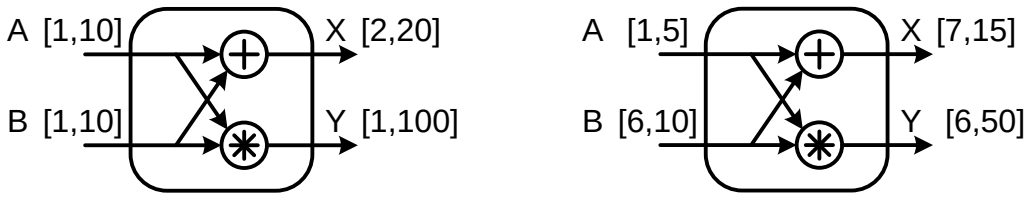
Um dos aspectos discutidos na seção 1.1 (Capítulo 1, página 3) é a modelagem da cobertura funcional, o qual depende muito da experiência e conhecimentos do engenheiro verificação. Não existe uma maneira formal amplamente aceita de se definir a cobertura, o que acaba por se refletir em deficiências em modelos obtidos manualmente. Neste capítulo, são discutidos os problemas que podem surgir na geração da cobertura de entrada e como estes afetam a cobertura de saída (foco desta dissertação). Com as observações apresentadas, estabelece-se a metodologia utilizada para gerar o modelo da cobertura de saída assim como o tratamento teórico deste.

4.1 O modelo de cobertura

4.1.1 A necessidade de cobertura de entrada e saída

Um dos maiores problemas no plano de verificação é a definição do modelo de cobertura. No seção 2.3.3.1(capítulo 2, página 14), o modelo por itens foi definido como uma das formas mais utilizadas de cobertura, sendo os itens associados a parâmetros do sistema sob verificação os quais representam as propriedades do sistema. Estes parâmetros são estabelecidos tanto para as entradas como para as saídas do circuito: a modelagem de cobertura para os primeiros dá-se por uma correspondência direta com a forma de geração de estímulos, enquanto a modelagem dos últimos é difícil, uma vez que dependem dos primeiros, existindo uma relação entrada-saída, cujo estabelecimento é complexo.

Na modelagem dos parâmetros de entrada, deve-se definir, num primeiro momento, o espaço funcional que se deseja observar e, assim, definir os vetores que estimulem as zonas que o compõem. Na maioria de casos, utiliza-se a geração de vetores aleatórios com restrições (em inglês, constrained random generation) definida através das especificações. Pode-se adotar formas mais automatizadas como a proposta por Castro et al. (34), mas uma discussão mais aprofundada sobre elas é irrelevante irrelevantes no contexto deste



(a) Circuito sem Restrições (b) Circuito com Restrições

Figura 4.1: Exemplo de Modelamento de Cobertura

trabalho.

Diferentes definições do espaço a ser observado podem gerar variadas possíveis respostas à saída. Considere-se, por exemplo, um circuito composto por um somador e um multiplicador, como mostrado na figura 4.1(a). Originalmente o intervalo possível de valores de entrada de A é $[1, 10]$ e de B, $[1, 10]$, o que gera um grupo de valores que estão no intervalo $[2, 20]$ para o caso de X e $[1, 100]$ no caso de Y.

Estabelecido o intervalo de entrada, o engenheiro de verificação projeta a cobertura de saída, podendo basear-se no número de bits que a saída tem ou, no caso de valores, determinar aqueles que são possíveis de ocorrer. No caso do exemplo apresentado, sabendo-se que os possíveis valores de saída estão no intervalo $[2, 20]$ para o caso da saída X e $[1, 100]$ para a saída Y, pode-se estabelecer que se deseja observá-los, distribuindo-os em cinco intervalos de aproximadamente igual dimensão:

1. Para o espaço amostral de X: $[2, 4]$, $[5, 8]$, $[9, 12]$, $[13, 16]$, $[17, 20]$.

2. Para o espaço amostral de Y: $[1, 20]$, $[21, 40]$, $[41, 60]$, $[61, 80]$, $[81, 100]$.

Por outro lado, vamos supor que existam restrições quanto às entradas que são desconhecidas do engenheiro de verificação; por exemplo, se a entrada A estiver em $[1, 5]$ e a entrada B terá que estar em $[6, 10]$ (e vice-versa), como mostrado na Figura 4.1(b). Desta forma, alguns dos eventos estabelecidos não serão observados, gerando um novo de espaço de valores possíveis na saída. Basicamente, para a saída X não será observado nenhum item os intervalos $[2, 4]$ e $[17, 20]$ e, no caso de Y, os eventos que não serão observados estarão definidos pelos intervalos $[61, 80]$ e $[81, 100]$. Portanto a contagem de itens associados a estes intervalos nunca terá um valor maior que zero e a progressão da cobertura ficaria estagnada a partir de certo número de vetores (Figura 4.2).

Dada a dificuldade em se definir um modelo de cobertura de saída, é comum que, em métodos manuais, se adote soluções simplificadas; uma solução trivial utilizada é modelar a cobertura com uma distribuição uniforme (número igual de amostras ou itens) para cada evento. Nos circuitos reais é pouco provável que a saída tenha uma distribuição uniforme,

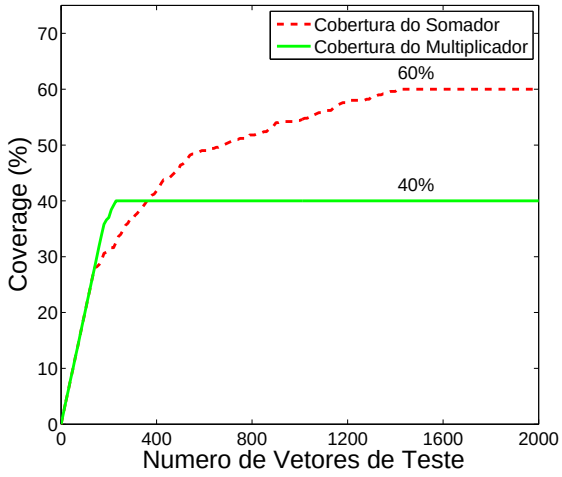
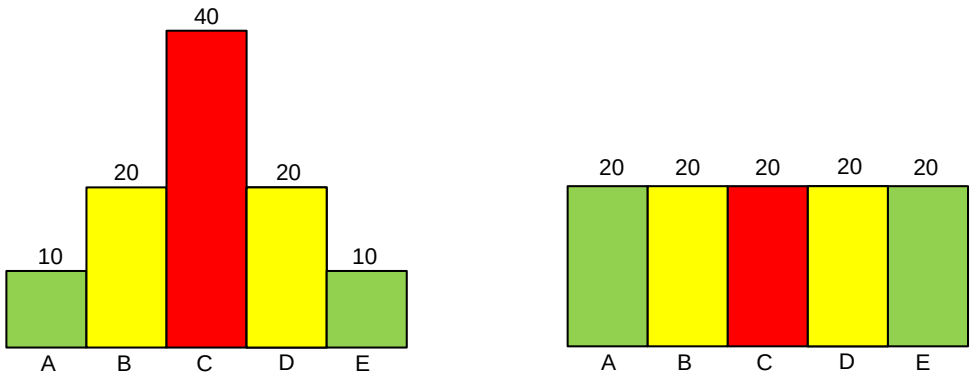


Figura 4.2: Progressão da cobertura das saídas para o exemplo mostrado na figura 4.1(b)



(a) Real (b) Projetado pelo Eng.

Figura 4.3: Distribuição de saída estabelecida para o circuito

mesmo que para as entradas este tipo de distribuição seja definida, afetando-se assim o tempo que se precisa para avaliar o sistema (atingir 100% de cobertura). Vamos supor que circuito tenha uma distribuição de saída igual à apresentada na figura 4.3(a), mas o engenheiro de verificação tenha definido uma distribuição uniforme (Figura 4.3(b)) para o modelo de cobertura. Na simulação (real), vamos assumir que a geração do número de vetores $N = 100$ permitirá o perfil de saída da primeira figura e precisará de um tempo τ para o processamento. Porém, pelas métricas de cobertura estabelecidas, só 80% desta foi alcançada, uma vez que os eventos F e J só terão sido atendidos em 50%, enquanto G, H e I terão já atingido 100% de cobertura individual (na verdade, o evento H terá muitas ocorrências acima do estabelecido pela cobertura uniforme). No processo de verificação, espera-se alcançar 100% da cobertura total, porém para se conseguir este objetivo haveria a necessidade de um tempo 2τ (Figura 4.4).

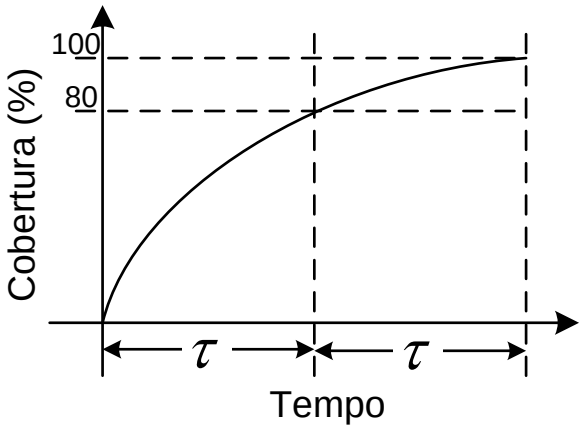


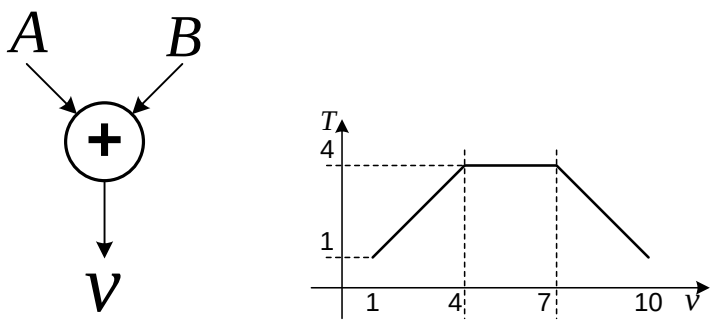
Figura 4.4: Exemplo de progressão da Cobertura de Saída

4.1.2 Funções probabilísticas para antecipar perfil de saída

Nesta seção iniciamos a discussão sobre o emprego de cálculos matemáticos no desenvolvimento de uma metodologia para a obtenção da distribuição de saída de um circuito. A base matemática sobre o qual repousa este trabalho consiste de noções de probabilidade e variáveis estocásticas, que serão utilizadas em seções seguintes. Não é propósito desta dissertação discorrer sobre tais noções, uma vez que há para eles uma extensa bibliografia como em (35), (36) e (37); entretanto, para breve referência, apresentamos algumas definições básicas no apêndice A.

Os parâmetros e sua distribuição no modelo de cobertura de saída são muitas vezes decorrentes de decisões de projetistas ante às especificações do sistema, porém estas devem estar balizadas pela distribuição real dos itens, o que é dependente do sistema e da distribuição dos estímulos à entrada. O exemplo da sub-seção anterior evidencia uma possível ineficiência quando o modelo de cobertura de saída é definida de forma arbitrária. Para evitar este tipo de situação deve-se conhecer (modelar) qual é a relação entrada-saída. Para isto, pode-se tratar o problema baseando-se em probabilidade de ocorrência de valores. Por exemplo, ao se observar o comportamento de um somador (Figura 4.5(a)) e estimulá-lo por todas as combinações possíveis de suas entradas nos intervalos $A \in [0, 6]$ e $B \in [1, 4]$, pode-se obter uma resposta similar à mostrada na figura 4.5(b), onde o valor do eixo vertical T representa o número de vezes que o valor de saída V (eixo horizontal) se repete. Este número de repetições representa a possibilidade de que o valor de saída aconteça. Pode-se aplicar métodos matemáticos probabilísticos para poder determinar a distribuição de saída, sem precisar fazer um cômputo um a um.

Em cálculo probabilístico, variáveis aleatórias podem ser de tipo discreto ou contínuo. A função de densidade probabilística discrimina a probabilidade individual de cada valor



(a) Bloco Somador (b) Distribuição da Soma

Figura 4.5: Exemplo de Somador e Distribuição de possíveis valores

válido de uma variável aleatória. A figura4.5(b) apresenta um perfil correspondente à da função densidade resultante da soma (considerado que a densidade das entradas seja constante), tanto para variáveis discretas ou contínuas. Por outro lado, no caso de uma operação de multiplicação usando variáveis discretas, os possíveis valores de saída estão espalhados e, por não seguirem um padrão mais específico e sequencial, como no caso da soma, dificultam a computação eficiente da densidade de saída. Para solucionar este inconveniente, será assumido que os intervalos são sempre contínuos.

Apesar da função densidade descrever adequadamente a probabilidade de eventos elementares, a sua manipulação não é adequada no contexto deste trabalho uma vez que, como mostrado na figura 4.3(a), o modelo de cobertura é composto por eventos de observação agrupados. São conjuntos agrupados de eventos elementares, definindo-se assim conjuntos de intervalos de valores (de variáveis aleatórias). Como o interesse é pela probabilidade conjunta de um evento, é mais adequada a manipulação da função de distribuição de probabilidade ou, por simplicidade, distribuição de probabilidade (PD, do inglês Probability Distribution). Formalmente a PD é representada por $F(x)$, para o intervalo $(-\infty, x]$, que corresponde probabilidade acumulada até x; para o nosso caso, definimos para um dado intervalo $[a, b]$, a PD de evento, calculada como $F(b) - F(a)$.

4.1.3 Representação intermédia - FSMD

Como observado na seção 4.1.1, conhecendo-se o modelo da operação que realiza o circuito (exemplo da figura 4.5) pode-se calcular a distribuição da resposta de saída. Deve-se ter um modelo do circuito evidenciando tais operações e, no contexto da verificação funcional, ele pode ser obtido do modelo de referência do testbench (seção 2.2.2, capítulo 2).

O modelo de referência é uma representação em alto nível de abstração do circuito desenvolvido, não contendo propriedades de tempo, na maioria de casos. Para a determinação

da cobertura de saída a partir do modelo do circuito, deve-se entender a linguagem que descreve seu comportamento. Em princípio qualquer linguagem de descrição de hardware é aceitável no âmbito desta dissertação, mas segue-se a tendência adotada no Grupo SEIS do LME-USP de usar modelos de referência em HDL SystemC (38).

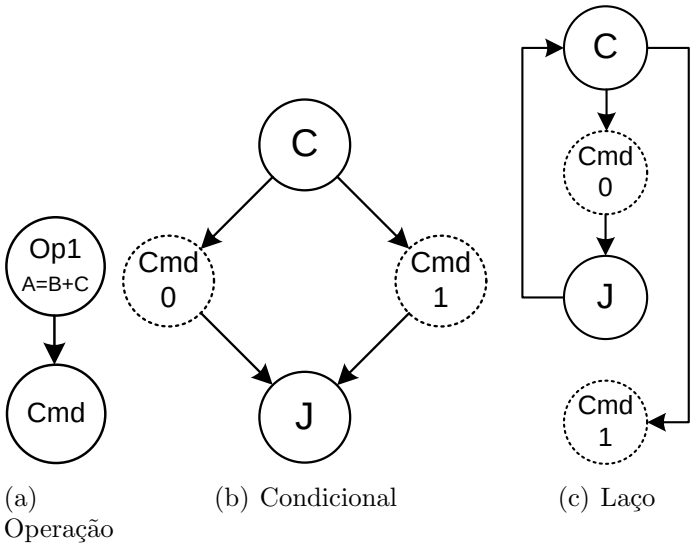
Os comandos a serem considerados apresentam propriedades específicas a serem consideradas no momento da análise. Estas propriedades podem ser divididas em dois grupos:

1. Comandos de execução, que contemplam operações:
   (a) Aritméticas ($+,-,*,/,\%$). Por exemplo: $s = a + b$.
   (b) Lógicas ($\neg,\wedge,\vee$). Por exemplo: $s = a \vee b$.
2. Comandos de Controle que permitem tomadas de decisão baseadas em sentenças com operações relacionais ($=,\neq,<,\leq,>,\geq$), como por exemplo, $a > b$. Tratamos aqui de dois tipos:
   (a) Condicionais: determinam que ramo de execução seguir dada uma condição. Por exemplo: *if/else; switch/case/default.*
   (b) Laços: executam um grupo de linhas de código por um número determinado de vezes. Por exemplo: *for; while; do/while.*

Nesta dissertação, por limitação de tempo reduzimos o escopo da análise, estabelecendo uma série de restrições como aceitar apenas as operações aritméticas ou usar apenas o laço determinístico, o que será detalhado mais à frente.

Por praticidade formalizamos a representação do circuito em forma equivalente de uma máquina de estados finitos com dados (FSMD,do inglês, Finite State Machine with Data) como apresentada em (39) e (40). A FSMD é uma representação sequencial de um algoritmo, próxima ao estilo sequencial de programação, permitindo a representação de cada comando de execução e de controle, além de permitir a representação dos dados de entrada e saída de cada operação. Diferente de uma máquina de estados finitos (FSM, do inglês Finite State Machine) que só emprega o valor do estado atual e suas entradas para determinar o valor de saída e o estado seguinte, a FMSD utiliza variáveis internas para determiná-los. Uma FSMD, na forma definida em (40) pode englobar os três tipos de comandos antes mencionados: execução (designação de valores), controle condicional e controles de laço.

Para o comando de execução é definido um estado (nó) com uma operação aritmética/Booleana associada (ação a ser realizada neste estado). A partir deste estado, há uma transição (arco) ao comando seguinte. Este caso é apresentado na figura 4.6(a).



(a) Operação (b) Condicional (c) Laço

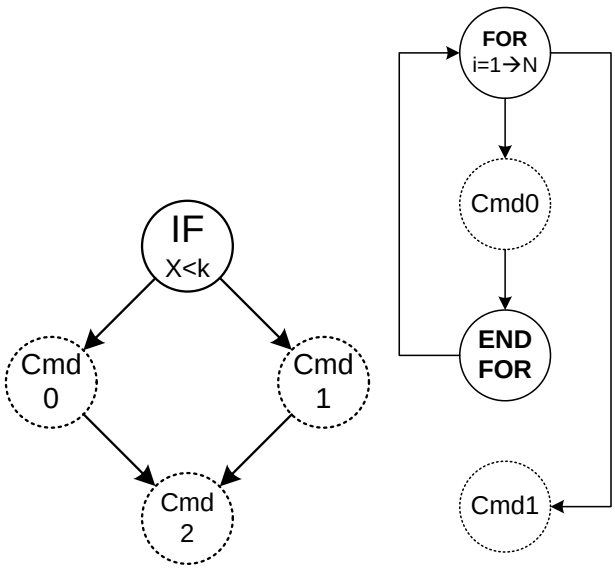
Figura 4.6: Representação em grafos (FSMD) de comandos

No caso de comandos condicionais, é definido um estado que contém a condição (C) a ser analisada e o estado de união (J). O primeiro não realiza designação de valores, mas determina o caminho que o fluxo de algoritmo associado deve seguir. O segundo une os possíveis caminhos gerados pela condição, indicando os estados comuns aos quais estes seguem. A figura 4.6(b) mostra como estes nós interagem entre si, com cada um dos caminhos associado a um conjunto de estados. Para esta dissertação, o nó de condição será especificado na forma $IF(X < k)$, como mostrado na figura 4.7(a), onde X é a variável a analisar e k uma constante sobre a qual X vai ser avaliado. Pode-se observar que o nó de união (J) não é necessário, na prática, e pode ser descartado, uma vez que este não realiza tarefa efetiva na FSMD.

No caso de laços, também são gerados dois nós, 'C' e 'J', mas diferente do caso anterior, o nó união (J) retorna ao nó condição ao finalizar a série de comandos que estes contém, com o propósito de repeti-los um número determinado de vezes como mostrado na figura 4.6(c). A estrutura segue o padrão de uma função *while* de C/C++.

No caso desta dissertação, por simplificação, trataremos de laços determinísticos com comandos do tipo *for* (ex. *for (i=0; i ¡ k; i=i+1)*), onde se conhece de antemão o número de vezes que os comandos se repetem. Este tipo de laço contém quatro etapas:

1. Inicializa Contador ($i = 0$)
2. Condição ($i < k$)
3. Incrementa/Decrementa contador ($i = i + 1$)
4. Retorna a condição



(a) Condicional (b) Laço

Figura 4.7: Representação específica em grafos dos comandos de controle na dissertação

Para manter a estrutura similar ao caso apresentado por Vahid et al. em (40), as etapas de inicialização e da condição vão ser enquadradas em um mesmo nó, como mostrado na figura 4.7(b). As etapas de incremento (ou decremento) e de retorno também estarão em um no mesmo nó, de união 'J' (ou ENDFOR).

Para maiores detalhes, a apresentação formal do conceito da FSMD pode ser vista em (39) (40).

4.2 Cômputo da distribuição de probabilidade de uma operação aritmética

Como mencionado anteriormente, assumimos que a modelagem da Distribuição de Probabilidade em operações aritméticas é realizada para intervalos contínuos. Tal medida permite uma facilidade de cálculo, utilizando a equação 4.1 para o caso onde a operação aritmética dependa somente de uma variável (onde $v = g(x)$ e $g^{'}(x) = \frac{\partial g(x)}{\partial x}$) e 4.2 quando depender de duas (considerando-se que as variáveis são independentes[1], $v = g(x, y)$). Ambas foram extraídas do livro de Papoulis (36) e são demostradas no apêndice A.

$$F(v) = \int_{-\infty}^{\infty} \frac{f_x}{|g^{'}(x)|} \mathrm{d}x \tag{4.1}$$

[1]A mudança de valor de uma variável não interfere no valor da outra para nenhum valor do espaço amostral onde estes estão definidos

$$F(v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_x(x) f_y(y) \mathrm{d}x\mathrm{d}y \tag{4.2}$$

onde ficam definidos:

X e Y : Variáveis aleatórias de entrada

V : Variável aleatória de saída

Ω_x e Ω_y : Espaço de eventos das entradas

Ω_v : Espaço de eventos da saída

x : Valor específico dentro do domínio estabelecido pela variável aleatória X.

y : Valor específico dentro do domínio estabelecido pela variável aleatória Y.

v : Valor específico dentro do domínio estabelecido pela variável aleatória V.

A partir dos aspectos comentados anteriormente, o cômputo da Distribuição de Probabilidade dos operadores matemáticos, soma $(+)$ e multiplicação $(*)$, é tratado. A divisão $(/)$ não é tratada; portanto, apenas um caso particular pode ser considerado (de fato uma multiplicação), no qual o divisor deve ser uma constante. Por outro lado, a subtração $(-)$ tem as mesmas propriedades que a soma, portanto a sua explicação já é implícita quando do tratamento do caso da soma.

Para todos os casos considera-se que a distribuição de cada uma das variáveis é uniforme a fim se simplificar os cálculos. A distribuição de saída, no entanto, é provavelmente não uniforme. Um caso de distribuição não uniforme é considerado como um conjunto de eventos com diferentes probabilidades para cada um deles, mas de distribuição uniforme (igual densidade) cada um dos eventos elementares que compõe o evento. Exemplo disto é apresentado na figura 4.8.

As fórmulas a serem apresentadas nas próximas seções consideram um evento (por variável) ao se realizar o cálculo. Para calcular a PD resultante de uma operação que tem variáveis com mais de um evento, a PD resultante deve considerar a união de todos os casos possíveis.

4.2.1 Distribuição de probabilidade da soma

A operação de soma pode ser entre uma variável aleatória e uma constante ou entre duas variáveis aleatórias. Cada uma destas formas apresenta um comportamento diferente, por-

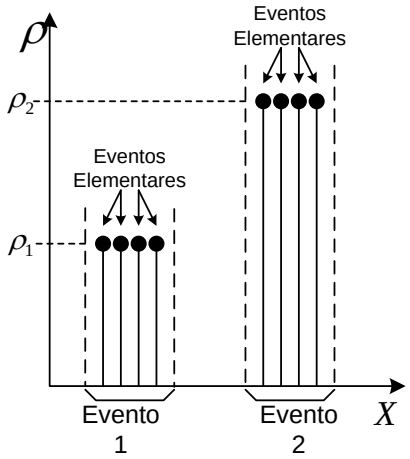


Figura 4.8: Exemplo de Distribuição não Uniforme entre eventos

tanto a função para calcular a PD muda. Os dois casos são apresentados nas seções abaixo.

4.2.1.1 Soma de uma variável aleatória e uma constante

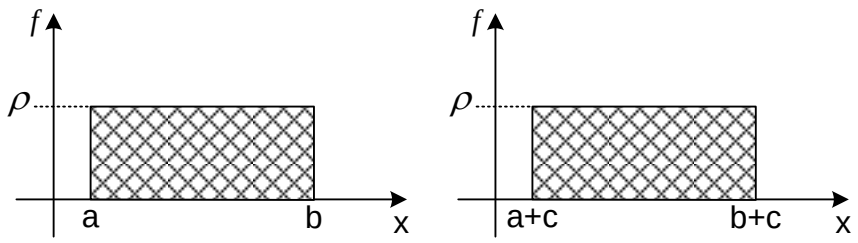
Uma constante C pode ser definida como uma variável aleatória tendo como espaço de possíveis valores um só elemento c, cuja densidade probabilística é $f_{cnst}(cnst) = \beta$, com $0 < \beta \leq 1$. Em se tratando de uma variável de entrada de um circuito, tipicamente $\beta = 1$, uma vez que a probabilidade de a constante assumir o seu valor seria absoluta. Ocorre que a constante pode derivar do resultado de uma operação qualquer, quando surge um intervalo de apenas um único valor; neste caso a probabilidade dependerá do resultado daquela operação. Por outro lado, se uma variável aleatória X, cujos eventos estão definidos no espaço $\Omega_x = [L_x, H_x]$ (L_x = Limite inferior do evento da variável X e H_x = Limite superior do evento da variável X), de densidade constante e uniforme igual a $f_x(x) = \alpha$, onde $0 < \alpha \leq \frac{1}{H_x - L_x}$, for operada com C ($V = C + X$), então define -se o novo espaço:

$$\Omega_v = [L_v, H_v] = [L_x + c, H_x + c] \tag{4.3}$$

A função distribuição é calculada pela equação 4.4.

$$F(v) = \begin{cases} 0 & \text{se } v \leq L_v \\ \alpha\beta(v - L_v) & \text{se } L_v < v < H_v \\ \alpha\beta(H_v - L_v) & \text{se } H_v \leq v \end{cases} \tag{4.4}$$

Na figura 4.9, é apresentado um exemplo onde $f_c = f_{cnst}(cnst) = \beta = 1$. Na figura 4.9(a), é mostrada a densidade de probabilidade da variável X enquanto a figura 4.9(b) mostra a densidade da variável resultante V, onde $V = X + c$. Pode-se observar que a



(a) Densidade de Probabilidade de X (b) Densidade de Probabilidade de $V = X + c$

Figura 4.9: Exemplo da Densidade Probabilidade no caso da soma de uma variável aleatória e uma constante

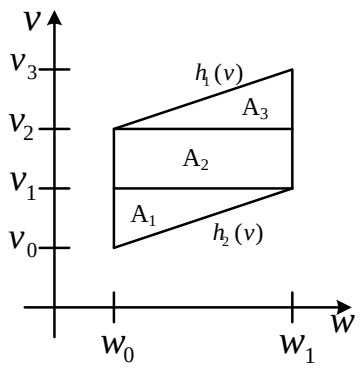


Figura 4.10: Áreas de Integração para o computo da PD da soma

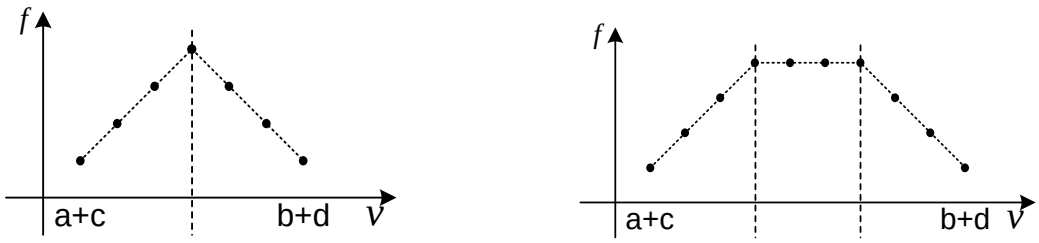
forma original é mantida e o que muda são os intervalos onde está definida a variável. Além disso, caso $\rho = \alpha = \frac{1}{H_x - L_x}$, teríamos $F(v^{'}) = 1$ para $H_v \leq v^{'}$.

4.2.1.2 Soma de duas variáveis aleatórias independentes

As equações propostas nesta seção consideram variáveis aleatórias independentes, ou seja, $P(X|Y) = P(X)$ e $P(Y|X) = P(Y)$, onde $P(A)$ denota a probabilidade de ocorrência do evento A e $P(A|B)$, a probabilidade de ocorrência do evento A dado que B ocorra. Para o computo da PD, emprega-se a equação 4.2, onde duas variáveis aleatórias independentes $X \in \Omega_x = [a, b]$ e $Y \in \Omega_y = [c, d]$ são consideradas. A partir delas determina-se o gráfico da figura 4.10, na qual o eixo vertical V representa todos os valores possíveis que a operação $X + Y$ pode adquirir e W representa uma das variáveis de entrada (X ou Y). Nesta figura aparecem três áreas $A1$, $A2$ e $A3$. Estas definem os limites da integral da equação 4.2 (página 42) para o cálculo da PD ($F(v)$) da soma, representada na equação 4.5. Nela, além das variáveis mencionadas, aparecem w_2 e, que representam os limites inferiores de X ou Y, ao considerar W como Y ou X, respectivamente. Ademais, pode-se observar a presença dos valores v_0, v_1, v_2 e v_3, resultantes da operação entre os limites de X e Y, e que correspondem aos pontos no eixo X que definem o comportamento de $F(v)$.

Tabela 4.1: Condições no Gráfico da Figura 4.10 no caso da Soma

Caso $(b-a) < (d-c)$	**Caso $(d-c) < (b-a)$**
$W = X$	$W = Y$
$w_0 = a$	$w_0 = c$
$w_1 = b$	$w_1 = d$
$w_2 = c$	$w_2 = a$
$w_3 = d$	$w_3 = b$
$v_0 = a + c$	$v_0 = a + c$
$v_1 = b + c$	$v_1 = a + d$
$v_2 = a + d$	$v_2 = b + c$
$v_3 = b + d$	$v_3 = b + d$



(a) Distribuição Triangular (b) Distribuição Trapezoidal

Figura 4.11: Padrão da Densidade de Probabilidade da soma de variáveis de distribuição uniforme

$$F(v) = \begin{cases} 0 & \text{se } v < v_0 \\ F_1(v) = \alpha\beta\dfrac{(v-v_0)^2}{2} & \text{se } v_0 < v \leq v_1 \\ F_2(v) = \alpha\beta(w_1 - w_0)(v - z_1) + F_1(v_1) & \text{se } v_1 < v \leq v_2 \\ F_3(v) = \alpha\beta\left[v_3 v - \dfrac{v^2}{2} - v_3 v_2 + \dfrac{v_2^2}{2})\right] + F_2(v_2) & \text{se } v_2 < v \leq v_3 \\ \alpha\beta(w_1 - w_0)(w_3 - w_2) & \text{se } v_3 < v \end{cases} \tag{4.5}$$

Os valores de v_0, v_1, v_2, v_3, w_0, w_1, w_2 e w_3 vão depender da relação estabelecida pelos limites que definem os intervalos de X e Y para o cálculo da $F(v)$. Esta relação está dada por $(b-a) < (d-c)$, que define os valores mostrados na tabela 4.1.

A distribuição da soma segue um determinado padrão quando as entradas são independentes e distribuídos uniformemente, como mostrado na figura 4.11. Considerando $X \in [a, b]$ e $Y \in [c, d]$, caso $(a + d) = (b + c)$, a forma da função de densidade de probabilidade da função $V = X+Y$ é igual à mostrada na figura 4.11(a). No caso de $(a+d) \neq (b+c)$, então a função densidade da resposta seria a da mostrada na figura 4.11(b).

4.2.2 Densidade de probabilidade da multiplicação

A operação de multiplicação pode acontecer em três casos diferentes:

1. Uma variável aleatória é multiplicada por uma constante cX.
2. Multiplicação entre duas variáveis aleatórias XY.
3. Uma variável aleatória é multiplicada por ela mesma n vezes, $X * X * \cdots = X^n$.

Para mostrar o comportamento da PD em cada um dos casos, esta seção divide-se em três partes, apresentadas a seguir.

4.2.2.1 Multiplicação de uma variável aleatória com uma constante

Uma constante C pode ser definida como uma varável aleatória que tem como espaço de possíveis valores um só elemento c ($\Omega_c = \{c\}$) e com uma distribuição probabilística $f_c(c) = \beta$. Por outro lado, se uma variável aleatória X, que tem como espaço de eventos a $\Omega_x = [L_x, H_x]$ de distribuição uniforme igual à $f_x(x) = \alpha$, onde $0 < \alpha \leq \frac{1}{H_x - L_x}$, for operada com C ($V = CX$), então define-se:

$$\begin{cases} \begin{cases} L_v = cL_x \\ H_v = cH_x \end{cases} & \text{se } 0 \leq c \\ \begin{cases} L_v = cH_x \\ H_v = cL_x \end{cases} & \text{se } c < 0 \end{cases} \tag{4.6}$$

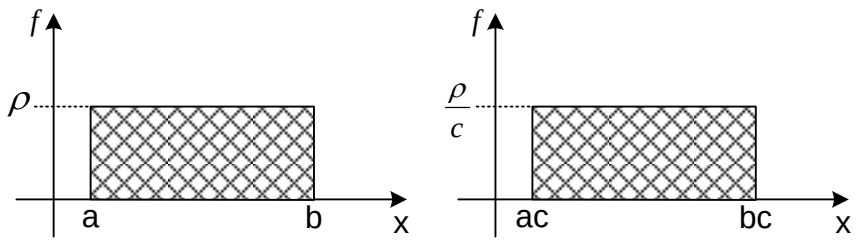
sendo a distribuição, usando a equação 4.1, dada por:

$$f_v(v) = \frac{f_x(\frac{v}{c})}{|c|} \tag{4.7}$$

Portanto, a PD para V é:

$$F(v) = \begin{cases} 0 & \text{se } v \leq L_v \\ \dfrac{\alpha\beta}{|c|}(v - L_v) & \text{se } L_v << H_v \\ \dfrac{\alpha\beta}{|c|}(H_v - L_v) & \text{se } H_v \leq v \end{cases} \tag{4.8}$$

Na figura 4.12, é apresentado o exemplo o caso $0 \leq c$. Na figura 4.12(a), é mostrada a densidade de probabilidade da variável X, onde $X \in [a, b]$. A figura 4.12(b) mostra a



(a) Densidade de Probabilidade de X (b) Densidade de Probabilidade de cX

Figura 4.12: Exemplo da Densidade Probabilidade no caso da multiplicação de uma variável aleatória e uma constante

densidade da variável V, onde $V = cX$. Pode-se observar que o valor da densidade muda para $\frac{\rho}{c}$ e os intervalos para $V \in [ca, cb]$.

4.2.2.2 Multiplicação de duas variáveis aleatórias independentes

Similar ao caso da soma, para fazer o cômputo da distribuição de probabilidade da multiplicação de duas variáveis aleatórias independentes utiliza-se a equação 4.2. Por ela, determina-se o gráfico da figura 4.10 (similar ao caso da seção 4.2.1.2, na página 44), na qual o eixo vertical V representa todos os valores possíveis que a operação XY pode adquirir e W representa uma das variáveis de entrada (X ou Y). Pode-se observar no grafo, as funções $h_1(v)$ e $h_2(v)$, que representam as declividades das retas entre V_2-V_3 e V_0-V_1 respectivamente. Tais retas definem os limites das integrais da equação 4.2.

O desenvolvimento da equação acima leva à função que define a PD, representada pela equação 4.9, da página 47.

$$F(v) = \begin{cases} 0 & \text{se } v < v_0 \\ F_1(v) = \alpha\beta\left[v\ln\left(\frac{h_2(v)}{w_0}\right) - v - v_0\ln\left(\frac{h_2(v_0)}{w_0}\right) + v_0\right] & \text{se } v_0 < v \leq v_1 \\ F_2(v) = \alpha\beta\,(v - v_1)\ln\left(\frac{w_1}{w_0}\right) + F_1(v_1) & \text{se } v_1 < v \leq v_2 \\ F_3(v) = \alpha\beta\left[v_2\ln\left(\frac{h_1(v_2)}{w_1}\right) - v_2 - v\ln\left(\frac{h_1(v)}{w_1}\right) + v\right] + F_2(v_2) & \text{se } v_2 < v \leq v_3 \\ \alpha\beta(w_1 - w_0)(w_3 - w_2) & \text{se } v_3 < v \end{cases} \tag{4.9}$$

onde

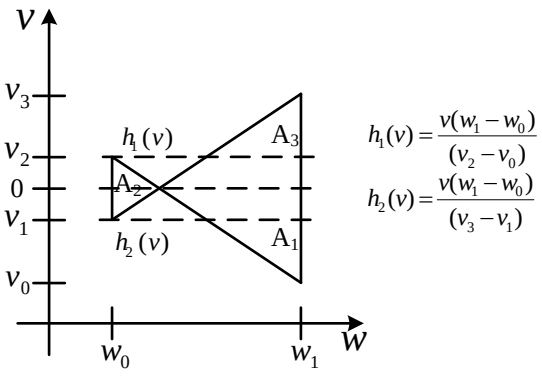


Figura 4.13: Áreas de Integração para o caso $L_x \in \mathbb{R}^- \wedge L_y \in \mathbb{R}^- \wedge H_x \in \mathbb{R}^+ \wedge H_y \in \mathbb{R}^+$ e $L_yH_x < L_xH_y$

α : Distribuição da variável X.

β : Distribuição da variável Y.

w_0 : Limite inferior do Intervalo do L_x.

w_1 : Limite superior do Intervalo do H_x.

w_2 : Limite inferior do Intervalo do L_y.

w_3 : Limite superior do Intervalo do H_y.

v_0 : L_xL_y

v_1 : H_xL_y

v_2 : L_xH_y

v_3 : H_xH_y

No caso em que $L_x \in \mathbb{R}^-$, $L_y \in \mathbb{R}^-$, $H_x \in \mathbb{R}^+$, $H_y \in \mathbb{R}^+$ e $L_yH_x < L_xH_y$, o gráfico resultante muda (Figura 4.13) em relação ao apresentado na figura 4.10. Diferentemente da soma, os limites de intervalo das variáveis de entrada envolvidas afeta a forma das áreas de integração, fazendo com que a função que define $F(v)$ mude. Pode-se perceber isto na figura 4.13 onde o gráfico é diferente da apresentada na figura 4.10. Com a mudança, são estabelecidos seis intervalos e portanto seis equações, onde $F(v)$ (Equação 4.10) é definida (diferente das cinco definidas para a equação 4.9). Nela, a forma $\iint$ é uma representação simplificada de $\iint \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v$, que é a equação resultante de uma traslação linear de variáveis da equação 4.2, (da página 42), mostrada em (36), para o cômputo da PD de uma multiplicação entre duas variáveis independentes.

$$F(v) = \begin{cases} 0 & \text{se } 0 < v \leq v_0 \\ F_1(v) = \int_{v_0}^{v} \int_{h_1(v)}^{w_1} & \text{se } v_0 < v \leq v_1 \\ F_2(v) = \int_{v_0}^{v} \int_{h_1(v)}^{w_1} + \int_{v_1}^{v} \int_{w_0}^{h_2(v)} & \text{se } v_1 < v \leq 0 \\ F_3(v) = \int_{0}^{v} \int_{w_0}^{h_1(v)} + \int_{0}^{v} \int_{h_2(v)}^{w_1} + F_2(0) & \text{se } 0 < v \leq v_2 \\ F_4(v) = \int_{0}^{v_2} \int_{w_0}^{h_1(v)} + \int_{0}^{v} \int_{h_2(v)}^{w_1} + F_2(0) & \text{se } v_2 < v \leq v_3 \\ \alpha\beta(w_1 - w_0)(w_3 - w_2) & \text{se } 0 < v \leq v_0 \end{cases} \tag{4.10}$$

A demostração completa para a obtenção da distribuição de todos os casos encontrados na multiplicação entre duas variáveis aleatórias independentes pode ser encontrada no apêndice D.

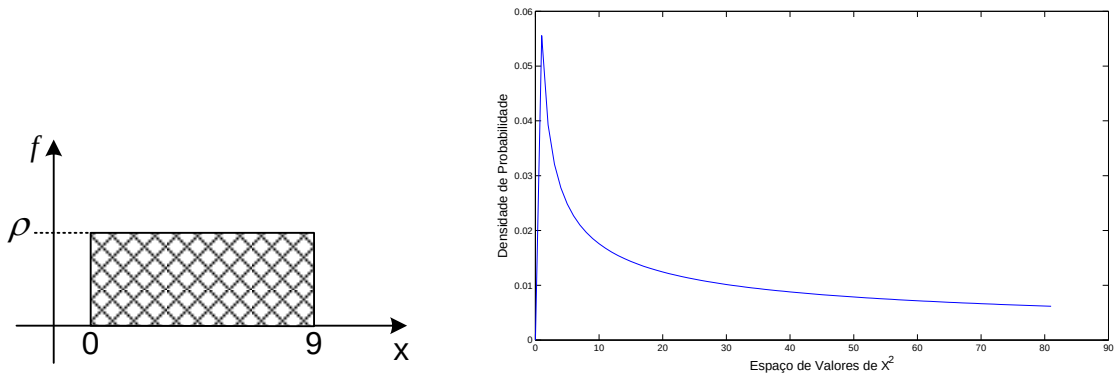
4.2.2.3 Obtenção da PD da potência n de uma variável aleatória (X^n)

Para a obtenção da $F_v(v') = \int f_v(v)\mathrm{d}v$ quando $v = g(x)$, sendo x um valor definido no espaço da variável aleatória X, emprega-se a equação 4.1 (da página 41), utilizada para calcular a PD de uma função que depende somente de uma variável aleatória.

Neste caso $v = g(x) = x^n \rightarrow x = v^{\frac{1}{n}}$ e portanto $g'(x) = nx^{n-1} = nv^{\frac{n-1}{n}}$. A expressão resultante é mostrada abaixo e a função que computa a PD ($F_v(v') = \int f_v(v)\mathrm{d}v$) é apresentada na equação 4.11, onde $v_0 = L_x^n$.

$$F_v(v) = \begin{cases} 0 & \text{se } v \leq v_0 \\ \alpha\left[v^{\frac{1}{n}} - v_0^{\frac{1}{n}}\right] & \text{se } v_0 < v < v_1 \\ \alpha(H_x - L_x) & \text{se } v_1 \leq v \end{cases} \tag{4.11}$$

Observe-se que a equação 4.11 só pode ser usada no caso de n ser ímpar. No caso de ser par, a equação resultante é mostrada na equação 4.12, contemplando duas possibilidades. A primeira quando $(L_x < 0) \wedge (0 < H_x)$ e a segunda para as demais possibilidades.



(a) Densidade de Probabilidade de X (b) Densidade de Probabilidade de X^2

Figura 4.14: Exemplo da Densidade Probabilidade no caso de X^n, sendo $n = 2$ para o exemplo

$$F(v) = \begin{cases} \begin{cases} 0 & \text{se } v < 0 \\ F_1(v) = 2\alpha\left[v^{\frac{1}{n}}\right] & \text{se } 0 < v < v_0 \\ F_2(v) = \alpha\left[v^{\frac{1}{n}} + v_0^{\frac{1}{n}}\right] & \text{se } v_0 < v < v_1 \\ \alpha(H_x - L_x) & \text{se } v_1 \leq v \end{cases} & \text{se } (L_x < 0) \wedge (0 < H_x) \\ \begin{cases} 0 & \text{se } v \leq v_0 \\ \alpha\left[v^{\frac{1}{n}} - v_0^{\frac{1}{n}}\right] & \text{se } v_0 < v < v_1 \\ \alpha(H_x - L_x) & \text{se } v_1 \leq v \end{cases} & \textit{outros} \end{cases} \tag{4.12}$$

Para ambos os casos, o valor de v_0 e v_1 é determinado pela equação:

$$\begin{cases} v_0 = L_x^n, v_1 = H_x^n & \text{se } |L_x| < |H_x| \\ v_1 = H_x^n, v_1 = L_x^n & others \end{cases} \tag{4.13}$$

Para exemplificar os aspectos apresentados nesta seção, a figura 4.14 mostra como variam a Densidade de Probabilidade da variável X (Figura 4.14(a)) e a densidade resultante da função X^2 (Figura 4.14(b)) dentro de seus espaços de valores; o primeiro de [0,9] e, o segundo, consequentemente de [0,81].

4.2.3 Considerações para o cômputo da PD

A teoria apresentada nas seções anteriores para o cálculo de PDs considera que as variáveis são contínuas. Entretanto, no contexto da verificação e na aplicação em simulação, elas são discretas em grande parte das instâncias ou, no limite, todas são discretas, se considerada

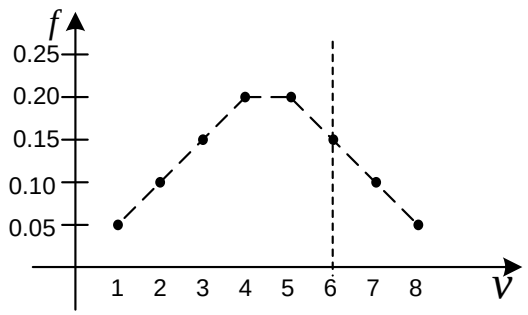


Figura 4.15: Exemplo de densidade de probabilidade da Soma

a resolução para a representação nos sistemas computadorizados de simulação. Por outro lado, as equações também consideram que cada variável contém somente um evento, o qual define o espaço de possíveis valores que a variável pode ter; na prática, entretanto, durante o processo de computação dos PDs, surgem múltiplos eventos para as variáveis envolvidas. Estas duas questões serão tratadas nesta seção.

4.2.3.1 Considerações para o cálculo de eventos de itens discretos

Como mencionado anteriormente, as equações utilizadas estão orientadas para eventos definidos por intervalos contínuos, mas devem ser adaptadas para operarem na forma discreta. No caso, pode-se "adaptar" a resposta gerada por aquelas equações para definir a probabilidade de ocorrência do evento.

O gráfico da figura 4.15 é utilizado para ilustrar o problema e os aspectos de resolução, e nele pode-se observar que a Densidade de Probabilidade da função contínua $v = g(x, y) = x + y$ é dada pelas linhas contínuas. De fato, a curva é utilizada pra representar a operação entre variáveis discretas $X \in [0, 4]$ e $Y \in [1, 4]$ e, nela pode-se observar que a probabilidade específica de $V = 6$ é igual a 0.15, mas se for calculado usando a forma contínua que propomos, a equação 4.14 se aplicaria.

$$P(v = k) = \int_{k}^{k} f_v(v)dv = 0 \tag{4.14}$$

Como o cálculo da PD contínua se faz pela probabilidade de ocorrência em intervalos de valores, o resultado obtido é 0 quando feito pontualmente. Para solucionar isto, é seguida uma recomendação proposta por Papoulis em (36), realizando-se um cálculo de valor aproximado da "probabilidade discreta" num ponto k, usando-se a função de densidade contínua através da equação 4.15.

$$P(v = k) = \int_{k-0.5}^{k+0.5} f_v(v)dv \tag{4.15}$$

O uso das extensões +0,5 e -0,5 deve ser estendido para um caso geral de cálculo de PD, de limites inferior (L_v) e superior (H_v) do intervalo a ser calculado, sendo utilizada a equação 4.16.

$$P(L_v \leq v \leq H_v) = \int_{L_v-0.5}^{H_v+0.5} f_v(v)dv \tag{4.16}$$

4.2.3.2 Cômputo da PD de variáveis compostas por mais de um evento

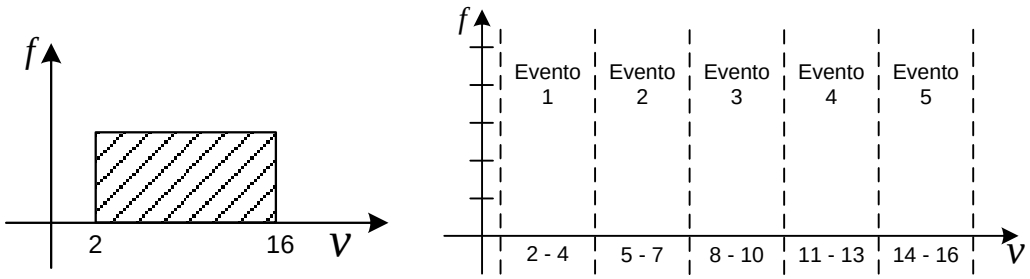
É comum em modelagem de cobertura por itens que o espaço de valores de parâmetros (variáveis dos circuitos, por exemplo) seja divido em intervalos e estes definidos como eventos. Por exemplo, os espaços de eventos $\Omega_x = [1, 7]$ e $\Omega_y = [1, 9]$ poderiam estar divididos em novos espaços de $\Omega'_x = [1, 3] \cup [4, 7]$ e $\Omega'_y = [1, 4] \cup [5, 9]$, respectivamente. Estende-se o cálculo da operação entre variáveis X eY por meio de operações entre cada um dos novos eventos de X e Y que denominaremos X_1, $X_2 \ldots X_n$ e Y_1, $Y_2 \ldots Y_m$. Apresentamos a extensão em forma de uma sequência de passos:

1. **Seleção dos eventos de saída**

   Deve-se selecionar os eventos de saída a serem observados, correspondendo ao espaço de valores que podem ser obtidos pela operação aritmética entre X e Y. O espaço de valores possíveis de saída depende do espaço que define cada uma das variáveis de entrada. No exemplo acima, considerando-se uma operação de soma, o espaço da saída de V é dado por seus valores extremos possíveis, ou seja, com $L_v = 2$ e o superior $H_v = 16$,como mostrado na figura 4.16(a).

   Por simplicidade, o intervalo obtido é dividido em pequenos intervalos de mesmo tamanho, definindo-se assim os eventos de saída (que poderá ser uma entrada de outra operação). A questão de resolução está ligada, no contexto desta tese, ao número de intervalos, sendo definido da forma a seguir.

   Resolução é uma métrica da metodologia/ferramenta de geração de modelo de cobertura, dada pelo número de intervalos em que o espaço de saída considerado é dividido. Trata-se de um fator determinante da precisão, uma vez que a saída considerada é, também, a entrada de alguma outra função ou bloco, cuja PD de saída é por sua vez



(a) Espaço dos Valores de Saída de V (b) Intervalos (Eventos) Estabelecidos no Espaço Total de V

Figura 4.16: Definição do Espaço de Eventos Possíveis para a saída V

computada[2]. A figura 4.16(b) ilustra o espaço da variável de saída para a variável V, de $\Omega'_v = [2, 16]$, da função de soma entre os eventos do exemplo acima. O espaço é divido em cinco intervalos, equivalente a cinco eventos, ou seja, com resolução 5, neste caso.

2. **Computação do PD de todas as combinações**

   Para cada uma das combinações de par de eventos, a PD é calculada selecionado usando as equações definidas na seção 4.2. Para o exemplo dado, as combinações de eventos das entradas X e Y são:

   (a) $X = [1, 3]$ e $Y = [1, 4]$

   (b) $X = [1, 3]$ e $Y = [5, 9]$

   (c) $X = [4, 7]$ e $Y = [1, 4]$

   (d) $X = [4, 7]$ e $Y = [5, 9]$

   O conjunto de densidades de probabilidade obtidas a partir de cada uma das combinações é apresentada na figura 4.17. A computação da PD para cada combinação já deve ser delimitada por eventos, dentro dos intervalos definidos para a saída.

3. **A união de todas as PDs**

   Após os dos procedimentos anteriores, procede-se a somar cada uma das PDs, computadas por evento. Como resultado se obtêm a PD da saída como a vista na figura 4.18 na resolução escolhida.

[2]Este valor para o caso desta dissertação é escolhido o valor de 20000, número, determinado por observação, que mostrou permitir uma boa aproximação para os exemplos usados na avaliação da ferramenta desenvolvida.

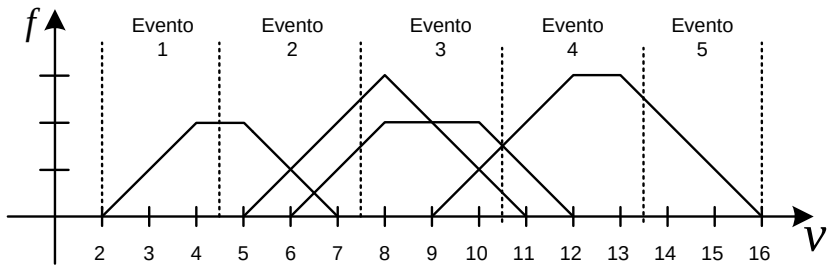


Figura 4.17: PD's sobrepostos das possíveis combinações dos eventos de X e Y

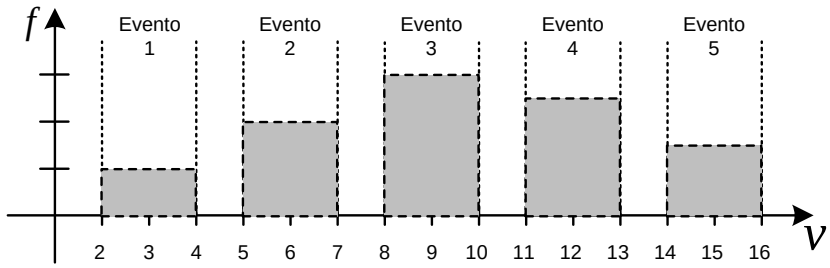


Figura 4.18: PD da saída V resultante da soma das PD de todas combinações de evento das entradas X e Y

4.3 Cômputo da PD de uma equação algébrica

Na seção anterior foi apresentada com detalhes a forma de se realizar o cômputo da PD de uma operação de uma ou entre duas variáveis. Entretanto, ao se modelar um sistema, as equações que o descrevem podem ser maiores, compostas por várias variáveis e, portanto, por mais de uma operação. Uma equação pode ser descrita genericamente como na equação 4.17.

$$v = g(x_0, x_1, ...) = c_0 x_0^{m_0} x_1^{m_1} \dots x_k^{m_k} + c_1 x_{k+1}^{m_{k+1}} x_{k+2}^{m_{k+2}} \cdots + \dots \tag{4.17}$$

Para calcular a PD de uma equação, faz-se de forma similar ao cálculo de equações algébricas; para se obter a resposta de um ponto do domínio $(x_0, x_1, \dots)$, primeiro, deve-se avaliar os valores das multiplicações e depois a soma das resultantes anteriores. Por exemplo, a figura 4.19 apresenta um diagrama de fluxo de uma FSMD contendo somente ***comandos de execução***; considerando que as entradas da FSMD são X_0, X_1, X_2, X_3, X_4 e X_5, a função expressa na equação 4.18 é gerada, contendo três componentes[3]: $4X_0$, $-2X_1^2X_2$ e $X_3X_4X_5$

[3]Nesta versão da dissertação, por simplicidade, os componentes são considerados independentes, ou seja, qualquer variável está presente em apenas um componente.

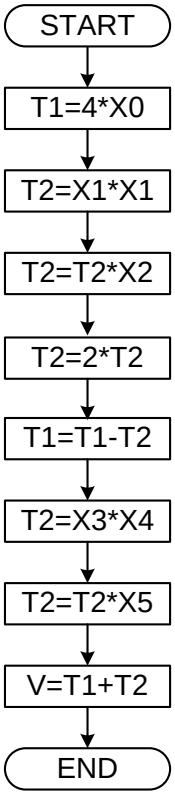


Figura 4.19: Diagrama de fluxo com comandos de execução

$$v = g(X_0, X_1, X_2, X_3, X_4, X_5) = 4X_0 - 2X_1^2X_2 + X_3X_4X_5 \tag{4.18}$$

A equação do exemplo pode ser escrita como:

$$v = g(Y_0, Y_1, Y_2) = Y_0 + Y_1 + Y_2$$

onde

$$\begin{aligned} Y_0 &= h_0(X_0) = 4X_0 \\ Y_1 &= h_1(X_1, X_2) = -2X_1^2X_2 \\ Y_2 &= h_2(X_3, X_4, X_5) = X_3X_4X_5 \end{aligned}$$

Portanto, a PD de $v = g(X_0, X_1, X_2, X_3, X_4, X_5)$ pode ser obtida pela equação4.19.

$$F(v) = \int\int\int f_{y_0} f_{y_1} f_{y_2} dy_0 dy_1 dy_2 \tag{4.19}$$

Para realizar o cômputo da equação, deve-se seguir os seguintes passos:

1. Obter a distribuição de X^n no caso o expoente da variável do componente ter $n > 1$.

2. Calcular a PD de cada componente - para isto deve-se usar as equações estabelecidas na seção 4.2.2 (Cômputo da PD da multiplicação).

3. Calcular a PD da soma dos componentes, calculada no passo anterior - para isto deve-se usar as equações estabelecidas na seção 4.2.1 (cômputo da PD da soma).

Para o primeiro passo emprega-se a equação proposta na seção 4.2.2.3. Para o exemplo apresentado, somente uma das variáveis tem um expoente maior a 1, com o termo X_1^2. Na figura 4.20(a) é apresentada a mudança da distribuição, gerando a variável $X_1' = X_1^2$. Esta "nova" variável é usada na etapa seguinte de cálculo.

Para realizar o segundo passo é estabelecido o fluxo de cálculo para usar as fórmulas definidas na seção 4.2.2. Primeiramente, somente duas variáveis são usadas no cálculo, o que permite usar as equações estabelecidas anteriormente. Para o exemplo apresentado anteriormente, nas figuras 4.20(b), 4.20(c) e 4.20(d) são ilustradas a realização desta etapa para os três componentes (Y_0, Y_1 e Y_2), com as distribuições resultantes.

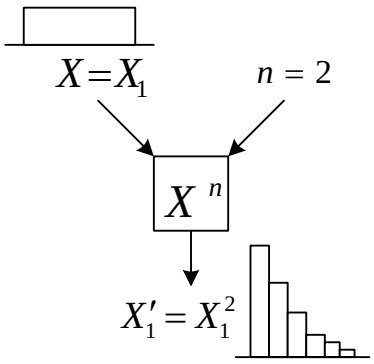
O passo três segue o mesmo procedimento que o anterior, com a diferença que neste caso são consideradas somas. Para o exemplo apresentado anteriormente, a figura 4.20(e) ilustra a realização desta etapa, considerando-se as distribuições obtidas no segundo passo.

4.4 Cálculo da PD na presença de comandos de controle

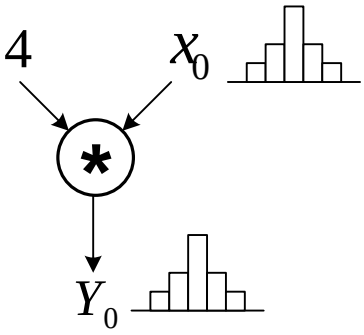
4.4.1 Análise de comandos condicionais

Na modelagem de um sistema, existem comandos que estabelecem diferentes caminhos a serem seguidos para a determinação da sua resposta, conhecidos como ***"Comandos de Controle Condicional"***. Tais estruturas utilizam uma operação relacional como parte de sua declaração, com a qual, após a determinação da falsidade ou tautologia, pode-se selecionar o caminho a seguir. Na figura 4.21, é mostrado um exemplo de código de um programa com um comando condicional ***"IF"***, cuja condição é $(X < 0)$. Designando-se $X = 4$ e $Y = 5$, o programa executa o comando $V = Y - X$ com valor de saída $V = 1$. Em um outro caso em que o valor de X fosse $X = -1$, o comando a ser executado seria $V = Y + X$ e a saída $V = 4$.

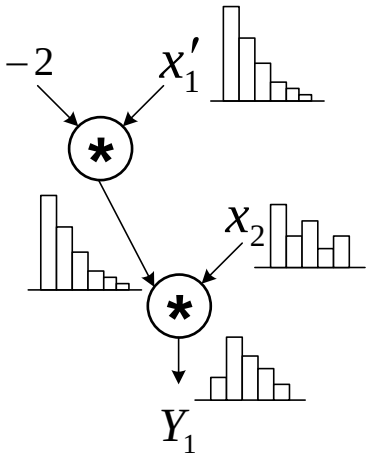
O exemplo indica que, dependendo do caminho a seguir pelo programa, a equação de saída do modelo muda e o fato de existirem mais de um caminho implica em que se deve



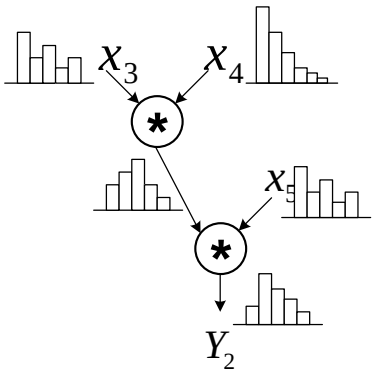
(a) PD de $X_1' = X_1^2$



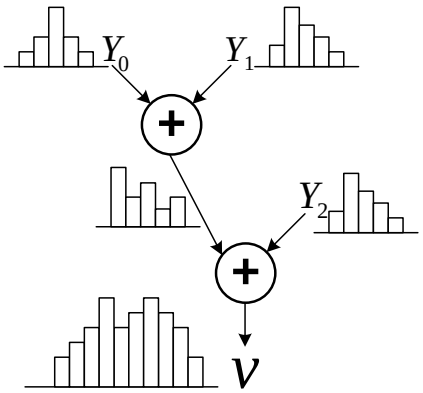
(b) PD de $Y_0 = 4X_0$



(c) PD de $Y_1 = -2X_1^2X_2$



(d) PD de $Y_2 = X_3X_4X_5$



(e) PD de $v = g(Y_0, Y_1, Y_2) = Y_0 + Y_1 + Y_2$

Figura 4.20: Exemplo de cálculo da PD da função $v = g(X_0, X_1, X_2, X_3, X_4, X_5) = 4X_0 - 2X_1^2X_2 + X_3X_4X_5$

if $(X < 0)$ **then**
  $V = Y + X$
else
  $V = Y - X$
end if

Figura 4.21: Exemplo de condicional

haver uma probabilidade associada à ocorrência de cada um deles. Ainda usando o exemplo anterior, pode-se estabelecer que as equações que determinam a saída do código anterior são:

$$V = \begin{cases} \boxed{Y + X} & \text{se } X < 0 \\ \boxed{Y - X} & \text{se } X \geq 0 \end{cases}$$

As equações que estão enquadradas nas caixas são conhecidas como equações objetivo e as que estão fora das caixas são restrições; a saída, dependente das restrições, tem como PD a função expressa por

$$P(V \leq v) = P((Y + X \leq v) \cap (X < 0)) + P((Y - X \leq v) \cap (X \geq 0))$$

A resolução deste tipo equação é bastante complexa quando as restrições e a função objetivo não são independentes entre si. Isto pelo fato de que a(s) variável(eis) que afeta(m) a equação objetivo estão também na(s) equação(es) de restrição. Portanto, para calcular $F(v)$ desta equação teria-se que obter as áreas de integração que definem as restrições sobre o Ω_x e Ω_y, de X e Y respectivamente. Isto difere do caso independente, onde, para calcular a $F(v)$ resultante, só se precisa calcular o produto da PD da função objetivo e a probabilidade de ocorrência desta, a qual está definida pelas restrições. Por este motivo, nesta dissertação somente o caso de condicionais independentes é considerado.

4.4.1.1 Cômputo da probabilidade de cada caminho de um nó condicional

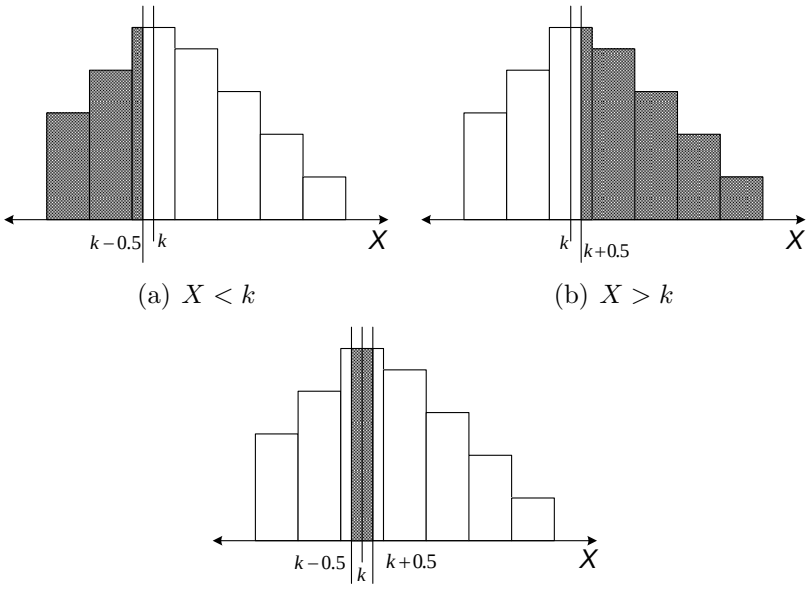
O cômputo da probabilidade de ocorrência de um dos caminhos de um nó condicional depende da operação de relacionamento, $<$, $>$, $=$, $\leq$, $\geq$, $\neq$. Tendo em conta os conceitos de probabilidade, pode-se definir que:

$$P(X < k) = 1 - P(X \geq k) \tag{4.20a}$$

$$P(X > k) = 1 - P(X \geq k) \tag{4.20b}$$

$$P(X = k) = 1 - P(X \neq k) \tag{4.20c}$$

Conhecendo-se a forma de computar $P(X < k)$, $P(X > k)$ e $P(X = k)$, pode-se obter a probabilidade das outras funções de relacionamento. Abaixo são apresentadas as



(a) $X < k$ (b) $X > k$

(c) $X = k$

Figura 4.22: Áreas consideradas no cômputo da probabilidade de caminhos em operações de relacionamento

equações utilizadas para o cálculo da probabilidade de ocorrência de valores nas funções de relacionamento mencionadas acima, segundo as considerações feitas na seção 4.2.3 sobre aproximações no uso de distribuição contínua e sobre a resolução adequada. A representação gráfica das áreas que consideram estas equações são mostradas na figura 4.22.

$$P(X < k) = \int_{-\infty}^{k-0.5} f_x(x)dx \tag{4.21a}$$

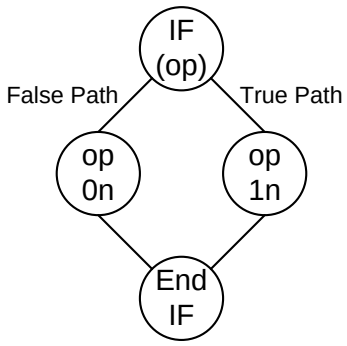
$$P(X > k) = \int_{k+0.5}^{\infty} f_x(x)dx \tag{4.21b}$$

$$P(X = k) = \int_{k-0.5}^{k+0.5} f_x(x)dx \tag{4.21c}$$

4.4.1.2 Analise do nó condicional

Para computar a PD de saída modelada por uma FSMD que tem somente um nó condicional, segue-se os passos abaixo:

1. Extrair a equação do caminho verdadeiro e do caminho falso. Por exemplo na figura 4.23 a equação de saída no caminho de condição *verdadeira* é gerada pelo comando



if $(< Condition >)$ **then**
$< Statement1 >$
else
$< Statement0 >$
end if

Figura 4.23: Exemplo generalizado de um condicional de dois caminhos

de execução $< Statement1 >$ e a equação resultante para o caso *falso* é gerada por $< Statement0 >$. Isto gera a equação abaixo.

$$Saida = \begin{cases} < Statement0 > & \text{se } < Condition >= False \\ < Statement1 > & \text{se } < Condition >= True \end{cases} \tag{4.22}$$

2. Extrair a probabilidade associada à ocorrência daquele caminho.

3. Computar a densidade de probabilidade de cada equação e multiplicar cada uma com sua probabilidade de ocorrência. A fórmula resultante deste procedimento é apresentada na equação 4.23.

$$F_{saida} = \int f_{saida} = P_{Condition_{True}} \int f_{<Statement0>} + P_{Condition_{False}} \int f_{<Statement1>} \tag{4.23}$$

onde:

F_{saida}	:	Função da Distribuição de Probabilidade de saída total.
f_{saida}	:	Função da Densidade Probabilística de saída total.
$f_{<Statement0>}$	:	Função de Densidade Probabilística da sentença $< Statement0 >$.
$f_{<Statement1>}$	:	Função de Densidade Probabilística da sentença $< Statement1 >$.
$P_{Condition_{True}}$	:	Probabilidade do caminho com a condição verdadeira acontecer.
$P_{Condition_{False}}$	:	Probabilidade do caminho com a condição falsa acontecer.

No modelo apresentado pela FSMD da figura 4.24 tem-se um fluxo que contém somente um comando condicional. Para calcular a PD da saída Z modelada na figura deve-se calcular densidade de probabilidade desta. Para isto, observa-se que o comando condicional está definido por $W > 0$, onde sua probabilidade de ser *verdadeiro* é α_1 e *falso* é α_0. O cálculo destes valores, observando a figura 4.25(a) e baseado nos conceitos de probabilidades de variáveis contínuas, é dado pela equação 4.21. Entretanto, tendo-se em conta as

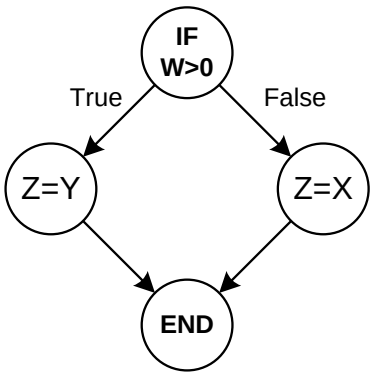
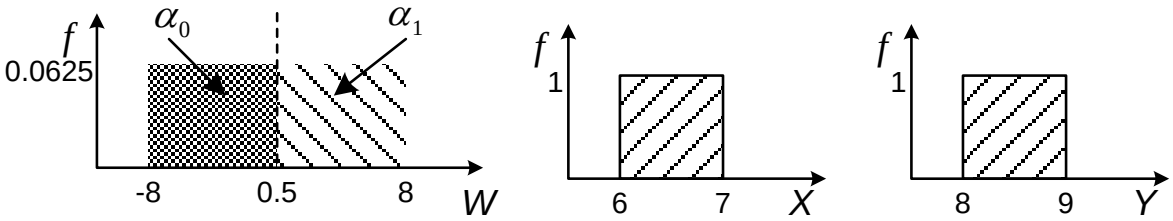
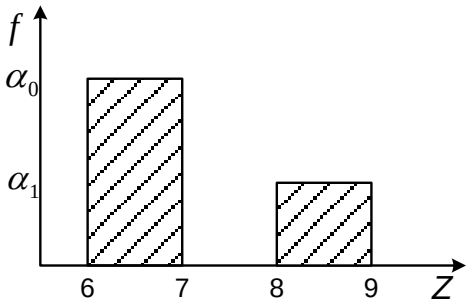


Figura 4.24: Exemplo de FSMD com um condicional



(a) Densidade de probabilidade de W (b) Densidade de probabilidade de X (c) Densidade de probabilidade de Y



(d) Densidade de Probabilidade de Z

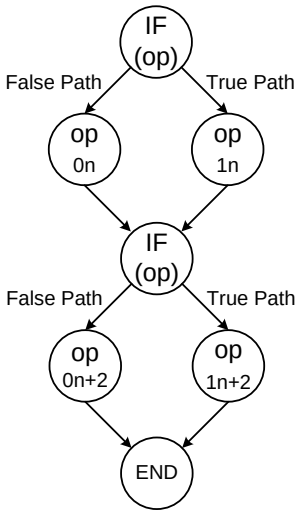
Figura 4.25: Exemplo de cômputo da densidade de probabilidade da saída Z do modelo da figura 4.24

considerações de cálculo apresentadas na seção 4.4.1.1 (na página 58) os limites da integral mudam, gerando as equações 4.25.

$$\alpha_0 = \int_{-8}^{0} 0.0625dw \qquad (4.24a) \qquad\qquad \alpha_0 = \int_{-8}^{0.5} 0.0625dw \qquad (4.25a)$$

$$\alpha_1 = \int_{0}^{8} 0.0625dw \qquad (4.24b) \qquad\qquad \alpha_1 = \int_{0.5}^{8} 0.0625dw \qquad (4.25b)$$

Conhecendo os valores de α_0 e α_1 (obtidos com a equação 4.25) e conhecendo as densidades de probabilidade das variáveis X e Y (Figuras 4.25(b) e 4.25(c)) procede-se a substituir estes valores na equação 4.23, obtendo o gráfico mostrado na figura 4.25(d).



if $(< Condtion0 >)$ **then**
$< Statement01 >$
else
$< Statement00 >$
end if
if $(< Condtion1 >)$ **then**
$< Statement11 >$
else
$< Statement10 >$
end if

Figura 4.26: Dois condicionais em serie

4.4.1.3 Análise de dois nós condicionais encadeados

É comum depararmo-nos com uma descrição de um programa com condicionais em série, como no exemplo de código apresentado a seguir e cuja representação em forma de grafo de estados é mostrada na figura 4.26. Fazendo-se uma análise do fluxo do programa pode-se extrair todos os caminhos possíveis que se pode percorrer. No caso considerado, pode-se observar que existem quatro caminhos possíveis, gerando a função de distribuição de saída da equação 4.26.

$$\begin{aligned} F_{saida} = & P_{Condtional0_{True}} P_{Contional1_{True}} \int f_{g_{<Statement01>,<Statement01>}} + \\ & P_{Condtional0_{True}} P_{Contional1_{False}} \int f_{g_{<Statement01>,<Statement00>}} + \\ & P_{Condtional0_{False}} P_{Contional1_{True}} \int f_{g_{<Statement00>,<Statement01>}} + \\ & P_{Condtional0_{False}} P_{Contional1_{False}} \int f_{g_{<Statement00>,<Statement00>}} \end{aligned} \tag{4.26}$$

onde:

$f_{g_{<Statement00>,<Statement10>}}$: Distribuição Probabilística da função gerada pelos comandos $< Statement00 >$ e $< Statement10 >$

$f_{g_{<Statement00>,<Statement11>}}$: Distribuição Probabilística da função gerada pelos comandos $< Statement00 >$ e $< Statement11 >$

$f_{g_{<Statement01>,<Statement10>}}$: Distribuição Probabilística da função gerada pelos comandos $< Statement01 >$ e $< Statement10 >$

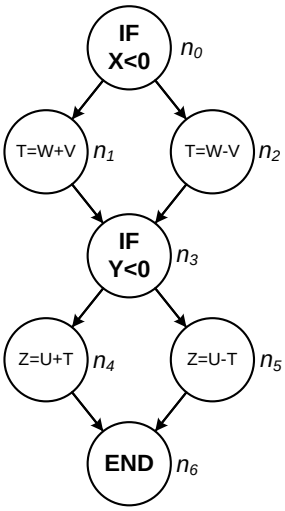


Figura 4.27: Exemplo de Análise de nós condicionais

$f_{g_{<Statement01>,<Statement11>}}$	: Distribuição Probabilística da função gerada pelos comandos $< Statement01 >$ e $< Statement11 >$
$P_{Conditional0_{True}}$	: Probabilidade de que a condição $< Condition0 >$ seja verdadeira
$P_{Conditional0_{False}}$	: Probabilidade de que a condição $< Condition0 >$ seja falsa
$P_{Conditional1_{True}}$	: Probabilidade de que a condição $< Condition1 >$ seja verdadeira
$P_{Conditional1_{False}}$	: Probabilidade de que a condição $< Condition1 >$ seja falsa

Para mostrar como se faz o cômputo da PD de um modelo que contém comandos de controle condicionais, o exemplo da figura 4.27 é apresentado. Nele, procura-se obter a distribuição da saída Z, conhecendo-se o comportamento de X, Y, U, V, W. Na figura, os nós n_1, n_2, n_5 e n_6 representam operações aritméticas (comandos de execução), os nós n_0 e n_4 comandos de controle condicional, n_3 e n_7 representam o final dos condicionais e n_8 é o final do programa. Os caminhos das condicionais têm as seguinte probabilidades:

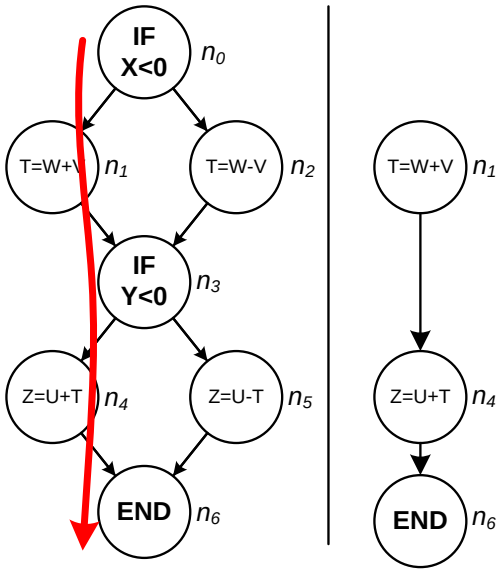
α_0 : Probabilidade de $X < 0$ ser falso.

α_1 : Probabilidade de $X < 0$ ser verdadeiro.

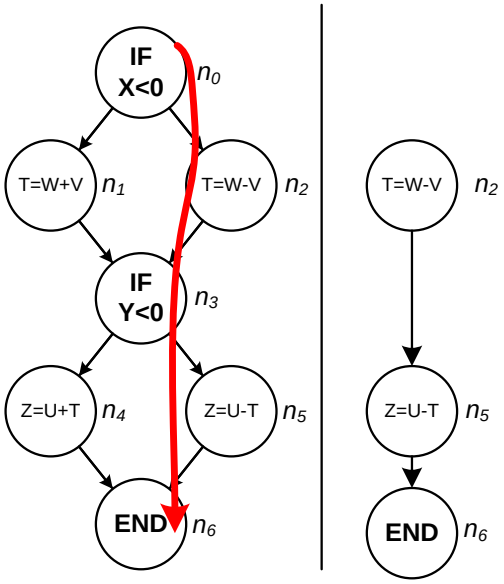
β_0 : Probabilidade de $Y < 0$ ser falso.

β_1 : Probabilidade de $Y < 0$ ser verdadeiro.

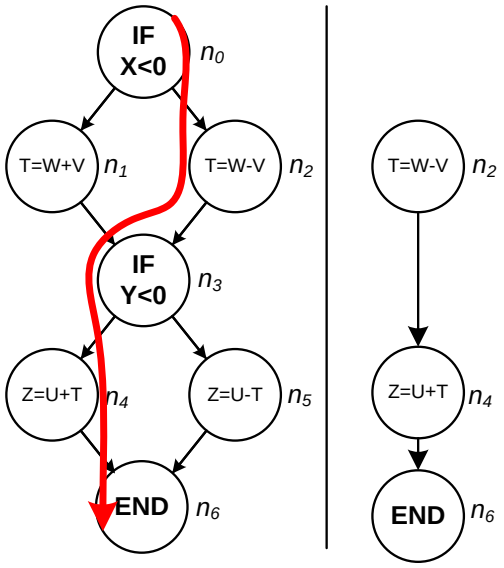
Para se determinar o PD do grafo apresentado na figura 4.27, deve-se de conhecer todos os possíveis caminhos que este tem. No caso apresentado, existem quatro caminhos. O primeiro analisado é mostrado na figura 4.28(b) através da linha vermelha.



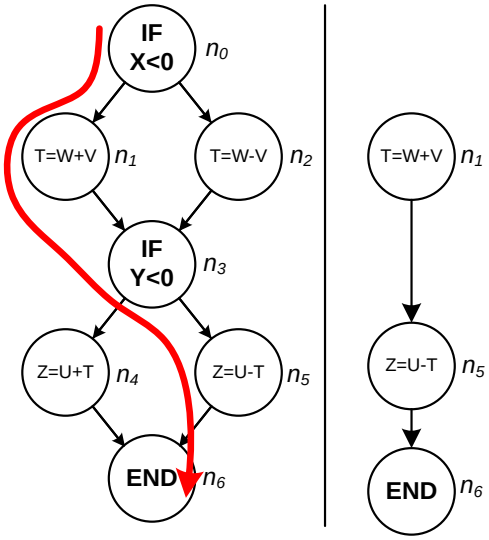
(a) Primeiro caminho



(b) Segundo caminho



(c) Terceiro caminho



(d) Quarto caminho

Figura 4.28: Caminhos possíveis do exemplo da figura 4.27

Ao longo do trajeto, mostra-se os nós das operações aritméticas executadas. Este caminho gera a função:

$$z = g_1(U, V, W) = U + V + W \tag{4.27}$$

Entretanto, para este caminho acontecer, as condições ***if*** $(X < 0)$ e ***if*** $(Y < 0)$ devem ser falsas. O PD é então:

$$F_{g_1}(z) = \alpha_0\beta_0 \int_{-\infty}^{z} f_{g_1} = \alpha_0\beta_0 P(U + V + W \leq z) \tag{4.28}$$

No caso do segundo caminho, a equação de saída resultante é:

$$z = g_2(U, V, W) = U - V - W \tag{4.29}$$

Para este caso acontecer, as condições ***if*** $(X < 0)$ e ***if*** $(Y < 0)$ devem ser falsa e verdadeira, respetivamente. Portanto a PD resultante é:

$$F_{g_2}(z) = \alpha_0\beta_1 \int_{-\infty}^{z} f_{g_2} = \alpha_0\beta_1 P(U - V - W \leq z) \tag{4.30}$$

Para o terceiro caminho, a equação de saída resultante é:

$$z = g_3(U, V, W) = U - V + W \tag{4.31}$$

Neste caso, as condições ***if*** $(X < 0)$ e ***if*** $(Y < 0)$ devem ser verdadeira e falsa, respetivamente. Portanto a PD resultante é:

$$F_{g_3}(z) = \alpha_1\beta_0 \int_{-\infty}^{z} f_{g_3} = \alpha_1\beta_0 P(U - V + W \leq z) \tag{4.32}$$

Para o quarto caminho, a equação de saída resultante é:

$$z = g_4(U, V, W) = U + V - W \tag{4.33}$$

Para tal, as condições ***if*** $(X < 0)$ e ***if*** $(Y < 0)$ devem que ser verdadeiras. Portanto o PD resultante é:

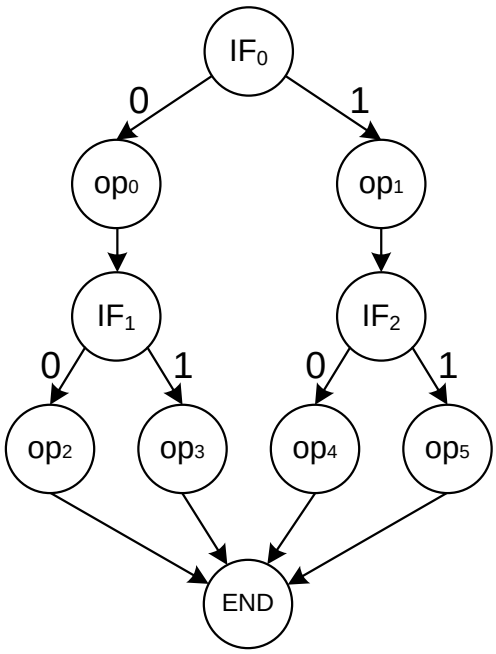


Figura 4.29: Grafo generalizado de nós condicionais encadeado

$$F_{g_4}(z) = \alpha_1\beta_1 \int_{-\infty}^{z} f_{g_4} = \alpha_1\beta_0 P(U + V - W \leq z) \tag{4.34}$$

Como resultado da combinação de todos os caminhos, o PD resultante do modelo mostrado é igual a:

$$\begin{aligned} F(z) &= F_{g_1}(z) + F_{g_2}(z) + F_{g_3}(z) + F_{g_4}(z) \\ &= \alpha_0\beta_0 \int_{-\infty}^{z} f_{g_1} + \alpha_0\beta_1 \int_{-\infty}^{z} f_{g_2} + \alpha_1\beta_0 \int_{-\infty}^{z} f_{g_3} + \alpha_1\beta_1 \int_{-\infty}^{z} f_{g_4} \end{aligned} \tag{4.35}$$

Esta resposta é equivalente à apresentada na equação 4.26.

4.4.1.4 Generalização da análise de nós condicionais

Pode-se estender o caso da seção anterior para uma forma generalizada de nós condicionais encadeados, como na figura 4.29 onde IF_0, IF_1 e IF_2 representam nós de decisão e op_0a op_5 representam as operações nos ramos. Pode-se observar agora que cada ramo do primeiro comando condicional leva a outros dois totalmente independentes, com a função de distribuição de saída igual à forma expressa na equação 4.36.

$$\begin{aligned} f_g(X) = & P(\overline{IF_0})P(\overline{IF_1})f_{g_1} + \\ & P(\overline{IF_0})P(IF_1)f_{g_2} + \\ & P(IF_0)P(\overline{IF_2})f_{g_3} + \\ & P(IF_0)P(IF_2)f_{g_4} \end{aligned} \tag{4.36}$$

onde

$$g(X) = \begin{cases} g_1(op_0, op_2) & \text{if } \overline{(IF_0)} \wedge \overline{(IF_1)} \\ g_2(op_0, op_3) & \text{if } \overline{(IF_0)} \wedge (IF_1) \\ g_3(op_1, op_4) & \text{if } (IF_0) \wedge \overline{(IF_2)} \\ g_4(op_1, op_5) & \text{if } (IF_0) \wedge (IF_2) \end{cases} \tag{4.37}$$

$f_{g_1} \ldots f_{g_4}$: Distribuição Probabilística da função $g_1 \ldots g_4$

A seguir é indicada a probabilidade para que aconteça um dos caminhos em cada condicional, sendo $P(IF_i) = 1 - P(\overline{IF_i})$.

$P(IF_0) \ldots P(IF_2)$: Probabilidade que a condição $IF_0 \ldots IF_2$ seja verdadeira.
$P(\overline{IF_0}) \ldots P(\overline{IF_2})$: Probabilidade que a condição $IF_0 \ldots IF_2$ seja falsa.

A forma descrita na figura pode ser generalizada com

$X = \{x_0, x_1, ..., x_n\}$: Grupo de entradas do modelo.
$OP = \{op_0, op_1, ..., op_m\}$: Grupo de comandos do fluxo do modelo.
$IF = \{IF_0, IF_1, .., IF_l\}$: Grupo de condicionais do fluxo do modelo.

A partir dos exemplos mostrados anteriormente, pode-se definir como equação geral 4.38 para obter a distribuição de saída do modelo.

$$F_{g_x} = \int_{-\infty}^{\infty} f_{g_x} = \sum_{i=0}^{L-1} \int_{-\infty}^{\infty} f_{g_{x_i}} \prod_{k=0}^{M_i-1} P(IF_{i_k}) \tag{4.38}$$

L	:	Número de caminhos possíveis para determinar o valor saída do modelo do circuito.
M	:	Número total de condicionais do modelo do circuito.
M_i	:	Número total de condicionais que participam no caminho i.
X	:	Grupo de entradas que participam do processamento do modelo.
g_x	:	Equação de saída do modelo.
g_{x_i}	:	Equação de saída formada pelo caminho i do modelo.
f_{g_x}	:	Distribuição probabilística de g_x.
$f_{g_{x_i}}$	:	Distribuição probabilística de g_{x_i}.
IF_i	:	Conjunto de condicionais que participam do caminho i.
IF_{i_k}	:	Condicional k que participa do caminho i.
$P(IF_{i_k})$	:	Probabilidade de IF_{i_k}, seja esta verdadeira ou falsa (segundo o caminho).

4.4.2 Comandos de laços

Laço é o comando de controle nas linguagens de programação que permite repetir um número determinado de linhas de código baseado em condições. O número de repetições pode ser determinístico, com o número de repetições conhecido, caso que consideraremos nesta dissertação; os casos não determinísticos são mais complexos, requerendo análise similar aos condicionais não independentes. A razão para isto é que um laço pode ser modelado também como um comando condicional onde a condição *verdadeira* indica a execução dos comandos dentro do laço e a condição *falsa* indica a execução daqueles fora do laço.

O exemplo na figura 4.30 representa o modelo de um circuito genérico e em um diagrama de fluxo FSMD, com saída Z entradas X e Y; T e i são variáveis temporárias. Nesta figura pode-se observar a expressão $T = T + Y$, a qual é repetida cinco vezes, de forma determinística, gerando a resposta de saída $Z = X + 5Y$.

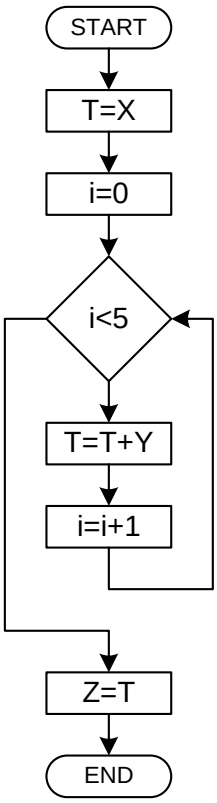


Figura 4.30: Exemplo com o comando de controle de laço

5 Desenvolvimento da ferramenta

No Capítulo 4 foram apresentadas as técnicas probabilísticas que nos permitirão calcular a PD de saída do circuito, gerando função esta que tem como domínio as entradas do circuito. O cálculo da PD pode ser realizado através da análise dos modelos algorítmicos descritos por alguma HDL (por exemplo, usando-se modelos de referência de *testbenches*). Para este propósito o modelo algorítmico do circuito deve ser representado adequadamente e foi decidido pelo uso de uma FSMD dada a sua similaridade com o fluxo de um programa. Através desta representação é possível avaliar cada um dos passos dos comandos para estabelecer a equação de saída. Deve-se mencionar que as tarefas de conversão para FSMD não são contempladas neste trabalho, dado o tempo a ser dispendido no seu desenvolvimento. Existem, de fato, um número grande de compiladores no mercado para este propósito, apesar de que não há muitos de uso livre disponíveis. Optamos por construir a representação de dados manualmente para daí utilizá-los como ponto de partida.

A partir destes conceitos e restrições estabelecidas, neste capítulo são apresentados os algoritmos e ferramentas que nos permitem analisar FSMDs e computar as probabilidades de saída dos nós. Neste capítulo apresentamos inicialmente a representação intermediária definida para as FSMDs e depois as técnicas e algoritmos para a computação das distribuições de dados junto às saídas de circuitos.

5.1 Representação intermediária dos FSMDs

Como mencionado no capítulo 4, uma FSMD é composta de nós (ou estados) que podem ser classificados de acordo com a sua função. Estas funções estabelecem propriedades e através destas pode-se definir processos para a análise de cada tipo de estado através dos algoritmos a serem propostos nas próximas seções.

Nesta seção são apresentadas as propriedades de cada um dos tipos de nós (com as restrições estabelecidas no capítulo 4). Estas propriedades são apresentadas como classes para poderem ser utilizadas numa linguagem de programação orientada a objetos (para esta dissertação é utilizada a linguagem C++). Através das classes definidas nesta seção,

$$
\begin{aligned}
formula &: formula\ op\ formula \mid atom \\
atom &: term\ op\ term \mid term \\
term &: variavel \mid constante \\
op &: + \mid - \mid * \mid / \mid << \mid >>
\end{aligned}
$$

Figura 5.1: Sintaxe geral dos comandos de execução

são projetados os algoritmos da seção 5.2.

5.1.1 Nó de designação

Comandos de execução são encontrados num programa para a designação de valores. Em uma FSMD equivalente ao programa, eles ocorrem em nós de operações aritméticas ou simplesmente na designação de valores de constantes ou de variáveis a outras variáveis. Este tipo de nó é definido pela sintaxe apresentada na figura 5.1, na qual as entidades *formula*, *atom*, *term* e *op* são definidas. Por exemplo, a sintaxe define que *formula* pode ser resultado de uma operação entre *formulas* ou um *atom*, e assim por diante. Neste caso, os *ops* indicam os operadores aritméticos que o comando pode utilizar.

A partir das considerações anteriores, é proposta a classe ***ExeCmdClass***, que contém as propriedades mencionadas para este tipo de comando ou nó. A lista de membros desta classe é mostrada na tabela 5.1, os quais são detalhados abaixo.

1. Op: se encarrega de armazenar a operação (+,-,/,*) que deve ser realizada no nó (ou estado) que representa.

2. InputA/B: é um ponteiro à variável A/B (ou arranjo de variáveis), que contém uma equação (ou uma constante) a qual vai ser usada na operação do presente estado.

3. OutputB: representa o ponteiro de uma variável (ou um arranjo de variáveis) de saída, onde será armazenada a resposta.

4. EnArrayA/B/O: flag que está ativo no caso em que as entradas (ou saídas) estejam representadas por um arranjo de variáveis.

5. VarIndexA/B/O: variável usada para selecionar um elemento do arranjo da entrada A/B ou da saída O.

6. NextPath: ponteiro para o estado (ou nó) seguinte da FSMD que deve ser executado.

Tabela 5.1: Membros da Classe ***ExeCmdClass***

Membros	Caraterísticas
Op	Operação do comando de execução
InputA	Variável de entrada A (emprega a Estrutura ***VarClass***)
InputB	Variável de entrada B (emprega a Estrutura ***VarClass***)
Output	Variável de saída (emprega a Estrutura ***VarClass***)
EnArrayA	Habilita a entrada A como arranjo
EnArrayB	Habilita a entrada B como arranjo
EnArrayO	Habilita a saída como arranjo
VarIndexA	Variável usada de índice do arranjo da entrada A (emprega a estrutura ***VarClass***)
VarIndexB	Variável usada de índice do arranjo da entrada B (emprega a estrutura ***VarClass***)
VarIndexO	Variável usada de índice do arranjo de saída (emprega a estrutura ***VarClass***)
NextPath	Ponteiro a seguinte nó do grafo G que contém o modelo do circuito

$$
\begin{aligned}
expression &: \textbf{\textit{if}}\ (atom)\ DoSomething\ \textbf{else}\ DoSomethingElse \\
atom &: term1\ op\ term2 \\
term1 &: Variavel \mid constante \\
term2 &: constante \\
op &: < \mid > \mid \leq \mid \geq \mid = \mid \neq
\end{aligned}
$$

Figura 5.2: Sintaxe geral do comando de controle condicional

Vários dos elementos da tabela 5.1 usam a estrutura *VarClass*. Esta estrutura define as propriedades usadas para construir uma variável nesta dissertação. Os seus elementos estão na tabela 5.7 e a explicação correspondente na seção 5.2.1.1 deste capítulo.

5.1.2 Nós condicionais

Os nós condicionais dentro da FSMD determinam caminhos de execução de um algoritmo. A seleção do caminho depende da operação estabelecida no nó. Esta operação deve seguir a sintaxe apresentada na figura 5.2, composta pela expressão ***if***, o qual indica que é um comando de controle condicional. Entre parênteses tem-se a condição que determina o caminho que deve ser executado. As expressões *DoSomething* e *DoSomethingElse* apontam para nós ou conjunto de comandos (Execução ou Controle) que são as "atividades" que o algoritmo deve seguir.

A partir das considerações feitas, a classe ***CondCmdClass*** é proposta. Ela é composta pelas propriedades mostradas na tabela 5.2.

Tabela 5.2: Membros da Classe ***CondCmdClass***

Membros	Caraterísticas
Expression	Armazena a expressão condicional (Usa estrutura ***ExeCmdClass***)
Path	Possíveis caminhos de trabalho

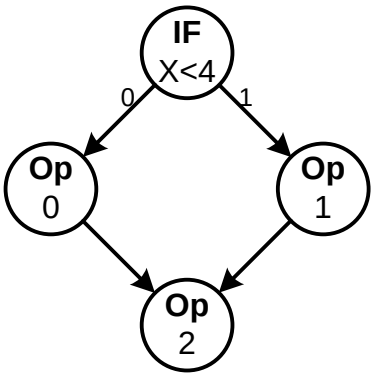


Figura 5.3: Exemplo de *comando de controle condicional* numa FSMD

O primeiro membro contém a expressão de condição. Por exemplo na figura 5.3 a condição é expressada no nó $if(X < 4)$. Esta expressão é usada para calcular a probabilidade de ocorrência dos caminhos. A expressão é armazenada usando a estrutura ***ExeCmdClass*** mostrada na tabela 5.1 e explicado na seção 5.1.1. O procedimento de análise e de cálculo é apresentado na seção 5.2.1.3 deste capítulo.

O segundo membro contém os dois possíveis caminhos de execução do nó condicional. Por exemplo na figura 5.3 estes nós estariam representados por *Op* 0 e *Op* 1.

5.1.3 Nós de laços

Um outro nó de controle existente numa FSMD (também como comando de controle nas linguagens de programação) é o nó de laço. Este permite repetir um número determinado de outros nós (ou linhas de código no caso de programas) baseado em condições. Neste caso, como mencionado em 5.1.2, usam-se laços determinísticos que seguem a sintaxe de linguagens do tipo C/C++ com algumas restrições, como expressa na figura 5.4, para um laço ***for***.

Na figura 5.5, a sintaxe do laço (figura 5.4) é representada, na forma de uma FSMD, permitindo assim observar o fluxo de execução de *formula1*, *formula2*, *formula3* e *DoSomething*. A *formula1* é somente executada num primeiro instante, após o qual, o nó condicional (cuja expressão está dada por *formula2*) é analisado. No caso de a condição ser válida, os nós internos do laço são executados (definido na figura 5.4 como *DoSomething*). Caso a condição for falsa, o estado externo ao laço dentro da FSMD é executado.

$$
\begin{aligned}
expression &: \textbf{\textit{for}}(formula1;\ formula2;\ formula3)\ DoSomething \\
formula1 &: item1\ op1\ item2 \\
formula2 &: item1\ op2\ item2 \\
formula3 &: item1\ op1\ item2 \\
op1 &: +|- \\
op2 &: < \mid > \mid \leq \mid \geq \\
item1 &: Variavel \mid constante \\
item2 &: constante
\end{aligned}
$$

Figura 5.4: Sintaxe geral do comando de controle de laço

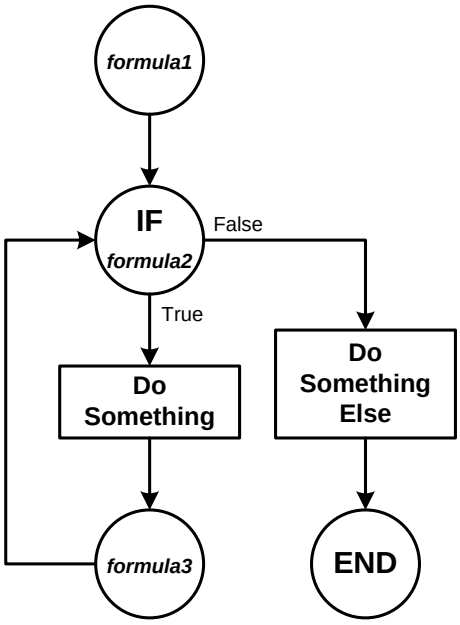


Figura 5.5: Exemplo da FSMD estendida do laço usando o formalismo da figura 5.4

Nas figuras 5.4 e 5.5 observa-se quatro estados no laço *for*:

1. INICIALIZAÇÃO: Nesta etapa, executa-se a operação definida por *formula1*. Esta operação atribui o valor inicial ao contador do repetições do laço.

2. COMPARAÇÃO: Analisa a condição estabelecida na *formula2*. Antes de executar os comandos dentro do laço, avalia se o contador cumpre com os limites que a condição define.

3. CONTAGEM: Nesta estado é executada a *formula3*, que descreve a sentença de incremento ou decremento do contador.

4. RETORNO: Ao finalizar as sentenças do laço, o fluxo da FSMD retorna ao estado de COMPARAÇÃO para verificar se há necessidade de se repetir os comandos do laço.

Como o laço ser considerado nesta dissertação é determinístico, diferentemente do caso do nó condicional, não há necessidade de se calcular a probabilidade de ocorrência de algum dos caminhos ao de se analisar a condição dada pela *formula2*. Isto permite evitar várias etapas de cálculo de probabilidade ao se avaliar o laço e consequentemente reduzir o tempo de cômputo.

Para se obter o estabelecido no parágrafo anterior, nesta dissertação o laço ***for*** é modelado como na figura 4.7(b) da seção 4.1.3 (página 41) do capítulo anterior. Pela figura, o nó ***FOR*** realiza as etapas de INICIALIZAÇÃO e COMPARAÇÃO e o nó ***ENDFOR*** as etapas de CONTAGEM e RETORNO.

Desta forma, define-se a classe ***ForCmdClass*** e ***EndForCmdClass*** com as propriedades acima. As variáveis usadas para estes fins estão mostradas nas tabelas 5.3 e 5.4, respetivamente.

No caso de ***ForCmdClass***, são elas:

1. EnableFor: este flag é ativado no estado do INICIALIZAÇÃO do laço ***for***. Ele explicita se *formula1* foi executada. A fórmula usada está armazenada em OpInitial da classe ***ForCmdClass***.

2. ExtNextNode: contém o nó (ou estado) que deve ser lido após se finalizar a análise do laço **for** e os comandos que este contém. Na figura 5.6, este estado é representado pelo nó $Z = X$.

3. IntNextNode: representa o primeiro nó (ou estado) a ser analisado dentro do laço FOR; no exemplo da figura 5.6, trata-se do nó de $X = X + Y$.

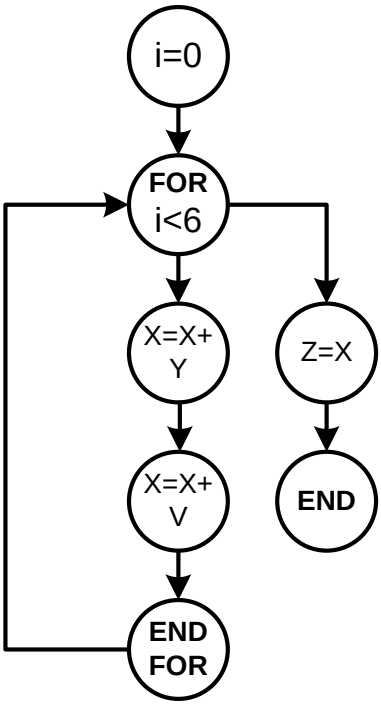


Figura 5.6: Exemplo de FSMD com laço ***FOR***

4. OpInitial: contém a operação de inicialização da variável contador que é usada para se determinar o inicio e fim do laço. Na figura 5.5 representa a *formula1* e na figura 5.6 esta operação está representado pela operação $i = 0$. A estrutura *ExeCmdClass* é empregada para armazenar a operação.

5. OpComp: contém a operação de comparação. Esta é usada para determinar se foram completadas todas as repetições do laço. Na figura 5.5, corresponde à *formula2* e, na figura 5.6, à expressão $i < 6$. A estrutura *ExeCmdClass* é empregada para armazenar a operação.

No caso de ***EndForClass***, as variáveis são:

1. IniForNode: é um ponteiro do nó FOR correspondente.

2. OpStep: representa a operação de incremento/decremento do contador para definir o número de vezes que se repete o laço. Na figura 5.5, corresponde à *formula3* e, na figura 5.6, à expressão $i = i + 1$, embutida em ***ENDFOR***. A estrutura *ExeCmdClass* é empregada para armazenar a operação.

Tabela 5.3: Membros da classe ***ForCmdClass***

Membros	Caraterísticas
EnableFor	Indica se o laço foi inicializado
ExtNextNode	Contém o nó seguinte fora do laço
IntNextNode	Contém o nó seguinte dentro do laço
OpIntial	Contém a operação para inicialização do laço (usa a estrutura ***ExeCmdClass***)
OpComp	Contém a operação para execução do laço (usa a estrutura ***ExeCmdClass***)

Tabela 5.4: Membros da classe ***EndForCmdClass***

Membros	Caraterísticas
IniForNode	Contém a posição do nó **for** respetivo
OpStep	Contém a operação de incremento/decremento da variável usada no laço (usa estrutura ***ExeCmdClass***)

5.2 Metodologia e algoritmos para a computação de distribuições de valores à saída

Definidas as representações dos tipos de nós que compõem uma FSMD (Seção 5.1 deste capítulo) e as estratégias para o cálculo da PD de saída (Seção 4.2 do capítulo 4, página 41), nesta seção estabelece-se os algoritmos usados para a sua computação, que estão consolidados em uma ferramenta denominada PrOCov. Para isto, dois dados de entrada são necessários:

1. A FSMD do modelo de referência do circuito, da qual são extraídas as equações que definem cada uma das saídas do circuito.

2. O conjunto de eventos (intervalos I) de cada entrada E_k e suas densidades de ocorrências. Todos os eventos são independentes entre si ($I_{k,i} \cap I_{k,j} = \varnothing$, $\forall i \neq j$), significando que nenhum dos elementos do intervalo $I_{k,i}$ está definido no $I_{k,j}$.

A partir destes dados a ferramenta PrOCov trabalha em duas etapas:

1. Extração das equações que define cada uma das saídas. No caso de existir vários caminhos para definir uma saída, neste processo também é calculada a probabilidade de ocorrência de cada caminho. A figura 5.7 apresenta um exemplo de FSMD com quatro caminhos, de ρ_0 a ρ_4. Para cada caminho, uma equação específica existe, sendo o conjunto de equações para o exemplo dado por:

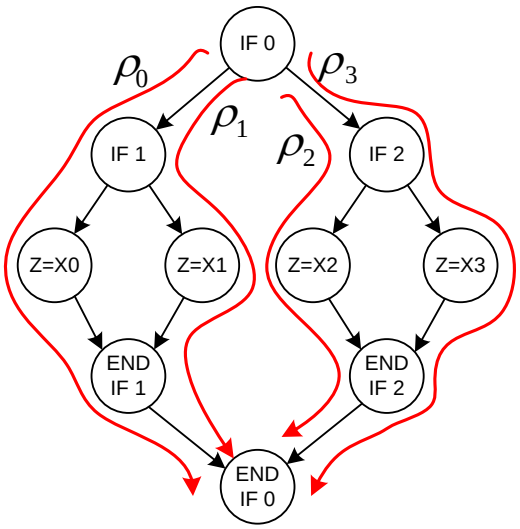


Figura 5.7: Exemplo de percurso numa FSMD

$$z = \begin{cases} g_0(X) = X_0 & \text{com probabilidade } P_0 = \rho_0 \\ g_1(X) = X_1 & \text{com probabilidade } P_1 = \rho_1 \\ g_2(X) = X_2 & \text{com probabilidade } P_2 = \rho_2 \\ g_3(X) = X_3 & \text{com probabilidade } P_3 = \rho_3 \end{cases}$$

onde Z é a saída do circuito; X_0,X_1,X_2 e X_3 são as entradas do circuito e $P_i = \rho_i$ corresponde à probabilidade de acontecer cada caminho obtida pela observação do *Comando de Controle Condicional*. Por comodidade, usamos a variável α_i tanto para a identificação do caminho como para a probabilidade de sua ocorrência.

Todas as equações encontradas durante a análise da FSMD são armazenadas numa base de dados, com sua respectiva probabilidade de ocorrência, como explicado na seção 5.1.2 deste capítulo.

2. Cálculo da PD por evento de cada uma das saídas. Para realizar esta tarefa, com cada equação obtida no passo anterior, gera-se um grafo do fluxo de dados (DFG, do inglês *Data Flow Graph*) correspondente, que explicita a dependência das funções de saída às entradas.

5.2.1 Extração de equação das saídas

Como comentado anteriormente, a primeira parte do algoritmo para a obtenção das distribuições de saída de uma FSMD a partir de distribuições de entrada, consiste da extração das equações de cada saída. Para entender a metodologia adotada, vamos primeiramente mostrar como as equações são armazenadas e como a FSMD é percorrida. Só então, apre-

Tabela 5.5: Membros da Estrutura ***CompStruct***

Membros	Caraterísticas
MultConst	Constante da multiplicação
NumOfInputs	Número de entradas envolvidas no cômputo do componente
InputAndExp	Arranjo da estrutura ***ElementStruct*** .

sentaremos na seção 5.2.1.5 o algoritmo EVALUATE() que gera as equações das saídas.

5.2.1.1 Representação e armazenamento das equações

À medida que o algoritmo que percorre os nós da FSMD evolui, equações vão sendo montadas e armazenadas para as diversas variáveis. Uma saída z qualquer deverá ser capaz de armazenar equações do seguinte tipo:

$$z = comp_0 + comp_1 + \cdots + comp_{m-1}$$

em que cada componente é da forma

$$comp_i = C_i X_{i_0}^{i_{c_0}} * X_{i_1}^{i_{c_1}} * \ldots$$

A estrutura ***CompStruct***, ilustrada na tabela 5.5, representa as características do componente de uma equação. Considerando-o na forma $CX_0^{c_0}X_1^{c_1}\ldots$ C é uma constante multiplicativa (MultConst); se X_0, X_1, etc. são as entradas do modelo envolvidas no cômputo do componente, então o número de entradas é registrado em NumOfInputs (2, neste caso).

Da expressão acima, c_i corresponde aos expoentes das variáveis X_i. A representação do conjunto $X_i^{c_i}$ é feita através da estrutura ***ElementStruct*** (tabela 5.6), onde Input= X_i e Exp= p_i. Para armazenar a equação de uma variável a classe ***VarClass*** é definida (Tabela 5.7). O primeiro membro (NumOfComp) define o número de componentes que a equação associada possui. O segundo membro (Comps) tem armazenado os componentes da equação, composto por um arranjo da estrutura ***CompStruct***, visto anteriormente. O último membro (Rslt) armazena a PD da equação, seja no cômputo de alguma saída ou quando está se procurando a probabilidade de caminhos.

Como mencionado no início da seção, as saídas podem ser definidas por mais de uma equação, correspondentes a caminhos gerados pelas estruturas condicionais. Ademais, cada equação é acompanhada pela probabilidade de ocorrência do caminho correspondente.

Tabela 5.6: Membros da Estrutura ***ElementStruct***

Membros	Caraterísticas
Input	Ponteiro à entrada do modelo
Exp	Valor do expoente da entrada do modelo

Tabela 5.7: Membros da Classe ***VarClass***

Membros	Caraterísticas
NumOfComp	Número de componentes que determinam a equação
Comps	Arranjo da estrutura ***CompStruct***
Rslt	Contém a densidade probabilística de cada intervalo dos possíveis valores da equação.

Como a classe ***VarClass*** é restrita a uma única equação, estabelece-se, então, a classe ***VarOutClass*** (tabela 5.8). O primeiro membro (NumOfPathEq) define o número de equações, portanto o número de caminhos que determina a saída. O segundo membro da classe (PathEq) é um arranjo de equações que são declarados com a classe ***VarClass***. O terceiro membro contém a probabilidade de ocorrência de cada um dos caminhos, definido por NumOfPathEq. O último armazena a PD resultante da sobreposição das PDs com sua as respectivas probabilidades de ocorrência, de todos os caminhos de execução da FSMD como apresentado no capítulo anterior (Seção 4.4.1, página 56).

Exemplificando a estrutura de armazenamento de dados, vamos assumir que as equações associadas a uma saída Z, são tal, que

$$Z = \begin{cases} 2 * X3 + 3X4 \\ 0,2 * X1 * X2 + 0,2 * X3 * X3 + 0,8 \end{cases} \tag{5.1}$$

A Figura 5.8 ilustra a representação da equação anterior na estrutura *VarOutClass*, onde fica evidente:

Tabela 5.8: Membros da Classe ***VarOutClass***

Membros	Caraterísticas
NumOfPathEq	Número de equações que determina o valor de saída do circuito
PathEq	Arranjo de equações (utiliza a Classe ***VarClass***)
PathProb	Arranjo que corresponde à probabilidade de ocorrência de cada um dos caminhos
Rslt	Contém o perfil (PD) da saída. Sobreposição (união) das distribuições de cada um dos caminhos possíveis com sua respectiva probabilidade de ocorrência.

1. Número de variáveis: participam na primeira equação, duas variáveis, X_3 e, X_4, enquanto que na segunda equação são três variáveis, X_1, X_2 e , X_3.

2. Número de componentes: para a primeira equação, há dois componentes, $2 * X_3$ e $3 * X_4$, enquanto que, para segunda, existem três, $0,2 * X_1 * X_2$, $0,2X_3 * X_3$ e $0,8$.

5.2.1.2 Análise dos comandos de execução

Os comandos de execução são encontrados num programa quando da designação de valores. Estes utilizam a sintaxe apresentada na figura 5.1 (página 71), onde *"op"* indica os operadores aritméticos que o comando pode utilizar. Estas considerações permitiram desenvolver a classe ***ExeCmdClass*** 5.1 apresentada na tabela 5.1 e explicada na seção 5.1.1(da mesma página 71).

Ao se percorrer uma FSMD, toda vez que se depara com um nó de designação de valor para uma ou mais variáveis, o cômputo da equação da(s) variável(eis) correspondente(s) é realizado pela função FindEq(), através do recurso de composição de equações, como explicado a seguir.

FindEq(n, ρ): encarrega-se de gerar a equação resultante de composição da expressão armazenada no nó de operação n. Caso esta resposta seja armazenada numa variável de saída, a probabilidade de ocorrência ρ (que é a probabiliade do caminho) é armazenada. Para entender o mecanismo de composição utilizado, na figura 5.9 temos um exemplo de dois nós em sequência. Quando o primeiro nó é alcançado, a equação da variável $\underline{T}$ é computada e armazenada; quando o segundo nó é alcançado, $\underline{Z}$ é computado através da composição de $\underline{T}$, ao se checar que a variável $\underline{T}$ já tem uma equação associada.

Por outro lado tendo-se em conta os dois tipos de variáveis, internas e de saída, como visto na seção anterior, a função FindEq() deve atuar de duas formas diferentes. A razão para isto é que o algoritmo de busca percorre um caminho até a saída do circuito, quando as informações da equação correspondente a este caminho são armazenadas; para a busca de um outro caminho, por este ser independente do primeiro a partir do último ponto (nó) de desvio, os dados do caminho anterior são desconsiderados. As duas formas são:

1. ***Z é variável interna:*** neste caso a estrutura *VarClass* deve manter somente a última equação com que foi atualizada. Por exemplo, na figura 5.10 existem dois caminhos de análise. Assumindo que Z é uma variável interna, então na análise do primeiro caminho ($Z = X0$, a base de dados de Z é a apresentada na figura 5.11(a). Daí o algoritmo prossegue com a busca para os nós sucessores. Depois, quando se faz

$$
\begin{cases}
NumOfPathEq = 2 \\
PathEq = \begin{cases}
NumOfComp = 2 \\
Comps = \begin{cases}
\begin{cases} MultCnst & = 2 \\ NumOfInputs & = 1 \\ InputAndExp & = \begin{cases} \begin{cases} Input & = X3 \\ Exp & = 1 \end{cases} \end{cases} \end{cases} \\
\begin{cases} MultCnst & = 3 \\ NumOfInputs & = 1 \\ InputAndExp & = \begin{cases} \begin{cases} Input & = X4 \\ Exp & = 1 \end{cases} \end{cases} \end{cases}
\end{cases} \\
\begin{cases}
NumOfComp = 3 \\
Comps = \begin{cases}
\boxed{\begin{cases} MultCnst & = 0,2 \\ NumOfInputs & = 2 \\ InputAndExp & = \begin{cases} \boxed{\begin{cases} Input & = X1 \\ Exp & = 1 \end{cases}} \\ \boxed{\begin{cases} Input & = X2 \\ Exp & = 1 \end{cases}} \end{cases} \end{cases}} \\
\boxed{\begin{cases} MultCnst & = 0,2 \\ NumOfInputs & = 1 \\ InputAndExp & = \begin{cases} \boxed{\begin{cases} Input & = X3 \\ Exp & = 2 \end{cases}} \end{cases} \end{cases}} \\
\boxed{\begin{cases} MultCnst & = 0,8 \\ NumOfInputs & = 0 \\ InputAndExp & = \end{cases}}
\end{cases}
\end{cases} \\
PathProb = \begin{cases} \rho_0 \\ \rho_1 \end{cases}
\end{cases}
$$

Figura 5.8: Conteúdo do Objeto da classe $VarOutClass$ para a equação 5.1

$$T = X1 * X2 + 4$$
$$Z = T + X3 * X3$$

Figura 5.9: Exemplo de dois nós em FSMD código do programa

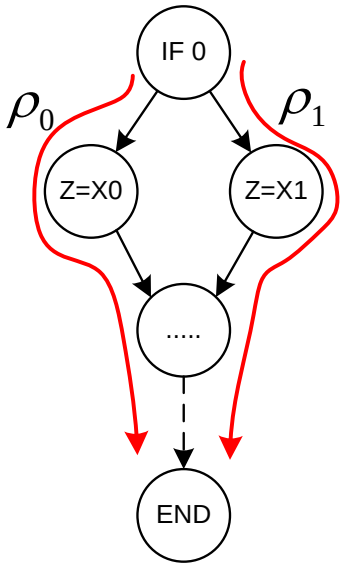


Figura 5.10: FSMD exemplo de armazenamento numa base de dados de uma variável

a análise do segundo caminho ($Z = X1$), os valores anteriores na base de dados de Z são sobrepostos, resultando nos valores apresentados na figura 5.11(b).

2. ***Z é variável de saída:*** Diferente do caso anterior, a estrutura *VarOutClass* deve armazenar todos as designações que receba. Cada uma das designações, pela forma em que o algoritmo foi planejado, corresponde a um dos caminhos que tem a FSMD. No exemplo da figura 5.10, se considerarmos que Z seja uma variável externa, a base de dados armazena a equação referente à designação pelo primeiro caminho e depois, quando do processamento do segundo caminho, acrescenta também a segunda equação. A base de dados de Z resultante é a apresentada na figura 5.12. Pode-se observar que, além de armazenar as equações, tem-se também armazenadas as probabilidade de ocorrência ρ de cada uma.

5.2.1.3 Equações de nós condicionais

Como vimos no capítulo 4, os nós condicionais dentro da FSMD determinam caminhos de execução de um programa. Na seção 5.2.1.1 foi definida a forma de representação deste tipo de nó, tendo como propriedade que uma equação (baseada em operação de comparação) é determinante para o caminho a seguir.

Foi visto também, no capítulo anterior, que os nós condicionais possuem condições independentes, sendo necessário que as PDs de cada caminho sejam computadas indepen-

$$\begin{cases} NumOfComp = 1 \\ Comps = \begin{cases} \begin{cases} MultCnst & = 1 \\ NumOfInputs & = 1 \\ InputAndExp & = \begin{cases} \begin{cases} Input & = X0 \\ Exp & = 1 \end{cases} \end{cases} \end{cases} \end{cases} \end{cases}$$

(a) Resultado do análise do primeiro caminho

$$\begin{cases} NumOfComp = 1 \\ Comps = \begin{cases} \begin{cases} MultCnst & = 1 \\ NumOfInputs & = 1 \\ InputAndExp & = \begin{cases} \begin{cases} Input & = X1 \\ Exp & = 1 \end{cases} \end{cases} \end{cases} \end{cases} \end{cases}$$

(b) Resultado da análise do segundo caminho

Figura 5.11: Base dados de Z, considerando esta uma variável interna

$$\begin{cases} NumOfPathEq = 1 \\ PathEq = \begin{cases} \begin{cases} NumOfComp = 1 \\ Comps = \begin{cases} \begin{cases} MultCnst & = 1 \\ NumOfInputs & = 1 \\ InputAndExp & = \begin{cases} \begin{cases} Input & = X0 \\ Exp & = 1 \end{cases} \end{cases} \end{cases} \end{cases} \end{cases} \\ PathProb = \begin{cases} \rho_0 \end{cases} \end{cases}$$

(a) Resultado do análise do primeiro caminho

$$\begin{cases} NumOfPathEq = 2 \\ PathEq = \begin{cases} \begin{cases} NumOfComp = 1 \\ Comps = \begin{cases} \begin{cases} MultCnst & = 1 \\ NumOfInputs & = 1 \\ InputAndExp & = \begin{cases} \begin{cases} Input & = X0 \\ Exp & = 1 \end{cases} \end{cases} \end{cases} \end{cases} \\ \begin{cases} NumOfComp = 1 \\ Comps = \begin{cases} \begin{cases} MultCnst & = 1 \\ NumOfInputs & = 1 \\ InputAndExp & = \begin{cases} \begin{cases} Input & = X1 \\ Exp & = 1 \end{cases} \end{cases} \end{cases} \end{cases} \end{cases} \\ PathProb = \begin{cases} \rho_0 \\ \rho_1 \end{cases} \end{cases}$$

(b) Resultado da análise do segundo caminho

Figura 5.12: Base dados de Z, considerando esta uma variável de saída

dentemente e adicionadas no final, junto ao nó de saída. Portanto todas as equações para um determinado caminho até a saída do circuito carregam uma etiqueta da equação correspondente ao caminho. Tal etiqueta contém a probabilidade acumulada (por operações multiplicativas) de todas as condições associadas aos condicionais que definem a ocorrencia aquele caminho.

Ao se percorrer uma FSMD, tão logo um nó condicional é alcançado, a função FOLLOWINGPATH(n,n_{next},ρ), cujo pseudo-código é apresentado na figura 5.13, é chamada. O parâmetro n é a entrada da função, indicando o nó da FSMD em processamento. As variáveis n_{next},ρ são saídas; ademais, a função retorna *falso* ao se fechar a análise de todos os caminhos de execução. A função tem como objetivo, através do análise da condição (o objeto *Expression*, da tabela 5.2, página 73) armazenada no nó n, estabelecer o caminho de execução a ser analisado (n_{next}, armazenados em *Path*), assim como probabilidade de ele acontecer (ρ). O algoritmo está dividido basicamente em três fases:

1. Inicialização de Variáveis: Nesta etapa os valores da expressão de condição são armazenados, como mostrados nas linhas de 3 até 12 da figura 5.13. A expressão segue da sintaxe da figura 5.2 (página 72), onde $InputA$ é um ponteiro a uma variável ($term1$), enquanto $InputB$ é um ponteiro a uma constante ($term2$). Como $InputA$ corresponde a uma variável, esta pode ser uma equação (como visto em 5.2.1.2) ou uma variável de entrada do circuito. Caso seja uma equação, computa-se a PD definida pela equação contida nele com a função CompPDofVar(). Esta é uma função genérica de se obter a PD de uma variável qualquer e será descrita em detalhes na seção 5.2.2.

2. Cômputo da probabilidade de ocorrência de um caminho, ρ, realizado pelas linhas 15 até 19 da figura 5.13. Observe-se que o código foi simplificado para representar com RELsymbol e inversoRELsymbol, os símbolos $[=, <, >]$ e $[\neq, \geq, \leq]$, respectivamente. De maneira similar, nas chamadas às funções, REL significa $EQ, LESS, GRE$. A figura 5.14 apresenta, por exemplo, o pesudocódigo da função COMPPROBOFLESS(c,X) que gera a probabilidade de ocorrência de $X < c$, basicamente seguindo as equações vistas na seção 4.4.1.1 do Cap.4, página 58. Pode-se notar que COMPPROBOFLESS(c,X) utiliza-se, como indicado na sua linha 6, de intervalos de valores, cuja distribuição probabilística é dada (ou previamente obtida). Todas as outras condições são construídas de maneira similar- os algoritmos correspondentes podem ser encontrados no Apêndice B.

3. Determinação do próximo nó (do caminho) a se seguir e analisar. O objetivo desta fase é passar como saída da função o nó correspondente ao caminho escolhido, como

```
FollowingPath(n,n_next,ρ)
1:
2: ///////Extrai parâmetros de cálculo
3: exp = n.expression
4: InputA = exp.InputA
5: InputB = exp.InputB
6: cond = exp.op
7:
8: ///////Computa a Distribuição da variável InputA
9: InputA.Rslt =CompPDofVar(InputA)
10:
11: ///////Extrai constante da entrada InputB
12: c =CnstVal(InputB)
13:
14: ///////Computa a Probabilidade da operação de Relação
15: if cond ='RELsymbol' then
16:    ρ =CompProbOfRel(c, InputA)
17: else if cond ='inversoRELsymbol' then
18:    ρ =1- CompProbOfRel(c, InputA)
19: end if
20:
21: ///////Analisa o valor de ρ
22: if ρ = 1 then
23:    n_next =SelTruePath(n.Path) ///////Seleciona o caminho verdadeiro
24:    return 0
25: else if ρ = 0 then
26:    n_next =SelFalsePath(n.Path) ///////Seleciona o caminho falso
27:    return 0
28: else
29:    if Enable = 1 then
30:       n_next =SelTruePath(n.Path) ///////Seleciona o caminho verdadeiro
31:       Enable = 0
32:       return 0
33:    else
34:       n_next =SelFalsePath(n.Path) ///////Seleciona o caminho falso
35:       ρ = 1 − ρ
36:       Enable = 1
37:       return 1
38:    end if
39: end if
```

Figura 5.13: Algoritmo para a seleção de caminho e sua probabilidade

```
CompProbOfLess(c, X)
1:
2: ρ = 0 ///////Inicializa a Probabilidade
3:
4: ///////Lê cada um dos intervalos de X
5: for all I ∈ X do
6:    ///////Executa a equação 4.4, página 43
7:    ρ = ρ+PDofSumSubVarCnst(c − 0.5, I.ρ, I.L, I.H)
8: end for
9: return ρ
```

Figura 5.14: Algoritmo para o cômputo da operação de relação '$X < c$'

apresentado nas linhas 22 até 38. No caso de um dos caminhos da nó de decisão ser absoluto (probabilidade 1 ou 0), então somente o caminho de probabilidade 1 é tomado. Caso contrário, primeiramente o caminho falso, isto é, de a condição não ocorrer, é selecionado (Enable é iniciado em '0') com a sua probabilidade. Quando o Enable é '1', o caminho verdadeiro com a sua probabilidade é selecionado. Isto indica que, quando a probabilidade não é absoluta, para que ambos caminhos sejam percorridos, a função deve ser chamada duas vezes, uma vez com Enable='0' e outra, com Enable='1'.

5.2.1.4 Nós de laços

Como vimos no Cap 4, os nós de Laço de uma FSMD permite repetir um número determinado de linhas de código baseados na efetivação de certas condições. Neste caso, como mencionado em 5.1.3, usam-se laços determinísticos com a sintaxe das linguagens do C/C++ sob algumas restrições (da Figura 5.2, página 72).

Baseadas nestas propriedades, as funções abaixo são definidas para a análise dos comandos de controle de laço no algoritmo geral de obtenção de equações, Evaluate(), da figura 5.15.

1. IsForLoop(): esta função determina se o nó em observação é um comando de controle de laço.

2. WasForLoopEnable(): esta função é usada para determinar se o nó ***FOR*** foi inicializado (utiliza o flag EnableFor, da tabela 5.3, na página 73. Caso o nó não tenha sido inicializado, realizam-se duas operações:

   (a) Habilitar o nó ***FOR*** com a função EnableForLoop().

   (b) Inicializar o Contador com a função SetForLoop().

3. EnableForLoop(): ativa o nó ***FOR***, mudando o valor de EnableFor para '1'.

4. SetForLoop(): inicializa o contador do laço. Para isto, tem embutida a função FindEq() -"método" da classe *ExeCmdClass*- (seção 5.2.2), com a qual faz a operação da *formula1* da figura 5.4, página 74) armazenada em OpInitial (tabela 5.3).

5. ExecuteLoop(): esta função realiza a etapa de comparação do contador do laço para definir se é caso de execução dos comandos internos do laço ou da execução daqueles que seguem o laço. Para isto tem embutida duas operações:

   (a) Selecionar o seguinte o nó a executar: com a operação de comparação *formula2* (da figura 5.4) estabelece-se o próximo comando a ser executado. O nó que deve ser analisado (Interno ou Externo) é armazenado em variável auxiliar n_{next} e a ser utilizado no algoritmo geral Evaluate(), a ser detalhada na seção 5.2.1.5.

   (b) Desabilitar nó ***FOR***: esta operação só é executada caso o laço tenha terminado de executar todas as repetições estabelecidas. Para isto a função ExecuteLoop() retorna o valor *falso*. Nesta condição a função DisableForLoop() é executada.

6. DisableForLoop(): desativa o nó ***FOR*** mudando o valor de EnableFor a '1', permitindo que o laço possa executar o estado de INICIALIZAÇÃO novamente (necessário quando se tem laços aninhados).

7. IsEndOfForLoop(): esta função processa dois estados do laço ***FOR***:

   (a) CONTAGEM: neste estado, a *formula3*, armazenada em OpStep (Tabela 5.4, página 77), é executada com a função FindEq(), uma vez que é um "método" da classe *ExeCmdClass*.

   (b) RETORNO: a função retorna em n_{next} o nó do estado ***FOR*** correspondente.

5.2.1.5 Algoritmo para a extração de equações das saídas

O algoritmo geral de geração de equações, Evaluate(), apresentado na figura 5.15, tem como objetivo a geração das equações das saídas de um circuito em função das entradas. O algoritmo tem como entrada o nó a ser processado e o correspondente do caminho então percorrido; como saída é(são) apresentada(s) a(s) equação(ões) associada(s) de cada saída primária. O algoritmo inicia-se com o nó n como o primeiro nó da FSMD e com a probabilidade de ocorrência do caminho a ser trilhado, $\rho_{acc} = 1$. O algoritmo busca realizar as tarefas de acordo com o tipo de nó que encontra, gerando os seguintes casos de acordo com a função encontrada:

```
EVALUATE(n,PO,ρ_acc)
1:
2: ///////////ANÁLISE DE FIM DO GRAFO
3:
4: ///////O nó simboliza o fim do código?
5: while ¬ENDNODE(n) do
6:
7:   ///////////ANÁLISE DOS COMANDOS DE EXECUÇÃO
8:
9:   ///////O nó é um comando de execução?
10:  if ISOPERATION(n,n_next) then
11:    FINDEQ(n, ρ_acc) ///////Encontra a Equação definida pela operação
12:
13:    ///////////ANÁLISE DOS COMANDOS DE CONTROLE CONDICIONAL
14:
15:    ///////O nó é um comando de controle condicional?
16:  else if ISCONDITIONAL(n) then
17:    while FOLLOWINGPATH(n,n_next,ρ) do
18:
19:      ///////Computa a Distribuição do caminho a ser analisado
20:      EVALUATE(n_next,PO,ρ * ρ_acc)
21:    end while
22:    ///////Todos os caminhos foram analisados, então sair de EVALUATE.
23:    break
24:
25:    ///////////ANÁLISE DOS COMANDOS DE CONTROLE DE LAÇO
26:
27:  else if ISFORLOOP(n) then
28:    if ¬WASFORLOOPENABLE(n) then
29:      ENABLEFORLOOP(n)
30:      SETFORLOOP(n)
31:    end if
32:    if ¬EXECUTELOOP(n,n_next) then
33:      DISABLEFORLOOP(n)
34:    end if
35:
36:    ///////O nó indica o fim do laço ou o fim do condicional?
37:  else if ISENDOFFORLOOP(n,n_next) then
38:    n = n_next
39:  end if
40:  n = n_next ///////Assina o seguinte nó a ser analisado
41: end while
```

Figura 5.15: Algoritmo para a Análise do Modelo e Cômputo da Distribuição de Saída

- IsOperation(n,n_{next}) (linha 10): caso n seja nó de operação (designação de valor alguma variável), a função acha o próximo nó n_{next} e retorna TRUE. Neste caso, a função FindEq(n,ρ_{acc}) vista na seção 5.2.1.2 é chamada para gerar a equação para a variável através da operação de composição de funções. No caso de a resposta da equação ser de uma saída primaria, esta deve armazenar sua probabilidade de ocorrência, ρ_{acc}, a qual será usada depois no cômputo da DP de saída.

- IsConditional(n) (linha 16): caso n seja nó condicional, retorna TRUE e segue. Neste caso, a função FollowingPath(n,n_{next},ρ) é chamada, gerando um caminho através do próximo nó n_{next} e sua probabilidade ρ, onde este corresponde a probabilidade da condição ser certa ou falsa. Como vimos na seção 5.2.1.3, caso o caminho não for absoluto (probabilidade dos caminhos forem diferentes de 0 ou 1), então, FollowingPath(n,n_{next},ρ) é chamada uma segunda vez pelo laço While associado e o segundo caminho é também gerado. Quando cada um dos caminhos é definido, a função Evaluate() é chamada recursivamente para que o caminho seja percorrido em direção à saída da FSMD.

- IsForLoop(n) (linha 27): caso n seja nó de laço, retorna TRUE e segue. Neste caso, a função WasForLoopEnable() analisa se o estado de INICIALIZAÇÃO já foi ativado no laço ***FOR***. Como foi visto na seção 5.2.1.4, tal estado corresponde à inicialização do contador de repetições. Caso aquele estado ainda não tenha sido ativado, a função EnableForLoop() ativa o flag EnableFor.

  Após esta análise, realiza-se a avaliação do contador. Neste processo, com a função ExecuteLoop(), é determinado se o contador está nos limites estabelecidos para sua execução. Esta função retorna TRUE caso o contador cumpra com os limites, retornando também o nó n_{next} que indica o próximo nó (comando) dentro do laço (ver a atualização $n = n_{next}$ que acontece na linha 38 do Evaluate()). Caso a função ExecuteLoop() retorne FALSE, o nó n_{next} contém a posição do primeiro comando fora do laço. Nesta situação, a condição da linha 31 de Evaluate() é verdadeira e a desabilitação do laço, com a função DisableForLoop(), é realizada.

- IsEndOfForLoop(n,n_{next}) (linha 37): Caso n seja nó de fim do laço, retorna TRUE. Neste caso, o contador do laço incrementa/decrementa a contagem. A função também retorna o estado n_{next}, o qual corresponde ao nó inicial ***FOR*** correspondente.

A finalização da computação de EVALUATE(n) se dá ao quando se alcançar o nó de saída da FSMD e, todas as saídas deverão estar computadas quanto às equações e à probabilidade ρ_{acc} assinalada a cada uma delas segundo o caminho seguido. Para os k caminhos existentes para se alcançar uma variável de saída, esta terá associadas a si k

equações e k probabilidades ρ_{acc}, as quais serão processadas pelo algoritmo de computo das distribuições probabilísticas de saída CompOutProbDist().

5.2.2 Cômputo da distribuição de saída

Para o cômputo das distribuições de saída utiliza-se a equação ou as equações associadas a cada uma das variáveis de saída. Uma determinada equação é gerada na forma descrita na seção anterior. O algoritmo que calcula a distribuição de cada uma das equações de cada variável de saída é dado pela função CompPDofVar(), apresentada na Figura 5.16.

O algoritmo considera que a estrutura da equação é dada através de componentes na seguinte forma:

$$z = comp_0 + comp_1 + \cdots + comp_{m-1}$$

e

$$comp_i = C_i * X_{i_0}^{i_{c_0}} * X_{i_1}^{i_{c_1}} * \cdots = C_i * X_{i_0}^{'} * X_{i_1}^{'} * \ldots \tag{5.2}$$

onde cada X' é conhecido como elemento. Os componentes da equação são independentes, ou seja, as entradas envolvidas no cálculo do componente $comp_i$ são diferentes de $comp_j$, $\forall i \neq j$. Uma equação de uma variável de saída pode ser representado em forma de grafo, como apresentado na figura 5.17 para auxiliar no entendimento de como a equação é processada.

A função CompPDofVar() tem como objetivo analisar uma equação associada a uma variável e calcular a sua PD. Para este propósito, o algoritmo primeiro faz o cálculo da PD de cada componente ($comp_i$) considerados somente os conceitos de cálculo da PD de uma multiplicação. Estes resultados são considerados entradas de uma seguinte etapa subsequente. Os detalhes de cada uma das etapas do algoritmo CompPDofVar() são comentadas a seguir.

1. *Cômputo da PD Individual dos Componentes*: esta etapa encarrega-se de calcular a PD de cada um dos componentes. Esta tarefa é realizada pelos comandos entre as linhas 4 a 28, que geram um DFG formado por nós de multiplicação, onde as entradas corresponde a um elemento do componente ($X_i^{'}$), como o apresentado na figura 5.18. A partir deste grafo o cálculo da PD do componente é a realizada com a função CompMultTreeDist(). Cada uma das PDs calculadas para cada um dos componentes é armazenada para ser usada na segunda etapa do cálculo da PD da

```
CompPDofVar(Var)
1:
2: NumOfComp = |Var.Comp|
3: A_t =GenAddTree(NumOfComp)
4: for all comp ∈ |Var.Comp| do
5:   NumOfElem = |comp.Elem|
6:   M_t =GenMultTree(NumOfElem)
7:   for all elem ∈ |comp.Elem| do
8:
9:     ///////Computa a nova distribuição de entrada do componente
10:
11:    if elem.exp > 1 then
12:      X = elem.var ///////Computo da distribuição X' = X^exp
13:      exp = elem.exp
14:
15:      ///////Detalhes na secção 4.2.2.3, página 4.2.2.3
16:      X'.Rslt =ProbDistOfXn(X,exp)
17:    else
18:      X = elem.var
19:      X'.Rslt = X.Rslt
20:    end if
21:
22:    ///////Estabelece a nova distribuição de entrada do componente
23:    SaveInputDist(X')
24:  end for
25:
26:  ///////Computa a distribuição do componente
27:  ProDist =CompMultTreeDist(M_t)
28:
29:  ///////Estabelece a distribuição de entrada do arvore de somadores
30:  SetAsInputDist(A_t,ProDist)
31: end for
32:
33: ///////Computa a distribuição da equação de saída
34: return CompAddTreeDist(A_t)
```

Figura 5.16: Algoritmo para o computo da Distribuição Probabilísticas de uma variável

equação. Uma explicação detalhada das funções utilizadas no algoritmo é apresentada a seguir continuação.

(a) GenMultTree: gera o DFG, cujos nós de entrada são os dados compostos por elementos X' (como mostrado na Figura 5.19). Observe que cada elemento X' corresponde a X^c, potência de X declarada.

(b) ProbDistOfXn: Esta função calcula a nova distribuição de uma das entradas do componente gerada pela expressão X^c, onde X é uma das entradas do circuito e c é uma constante $\in \mathbb{Z}^+$. O algoritmo correspondente em pseudo-código é apresentado nas figuras 5.20 e 5.21. Basicamente, para o cômputo de X^c, os diversos intervalos na entrada X com valores diferenciados de frequência dos dados são processados e a PD da multiplicação entre valores de cada dois intervalos é computado e acumulado. O cômputo das PDs é realizado com os conceitos apresentados na seção 4.2.2.3 (capitulo 4, página 49).

(c) CompMultTreeDist: computa a distribuição do componente quando todas as PDs de suas entradas tiverem sido estabelecidas. O algoritmo da função é apresentada no Apêndice C. Para este propósito, o algoritmo para computar a PD de uma multiplicação de duas variáveis independentes é utilizada, sendo apresentada no apêndice C (Figura C.2). Pode-se ter os casos de uma multiplicação de variável e uma constante (mostrado diretamente na Figura C.2), ou entre duas variáveis aleatória, os quais são tratados dentro do algoritmo PDofMult com a função PDofMultVarVar (algoritmo mostrada na Figura D.43 do apêndice D). O cômputo do PD é realizado com os conceitos apresentados na seção 4.2.2 (capitulo 4, página 46) e detalhes dos algoritmos estão no apêndice D.

2. *Cômputo da PD da Equação*: Para esta etapa já se deve ter-se gerado uma árvore de somadores, como na Figura 5.22, onde é armazenado, em cada um de seus nós de entrada, o resultado de cada execução de CompMultTreeDist, na forma apresentada na Figura 5.23. Através do cálculo da PD da soma encadeada de cada um dos componentes, pode-se calcular a PD da equação de saída. Para este propósito, a função CompAddTreeDist é utilizada e o algoritmo correspondente é apresentado no Apêndice C (na figura C.3). O procedimento utilizado é similar ao CompMultTreeDist, mas com a diferença de que o operador usado é o de soma, tarefa realizada pelo algoritmo da figura C.4 do Apêndice C.

Após a computação de uma equação associada a uma variável de saída, o processo deve ser repetido para outras equações da mesma variável, caso existam, devido a diferentes caminhos que foram computados. A PD total deverá ser computado neste caso como

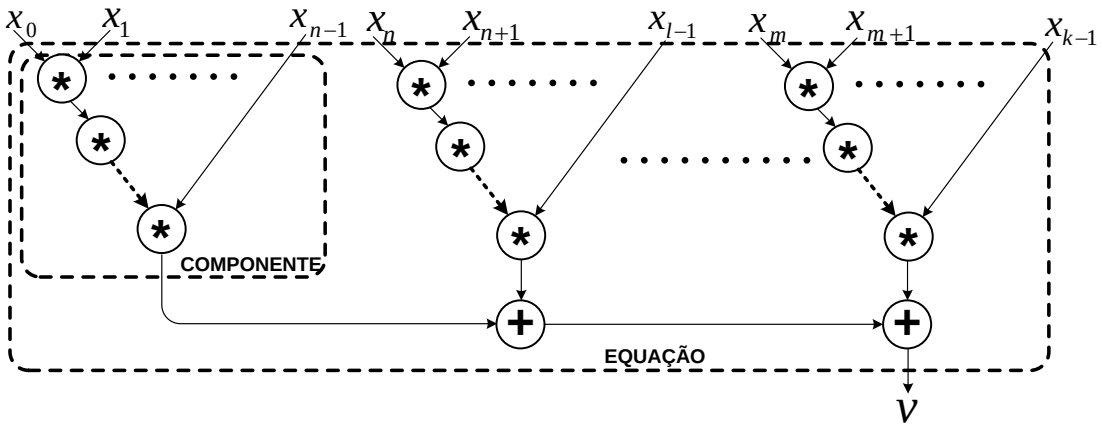


Figura 5.17: Exemplo de equação representada em grafos

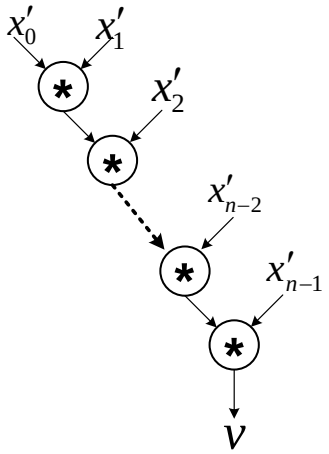


Figura 5.18: Árvore de multiplicadores para gerar os componentes da equação

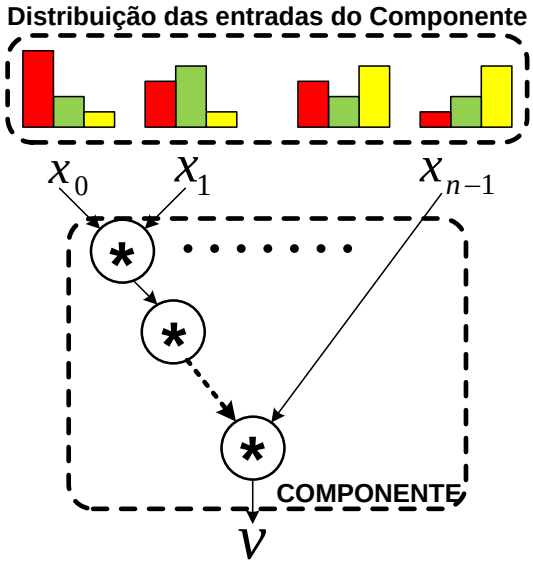


Figura 5.19: Exemplo de cômputo, em forma de grafos, da distribuição do componente

a soma do produto da PD de cada equação que define a saída ($F_i(v)$) com sua respectiva probabilidade de ocorrência (ρ_i). A equação 5.3 representa isto, onde L determina o número de caminhos.

$$F(v) = \sum_{0}^{L-1} \rho_i F_i(v) \tag{5.3}$$

O algoritmo CompOutProbDist. da Figura C.5, é o algoritmo topo que faz a busca da PD de todas as variáveis de saída obtidas pelo algoritmo de montagem das equações, Evaluate(), da seção 5.2.1.5 deste capítulo.

```
ProbDistOfXn(X,n)
1:
2: ///////Define os valores dos intervalos resultante X' = X^n,
3: ///////mas o valor da distribuição ρ de cada um é 0
4: X' =DefineIntervalsOfXn(X,n)
5:
6: ///////Percorre cada um dos intervalos que define a Distribuição
7: ///////de Probabilidade da entrada X
8: for all I_x' ∈ X' do
9:   for all I_x ∈ X do
10:     PD.L =PDofXn(I_x'.L − 0.5, I_x, n)
11:     PD.H =PDofXn(I_x'.H + 0.5, I_x, n)
12:     PD.ρ = (PD.H−PD.L) / (I_x'.H−I_x'.L)
13:     X'.ρ = X'.ρ + PD.ρ
14:   end for
15: end for
16: return X'
```

Figura 5.20: Algoritmo para calcular a PD Resultante de X^n

```
PDofXn(v',Interval,n)
1:
2: ///////n é impar?
3: if IsOdd(n) then
4:   v0 = Interval.L^n ///////Define o limite inferior do novo intervalo
5:   v1 = Interval.H^n ///////Define o limite superior do novo intervalo
6:
7:   ///////Computa o PD com a equação 4.11, página 49
8:   return PDofXn2(v',v0,v1,n)
9:
10:  ///////n é par
11: else
12:
13:  ///////Estabelece limites do novo intervalo, baseado na equação 4.13, página 50
14:  if |a| < |b| then
15:    v0 = Interval.L^n
16:    v1 = Interval.H^n
17:  else
18:    v0 = Interval.H^n
19:    v1 = Interval.L^n
20:  end if
21:
22:  ///////Computa o PD com a equação 4.12, página 50
23:  if Interval.L < 0 ∧ 0 < Interval.H then
24:    return PDofXn1(v',v0,v1,n)
25:  else
26:    return PDofXn2(v',v0,v1,n)
27:  end if
28: end if
```

Figura 5.21: Algoritmo para calcular a PD de v' no conjunto definido por X^n

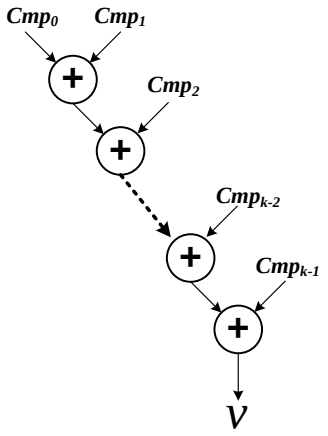


Figura 5.22: Árvore para somar os Componentes da Equação

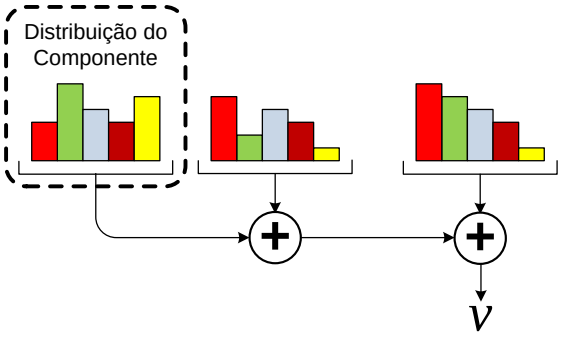


Figura 5.23: Exemplo de equação representada em grafos

6 Resultados

Neste capítulo são apresentados os resultados experimentais na aplicação da metodologia e no uso das ferramentas desenvolvidas para o cômputo da cobertura de saída por distribuições probabilísticas, baseados nos algoritmos apresentados no capítulo 5 e fundamentados no capítulo 4.

6.1 Implementação da metodologia de cômputo da distribuição de saída

No desenvolvimento da ferramenta projetada nesta dissertação, decidiu-se pela utilização da linguagem C++ que permite o uso de classes e objetos na sua implementação, além de possibilitar a expansão das "propriedades" que possam ter. A etapa de captura do código HDL foi realizada manualmente por razões práticas, uma vez que a implementação de um *parser* demandaria muito tempo e seria secundário dentro dos objetivos da dissertação.

Os algoritmos EVALUATE() e COMPOUTPROBDIST() foram implementados de forma apresentada no capítulo 5, onde o primeiro faz a análise da FSMD que representa ao modelo do circuito, gerando no processo uma equação de saída. Esta é processada por COMPOUTPROBDIST() quem apresenta, com valores probabilísticos, a relação entrada-saída do circuito.

6.2 Metodologia experimental

Para a avaliação da eficiência do modelo de cobertura proposto, gerado pela ferramenta ProCov, ela é comparada com modelos manuais (Distribuição Uniforme, onde é considerado que todos os eventos têm a mesma probabilidade de ocorrência e os intervalos para os eventos são definidos pela experiência do projetista). Os objetivos da metodologia experimental delineada nesta dissertação consistem em mostrar:

1. a correspondência e similaridade que o modelo de cobertura de saída deve ter com

a resposta do sistema ao se realizar uma simulação. Esta comparação pode ser feita entre o número vetores de teste por evento que a ferramenta desenvolvida calcula (pré-simulação) e o número de vetores requerido para a obtenção de 100% da cobertura nos eventos na simulação do modelo comportamental do sistema (pós-simulação).

2. a eficiência do modelo de cobertura de saída proposto nesta dissertação em termos de tempo de simulação. Isto é verificado com a observação que o tempo levado para atingir 100% da cobertura com o modelo gerado é menor que o requerido com o modelo manual de distribuição uniforme.

O fluxo de ações para alcançar os resultados dentro dos objetivos acima pode ser representado na figura 6.1. Nela, observa-se a presença de blocos marcados em cinza que estão associados às seguintes tarefas principais:

1. Definição dos experimentos
2. Cômputo da Cobertura e Geração do Modelo de Cobertura
3. Simulação com os modelos
4. Levantamento e Análise de resultados

Estas etapas são explicadas e detalhadas nas seções a seguir.

6.2.1 Definição dos experimentos

Esta etapa está associada ao bloco em cinza mais ao topo da figura 6.1, sendo nela definidos:

1. Modelo do circuito: trata-se da representação em alto nível do circuito que se deseja analisar. Como mencionado em seções anteriores, a extração das FSMDs correspondentes é feita manualmente. Os circuitos usados são:

   FIR: filtro de resposta finita a impulso (finite impulse response filter) de 16 coeficientes e saída única, cuja estrutura é composta por laços e operações aritméticas de soma e multiplicação. O circuito tem uma entrada e quinze registradores fazendo com que possa ser interpretado como um circuito de dezesseis entradas independentes, uma vez que os valores de cada um dos registradores são independentes no tempo.

   Elliptic: filtro de onda elíptica composto por oito entradas e oito saídas. A estrutura é composta somente por operadores aritméticos fazendo interatuar as entradas do modelo, gerando diferentes equações em cada uma das saídas.

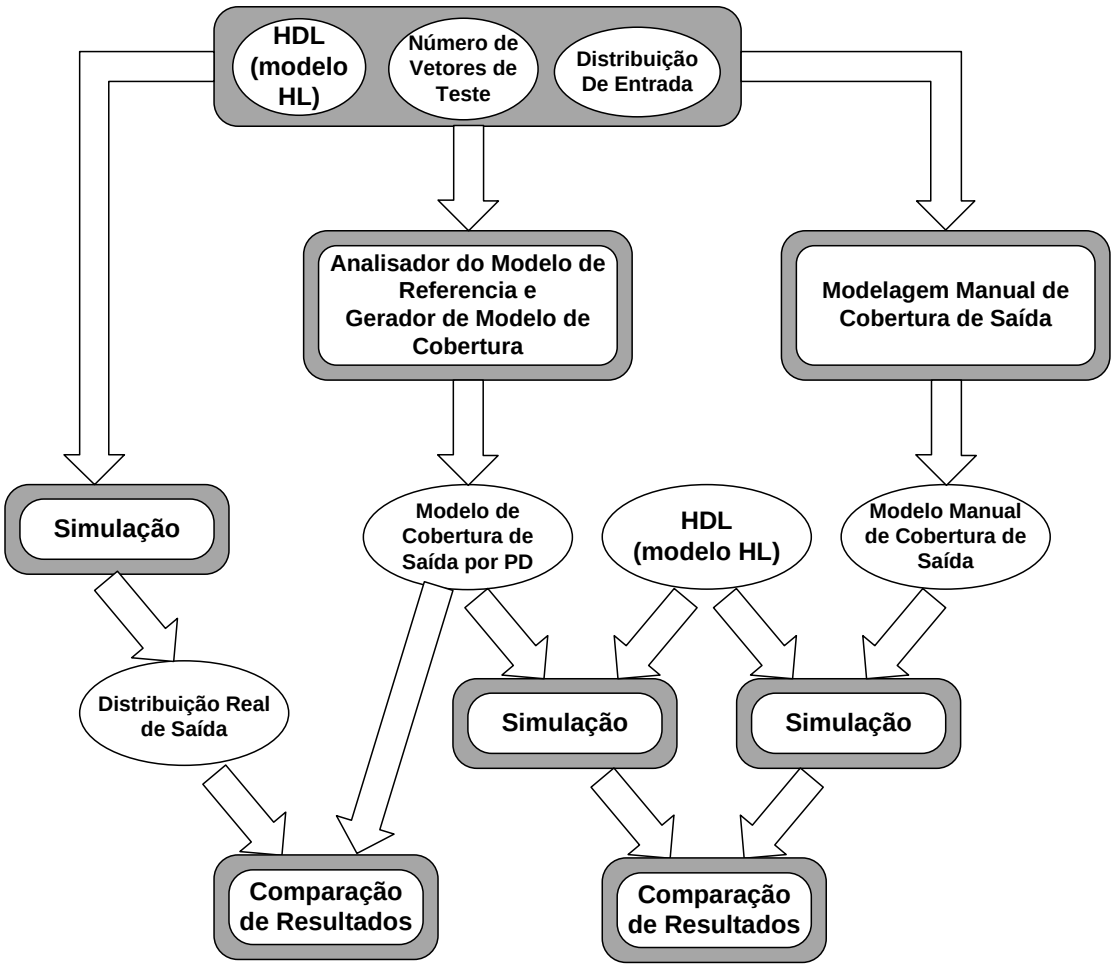


Figura 6.1: Fluxo de teste da metodologia

FFT4: cálculo da Transformada Rápida de Fourier (do inglês Fast Fourier Transform) de quatro pontos de entrada (Real e Imaginário) o que faz que sejam oito entradas e portanto tem-se oito saídas.

2. A distribuição de entrada: esta é uma tarefa usual ao se realizar o planejamento da verificação e montagem do *testbench*; considera-se os possíveis valores que cada uma das entradas do circuito pode ter, designando-se probabilidades aos valores que vão ser utilizados na simulação. Nesta dissertação considera-se a forma mais básica em que cada uma das entradas é composta por somente um intervalo com distribuição uniforme. A adoção deste modelo de distribuição de entrada é conveniente, já que facilita a demostração da funcionalidade da ferramenta; ademais, a mudança dos valores não afeta o procedimento de avaliação, pois somente mudaria a distribuição probabilística da saída, que por sua vez pode ser calculado pela ferramenta. Os valores selecionados para os três circuitos-exemplo são apresentados na Tabela 6.1, com uma, oito e oito saídas consideradas, respectivamente. Para cada saída é dado o limite inferior e superior do conjunto de valores válidos considerados e o valor probabilístico de cada intervalo de evento. Como todos os números inteiros são incluídos, o número de pontos, assim como de intervalos aos seus redores, fica implicitamente definido. Por exemplo, para qualquer saída do exemplo Elliptic, pelos limites inferior e superior, o espaço de valores de saída correspondente é dividido em de 65.536 pontos ou intervalos, o que corresponde, em uma distribuição uniforme, à probabilidade de $1,53x10^{-5}$ aproximadamente para cada ponto.

3. O número de vetores que se deseja observar: a definição do número de vetores estabelece o quanto do espaço funcional é efetivamente analisado em um *testbench*. Para se ter uma avaliação melhor do número adequado, a quantidade é escolhida pela observação dos intervalos gerados e das distribuições obtidas após a aplicação da ferramenta para a obtenção da densidade de probabilidades. Da nossa análise, para se avaliar a metodologia foi selecionado o valor de 10 milhões de vetores de teste. Resultados da aplicação da ferramenta levaram a observação de que alguns eventos têm uma distribuição igual a 10^{-7} %, significando que se precisaria aproximadamente de 10^9 vetores de teste para se observar pelo menos um evento extremo. Este número é bastante elevado considerando-se que o experimento é repetido várias vezes durante o processo de avaliação; foi escolhido, então, o valor já mencionado de 10 milhões, que garantiria a avaliação tanto do comportamento desejado como também da ferramenta projetada.

Tabela 6.1: Definição de cada saída dos modelos de referência a serem usadas

Modelo	Entrada					Saída
do Circuito	Posição	Nome	Limite Inferior	Limite Superior	Densidade	
FIR	0	data_in	-128	127	0,003922	data_out
Elliptic	0	inp	-32768	32767	$1,53x10^{-5}$	outp
	1	sv2	-32768	32767	$1,53x10^{-5}$	sv2_o
	2	sv13	-32768	32767	$1,53x10^{-5}$	sv13_o
	3	sv18	-32768	32767	$1,53x10^{-5}$	sv18_o
	4	sv26	-32768	32767	$1,53x10^{-5}$	sv26_o
	5	sv33	-32768	32767	$1,53x10^{-5}$	sv33_o
	6	sv38	-32768	32767	$1,53x10^{-5}$	sv38_o
	7	sv39	-32768	32767	$1,53x10^{-5}$	sv39_o
FFT4	0	ar[0]	-128	127	0,003922	Afr[0]
	1	ar[1]	-128	127	0,003922	Afr[1]
	2	ar[2]	-128	127	0,003922	Afr[2]
	3	ar[3]	-128	127	0,003922	Afr[3]
	4	ai[0]	-128	127	0,003922	Afi[0]
	5	ai[1]	-128	127	0,003922	Afi[1]
	6	ai[2]	-128	127	0,003922	Afi[2]
	7	ai[3]	-128	127	0,003922	Afi[3]

6.2.2 Cômputo da cobertura e geração dos modelos de cobertura

A etapa de cômputo e geração de modelos de cobertura envolve as modelagens da metodologia proposta e a manual (as duas caixas cinzas da direita na segunda fileira, do alto para baixo, da figura 6.1). No cômputo da cobertura de saída têm-se em conta os intervalos (eventos) designados a cada uma das entradas e as suas densidades probabilísticas, como descrito no item 2) da seção anterior. De outro lado, já no processamento do algoritmo (figura 5.15), é considerado que o número máximo de intervalos possíveis para a distribuição das variáveis em cada etapa de cálculo é 20.000, o que corresponderia ao número máximo considerado de eventos. Este valor, como é explicado no capítulo 4, é o fator de resolução para reduzir os erros de cálculo, isto é, com uma maior resolução o erro de cálculo diminui. Para verificar de que maneira a redução da resolução poderia afetar o modelo probabilístico da resposta de saída e a precisão dos cálculos, foram realizadas outras simulações com os mesmos casos de teste, mas variando os valores de resolução. Os resultados referentes a estes experimentos são apresentados na secção 6.3.3.

O fato de que o algoritmo apresentado na figura 5.15 utiliza 20.000 intervalos para a computação das PDs implica que ele geraria, ao final da computação, também 20.000 eventos a serem observados em cada saída. Esta quantidade é muito elevada para um modelo de cobertura por itens e também para se avaliar se modelo computado pela ferramenta desen-

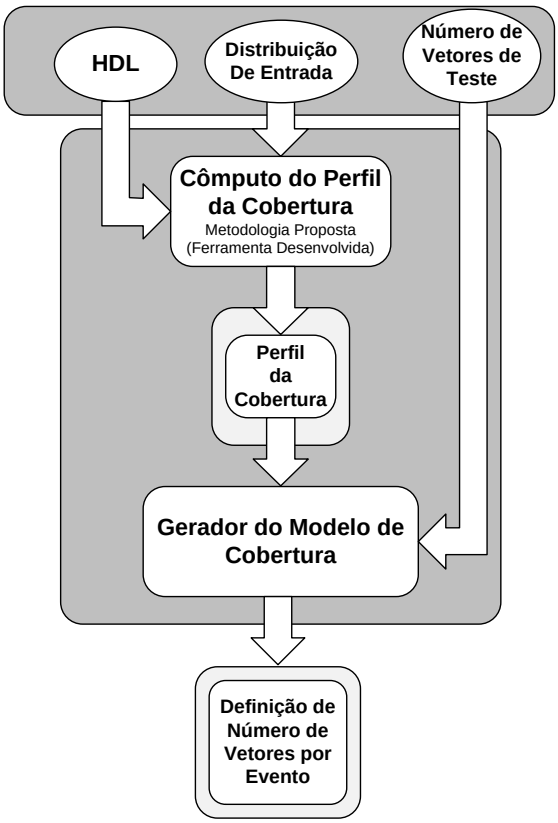


Figura 6.2: Cômputo do Perfil de Cobertura e Geração de Modelos de Cobertura

volvida é similar ao perfil gerado através da simulação. Para o ajuste e redução do número final de eventos, é adicionada, ao final do processamento, uma etapa de agrupamento que resulta em 20 eventos para cada saída, mostrada na Figura 6.2. Após a redução do número de eventos e computação das suas respectivas densidades, a ferramenta proposta gera o modelo de cobertura final. Em paralelo, considerando os novos eventos estabelecidos, o modelo manual de distribuição uniforme também é gerado (com a mesma densidade probabilística geral). A partir dos diferentes modelos de cobertura, é possível fazer uma comparação das duas metodologias. Nos dois casos emprega-se o número de vetores que se deseja observar (estabelecido no item 3 da seção anterior) para se calcular quantos vetores deve-se observar em cada um dos eventos aproximadamente.

6.2.3 Simulação do modelo de referência e avaliação dos modelos de cobertura

Quando falamos da etapa de simulação nos referimos não só à caixa cinza da esquerda na segunda fileira, do alto para baixo, da figura 6.1 para o modelo em alto nível do processo de verificação, mas também implicitamente, àquelas realizadas tendo como critérios de cobertura os modelos da seção anterior 6.2.2. Para demonstrar que a metodologia proposta pode gerar um modelo de cobertura adequado, ou seja, capaz de predizer razoavelmente

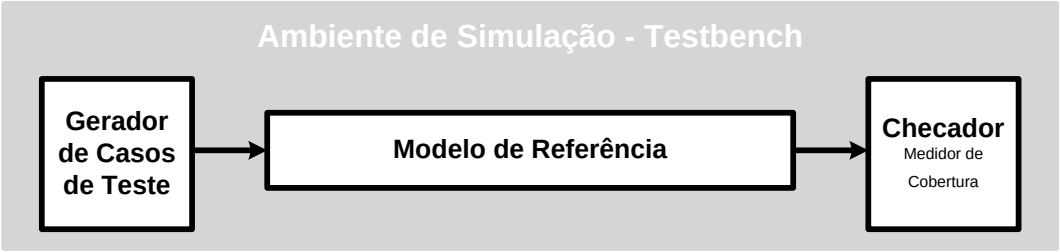


Figura 6.3: Testbench simplificado para validação da metodologia

a distribuição de saída do circuito, faz-se a simulação do circuito utilizando-se geração de vetores aleatórios na entrada do modelo de referência de alto-nível. Para isto foi implementado um *testbench* simplificado como apresentado na figura 6.3, acelerando-se o processo de simulação, uma vez que deseja-se apenas avaliar se o modelo gerado pela ferramenta é similar ao obtido no processo de simulação e não verificar o funcionamento do DUV em si.

A segunda comparação a ser feita é entre o modelo de cobertura obtido com a metodologia desenvolvida nesta dissertação e o modelo manual com distribuição uniforme; deve-se, neste caso, comparar o tempo (medido em número de vetores de teste[1] que a execução de testbench em cada caso leva para atingir o nível de 100% da cobertura. Como para a modelagem uniforme o tempo necessário para obter o 100% de cobertura mostrou ser muito alto (a demostração teórica e os cálculos para os casos de teste na seção 6.3.2), estabeleceu-se que as simulações dos modelos ocorreria da seguinte forma:

Modelo Manual: deve-se limitar o número total de vetores de teste a serem gerados no processo de simulação. Estes devem permitir observar a tendência da progressão da cobertura, embora os 10.000.000 de itens de cobertura para cada evento não foram possíveis de serem atingidos. Para este propósito estabelece-se então que o número total de vetores que deve-se gerar é 60.000.000 (6 vezes do valor de itens estabelecido como 100% da cobertura individual)

Modelo Proposto: deve-se atingir o índice de 100% com um total de 10.000.000 itens a serem contados. Pode-se observar que para atingir este objetivo precisa-se, na prática, gerar mais vetores de teste que o desejado durante a simulação, mas que esta variação é pouco relevante.

Ao finalizar a simulação são gerados dois arquivos:

Contagem de Itens por Evento: este arquivo contém o número de itens que foram contabilizados para cada evento ao se finalizar a simulação. Para o modelo proposto

[1]Não é calculado em unidade absoluta de tempo porque os valores podem ser influenciados pela capacidade do computador empregado.

nesta dissertação, a contagem refere-se ao momento em que 100% da cobertura é alcançado enquanto que, para o modelo manual, trata-se do momento da ocorrência de 60 milhões de vetores de teste.

Progressão de Cobertura: este arquivo é usado na observação da progressão da cobertura durante a simulação.

Um dos aspectos serem avaliados para o processo de modelagem é o tempo para a sua criação. Na modelagem proposta, este tempo depende não só da complexidade do modelo do circuito, como também da resolução utilizada para o seu processamento. Um elevado valor deste parâmetro diminui o erro de cálculo, mas ao mesmo tempo, eleva o tempo de computação. A questão que surge é qual a resolução que se deve usar para modelar a(s) PD da(s) saída(s) do circuito. Para o estudo da influência da resolução sobre a precisão do modelo de cobertura obtido e sobre o tempo necessário para se alcançar o índice de 100% da cobertura, três valores de resolução foram testados. Esta comparação é realizada na seção 6.3.3.

6.2.4 Levantamento e análise de resultados

Os resultados são obtidos pelas comparações propostas no fluxo da figura 6.1, indicados nos blocos cinzas inferiores. Para mostrar a eficiência da metodologia proposta as seguintes comparações são realizadas:

Consistência do modelo gerado: mostrando que o modelo de cobertura gerado apresenta perfil equivalente à resposta do modelo do circuito em alto-nível quando simulado.

Tempo de simulação: mostrando a eficiência do modelo gerado comparado aos modelos manuais. Espera-se observar que o nosso modelo requeira menor número de vetores de teste para atingir 100% de cobertura.

Variação do modelo ante variação da resolução: mostrar como o modelo de saída gerado é afetado quando a resolução para seu cálculo muda.

6.3 Resultados dos experimentos

Nesta seção são apresentados os resultados das comparações propostas. Todas as saídas dos casos de teste são consideradas, mas, uma vez que se verificou um comportamento similar

entre elas, neste capítulo somente são apresentados os resultados das saídas 0 de cada caso para não estender o tamanho do capítulo.

6.3.1 Consistência do modelo gerado

6.3.1.1 Aspectos do procedimento experimental

O modelo de cobertura gerado usando a metodologia proposta deve apresentar um perfil de distribuição de valores semelhante àquela da saída do circuito quando obtido por simulação do modelo de referência.

Para mostrar esta característica é apresentado um histograma, no qual é mostrada, em forma gráfica, a diferença entre os seguintes parâmetros:

- número de vetores esperados por evento (NVEE): é a quantidade computada pela ferramenta proposta nesta dissertação que gera o modelo de cobertura de saída. A quantidade de itens para cada evento do modelo de cobertura equivale à quantidade de vetores de teste que geram saídas no intervalo correspondente ao evento (cada novo caso de teste implica em apenas um único item adicional de cobertura de saída). A soma de todos os itens equivale ao total de vetores de teste esperados para simulação.
- número real de vetores (NRV): quantidade requerida de fato na simulação para se obter 100% da cobertura total na simulação com o modelo proposto, ou seja, até que todos os eventos e valores referentes a NVEE tenham sido alcançados.

Para observar mais detalhes de cada resultado são apresentadas também tabelas, as quais mostram individualmente os eventos tidos em conta na modelagem e na simulação de cada caso de teste. Nelas poder-se-á observar a variação percentual em cada um dos eventos entre NVEE e NRV.

6.3.1.2 Resultados sobre consistência de modelo

Para demonstrar visualmente que a metodologia proposta pode modelar, de forma aproximada, a(s) saída(s) de cada circuito, apresentamos, respectivamente, para os circuitos FIR, FFT e Elliptic, os histogramas das figuras 6.4, 6.5 e 6.6, seguindo a descrição da sub-seção anterior.

Pelas figuras 6.4, 6.5 e 6.6, pode-se observar que os valores de NVEE e de NRV são muito semelhantes, exceto no caso do Elliptic, para o qual a variação é maior. Isto ocorre devido aos erros de aproximação que a ferramenta faz em cada etapa do cálculo que dependem da resolução estabelecida (um estudo sobre este fenômeno é feito na seção 6.3.3).

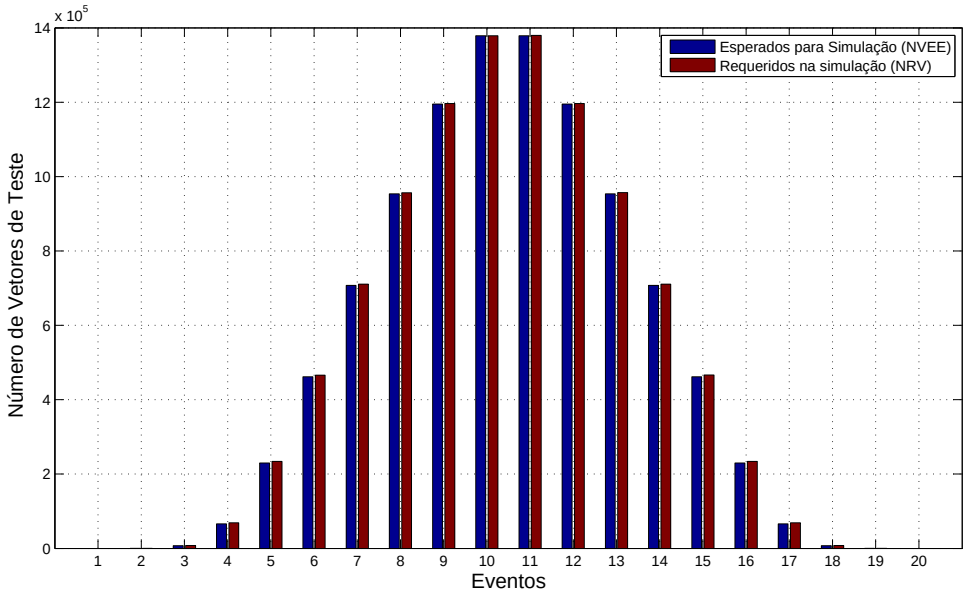


Figura 6.4: Comparação entre o número de vetores de teste esperados (computado pela ferramenta com resolução de 20.000) e o número de vetores que foram contados para obter 100% de cobertura na saída do exemplo FIR.

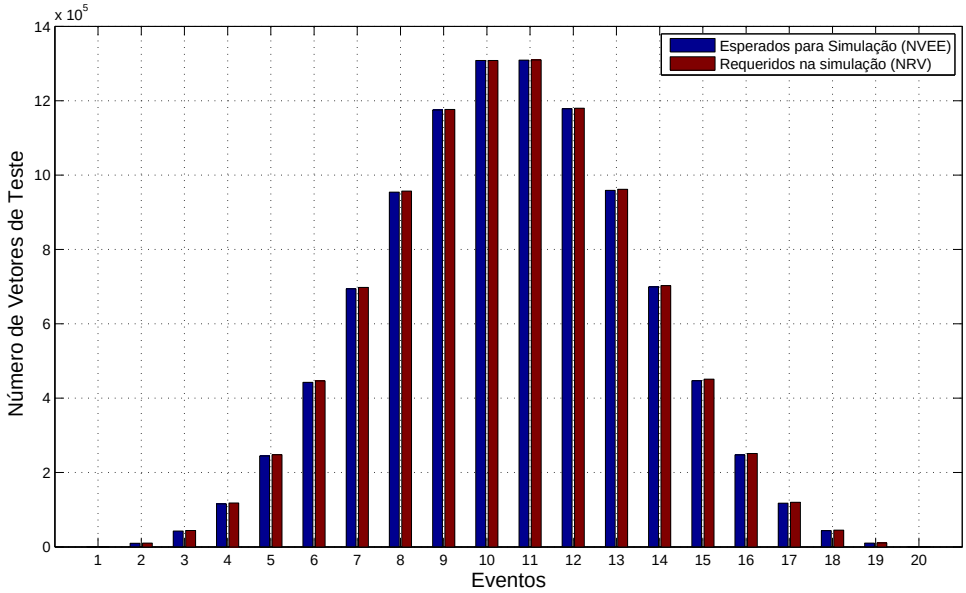


Figura 6.5: Comparação entre o número de vetores de teste esperados (computado pela ferramenta com resolução de 20.000) e o número de vetores que foram contados para obter 100% de cobertura na saída 0 do exemplo FFT.

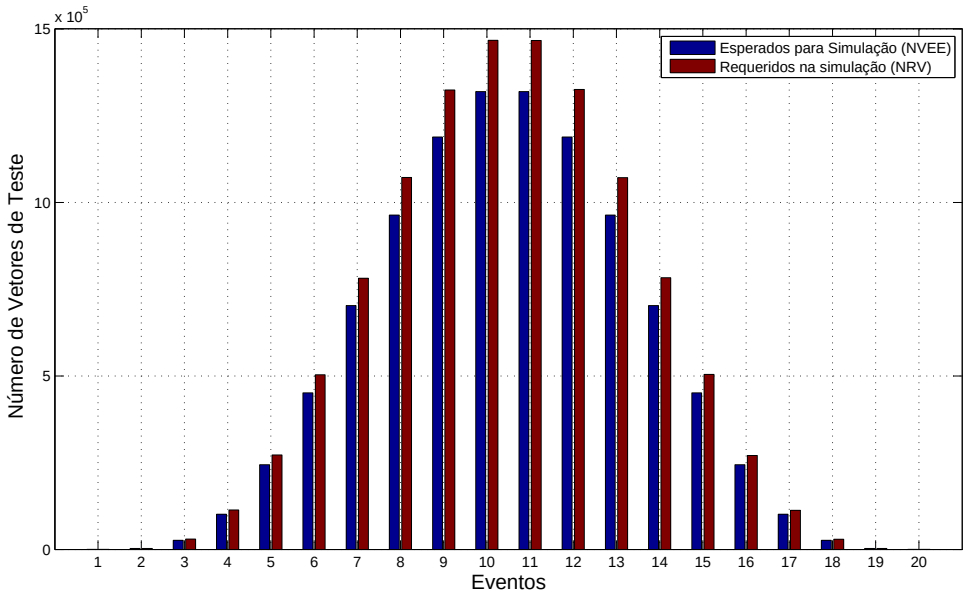


Figura 6.6: Comparação entre o número de vetores de teste esperados (computado pela ferramenta com resolução de 20.000) e o número de vetores que foram contados para obter 100% de cobertura na saída 0 do exemplo Elliptic.

Para observar outros detalhes de variação entre o modelo gerado com a nossa metodologia e o resultado obtido por simulações, as tabelas 6.2, 6.3 e 6.4 são apresentadas. Nelas, os seguintes parâmetros são mostrados:

- Eventos: os grupos de intervalos selecionados apresentados em separado.
- Distribuição: apresenta a distribuição de saída por evento do circuito segundo o modelo proposto, com a massa[2] total (soma da probabilidade de cada dos eventos) 1.
- Número de Vetores de Teste NVEE e NVR.
- Erro: a variação percentual entre VNEE e NVR para cada evento.

[2]Define-se massa de um evento como a probabilidade de sua ocorrência. A massa total do espaço de eventos de uma variável aleatória é igual a 1. A massa em termos numéricos (de itens) corresponde ao número total de itens considerados em todos os eventos.

Tabela 6.2: Resultados para obter 100% de cobertura na saída do FIR (metodologia proposta com resolução de 20.000)

Evento	Intervalo		Distribuição (%)	Número de Vetores Teste		Error (%)
	Inferior	Superior		Esperados para Simulação (NVEE)	Requeridos na simulação (NRV)	
1	-3.961.518	-3.566.090	$5,61x10^{-7}$	0	0	0,00
2	-3.566.089	-3.170.661	$1,59x10^{-3}$	158	192	21,52
3	-3.170.660	-2.775.232	0,07	7.438	8.033	8,00
4	-2.775.231	-2.379.803	0,66	66.223	68.843	3,96
5	-2.379.802	-1.984.374	2,30	229.566	234.162	2,00
6	-1.984.373	-1.588.945	4,62	461.715	465.734	0,87
7	-1.588.944	-1.193.516	7,07	707.381	710.766	0,48
8	-1.193.515	-798.087	9,54	953.540	956.278	0,29
9	-798.086	-402.658	11,95	1.195.196	1.196.315	0,09
10	-402.657	-7.229	13,79	1.378.794	1.378.794	0,00
11	-7.228	388.200	13,79	1.378.791	1.379.743	0,07
12	388.201	783.629	11,95	1.195.191	1.196.626	0,12
13	783.630	1.179.058	9,54	953.534	956.786	0,34
14	1.179.059	1.574.487	7,07	707.375	710.820	0,49
15	1.574.488	1.969.916	4,62	461.709	466.344	1,00
16	1.969.917	2.365.345	2,30	229.561	233.845	1,87
17	2.365.346	2.760.774	0,66	66.221	69.038	4,25
18	2.760.775	3.156.203	0,07	7.438	7.947	6,84
19	3.156.204	3.551.632	$1,59x10^{-3}$	158	197	24,68
20	3.551.633	3.947.052	$5,60x10^{-7}$	0	0	0,00
TOTAL				9.999.989	10.040.463	0,40

Tabela 6.3: Resultados para obter 100% de cobertura na saída 0 do exemplo FFT (metodologia proposta com resolução de 20.000)

Evento	Intervalo		Distribuição (%)	Número de Vetores Teste		Error (%)
	Inferior	Superior		Esperados para Simulação (NVEE)	Requeridos na simulação (NRV)	
1	-512	-462	$6,42x10^{-3}$	642	712	$10,90$
2	-461	-411	$0,10$	9.825	10.434	$6,20$
3	-410	-360	$0,43$	42.841	44.376	$3,58$
4	-359	-309	$1,16$	115.723	118.086	$2,04$
5	-308	-258	$2,44$	244.407	248.060	$1,49$
6	-257	-207	$4,42$	442.429	446.264	$0,87$
7	-206	-156	$6,94$	694.054	698.033	$0,57$
8	-155	-105	$9,54$	954.262	956.930	$0,28$
9	-104	-54	$11,75$	1.174.809	1.176.516	$0,15$
10	-53	-3	$13,08$	1.307.970	1.307.970	$0,00$
11	-2	48	$13,09$	1.309.305	1.309.807	$0,04$
12	49	99	$11,78$	1.178.487	1.179.725	$0,11$
13	100	150	$9,59$	959.054	961.758	$0,28$
14	151	201	$6,99$	699.282	702.675	$0,49$
15	202	252	$4,47$	446.926	451.234	$0,96$
16	253	303	$2,48$	247.622	251.150	$1,42$
17	304	354	$1,18$	117.633	120.231	$2,21$
18	355	405	$0,44$	43.841	45.328	$3,39$
19	406	456	$0,10$	10.185	10.779	$5,83$
20	457	508	$6,95x10^{-3}$	694	788	$13,54$
TOTAL				9.999.991	10.040.856	$0,41$

Tabela 6.4: Resultados para obter 100% na saída 0 do Exemplo Elliptic(metodologia proposta com resolução de 20.000)

Evento	Intervalo		Distribuição (%)	Número de Vetores Teste		Error (%)
	Inferior	Superior		Esperados para Simulação (NVEE)	Requeridos na simulação (NRV)	
1	-1.015.808	-914.229	$4,46x10^{-4}$	44	53	20,45
2	-914.228	-812.649	0,03	2.791	3.006	7,70
3	-812.648	-711.069	0,26	26.326	29.701	12,82
4	-711.068	-609.489	1,02	101.606	113.634	11,84
5	-609.488	-507.909	2,44	243.907	272.046	11,54
6	-507.908	-406.329	4,51	451.323	503.242	11,50
7	-406.328	-304.749	7,03	702.928	781.253	11,14
8	-304.748	-203.169	9,63	963.399	1.071.585	11,23
9	-203.168	-101.589	11,89	1.188.595	1.324.017	11,39
10	-101.588	-9	13,19	1.319.168	1.467.045	11,21
11	-8	101.571	13,19	1.319.158	1.466.750	11,19
12	101.572	203.151	11,89	1.188.570	1.325.296	11,50
13	203.152	304.731	9,63	963.364	1.071.532	11,23
14	304.732	406.311	7,03	702.892	783.071	11,41
15	406.312	507.891	4,51	451.290	504.225	11,73
16	507.892	609.471	2,44	243.883	270.686	10,99
17	609.472	711.051	1,02	101.591	112.864	11,10
18	711.052	812.631	0,26	26.320	29.200	10,94
19	812.632	914.211	0,03	2.790	3.025	8,42
20	914.212	1.015.777	$4,45x10^{-4}$	44	44	0,00
TOTAL				9.999.989	11.132.275	10,17

Pelas tabelas pode-se observar que a variação no número total de testes necessários para se obter o índice de 100% da cobertura é baixo para os casos dos circuitos FIR e da FFT4, com variações entre 0.4% até 1.1%. No caso do Elliptic, as variações são maiores, entre 2% até 33%. Como mencionado anteriormente, estas variações maiores são devidas, principalmente, aos erros de aproximação pela resolução usada. Observe-se que os eventos que mais demoraram para em atingir 100% de cobertura de evento (e, consequentemente, global) nos casos FIR e FFT, foram os eventos 10 com alta probabilidade de ocorrência, ou seja com menor influência relativa. Já para o caso do Elliptic, os eventos que mais demoraram para atingir 100% de cobertura de evento (e, consequentemente, global) foram aqueles, de menor probabilidade de ocorrência, como, por exemplo o evento 20 da saída 0, sendo o erro de cômputo nestes eventos crítico para controlar o tempo de simulação. Pelo estudo realizado na seção 6.3.3, será verificada a importância da resolução para controlar os erros.

6.3.2 Tempo de simulação

6.3.2.1 Aspectos do procedimento experimental

Espera-se na modelagem proposta nesta dissertação alcançar 100% de cobertura com um número de vetores de teste significativamente menor do que o necessário com o método manual (com distribuição uniforme). Para apresentar esta característica deve-se mostrar a progressão da cobertura em função do número de vetores de teste, explicitando-se o número requerido para se obter 100% da cobertura para ambos casos.

O número de itens (também vetores de teste) para cada evento correspondente ao índice de 100% de cobertura no modelo proposto são aqueles dados na sub-seção anterior; entretanto, o número de vetores de teste para 100% de cobertura por distribuição uniforme ser definido de uma maneira a se ter uma comparação adequada. Consideramos, então que a distribuição uniforme sobre os eventos (intervalos) se dá de forma em que a sua massaseja igual a da distribuição no modelo proposto nesta dissertação.

Dois modelos de cobertura com distribuições diferentes, apesar de mesma massa, poderão apresentar requisitos bem diferentes quanto à cobertura dos eventos individuais. Para exemplificar, a Tabela 6.5 mostra o caso Elliptic (saída 0), com as densidades probabilísticas para o modelo proposto (coluna 2) e para o modelo de distribuição uniforme (Modelo Manual, coluna 3). Para este último, todos os vinte eventos apresentam a mesma densidade probabilística e pode-se observar que o evento 1 tem uma densidade de 11.210 vezes maior que o evento 1 (de densidade mais baixa) no modelo proposto. Considerando-se que a cobertura total ocorre somente quando a cobertura em 100% de cada evento tenha

Tabela 6.5: Exemplo para computar o número de vetores requeridos na saída 0 do Elliptic para completar o 100% da cobertura

Evento	Densidade (Probabilidade)		Número de vetores requeridos	
	Modelo Proposto	Modelo Manual	Modelo Proposto	Modelo Manual
1	$4,46x10^{-6}$	$0,05$	4	50.000
2	$0,03x10^{-2}$	$0,05$	279	50.000
3	$0,26x10^{-2}$	$0,05$	2.632	50.000
4	$0,01$	$0,05$	10.160	50.000
5	$0,02$	$0,05$	24.390	50.000
6	$0,05$	$0,05$	45.132	50.000
7	$0,07$	$0,05$	70.292	50.000
8	$0,10$	$0,05$	96.339	50.000
9	$0,12$	$0,05$	118.859	50.000
10	$0,13$	$0,05$	131.916	50.000
11	$0,13$	$0,05$	131.915	50.000
12	$0,12$	$0,05$	118.857	50.000
13	$0,10$	$0,05$	96.336	50.000
14	$0,07$	$0,05$	70.289	50.000
15	$0,05$	$0,05$	45.129	50.000
16	$0,02$	$0,05$	24.388	50.000
17	$0,01$	$0,05$	10.159	50.000
18	$0,26x10^{-2}$	$0,05$	2.632	50.000
19	$0,03x10^{-2}$	$0,05$	279	50.000
20	$4,45x10^{-6}$	$0,05$	4	50.000
TOTAL	1,00	1,00	999.991	1.000.000

ocorrido, o tempo de simulação usando o modelo distribuição uniforme pode ser muito elevado. Por exemplo, ao se multiplicar todos os valores de densidade da Tabela 6.5 por 1.000.000 (e arredondando), temos nas colunas 4 e 5 os valores inteiros designando o número de itens de eventos dos dois modelos, ficando óbvio a diferença de requisitos para se atingir a cobertura de 100% individualmente. Para o evento 1, necessitaríamos da ocorrência de quatro itens no modelo proposto enquanto que para o modelo de distribuição uniforme, seria necessária a ocorrência de 50.000. Para limitar o tempo de simulação para o modelo de cobertura de distribuição uniforme, foi feita uma aproximação no número de vetores de teste requeridos, seguindo os passos abaixo.

1. Normalizar as probabilidades pelo evento com menor distribuição do modelo proposto nesta dissertação. O resultado desta operação retorna o número mínimo de vetores de teste requerido para observar pelo menos um item do evento com menor probabilidade de ocorrência.

2. Totalizar o número de vetores somando-se os números de itens de cada evento. O valor obtido determina o mínimo de vetores de teste que devem ser inseridos para se

obter a distribuição computada. Este valor corresponde à massa "normalizada", onde o evento de menor probabilidade de ocorrência apresenta um item.

3. O valor resultante da etapa anterior deve ser multiplicado pelo número de itens que se deseja observar em cada evento dentro do modelo de distribuição uniforme. O valor resultante é o número mínimo de vetores de teste que se requer para conseguir o 100% da cobertura com distribuição uniforme. Quanto ao número de itens na distribuição uniforme, é claro que o valor é idêntico para todos os eventos e é dado pelo número de total de vetores que se deseja observar no modelo proposto (definido como 10.000.000) dividido pelo número de eventos.

O procedimento mencionado anteriormente é formalizado na equação 6.1.

$$\frac{\sum_{i}^{n} P_i}{Min(ProbEvent)} * NumItens = \frac{\sum_{i}^{n} P_i.TotalNumVector}{Min(ProbEvent) * TotalEvents} \tag{6.1}$$

Onde:

- $P_i \in ProbEvent$
- $ProbEvent = \{P_0, P_1, ..., P_{n-1}\}$- probabilidade de ocorrência de itens em cada evento, onde n é o número de eventos.
- $Min(ProbEvent)$- probabilidade de menor valor do conjunto $ProbEvent$.
- $NumItens$- número de vetores requeridos para completar 100% da cobertura do evento com menor probabilidade de ocorrência no modelo de cobertura de distribuição uniforme.
- $TotalNumVector$- número total de vetores a serem aplicados correspondente a 100% de cobertura total no modelo proposto.
- $TotalEvents$- número eventos ou intervalos considerados para uma saída.

A partir da equação 6.1 pode-se ver pela tabela 6.6, qual o número de vetores que é requerido na simulação com o modelo de cobertura de distribuição uniforme. A tabela mostra, para cada função de saída dos exemplos considerados:

Número de vetores com a Metodologia Proposta (NVMP): este valor indica o número de vetores esperados para se atingir 100% de cobertura com o modelo proposto.

Tabela 6.6: Número de vetores requeridos para obter 100% de cobertura

Modelo do Circuito	Saída	Número de Vetores Teste		Variação
		Met. Proposta	Dist. Uniforme	(%)
FIR	0	10.000.000	$8,92x10^{13}$	$8,92x10^{8}$
FFT4	0	10.000.000	$7,79x10^{9}$	$7,79x10^{4}$
	1	10.000.000	$7,79x10^{9}$	$7,79x10^{4}$
	2	10.000.000	$1,04x10^{8}$	$1,04x10^{3}$
	3	10.000.000	$1,04x10^{8}$	$1,04x10^{3}$
	4	10.000.000	$7,79x10^{9}$	$7,79x10^{4}$
	5	10.000.000	$7,79x10^{9}$	$7,79x10^{4}$
	6	10.000.000	$1,04x10^{8}$	$1,04x10^{3}$
	7	10.000.000	$1,04x10^{8}$	$1,04x10^{3}$
Elliptic	0	10.000.000	$1,12x10^{11}$	$1,12x10^{6}$
	1	10.000.000	$8,66x10^{11}$	$8,66x10^{6}$
	2	10.000.000	$1,52x10^{12}$	$1,52x10^{7}$
	3	10.000.000	$4,79x10^{11}$	$4,79x10^{6}$
	4	10.000.000	$1,21x10^{12}$	$1,21x10^{7}$
	5	10.000.000	$1,72x10^{12}$	$1,72x10^{7}$
	6	10.000.000	$1,35x10^{11}$	$1,35x10^{6}$
	7	10.000.000	$1,16x10^{11}$	$1,16x10^{6}$

Número de vetores com Distribuição Uniforme (NVDU): este valor indica o número de vetores esperados para se atingir 100% de cobertura com um modelo uniforme, com base na equação 6.1 apresentado nesta seção. Os dados referentes aos eventos foram obtidos usando nossa ferramenta com uma resolução de 20.000.

Variação: representa relação percentual entre NVDU e NVMP $\left(\frac{NVDU}{NVMP} * 100\%\right)$.

Pode-se observar na tabela 6.6 que os valores teóricos para atingir o 100% da cobertura usando modelos com distribuição uniforme são muito elevados, comprovando desta forma a ineficiência destes para o processo de verificação. Esta diferença é crítica, por exemplo no caso do FIR, onde para atingir a cobertura precisa-se de $8.922834x10^{13}$ vetores de teste o que equivale ao $8.922834x10^{8}$% comparado com o modelo gerado com a metodologia proposta. Portanto, em processos industrias, utilizar modelos de cobertura uniforme elevariam os custos de projeto, além de aumentar o Time-to-Market.

Estas equações seriam ratificadas através de simulação, mas, como foi comentado anteriormente, o tempo de simulação envolvido seria muito elevado. Por isto, determinou-se limitar o tempo de simulação, restringindo-se o número de vetores de teste a que devem serem gerados. A quantidade escolhida para este processo foi 60.000.000 o que equivalente a 600% ao do número de vetores de teste estabelecido como cobertura global no modelo de cobertura proposto. Este número, denominado número de vetores de referência de excesso (NVRE), mostrou corresponder, na prática, a um valor suficientemente elevado para per-

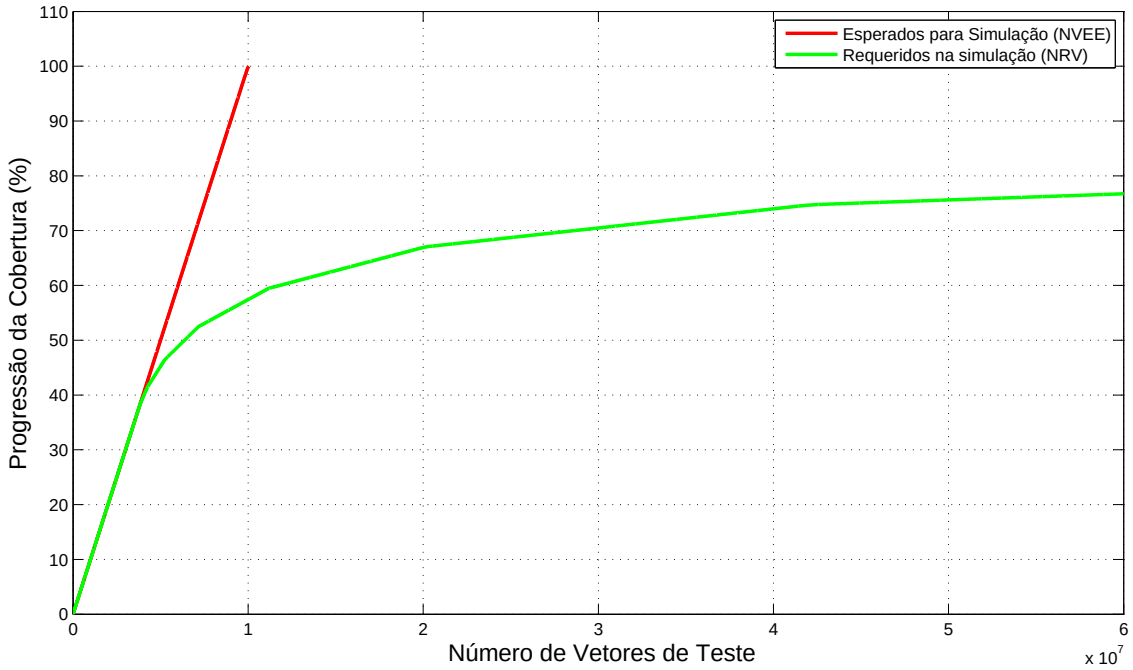


Figura 6.7: Progressão da cobertura da saída 0 do exemplo FFT utilizando o modelo gerado com a metodologia proposta (com resolução de 20000) e o modelo manual (distribuição uniforme para os eventos selecionados)

mitir a observação comparativa da progressão da cobertura para os modelos proposto de distribuição uniforme, além de não comprometer o tempo simulação.

6.3.2.2 Resultados sobre tempo de simulação

A figura 6.7 apresenta a progressão da cobertura com o modelo gerado com nossa metodologia (NVEE) e o modelo manual (NRV) para o caso de saída 0 FFT4. Curvas de progressão de outras saídas e de outros circuitos são omitidas uma vez que são bastante similares no comportamento. Pode-se observar pela figura que após inserir uma quantidade inicial de vetores de teste, a progressão usando o modelo com distribuição uniforme perde a linearidade, levando a um aumento expressivo no tempo de simulação. Após a simulação de 60 milhões de vetores de teste, a cobertura total não chega a 80%. De forma diferente, o modelo de cobertura gerado pela nossa metodologia mantém a linearidade até alcançar 100% de cobertura.

Para se entender com detalhes o que ocorreu com a cobertura do modelo manual no momento em que se alcança o NVRE de 60 milhões de vetores, é apresentada a tabela correspondente do caso FFT (tabela 6.7), com os parâmetros abaixo:

- Eventos: os grupos de intervalos selecionados a serem observados (estes são iguais aos

selecionados nas tabelas 6.2, 6.3 e 6.4)

- Número de Vetores de Teste:
  - Esperado para 100% cobertura: é a quantidade esperada para cada evento em uma distribuição uniforme, para 100% de cobertura global, portanto todos os valores são iguais.
  - Inseridos por evento após NVRE: é a quantidade de vetores contada para cada um dos eventos, após a simulação com o NVRE de 60 milhões de vetores.
- Cobertura: é a porcentagem de cobertura alcançada para cada evento; nos casos em que se tenha excedido o número de vetores de teste esperado, o valor de cobertura fica saturado em 100%.

Pode-se observar na tabela 6.7, que os eventos 4 a 17 atingiram 100% da sua cobertura. No momento que isto acontece durante o processo de simulação, os itens que continuam a ser observados nestes eventos não influenciam mais na progressão da cobertura total, tendo como consequência, a perda de linearidade e portanto a progressão torna-se mais lenta. Isto ocorre porque há eventos que estão longe de atingir 100% de cobertura individual. Por exemplo, na tabela 6.7, o Evento 1 conta com apenas 4.180 itens dos 500 mil necessários. Por estes fatos, na figura 6.7, a partir da simulação com 4.000.000 vetores de teste no modelo de referência, a progressão começa a se desviar. No momento que, pela na simulação usando nosso modelo de cobertura (linha vermelha), 100% da cobertura é atingida, a simulação usando o modelo manual só teria chegado a 57%.

Tabela 6.7: Resultados da Progressão de Cobertura na saída 0 do FFT(Métodos Manuais)

Evento	Intervalo		Distribuição (%)	Número de Vetores Teste		Cobertura (%)
	Inferior	Superior		Esperado para 100% de cobertura	Inseridos por evento após Número de Vetores de Referencia	
1	-512	-462	5,00	500.000	4.180	0,84
2	-461	-411	5,00	500.000	62.372	12,47
3	-410	-360	5,00	500.000	265.378	53,08
4	-359	-309	5,00	500.000	705.463	100,00
5	-308	-258	5,00	500.000	1.482.234	100,00
6	-257	-207	5,00	500.000	2.666.841	100,00
7	-206	-156	5,00	500.000	4.170.588	100,00
8	-155	-105	5,00	500.000	5.718.159	100,00
9	-104	-54	5,00	500.000	7.030.307	100,00
10	-53	-3	5,00	500.000	7.816.368	100,00
11	-2	48	5,00	500.000	7.827.144	100,00
12	49	99	5,00	500.000	7.049.317	100,00
13	100	150	5,00	500.000	5.747.201	100,00
14	151	201	5,00	500.000	4.199.156	100,00
15	202	252	5,00	500.000	2.696.283	100,00
16	253	303	5,00	500.000	1.500.390	100,00
17	304	354	5,00	500.000	718.486	100,00
18	355	405	5,00	500.000	271.064	54,21
19	406	456	5,00	500.000	64.373	12,87
20	457	508	5,00	500.000	4.696	0,94
TOTAL				10.000.000	60.000.000	76,72

6.3.3 Variações de resolução no cálculo da PD de saída

6.3.3.1 Aspectos do procedimento experimental

Um fator importante na aplicação da nossa metodologia é o tempo que se leva para gerar o modelo de cobertura proposto e um dos parâmetros envolvidos é a resolução utilizada. Sabe-se que quanto a maior resolução, mais aproximado será o modelo de cobertura gerado do comportamento real do circuito; porém, como consequência disto, ocorre também um maior tempo de cálculo. Para avaliar a influência da resolução no modelo de saída, e portanto na simulação do sistema, experimentos foram realizados seguindo-se os passos abaixo:

1. Geração dos modelos de cobertura para valores diferentes de resolução, de 200 e 2.000, que equivalem à centésima e décima partes da resolução usada nas simulações anteriores (20.000).

2. Simulação do modelo de referência do circuito utilizando-se os modelos de cobertura gerados.

6.3.3.2 Resultados sobre variações de resolução

Para observar como a resolução influencia no tempo de cômputo, estes tempos foram medidos para os três casos de teste (FIR, FFT e Elliptic) nas três resoluções estabelecidas e os seus resultados apresentados na figura 6.8, em uma curva de tempo de geração por resolução. Nela pode-se observar que o caso da FFT é o único que apresenta um comportamento linear, enquanto nos outro dois casos o comportamento é polinomial, razão pela qual os seus tempos de cálculo, com uma resolução de 20.000, é muito elevado. O comportamento polinomial deve-se aos laços aninhados contidos nos algoritmos de cálculo do perfil. No caso da FFT, há uma particularidade: na maioria das saídas, a quantidade de valores discretos é pequeno, por exemplo, para a saída 0, os valores vão de-512 a 508, o que gera um espaço de 1.021 eventos no máximo, se cada evento consistir de um único elemento. Portanto o número de vezes que se repetem os laços do algoritmo de cálculo do perfil de saída não muda, mesmo que se aumente a resolução a 20.000, fazendo-se que não haja, neste caso, uma variação tão elevada no tempo de cálculo.

Embora o tempo requerido para a ferramenta obter o perfil seja elevado, ele permite reduzir de forma substancial o tempo de simulação, como foi observado na seção anterior. Mas, como pode-se concluir pela da figura 6.8, caso o modelo do circuito a ser utilizado seja muito complexo, o tempo de cálculo poderia se elevar demasiadamente, fazendo com que, ao final, a metodologia não fosse tão benéfica. Por isto, a seleção de um valor adequado de

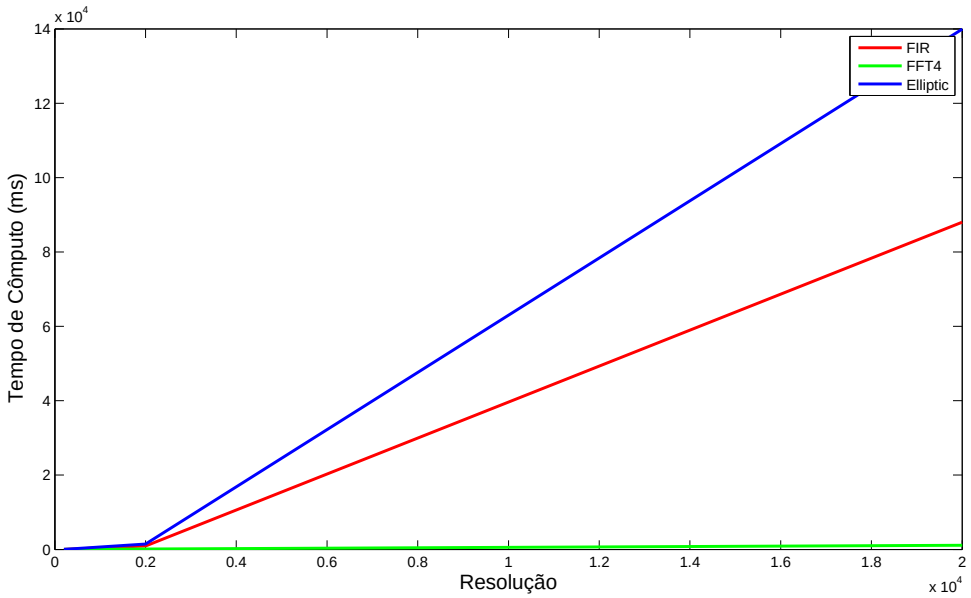


Figura 6.8: Variação de tempo de cálculo em função da resolução

resolução converte-se em um fator importante na modelagem da cobertura para o processo de verificação. Para observar como muda o número de vetores requeridos para obter 100% de cobertura, segundo as resoluções adotadas, a tabela 6.8 é apresentada. Nesta a primeira coluna mostra os modelos usados(FIR, FFT e Elliptic), a segunda coluna as saídas de cada um circuitos e da coluna três até cinco, o número de vetores que foram requeridos para obter o 100% de cobertura segundo a resolução (200, 2.000, 20.000).

Nos casos em que a resolução é de 200 pontos, pode-se observar que o número de vetores necessários para se obter 100% da cobertura aumentou em vários casos dos *benchmarks*, resultado de uma precisão mais baixa do modelo gerado. Pode-se notar isto, por exemplo, no caso do exemplo Elliptic, cuja saída 5 necessitou de 10.216.782 vetores quando é utilizada uma resolução de 20.000, enquanto que para a resolução de 200 são necessários 13.631.682 vetores. Isto significa em atraso adicional na entrega de resultados do processo de verificação, que pode-se tornar considerável em níveis industriais.

No caso de se usar uma resolução de 2.000, o número de vetores de teste para completar 100% da cobertura é menor que no caso de resolução de 200, para várias saídas, e também pode-se observar que para todos os casos de teste dos exemplos FIR, FFT e Elliptic, o número de vetores requeridos é igual aos mostrados com resolução de 20.000. Este resultado nos permite conjecturar que utilizar uma resolução de 100 vezes o número de eventos que se deseja observar permite ter uma modelagem adequada além de não elevar de forma significativa o tempo de cômputo.

Tabela 6.8: Número de Vetores usados para obter 100% de cobertura segundo a resolução usada

Modelo do Circuito	Saída	Resolução		
		200	2000	20000
FIR	0	10.033.801	10.040.367	10.040.463
FFT	0	10.040.193	10.040.856	10.040.856
	1	10.052.410	10.052.335	10.052.335
	2	10.106.679	10.106.679	10.106.679
	3	10.104.018	10.104.018	10.104.018
	4	10.037.931	10.038.432	10.038.432
	5	10.053.822	10.053.710	10.053.710
	6	10.103.570	10.103.570	10.103.570
	7	10.106.775	10.106.775	10.106.775
ELI	0	11.450.995	11.132.275	11.132.275
	1	11.927.258	11.605.705	11.605.705
	2	11.605.705	11.605.705	11.605.705
	3	10.422.245	10.422.245	10.422.245
	4	15.044.137	15.044.137	15.044.137
	5	13.631.682	10.216.782	10.216.782
	6	10.688.340	10.549.362	10.549.362
	7	10.599.764	10.478.131	10.478.131

7 Conclusões e trabalhos futuros

Nesta dissertação foi proposta uma metodologia de geração de um modelo de maior abrangência para a cobertura de saída com o intuito de auxiliar o engenheiro de verificação na tarefa de definir a cobertura de saída por itens e melhorar o desempenho do processo de verificação dinâmica por simulação. A metodologia baseou-se na utilização de representações do modelos de referência e no uso de métodos probabilísticos. Da pesquisa realizada na dissertação e dos resultados experimentais obtidos, as seguintes conclusões foram obtidas:

1. Foi definida uma metodologia para estabelecer a relação entrada-saída do modelo de referência do DUV dentro de um formato conveniente para a geração de modelo de cobertura de saída baseado em itens. Isto é possível uma vez que os modelos de referência utilizados nos testbenches podem ser interpretados como uma função que abstrai a lógica interna do sistema sob verificação. Desta forma, pode-se predizer o perfil dos eventos a serem observados na saída, tendo-se definido previamente o conjunto de eventos de entrada.

2. Foi estabelecido um arcabouço conceitual para sistematizar a metodologia proposta para a obtenção da cobertura de saída por itens. Para isto, ao se ter estabelecido a representação das variáveis do circuito em forma equações, pode-se usar métodos probabilísticos para se obter o conjunto de eventos de saída e a probabilidade de ocorrência de cada destes com boa precisão. Esta metodologia, portanto, tem permitido aumentar a abrangência dos modelos de cobertura na etapa de saída. É a partir da predição dos eventos de saída os engenheiro de verificação pode fazer variações na entrada para estimular zonas de especial interesse ou reduzir ineficiências devido a expectativas erradas de cobertura.

3. Baseado na metodologia desenvolvida, a ferramenta PrOCov foi projetada. Esta tem como objetivo a encontrar a(s) equação(ões) que define(m) a(s) saída(s) do circuito sob verificação e, a partir destas, encontrar a distribuição probabilística por evento observável, tendo-se definido previamente o espaço de parâmetros de entrada. A partir

desta resposta a ferramenta PrOCov cria o perfil de saída. Através desta resposta e indicando a quantidade de vetores de teste se deseja gerar, é criado um arquivo o qual contém o modelo de cobertura na linguagem SystemC.

4. Para a avaliação da metodologia, e da ferramenta gerada a partir desta, foram usados alguns exemplos, modelos em alto nível do filtro FIR, do processador FFT e do filtro Elliptic, todos descritos em SystemC. Nos três casos testados, o PrOCov encontrou satisfatoriamente os respectivos perfis de saída. Estes foram comparados com os perfis obtidos por simulação, mostrando uma excelente precisão; apenas pequenas variações foram encontradas devidas a erros de aproximação sendo elas associadas a eventos extremos (de menor probabilidade).

5. A falta de eficiência no uso de um modelo de distribuição uniforme para a cobertura de saída gerado manualmente foi verificada ao compararmos com o modelo de cobertura gerado com a nossa metodologia. O tempo de execução de testbench (simulação) utilizando-se o perfil gerado com a ferramenta PrOCov foi reduzido em $8,92x10^8\%$. Este efeito melhorou, portanto, o desempenho do processo de verificação. É este fator torna-se crucial na industria para a obtenção produtos finais e o ingresso destes ao mercado.

6. Para o êxito da modelagem obtida pela ferramenta PrOCov, a resolução utilizada no cálculo foi fundamental. Experimentos indicaram que se requer uma resolução de aproximadamente 100 vezes o número de eventos desejados. Isto permitiu não só se ter uma boa aproximação da saída senão também uma uma redução no tempo de cálculo do perfil, que no caso de sistemas de maior complexidade torna-se crucial.

A verificação de sistemas digitais é uma área de pesquisa que é cada dia mais de interesse da comunidade acadêmica e da indústria por representar um grande desafio na tentativa por aumentar a eficiência do ciclo de produção. Um dos pontos de grande interesse é a modelagem da cobertura, área que não tem sido muito explorada. Assim, além das propostas apresentadas nesta dissertação, sugere-se continuar a pesquisa tanto das técnicas específicas como das metodologias apresentadas neste trabalho, assim como nas propostas de outros autores. Algumas sugestões para trabalhos futuros são:

1. A ferramenta PrOCov na sua versão atual só considera casos de cálculo de componentes e condições independentes para sua facilidade de cálculo. De outro lado, grande parte dos circuitos contém condicionais dependentes, ou seja, uma variável usada numa determinada condição é usada, também, dentro do cálculo por ela determinada. Portanto, a extensão da metodologia para o cômputo casos deveria também

ser estabelecida. Por outro lado, deve-se adicionar as operações aritméticas de divisão e módulo usadas frequentemente no desenvolvimento de aplicações.

2. Nos resultados obtidos, a resolução usada permitiu obter um perfil de cobertura de boa aproximação. Esta considerou os valores máximo e mínimo para dividir o espaço de saída num grupo de eventos de aproximadamente igual quantidade de eventos elementares. Mas para um caso genérico, utilizando esta metodologia, pode existir espaços sem evento elementar algum. Estes seguindo a metodologia empregada nesta dissertação são eventos de probabilidade zero. Este "desperdício" de eventos que nunca serão observados poderia ser evitado e poderiam ser aproveitados para aumentar a resolução de outros eventos. Portanto, a obtenção de uma heurística desta etapa de cálculo melhoraria o cálculo do perfil de saída, reduzindo o erros de aproximação pelo aproveitamento de aqueles de evento de probabilidade nula de ocorrência.

3. Ao existir uma relação entrada-saída em um circuito, nesta dissertação estabeleceu-se a forma de se encontrar o perfil de saída baseado num conjunto de eventos de entrada. Entanto, existem casos onde se requer estabelecer os valores de entradas e sua probabilidade, para se obter um determinado perfil na saída, por exemplo, como o uso de técnicas de CDV. Por isso se propõe também usar a metodologia definida nesta dissertação como base para definir um método de estabelecer o perfil de entrada uma vez definida uma determinada distribuição na saída.

Apêndice A – Fundamentos de probabilidade

Esta seção tem como foco apresentar conceitos básicos de probabilidades usados para estabelecer as bases matemáticas da seção seção 4.2 do capítulo 4. Baseado nestes conceitos é desenvolvida a ferramenta de Cômputo da Distribuição de Saída do capítulo 5. Primeiramente apresentamos os conceitos de Probabilidades (Seção A.1), procurando definir o *espaço de probabilístico*. Após isto (Seção A.2), apresentamos o conceito, de forma geral, de *variável aleatória*. Com isto, na seção A.3, estabelece-se as fórmulas para o cálculo da distribuição de probabilidade de uma função que depende de uma ou dois variáveis aleatórias. Para mais detalhes revisar (35) (37) (36).

A.1 Conceito de probabilidade

O conceito fundamental na teoria de probabilidades é o ***"Experimento Aleatório"***. Este tem como premissa que a resposta de qualquer experimento é incerta (36) (41). Para o modelamento do experimento, deve-se de conhecer o conjunto de possíveis resultados. Este conjunto é conhecido como ***Espaço de Amostras*** (Ω). Por exemplo um dado tem 6 faces, portanto 6 possíveis resultados, os quais estão em Ω:

$$\omega_1,\ \omega_2,\ \omega_3,\ \omega_4,\ \omega_5,\ \omega_6$$

Baseado nesses possíveis resultados, pode-se definir um grupo de interesse. Este é conhecido como ***Evento***. Ao conjunto dos possíveis grupos de interesse, o qual está constituído por todos os subgrupos de Ω, é conhecido como ***Espaço de Eventos*** ($\mathbb{E}$). Sendo A_0, A_1, ... todos os possíveis eventos (subgrupos de Ω), então sua união é também um evento (41):

$$\bigcup_{i=0}^{\infty} A_i \in \mathbb{E} \tag{A.1}$$

Por outro lado, procura-se uma métrica para quantificar a *chance* (***Probabilidade, P***) de ocorrência de um evento associado a um experimento aleatório. Esta métrica procura proporcionar a *"frequência"* com que o evento ocorre (35) (36) (41).

A partir dos conceitos apresentados, define-se o ***Espaço de Probabilidade*** (Ω, $\mathbb{E}$, P), o qual procura modelar e permite medir o experimento (36) (41).

A.2 Variável aleatória

Os eventos representam muitas vezes objetos abstratos (ex. faces do dado), o que impede sua manipulação matemática de forma direta. Para remediar isto, deve-se introduzir um meio para representar estas variações em número real. Precisa-se, uma função que tenha como domínio os possíveis resultados do experimento e como intervalo um valor real ($\mathbb{R}$). A função que contém estas propriedades é conhecida como ***Variável Aleatória*** $X : \omega \rightarrow \mathbb{R}$.

Para determinar a probabilidade de um grupo de eventos, define-se o conceito de ***Distribuição de Probabilidade*** (PD). No caso em que se deseje calcular a PD até x, então:

$$F(x) = P(X \leq x) \tag{A.2}$$

A partir desta equação, define-se a densidade de probabilidade como:

$$f(x) = \frac{\partial F(x)}{\partial x} \tag{A.3}$$

No caso em que queira calcular a probabilidade um evento gerar um valor entre a e b, então:

$$P(a \leq X \leq b) = F(b) - F(a) = \int_a^b f(\tau)\mathrm{d}\tau \tag{A.4}$$

A.3 Funções de variáveis aleatórias

Supondo existir uma variável aleatória X e uma função $z = g(x)$ que tem o intervalo definido pela variável x, pode-se estabelecer uma variável aleatória Z, onde:

$$Z = g(X) \tag{A.5}$$

Conhecendo que a função de distribuição de probabilidade de X é $F_X(x)$, define-se que:

$$F_Z(z) = P(Z \leq z) = P(g(X) \leq z) \tag{A.6}$$

Considerando-se que $z = g(x)$ é monotônica, portanto tem uma solução e as densidade de probabilidades são f_x e f_z para x e z respectivamente, então:

$$f_z \, |dz| = f_x \, |dx| \tag{A.7}$$

$$f_z = \frac{f_x}{\frac{|dz|}{|dx|}} \tag{A.8}$$

No caso de $z = g(x)$ não ser monotônica ($z = g(x_0) = g(x_1) \cdots = g(x_{n-1})$), então

$$f_z(z) = \sum_{i=0}^{n-1} \frac{f_x(x_i)}{|g^{'}(x_i)|} \tag{A.9}$$

onde n é o número total de raízes da função $g(x)$ e $g^{'}(x_i)$ é a derivada de $g(x)$ em x_i.

Em geral pode-se calcular $F_Z(z)$ usando-se a densidade de probabilidade de X, $f_x(x)$, calculando:

$$F_Z(z) = \int_{S(z)} f_x(x) \mathrm{d}x \tag{A.10}$$

onde $S(z)$ é o conjunto de pontos $\{x : g(x) \leq z\}$. No caso em que a função Z dependa de duas variáveis aleatórias X e Y, com $Z = g(X, Y)$, então:

$$F_Z(z) = \iint_{S(z)} f(x, y) \mathrm{d}x \mathrm{d}y \tag{A.11}$$

onde $f(x, y)$ é densidade conjunta de X e Y e $S(z) = \{(x, y) : g(x, y) \leq z\}$. Para o caso que as variáveis aleatórias X e Y sejam independentes (37) (36):

$$F_Z(z) = \iint_{S(z)} f(x) f(y) \mathrm{d}x \mathrm{d}y \tag{A.12}$$

Apêndice B – Algoritmos para operações de relação

Este capítulo apresenta os algoritmos para o cômputo da probabilidade das operações de relacionamento.

```
CompProbOfLEq(c, X)

ρ = 0 ///////Inicializa a Probabilidade

///////Lê cada um dos intervalos de X
for all I ∈ X do
    ρ = ρ+PDofSumSubVarCnst(c + 0.5, I.ρ, I.L, I.H)///////Executa a equação 4.4
end for
return ρ
```

Figura B.1: Algoritmo para o computo da operação de relação '$X \leq c$'

```
CompProbOfGre(c, X)

return 1−CompProbOfLEq(c, X)
```

Figura B.2: Algoritmo para o computo da operação de relação '$X > c$'

```
CompProbOfEq(c, X)
1:
2: ///////Lê cada um dos
3: ///////intervalos de X
4: for all I ∈ X do
5:   if c ∈ I then
6:     if (c = I.L) ∨ (c = I.H) then
7:       return I.ρ/2
8:     else
9:       return I.ρ
10:    end if
11:  end if
12: end for
13: return 0
```

Figura B.3: Algoritmo para o computo da operação de relação '$X = c$'

Apêndice C – Algoritmos utilizados no cômputo da distribuição de saída

```
CompMultTreeDist(G_M)

//////////Computa a PD para cada nó do G_M sem considerar as entradas de dados
for all n ∈ TSort(N(G_M) − PI(G_M)) do
  InputA ← n.InputA
  InputB ← n.InputB
  DefinePoints(n.Rslt,InputA.Rslt,InputB.Rslt)
  for all IntervalA ∈ InputA do
    for all IntervalB ∈ InputB do
      for all IntervalRslt ∈ n do
        LowLimit = IntervalRslt.L − 0.5
        HighLimit = IntervalRslt.H + 0.5
        ProbLow =PDofMult(LowLimit,IntervalA,IntervalB)
        ProbHigh =PDofMult(HighLimit,IntervalA,IntervalB)
        IntervalRslt.ρ = IntervalRslt.ρ + (ProbHigh−ProbLow)/(HighLimit−LowLimit−1)
      end for
    end for
  end for
end for

//////////Retorna a PD de saída do Componente
return PO(G_M).Rslt
```

Figura C.1: Algoritmo para Cômputo da PD dos componentes da equação

```
PDofMult(v', IntervalA, IntervalB)
///////O IntervalA é uma constante?
if IsCnst(IntervalA) then
  c = IntervalA.L ///////Valor da Constante do IntervalA
  β = IntervalA.ρ ///////Distribuição da Constante de IntervalA
  if c = 0 then
    return 0
  else if C > 0 then
    Rslt.L = IntervalB.L * c ///////Limite inferior resultante
    Rslt.H = IntervalB.H * c ///////Limite superior resultante
  else
    Rslt.L = IntervalB.H * c ///////Limite superior resultante
    Rslt.H = IntervalB.L * c ///////Limite inferior resultante
  end if
  Rslt.ρ = (β*IntervalB.ρ)/|c| ///////Distribuição Resultante
  ///////Executa a equação 4.4, página 43
  if v' ≤ Rslt.L then
    return 0
  else if v' < Rslt.H then
    return Rslt.ρ * (v' − Rslt.L)
  else
    return Rslt.ρ * (Rslt.H − Rslt.L)
  end if
  ///////O IntervalB é uma constante?
else if IsCnst(IntervalB) then
  c = IntervalB.L ///////Valor da Constante do IntervalB
  β = IntervalA.ρ ///////Distribuição da Constante de IntervalB
  if c = 0 then
    return 0
  else if c > 0 then
    Rslt.L = IntervalA.L * c ///////Limite inferior resultante
    Rslt.H = IntervalA.H * c ///////Limite superior resultante
  else
    Rslt.L = IntervalA.H * c ///////Limite superior resultante
    Rslt.H = IntervalA.L * c ///////Limite inferior resultante
  end if
  Rslt.ρ = (β*IntervalA.ρ)/|c| ///////Distribuição Resultante
  ///////Executa a equação 4.4, página 43
  if v' ≤ Rslt.L then
    return 0
  else if v' < Rslt.H then
    return Rslt.ρ * (v' − Rslt.L)
  else
    return Rslt.ρ * (Rslt.H − Rslt.L)
  end if
  ///////O IntervalA e IntervalB não são constantes
else
  ///////Executa equação 4.5, página 45
  return PDofMultVarVar(v',w,v)
end if
```

Figura C.2: Algoritmo para calcular o PD da Multiplicação de duas variáveis independentes

```
CompAddTreeDist(G_A)
1:
2: ///////////Computa a PD para cada nó do G_A sem considerar as entradas de dados
3: for all n ∈ TSort(N(G_A) − PI(G_A)) do
4:    InputA ← n.InputA
5:    InputB ← n.InputB
6:    DefinePoints(n.Rslt,InputA.Rslt,InputB.Rslt)
7:    for all IntervalA ∈ InputA do
8:       for all IntervalB ∈ InputB do
9:          for all IntervalRslt ∈ n do
10:            LowLimit = IntervalRslt.L − 0.5
11:            HighLimit = IntervalRslt.H + 0.5
12:            ProbLow =PDofSum(LowLimit,IntervalA,IntervalB)
13:            ProbHigh =PDofSum(HighLimit,IntervalA,IntervalB)
14:            IntervalRslt.ρ = IntervalRslt.ρ + (ProbHigh−ProbLow)/(HighLimit−LowLimit−1)
15:         end for
16:      end for
17:   end for
18: end for
19:
20: ///////////Retorna a PD de saída da equação
21: return PO(G_A).Rslt
```

Figura C.3: Algoritmo para Cômputo da PD da equação

```
PDofSum(v', IntervalA, IntervalB)
///////O IntervalA é uma constante?
if IsCnst(IntervalA) then
  c = IntervalA.L ///////Valor da Constante do IntervalA
  β = IntervalA.ρ ///////Distribuição da Constante de IntervalA
  Rslt.L = IntervalB.L + c ///////Limite inferior resultante
  Rslt.H = IntervalB.H + c ///////Limite superior resultante
  Rslt.ρ = β * IntervalB.ρ ///////Distribuição Resultante
  if v' ≤ Rslt.L then
    return 0
  else if v' < Rslt.H then
    return Rslt.ρ * (v' − Rslt.L)
  else
    return Rslt.ρ * (Rslt.H − Rslt.L)
  end if
  ///////O IntervalB é uma constante?
else if IsCnst(IntervalB) then
  c = IntervalB.L ///////Valor da Constante do IntervalB
  β = IntervalA.ρ ///////Distribuição da Constante de IntervalB
  Rslt.L = IntervalA.L + c ///////Limite inferior resultante
  Rslt.H = IntervalA.H + c ///////Limite superior resultante
  Rslt.ρ = β * IntervalA.ρ ///////Distribuição Resultante
  if v' ≤ Rslt.L then
    return 0
  else if v' < Rslt.H then
    return Rslt.ρ * (v' − Rslt.L)
  else
    return Rslt.ρ * (Rslt.H − Rslt.L)
  end if
  ///////O IntervalA e IntervalB não são constantes
else
  v_0 = IntervalA.L + IntervalB.L ///////Estabelece os valores dos conjuntos v
  v_3 = IntervalA.H + IntervalB.H
  if (IntervalA.H − IntervalA.L) < (IntervalB.H − IntervalB.L) then
    w_0 = IntervalA.L w_1 = IntervalA.H
    w_2 = IntervalB.L w_3 = IntervalB.H
    v_1 = IntervalA.H + IntervalB.L
    v_2 = IntervalA.L + IntervalB.H
  else
    w_0 = IntervalB.L w_1 = IntervalB.H
    w_2 = IntervalA.L w_3 = IntervalA.H
    v_1 = IntervalA.L + IntervalB.H
    v_2 = IntervalA.H + IntervalB.L
  end if
  ///////Executa equação 4.5, página 45
  return PDofSumVarVar(v',w,v)
end if
```

Figura C.4: Algoritmo para calcular o PD da soma de duas variáveis independentes

```
CompOutProbDist(PO, ρ)
////////////////FAZ A ANÁLISE POR CADA UMA DAS SAÍDAS DO MODELO
for all po ∈ PO do
   ///////Calcula a PD de cada equação que define a saída (Seção 4.3, página 54)
   for i = 0 → po.NumOfPathEq − 1 do
      ProDist =CompPDofVar(po.PathEq(i))
      po.PathEq(i).Rslt = po.PathProb(i) ∗ ProDist
   end for

   ///////Calcula a PD da Saída pela superposição de PD's
   ///////de cada uma das equações definidas pelos caminhos da FSMD
   ///////calculadas na etapa anterior (Seção 4.4.1, página 56)
   po.Rslt = 0
   for i = 0 → po.NumOfPathEq − 1 do
      po.Rslt = po.Rslt + po.PathEq(i).Rslt
   end for
end for
```

Figura C.5: Algoritmo para o computo da Distribuição Probabilísticas das variáveis de saída

Apêndice D – Cômputo de todos os casos da PD da multiplicação

Neste capítulo mostram-se as considerações para a determinação da Densidade Probabilística (PD) da Multiplicação para o caso de intervalos limitados e de distribuição uniforme. Diferente da PD da soma, à multiplicação tem mais de um caso para se determinar a PD $F(v) = P(X + Y \leq v)$. Abaixo se mencionam as notações usadas são relacionadas:

$X =$ Variável aleatória de entrada X.

$Y =$ Variável aleatória de entrada Y.

$V =$ Variável aleatória de saída V.

$\Omega_x =$ Espaço de possíveis valores que a variável aleatória X tem.

$\Omega_y =$ Espaço de possíveis valores que a variável aleatória Y tem.

$\Omega_v =$ Espaço de possíveis valores que a variável aleatória V tem.

$[a, b] =$ Intervalo de a até b, onde $\Omega_x = [a, b] \in \mathbb{R}$.

$[c, d] =$ Intervalo de c até d, onde $\Omega_y = [c, d] \in \mathbb{R}$.

$x =$ Valor específico dentro do espaço definido para que variável aleatória X.

$y =$ Valor específico dentro do espaço definido para que variável aleatória Y.

$v =$ Valor específico dentro do espaço definido para que variável aleatória V.

Para determinar a $F(v)$ emprega-se a equação geral D.1.

$$F(v) = P(X + Y = V \leq v) = \int_{-\infty}^{\infty} f_x(x) f_y(y) dx dy \tag{D.1}$$

Fazendo uma translação linear, a equação D.1 é igual à equação D.2, nesta seção; em alguns casos, essa expressão vai-se simplificar por $\iint$.

$$\iint \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v \tag{D.2}$$

Para poder resolver o problema, este é divido em três condições principais:

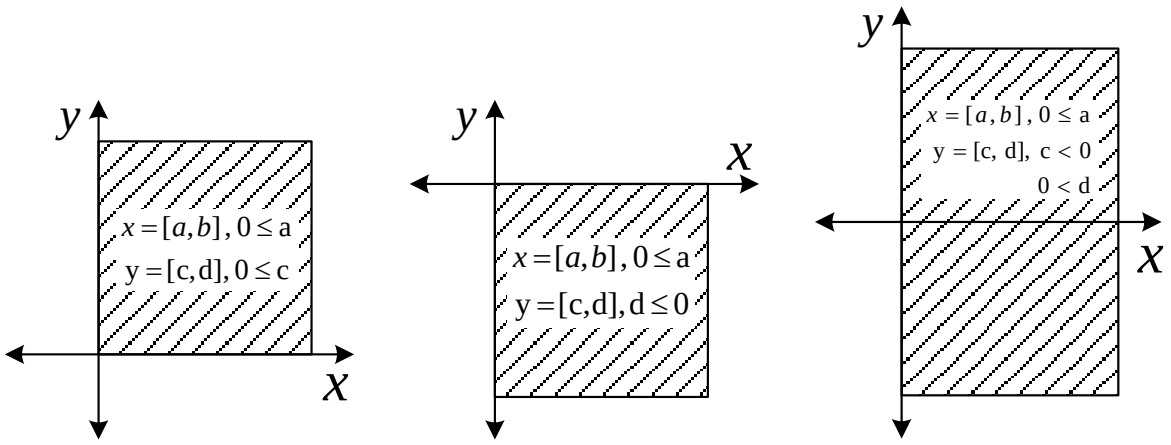
1. $x \in [a, b]$, $0 \leq a$
2. $x \in [a, b]$, $b \leq 0$
3. $x \in [a, b]$, $a < 0$, $0 < b$

Cada um destes casos define as áreas de integração que são empregadas no cálculo da PD; estas são explicadas na seção D.1. Na seção D.2 apresentam-se as similaridades entre os sub-casos e como, a partir deles podem-se gerar equações gerais. A última seção (D.3) mostra os algoritmos baseados nas equações determinadas em D.2.

D.1 Cômputo da integral

D.1.1 Caso $x \in [a, b]$, $0 \leq a$

O setor do plano XY usado neste caso é apresentado na figura D.1, a qual mostra 3 condições diferentes. A partir delas, definem-se as áreas de computo $F(v)$.



(a) $0 \leq a$, $0 \leq c$ (b) $0 \leq a$, $0 \leq c$ (c) $0 \leq a$, $0 \leq c$

Figura D.1: Espaço Amostral das variáveis aleatórias quando $x \in [a, b]$,$0 \leq a$

D.1.1.1 $a = 0$

D.1.1.1.1 Quando $y \in [c, d]$, $c = 0$ Neste caso, utiliza-se a área definida na figura D.1(a).

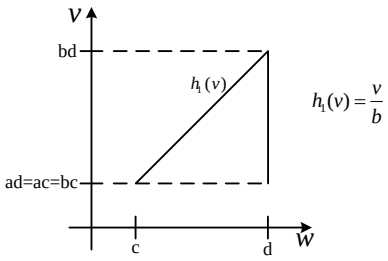


Figura D.2: Definição da área de Integração para o caso de $a = 0$ e $c = 0$

D.1.1.1.2 Quando $y \in [c, d]$, $0 < c$ Neste caso se utiliza a área definida na figura D.1(a).

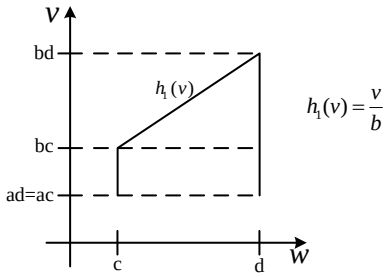


Figura D.3: Definição da área de Integração para o caso de $a = 0$ e $c < 0$

Para os casos apresentados nas figuras D.2 e D.3 o PDF destas pode ser obtido com a expressão apresentada na equação D.3, onde $v_0 = ac$, $v_1 = ad$, $v_2 = bc$, $v_3 = bd$ $w_0 = c$, $w_1 = d$ e $h_1(v) = \frac{v(w_1 - w_0)}{v_3 - v_2}$.

$$F(v) = \begin{cases} F_1(v) = \int_{v_1}^{v} \int_{w_0}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_1 < v \leq v_2 \\ F_2(v) = \int_{v_2}^{v} \int_{h_1(v)}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_1(v_2) & \text{if } v_2 < v \leq v_3 \end{cases} \tag{D.3}$$

D.1.1.1.3 Quando $y \in [c, d]$, $d = 0$ Neste caso se utiliza a área definida na figura D.1(b).

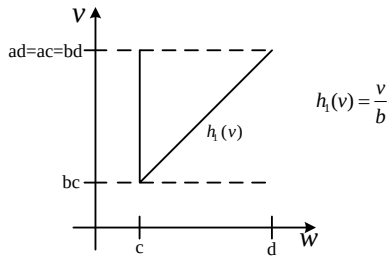


Figura D.4: Definição da área de Integração para o caso de $a = 0$ e $d = 0$

D.1.1.1.4 Quando $y \in [c, d]$, $d < 0$ Neste caso se utiliza a área definida na figura D.1(b).

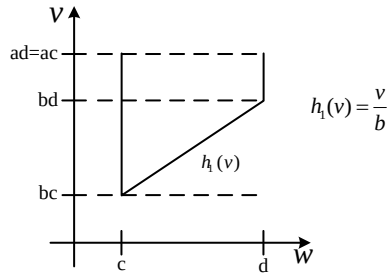


Figura D.5: Definição da área de Integração para o caso de $a = 0$ e $d < 0$

Para os casos apresentados nas figuras D.4 e D.5 o PDF destas pode ser obtido com a expressão apresentada na equação D.4, onde $v_0 = bc$, $v_1 = bd$, $v_2 = ac$, $v_3 = ad$ $w_0 = c$, $w_1 = d$ e $h_1(v) = \frac{v(w_1 - w_0)}{v_1 - v_0}$.

$$F(v) = \begin{cases} F_1(v) = \int_{v_0}^{v} \int_{w_0}^{h_1(v)} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int_{v_1}^{v} \int_{w_0}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_1(v_1) & \text{if } v_1 < v \leq v_2 \end{cases} \tag{D.4}$$

D.1.1.1.5 Quando $y \in [c, d]$, $c < 0$, $0 < d$ Neste caso se utiliza a área definida na figura D.1(c).

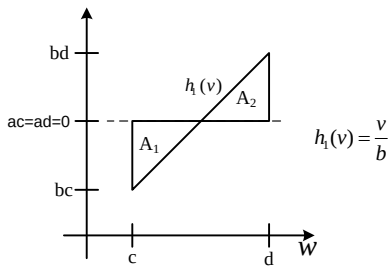


Figura D.6: Definição da área de Integração para o caso de $a = 0$, $c < 0$ e $0 < d$

O caso apresentado na figura D.6 também pode-se generalizar com os casos apresentados nas figuras D.2 e D.4, gerando a equação D.5, onde $v_0 = bc$, $v_1 = ac$, $v_2 = ad$, $v_3 = bd$ $w_0 = c$ e $w_1 = d$, e $h_1(v) = \frac{v(w_1 - w_0)}{v_3 - v_0}$.

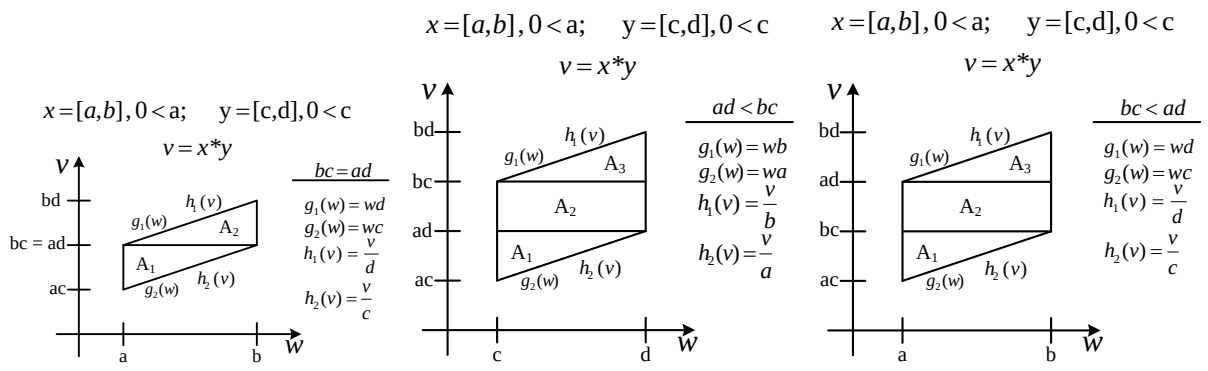
$$F(v) = \begin{cases} F_1(v) = \int_{v_0}^{v} \int_{w_0}^{h_1(v)} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int_{v_1}^{v} \int_{h_1(v)}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_1(v_2) & \text{if } v_1 < v \leq v_3 \end{cases} \tag{D.5}$$

D.1.1.2 $0 < a$

D.1.1.2.1 Quando $y \in [c, d]$, $c = 0$ Neste caso se utiliza a área definida na figura D.1(a).

Este caso é similar ao apresentado na seção D.1.1.1 quando $0 < c$ na figura D.3, mas os valores dos parâmetros diferem: $v_0 = ac$, $v_1 = bc$, $v_2 = ad$, $v_3 = bd$, $w_0 = a$, $w_1 = b$ e $h_1(v) = \frac{v(w_1 - w_0)}{v_3 - v_2}$. e a equação que define o PDF é a D.3.

D.1.1.2.2 Quando $y \in [c, d]$, $0 < c$ Neste caso se utiliza a área definida na figura D.1(a).



(a) $ad = bc$ (b) $ad < bc$ (c) $bc < ad$

Figura D.7: Definição do espaço de integração quando $0 < a$, $0 < c$

Das figuras apresentadas, pode-se observar que duas delas (D.7(a) e D.7(c)), têm $w \in [a, b]$, mas na figura D.7(b), a variável $w \in [c, d]$. Isto foi feito para obter similaridade nas formas das áreas de integração, o qual permite generalizai-las como se apresentada na figura D.8 (igual às figuras D.7(b) e D.7(c)).

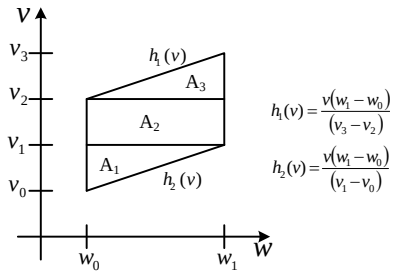


Figura D.8: Forma Geral das gráficas apresentadas na figura D.7

A partir desta figura pode-se gerar a equação da Função de Distribuição Probabilística (*PDF*, Probabilistic Density Function), a qual se mostra na equação D.6.

$$F(v) = \begin{cases} F_1(v) = \int_{v_0}^{v} \int_{w_0}^{h_2(v)} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int_{v_1}^{v} \int_{w_0}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_1(v_1) & \text{if } v_1 < v \leq v_2 \\ F_3(v) = \int_{v_2}^{v} \int_{h_1(v)}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_2(v_2) & \text{if } v_2 < v \leq v_3 \end{cases} \tag{D.6}$$

Neste caso não se está considerando o caso apresentado na figura D.7(a), já que se substitui os valores na equação D.6 a resposta desejada é a mesma.

D.1.1.2.3 Quando $y \in [c, d]$, $d = 0$ Neste caso utiliza-se a área definida na figura D.1(b).

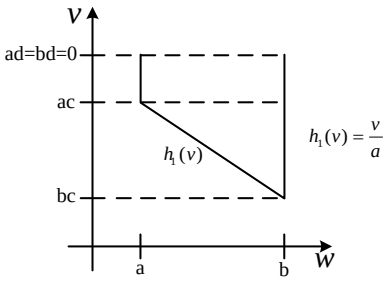
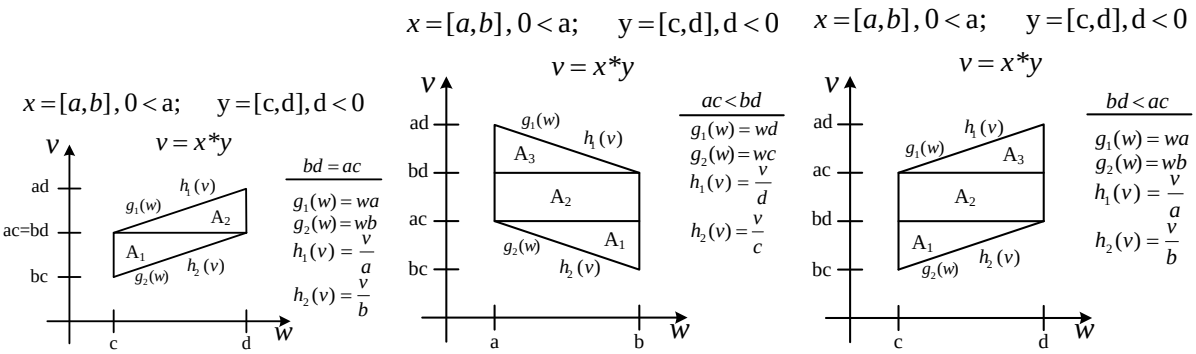


Figura D.9: Área de Integração para o caso $0 < a$ e $d = 0$

Para o caso apresentado na figura D.9, pode-se generalizar as equações utilizando: $w_0 = bc$, $w_1 = ac$, $w_2 = ad$, $w_3 = bd$, $w_0 = a$, $w_1 = b$ e $h_1(v) = \frac{v(v_1 - v_0)}{w_1 - w_0}$. A partir deles, valores a PDF é expressado na equação D.7

$$F(v) = \begin{cases} F_1(v) = \int_{v_0}^{v} \int_{h_1(v)}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int_{v_1}^{v} \int_{w_0}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_1(v_1) & \text{if } v_1 < v \leq v_2 \end{cases} \tag{D.7}$$

D.1.1.2.4 Quando $y \in [c, d]$, $d < 0$ Neste caso utiliza-se a área definida na figura D.1(b).



(a) $ad = bc$ (b) $ac < bd$ (c) $bd < ac$

Figura D.10: Definição do espaço de integração quando $0 < a$, $d < 0$

Neste caso existem duas formas da área de integração: a primeira igual a usada na figura D.8 que correspondem às figuras D.10(a) e D.10(c), e uma segunda apresentada na figura D.11.

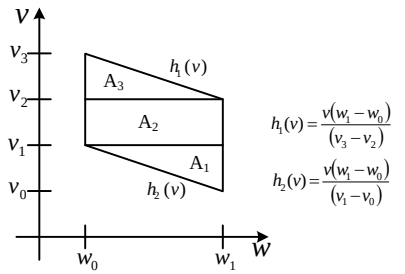


Figura D.11: Forma Geral da gráfica apresentada na figura D.10(b)

A área de integração definida pela figura anterior, determina a equação D.8.

$$F(v) = \begin{cases} F_1(v) = \int_{v_0}^{v} \int_{h_2(v)}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int_{v_1}^{v} \int_{w_0}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_1(v_1) & \text{if } v_1 < v \leq v_2 \\ F_3(v) = \int_{v_2}^{v} \int_{w_0}^{h_1(v)} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_2(v_2) & \text{if } v_2 < v \leq v_3 \end{cases} \tag{D.8}$$

D.1.1.2.5 Quando $y \in [c, d]$ $c < 0$, $0 < d$ Neste caso utiliza-se a área definida na figura D.1(c).

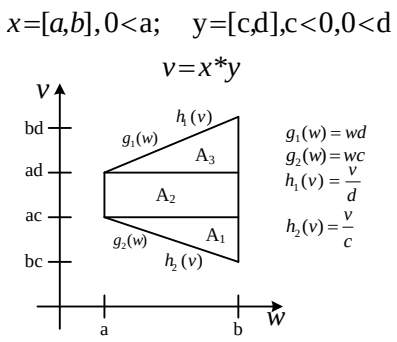


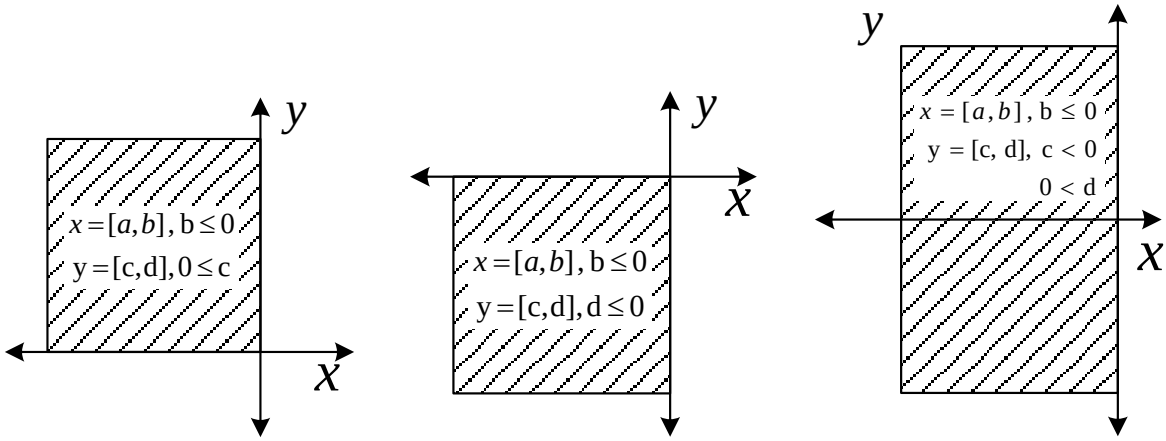
Figura D.12: Definição do espaço de integração quando $0 < a$, $0 < c$, $0 < d$

Tendo em conta as expressões usadas anteriormente, tanto na equação D.6 e D.8, faz-se as seguintes transformações: $bc = v_0$, $ac = v_1$, $ad = v_2$, $bd = v_3$, $a = w_0$, $b = w_1$. A partir delas obtém-se a equação D.9.

$$F(v) = \begin{cases} F_1(v) = \int\limits_{v_0}^{v} \int\limits_{h_2(v)}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int\limits_{v_1}^{v} \int\limits_{w_0}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_1(v_1) & \text{if } v_1 < v \leq v_2 \\ F_3(v) = \int\limits_{v_2}^{v} \int\limits_{h_1(v)}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_2(v_2) & \text{if } v_2 < v \leq v_3 \end{cases} \tag{D.9}$$

D.1.2 Caso $x \in [a, b]$, $b \leq 0$

O setor do plano XY usado neste caso é apresentado na figura D.13, a qual mostra 3 condições diferentes. A partir destas, se define as áreas de computo $F(v)$.



(a) $b \leq 0$, $0 \leq c$ (b) $b \leq 0$, $0 \leq c$ (c) $b \leq 0$, $0 \leq c$

Figura D.13: Espaço Amostral das variáveis aleatórias quando $\Omega_x \in \mathbb{R}$ e $\Omega_y \in \mathbb{R}^-$

D.1.2.1 $b = 0$

D.1.2.1.1 Quando $y \in [c, d]$, $c = 0$ Neste caso se utiliza a área definida na figura D.13(a).

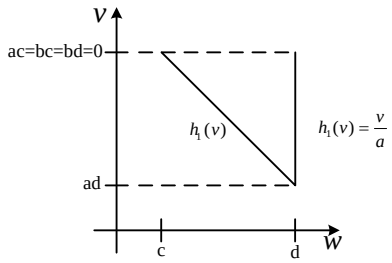


Figura D.14: Área de Integração para o caso de $b = 0$ e $c = 0$

D.1.2.1.2 Quando $y \in [c, d]$, $0 < c$ Neste caso se utiliza a área definida na figura D.13(a).

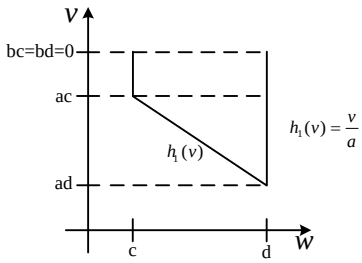


Figura D.15: Área de Integração para o caso de $b = 0$ e $0 < c$

Para os casos apresentados nas figuras D.14 e D.15 o PDF destas pode ser obtido com a expressão apresentada na equação D.10, onde $v_0 = ad$, $v_1 = ac$, $v_2 = bc$, $v_3 = bd$ $w_0 = c$, $w_1 = d$ e $h_1(v) = \frac{v(w_1 - w_0)}{v_3 - v_2}$.

$$F(v) = \begin{cases} F_1(v) = \int_{v_0}^{v} \int_{h_1(v)}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int_{v_1}^{v} \int_{w_0}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_1(v_1) & \text{if } v_1 < v \leq v_2 \end{cases} \tag{D.10}$$

D.1.2.1.3 Quando $y \in [c, d]$, $d = 0$ Neste caso se utiliza a área definida na figura D.13(b).

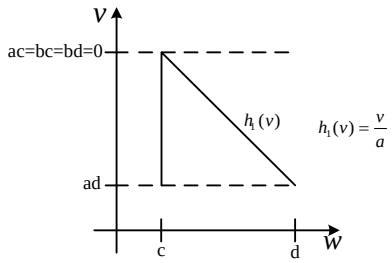


Figura D.16: Área de Integração para o caso de $b = 0$ e $d = 0$

D.1.2.1.4 Quando $y \in [c, d]$, $d < 0$ Neste caso se utiliza a área definida na figura D.13(b).

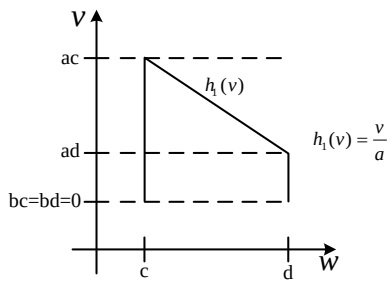


Figura D.17: Área de Integração para o caso de $b = 0$ e $d < 0$

Para os casos apresentados nas figuras D.21 e D.17 o PDF destas pode ser obtido com a expressão apresentada na equação D.11, onde $v_0 = bc$, $v_1 = bd$, $v_2 = bd$, $v_3 = bc$ $w_0 = c$, $w_1 = d$ e $h_1(v) = \frac{v(w_1 - w_0)}{v_3 - v_2}$.

$$F(v) = \begin{cases} F_1(v) = \int_{v_0}^{v} \int_{w_0}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_0 < v \leq v_2 \\ F_2(v) = \int_{v_2}^{v} \int_{w_0}^{h_1(v)} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_1(v_2) & \text{if } v_2 < v \leq v_3 \end{cases} \tag{D.11}$$

D.1.2.1.5 Quando $y \in [c, d]$, $c < 0$, $0 < d$ Neste caso se utiliza a área definida na figura D.13(c).

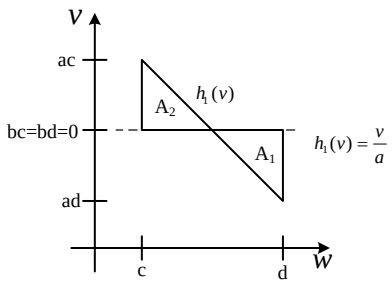


Figura D.18: Área de Integração para o caso de $b = 0$ e $c < 0$ e $0 < d$

O caso apresentado na figura D.18 também pode-se generalizar com os casos apresentados nas figuras D.14 e D.15, gerando a equação D.12, onde $v_0 = bc$, $v_1 = ac$, $v_2 = ad$, $v_3 = bd$ $w_0 = c$ e $w_1 = d$, e $h_1(v) = \frac{v(w_1 - w_0)}{v_3 - v_0}$.

$$F(v) = \begin{cases} F_1(v) = \int\limits_{v_0}^{v} \int\limits_{h_1(v)}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int\limits_{v_2}^{v} \int\limits_{w_0}^{h_1(v)} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_1(v_2) & \text{if } v_2 < v \leq v_3 \end{cases} \tag{D.12}$$

D.1.2.2 $b < 0$

D.1.2.2.1 Quando $y \in [c, d]$, $c = 0$ Neste caso se utiliza a área definida na figura D.13(a).

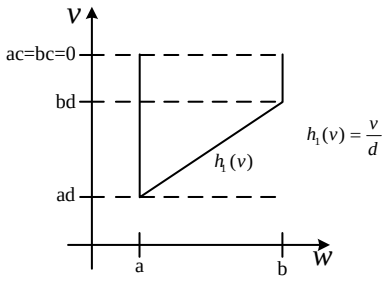
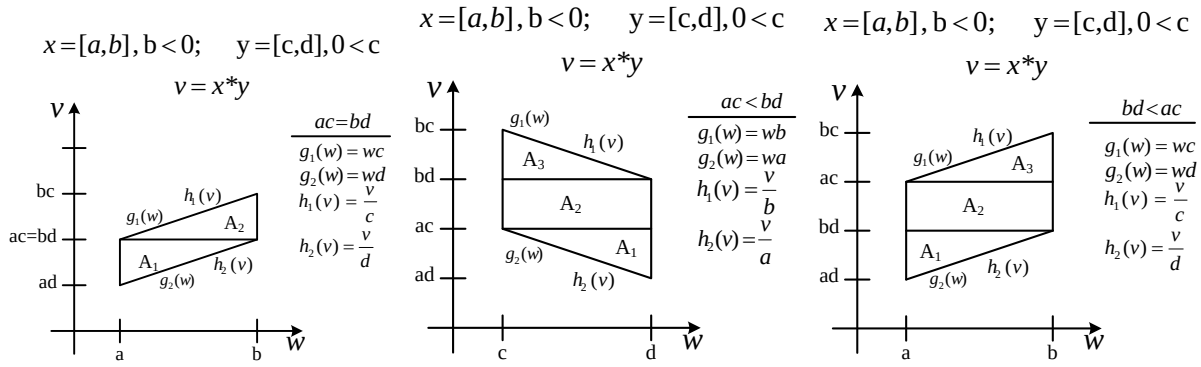


Figura D.19: Área de Integração para o caso de $b < 0$ e $c = 0$

O caso apresentado na figura D.19 é similar ao apresentado na seção D.1.1.1 quando $d < 0$. Baseado nesto se pode generalizar os parâmetros com os seguintes valores: $v_0 = ad$, $v_1 = bd$, $v_2 = ac$, $v_3 = bc$, $w_0 = a$, $w_1 = b$ e $h_1(v) = \frac{v(w_1 - w_0)}{v_1 - v_0}$.

D.1.2.2.2 Quando $y \in [c, d]$, $0 < c$ Neste caso se utiliza a área definida na figura D.13(a).



(a) $ac = bd$ (b) $ac < bd$ (c) $bd < ac$

Figura D.20: Definição do espaço de integração quando $b < 0$, $0 < c$

Ao igual que no caso apresentado na seção D.1.1.2 para o caso $d < 0$, as gráficas apresentadas na figura D.20 tem áreas de integração similares, e, da mesma forma que aquelas. As equações gerais que determinam o PDF são determinadas, por tanto, pelas equações D.6 e D.8.

D.1.2.2.3 Quando $y \in [c, d]$, $d = 0$ Neste caso se utiliza a área definida na figura D.13(b).

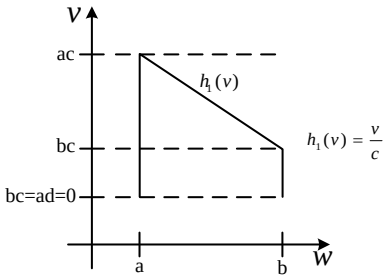
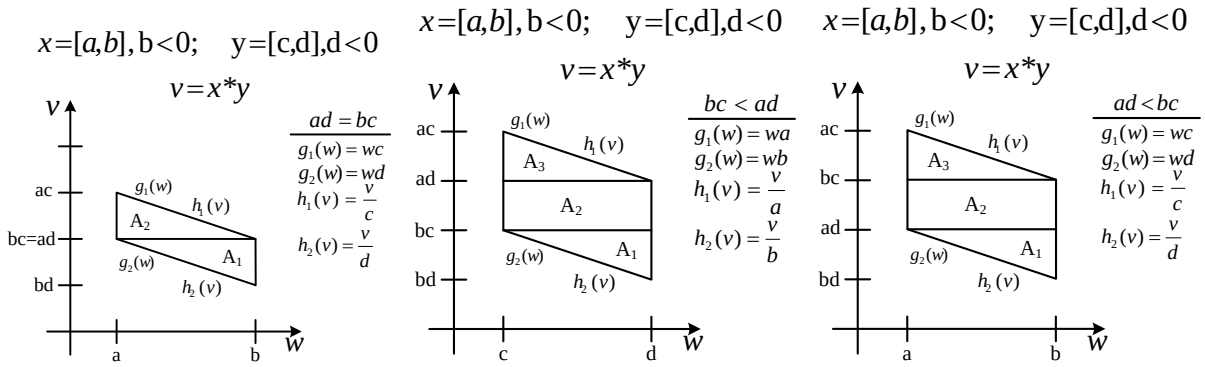


Figura D.21: Área de Integração para o caso de $b < 0$ e $d = 0$

D.1.2.2.4 Quando $y \in [c, d]$, $d < 0$ Neste caso se utiliza a área definida na figura D.13(b).



(a) $bc = ad$ (b) $bc < ad$ (c) $ad < bc$

Figura D.22: Definição do espaço de integração quando $b < 0$, $d < 0$

O caso apresentado na figura D.22 é similar ao caso apresentado na seção D.1.1.2 para o caso $0 < c$ (Figura D.7). Por tanto a equação que define este caso é a equação D.6.

D.1.2.2.5 Quando $y \in [c, d]$ $c < 0$, $0 < d$ Neste caso se utiliza a área definida na figura D.13(c).

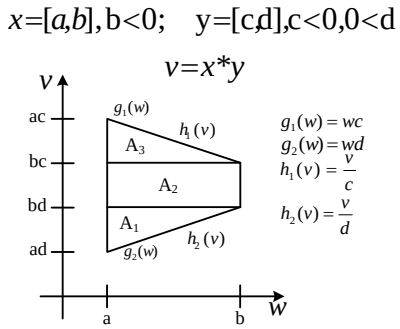


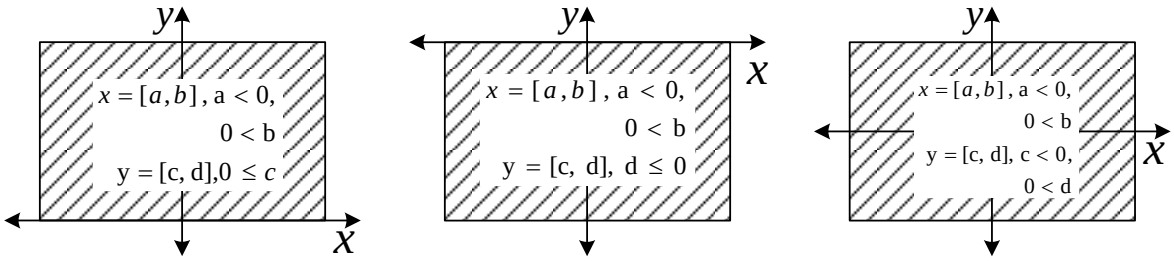
Figura D.23: Definição do espaço de integração quando $b < 0$, $c < 0$, $0 < d$

A partir da figura D.23 pode-se definir a PDF, expressada na equação D.13.

$$F(v) = \begin{cases} F_1(v) = \int_{v_0}^{v} \int_{w_0}^{h_2(v)} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int_{v_1}^{v} \int_{w_0}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_1(v_1) & \text{if } v_1 < v \leq v_2 \\ F_3(v) = \int_{v_2}^{v} \int_{w_0}^{h_1(v)} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v + F_2(v_2) & \text{if } v_2 < v \leq v_3 \end{cases} \tag{D.13}$$

D.1.3 Caso $x \in [a, b]$, $a < 0$, $0 < b$

O setor do plano XY usado neste caso é apresentado na figura D.24, a qual mostra 3 condições diferentes. A partir destas, se define as áreas de computo $F(v)$.



(a) $a < 0$, $0 < b$, $0 \leq c$ (b) $a < 0$, $0 < b$, $d \leq 0$ (c) $a < 0$, $0 < b$, $c < 0$, $0 < d$

Figura D.24: Espaço Amostral das variáveis aleatórias quando $\Omega_x \in [a, b]$,$a < 0$, $0 < b$

D.1.3.0.6 Quando $y \in [c, d]$, $c = 0$ Neste caso se utiliza a área definida na figura D.24(a).

Este caso é similar ao apresenta na seção D.1.1.1 quando $c < 0$ e $0 < d$ na figura D.6, mas os valores dos parâmetros diferem: $v_0 = ad$, $v_1 = bc$, $v_2 = ac$, $v_3 = bd$, $w_0 = a$, $w_1 = b$ e $h_1(v) = \frac{v(w_1 - w_0)}{v_3 - v_0}$. e a equação que define o PDF é a D.5.

D.1.3.0.7 Quando $y \in [c, d]$, $0 < c$ Neste caso se utiliza a área definida na figura D.24(a).

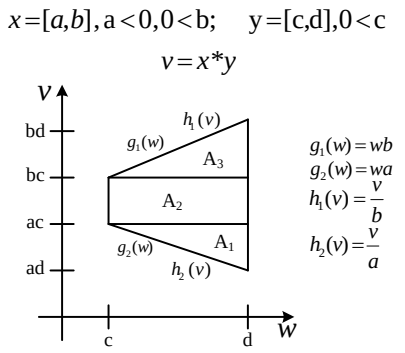


Figura D.25: Definição do espaço de integração quando $a < 0$, $0 < b$, $0 < c$

O caso apresentado na figura D.25 é similar ao caso apresentado na seção D.1.1.2 para o caso de $c < 0$, $0 < d$, mas com a diferença de que $v_0 = ad$, $v_1 = ac$, $v_2 = bc$, $v_3 = bd$, $w_0 = c$, $w_1 = d$. Por tanto a PDF deste caso é definido pela equação D.9.

D.1.3.0.8 Quando $y \in [c, d]$, $d = 0$ Neste caso se utiliza a área definida na figura D.24(b).

Este caso é similar ao apresenta na seção D.1.2.1 quando $c < 0$ e $0 < d$ na figura D.18, mas os valores dos parâmetros diferem: $v_0 = bc$, $v_1 = ad$, $v_2 = bd$, $v_3 = ac$, $w_0 = a$, $w_1 = b$ e $h_1(v) = \frac{v(w_1 - w_0)}{v_3 - v_0}$. e a equação que define o PDF é a D.12.

D.1.3.0.9 Quando $y \in [c, d]$, $d < 0$ Neste caso se utiliza a área definida na figura D.24(b).

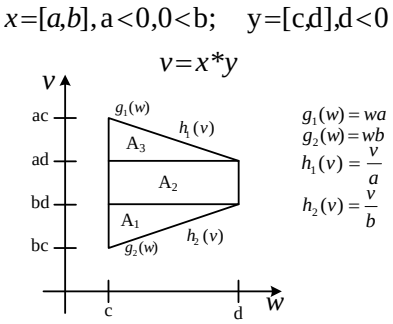
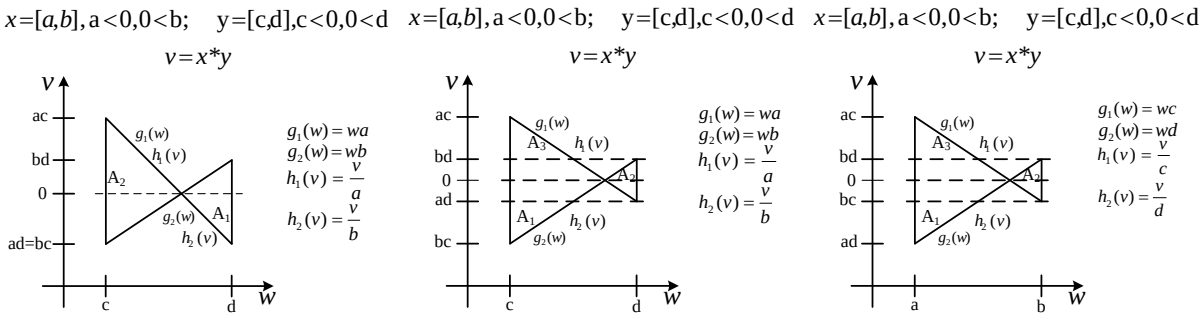


Figura D.26: Definição do espaço de integração quando $a < 0$, $0 < b$, $d < 0$

O caso apresentado na figura D.26 é similar ao caso apresentado na seção D.1.2.2 para o caso de $c < 0$, $0 < d$, mas com a diferença de que $v_0 = bc$, $v_1 = bd$, $v_2 = ad$, $v_3 = ac$, $w_0 = c$, $w_1 = d$. Por tanto o PDF deste caso é definido pela equação D.13.

D.1.3.0.10 Quando $y \in [c, d]$ $c < 0$, $0 < d$ Neste caso se utiliza a área definida na figura D.24(c).



(a) $bc = ad$ (b) $bc < ad$ (c) $ad < bc$

Figura D.27: Definição do espaço de integração quando $a < 0$, $0 < b$, $c < 0$, $0 < d$, $bd < ac$

Para o caso apresentado na figura D.27 se apresenta uma forma generalizada da área de integração determinada na figura D.28.

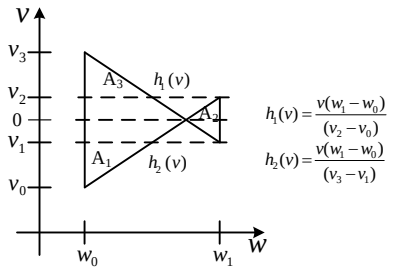
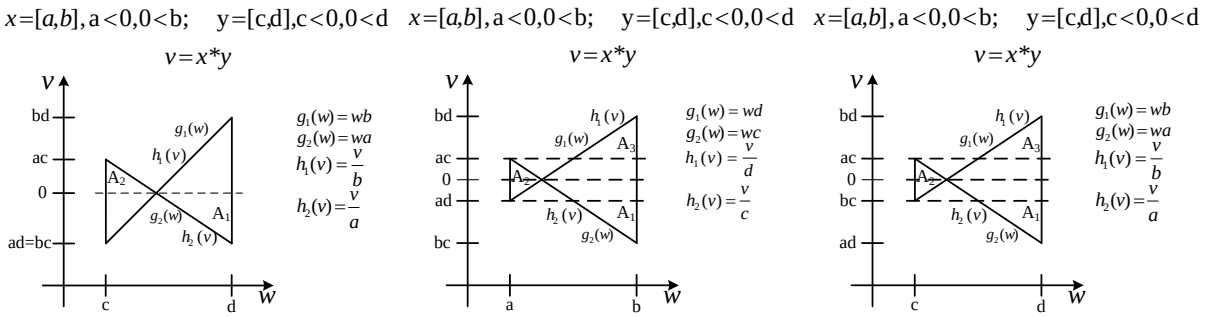


Figura D.28: Espaço de Integração Geral para os casos apresentados na figura D.27

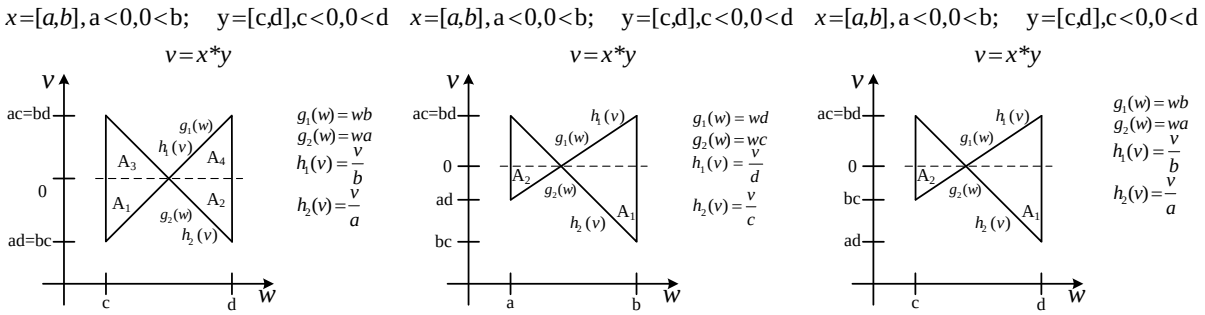
A partir da figura se determina a equação que determina o PDF para esta condição, a qual é mostrada a continuação.

$$F(v) = \begin{cases} F_1(v) = \int_{v_0}^{v} \int_{w_0}^{h_2(v)} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int_{v_0}^{v} \int_{w_0}^{h_2(v)} + \int_{v_1}^{v} \int_{h_1(v)}^{w_1} & \text{if } v_1 < v \leq 0 \\ F_3(v) = \int_{0}^{v} \int_{h_2(v)}^{w_1} + \int_{0}^{v} \int_{w_0}^{h_1(v)} + F_2(0) & \text{if } 0 < v \leq v_2 \\ F_4(v) = \int_{0}^{v_2} \int_{h_2(v)}^{w_1} + \int_{0}^{v} \int_{w_0}^{h_1(v)} + F_2(0) & \text{if } v_2 < v \leq v_3 \end{cases} \tag{D.14}$$



(a) $bc = ad$ (b) $bc < ad$ (c) $ad < bc$

Figura D.29: Definição do espaço de integração quando $a < 0$, $0 < b$, $c < 0$, $0 < d$, $ac < bd$



(a) $bc = ad$ (b) $bc < ad$ (c) $ad < bc$

Figura D.30: Definição do espaço de integração quando $a < 0$, $0 < b$, $c < 0$, $0 < d$, $ac = bd$

As figuras D.29 e D.30 podem-se generalizar com o gráfico mostrado na Figura D.31.

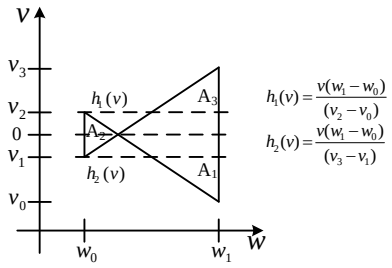


Figura D.31: Espaço de Integração Geral para os casos apresentados nas figuras D.29 e D.30

A partir desta se determina o PDF para este caso na equação D.15.

$$F(v) = \begin{cases} F_1(v) = \int_{v_0}^{v} \int_{h_1(v)}^{w_1} & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int_{v_0}^{v} \int_{h_1(v)}^{w_1} + \int_{v_1}^{v} \int_{w_0}^{h_2(v)} & \text{if } v_1 < v \leq 0 \\ F_3(v) = \int_{0}^{v} \int_{w_0}^{h_1(v)} + \int_{0}^{v} \int_{h_2(v)}^{w_1} + F_2(0) & \text{if } 0 < v \leq v_2 \\ F_4(v) = \int_{0}^{v_2} \int_{w_0}^{h_1(v)} + \int_{0}^{v} \int_{h_2(v)}^{w_1} + F_2(0) & \text{if } v_2 < v \leq v_3 \end{cases} \tag{D.15}$$

D.2 Definição de equações gerais

Nos casos das figuras apresentadas na seção D.2 se pode classificar em cinco casos das formas de áreas de integração. O primeiro caso, na figura D.32, cujos gráficos tem formas semejantes, mas a pendente das funções $h_1(v)$ e $h_2(v)$ variam. A segunda, figura D.33(a), referece ao caso $X = [a, b]$e $Y = [c, d]$, onde $a < 0$, $0 < b$ e $c = 0$ ou o segundo caso onde $c < 0$, $0 < d$ e $a = 0$ (Figura D.6). O terceiro, figura D.33(b), caso semilar ao anterior, mas com as condições restritas quando $a < 0$, $0 < b$ e $d = 0$ ou quando $c < 0$, $0 < d$ e $b = 0$ (Figura D.18). A quarta e a quinta forma, figuras D.34, correspondem aos mostrados na seção D.1.3 para o caso $a < 0$, $0 < b$,$c < 0$, $0 < d$.

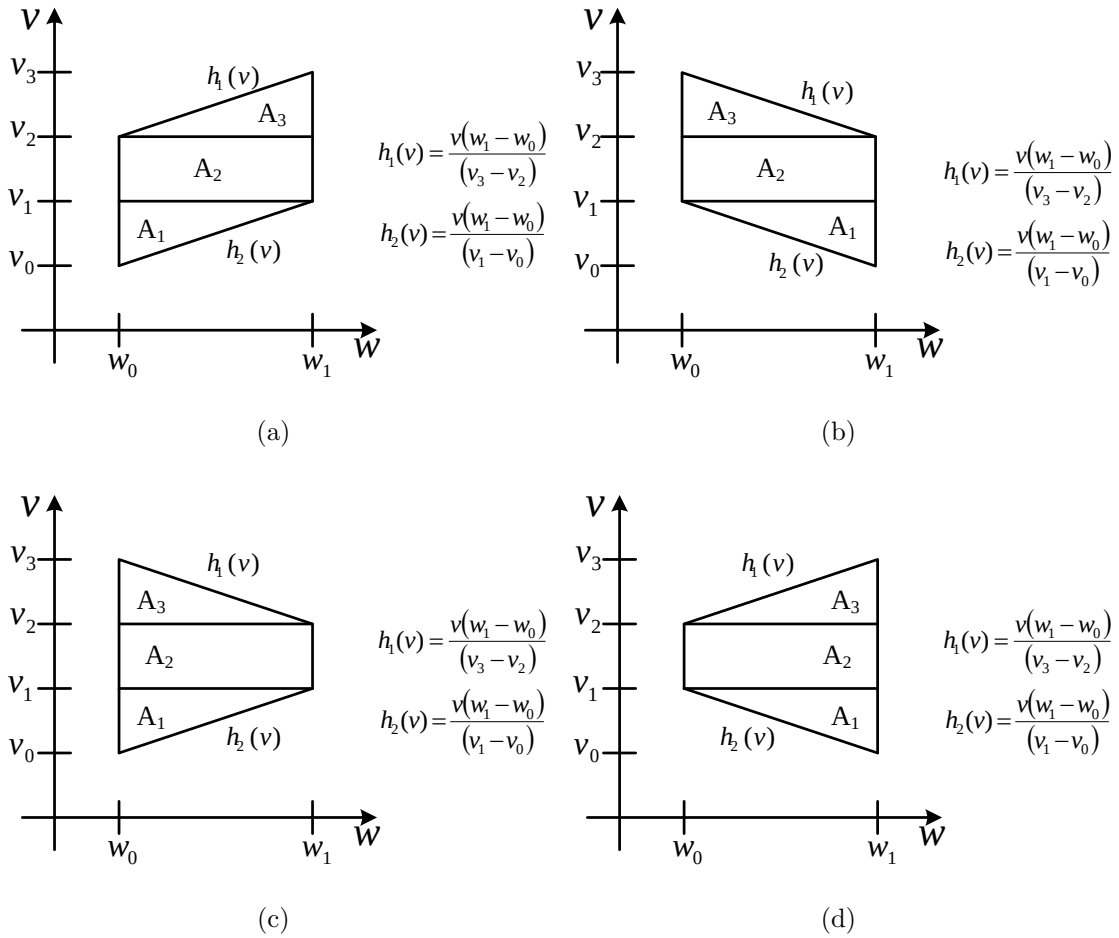


Figura D.32: Áreas de Integração Generalizada para o Caso 1

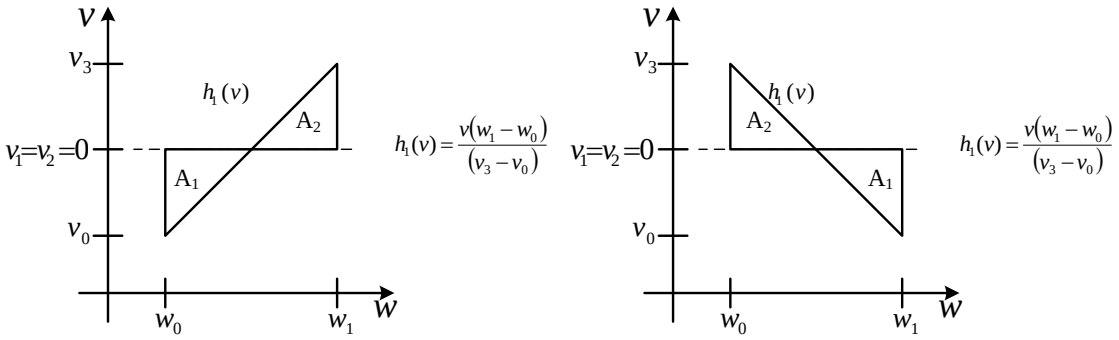
Para o primeiro caso (Figura D.32) pode-se observar que para os três casos $h_1(v)$ e $h_2(v)$ dependem dos mesmos valores para ser calculados, mas existe uma variação na pendente (Positiva ou Negativa). A variação da pendente determina os intervalos de integração, isto se pode observar, por exemplo, nas equaçãoes D.6, D.8, D.9 e D.13.

A partir da observação feita respeito ao analise da pendente pode-se determinar dois

tipos de equações para as áreas A_1 e A_3 expressadas na equação, onde $m_1 = \frac{w_1 - w_0}{v_3 - v_2}$ e $m_2 = \frac{w_1 - w_0}{v_1 - v_0}$

$$F(v) = \begin{cases} F_1(v) = \begin{cases} F_{1a}(v) = \int_{v_0}^{v} \int_{w_0}^{h_2(v)} & \text{if } m_2 > 0 \\ F_{1b}(v) = \int_{v_0}^{v} \int_{h_2(v)}^{w_1} & \text{if } m_2 < 0 \end{cases} & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \int_{v_1}^{v} \int_{w_0}^{w_1} + F_1(v_1) & \text{if } v_1 < v \leq v_2 \\ F_3(v) = \begin{cases} F_{3a}(v) = \int_{v_2}^{v} \int_{h_1(v)}^{w_1} + F_2(v_2) & \text{if } m_1 > 0 \\ F_{3b}(v) = \int_{v_2}^{v} \int_{w_0}^{h_1(v)} + F_2(v_2) & \text{if } m_1 < 0 \end{cases} & \text{if } v2 < v \leq v_3 \end{cases} \tag{D.16}$$

No segundo e terceiro caso, estes referem-se aos apresentados nas figuras D.6 e D.18. Estes têm como equações generalizadas: D.5 e D.12 respeitivamente. Nestas equações também pode-se observar o mesmo fenômeno do caso anterior, onde a pendente $m_1 = \frac{w_1 - w_0}{v_3 - v_0}$ pode ser positiva ou negativa, o que determina os intervalos de integração. Isto se mostra na equação D.17.

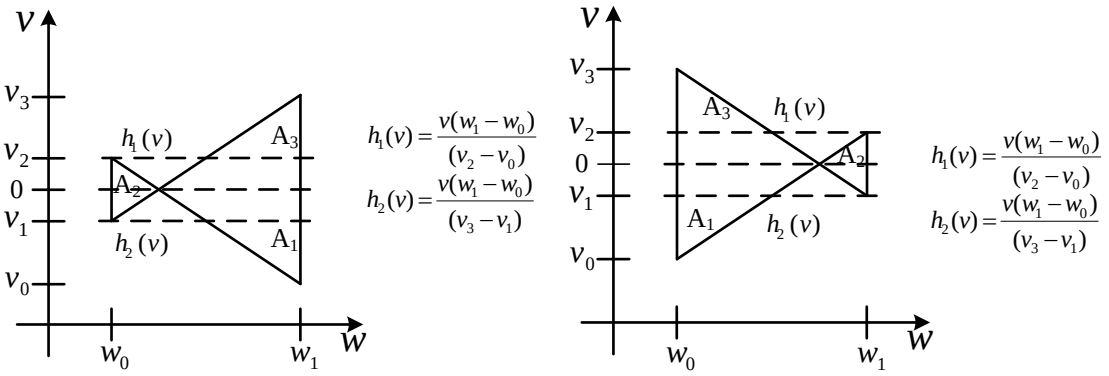


(a) Caso 2 (b) Caso 3

Figura D.33: Áreas de Integração Generalizada para os Casos 2 e 3

$$F(v) = \begin{cases} F_1(v) = \begin{cases} F_{1a}(v) = \int_{v_0}^{v} \int_{w_0}^{h_1(v)} \quad \text{if } m_1 > 0 \\ F_{1b}(v) = \int_{v_0}^{v} \int_{h_1(v)}^{w_1} \quad \text{if } m_1 < 0 \end{cases} & \text{if } v_0 < v \leq v_1 \\ F_2(v) = \begin{cases} F_{2a}(v) = \int_{v_2}^{v} \int_{h_1(v)}^{w_1} + F_1(v_2) \quad \text{if } m_1 > 0 \\ F_{2b}(v) = \int_{v_2}^{v} \int_{w_0}^{h_1(v)} + F_1(v_2) \quad \text{if } m_1 < 0 \end{cases} & \text{if } v2 < v \leq v_3 \end{cases} \tag{D.17}$$

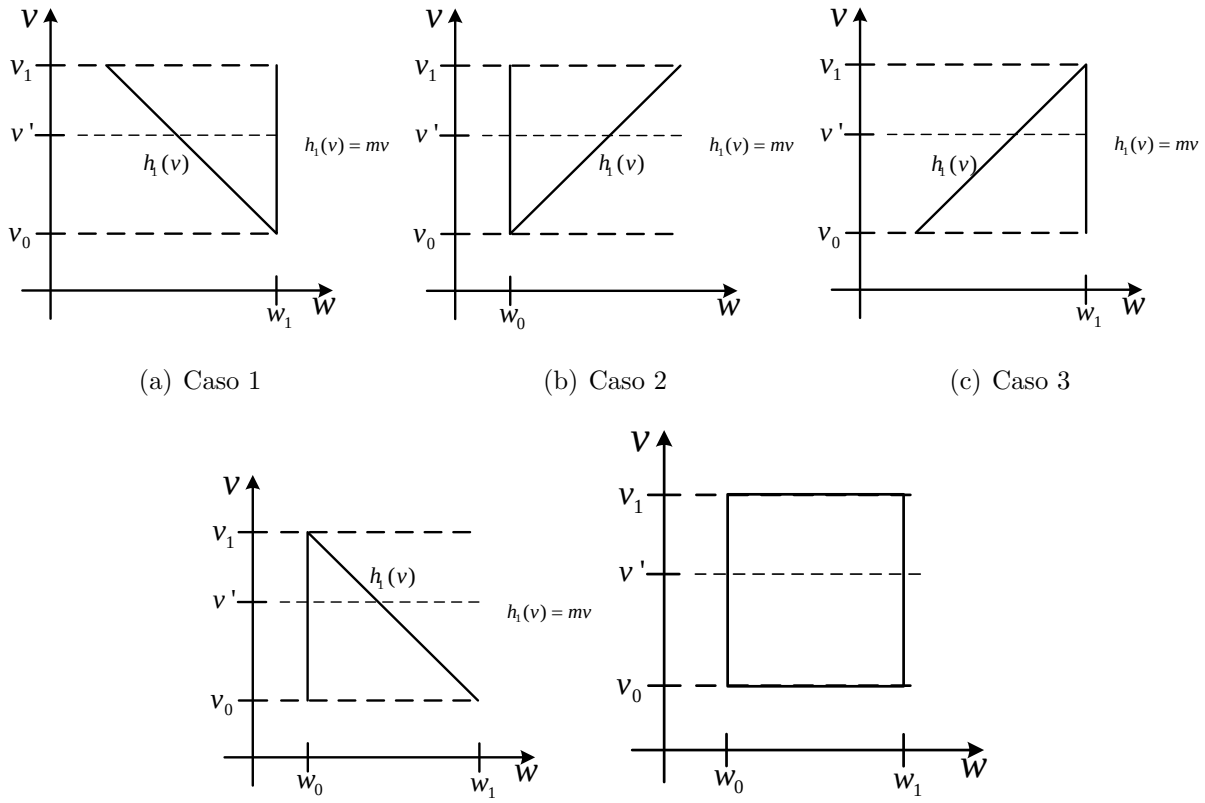
No quarto e quinto caso (Figura D.34) são representados na seção D.1.3 quando $c < 0$ e $0 < d$.



(a) Caso 4 (b) Caso 5

Figura D.34: Áreas de Integração Generalizada para os Casos 4 e 5

Nas equações propostas pode-se destingir 5 casos de limites de integração. Estes são apresentados na figura D.35, onde os 4 primeiros definem a pendente m da função $h_1(v)$.



(a) Caso 1 (b) Caso 2 (c) Caso 3

(d) Caso 4 (e) Caso 5

Figura D.35: Casos de Límites de Integração

Para os casos 1 e 3 da figura D.35 a equação D.18 define o PDF quando $f_x(x)$ e $f_y(y)$ tem uma distribuição uniforme em intervalos definidos.

$$F(v) = \int_{v_0}^{v} \int_{h_1(v)}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v = -\alpha\beta \left[v\ln\left(\left|\frac{h_1(v)}{w_1}\right|\right) - v - v_0\ln\left(\left|\frac{h_1(v_0)}{w_1}\right|\right) + v_0 \right] \tag{D.18}$$

Para os casos 2 e 4 da figura D.35 a equação D.19 define o PDF quando $f_x(x)$ e $f_y(y)$ tem uma distribuição uniforme em intervalos definidos.

$$F(v) = \int_{v_0}^{v} \int_{w_0}^{h_1(v)} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\, \mathrm{d}w\mathrm{d}v = \alpha\beta \left[v\ln\left(\left|\frac{h_1(v)}{w_0}\right|\right) - v - v_0\ln\left(\left|\frac{h_1(v_0)}{w_0}\right|\right) + v_0 \right] \tag{D.19}$$

As equações D.18 e D.19 são unidas através da simplificação $w_0 = w_1 = w$, gerando a equação D.20.

```
CompVln(v,m,w)
1:
2: if 0 = v then
3:    return 0
4: else
5:    temp = m * v/w
6:    temp = abs(temp)
7:    return v*ln(temp)
8: end if
```

Figura D.36: Computa o valor de $\lim_{v \to v_0} v\ln(\left|\frac{vm}{w}\right|)$

```
CompF1(v,v0,m,w)
1:
2: temp =CompVln(v,m,w)
3: temp = temp+CompVln(v0,m,w)
4: temp = temp − v + v0
5: return abs(temp)
```

Figura D.37: Computa o valor do PD baseado da equação D.20

$$F(v) = \left|\alpha\beta\left[v\ln\left(\left|\frac{h_1(v)}{w}\right|\right) - v - v_0\ln\left(\left|\frac{h_1(v_0)}{w}\right|\right) + v_0\right]\right| \tag{D.20}$$

Para o caso 5 da figura D.35 a equação D.21 define seu PDF quando $f_x(x)$ e $f_y(y)$ tem uma distribuição uniforme em intervalos definidos.

$$F(v) = \int_{v_1}^{v}\int_{w_0}^{w_1} \frac{1}{|w|} f_x(w) f_y(\frac{v}{w})\,\mathrm{d}w\mathrm{d}v = (v - v_1)\ln\left(\left|\frac{w_1}{w_0}\right|\right) \tag{D.21}$$

D.3 Algoritmos propostos para o cômputo da PD da multiplicação

```
CompF2(v,v0,w0,w1)
1:
2: temp =CompVln(v,w1,w0)
3: temp = temp−CompVln(v0,w1,w0)
4: return temp
```

Figura D.38: Computa o valor do PD baseado da equação D.21

```
CompPDCase1(v',v,w,α,β)
1:
2: v_0 = v(0) v_1 = v(1) v_2 = v(2) v_3 = v(3)
3: w_0 = w(0) w_1 = w(1) w_2 = w(2) w_3 = w(3)
4: m_1 = (w_1 − w_0)/(v_3 − v_2)
5: m_2 = (w_1 − w_0)/(v_1 − v_0)
6: if v' <= v_0 then
7:    return 0
8: else if (v' >= v_3) then
9:    return αβ(w_1 − w_0)(w_3 − w_2)
10: else if v' <= v_1 then
11:    if m_2 < 0 then
12:       return αβCompF1(v',v_0,m_2,w_1)
13:    else
14:       return αβCompF1(v',v_0,m_2,w_0)
15:    end if
16: end if
17: if m_2 < 0 then
18:    temp =CompF1(v_1,v_0,m_2,w_1)
19: else
20:    temp =CompF1(v_1,v_0,m_2,w_0)
21: end if
22: if (v_1 < v') ∧ (v' <= v_2) then
23:    return αβ (CompF2(v',v_1,w_0,w_1) + temp)
24: end if
25: temp =CompF2(v_2,v_1,w_0,w_1)+temp
26: if m_1 < 0 then
27:    return αβ(CompF1(v',v_2,m_1,w_0)+temp)
28: else
29:    return αβ(CompF1(v',v_2,m_1,w_1)+temp)
30: end if
```

Figura D.39: Computa o valor do PD baseado da equação D.16

```
CompPDCase23(v',v,w,α,β)
1:
2: v_0 = v(0) v_1 = v(1) v_2 = v(2) v_3 = v(3)
3: w_0 = w(0) w_1 = w(1) w_2 = w(2) w_3 = w(3)
4: m = (w_1 − w_0)/(v_3 − v_0)
5: if v' <= v_0 then
6:    return 0
7: else if (v' >= v_3) then
8:    return αβ(w_1 − w_0)(w_3 − w_2)
9: else if v' <= v_1 then
10:   if m < 0 then
11:      return αβCompF1(v',v_0,m,w_1)
12:   else
13:      return αβCompF1(v',v_0,m,w_0)
14:   end if
15: end if
16: if m < 0 then
17:   temp =CompF1(v_1,v_0,m,w_1)
18: else
19:   temp =CompF1(v_1,v_0,m,w_0)
20: end if
21: if m < 0 then
22:   return αβ(CompF1(v',v_2,m,w_0)+temp)
23: else
24:   return αβ(CompF1(v',v_2,m,w_1)+temp)
25: end if
```

Figura D.40: Computa o valor do PD baseado da equação D.17

```
CompPDCase4(v',v,w,α,β)
1:
2: v_0 = v(0) v_1 = v(1) v_2 = v(2) v_3 = v(3)
3: w_0 = w(0) w_1 = w(1) w_2 = w(2) w_3 = w(3)
4: m_1 = (w_1 − w_0)/(v_2 − v_0)
5: m_2 = (w_1 − w_0)/(v_3 − v_1)
6: if v' <= v_0 then
7:    return 0
8: else if (v' >= v_3) then
9:    return αβ(w_1 − w_0)(w_3 − w_2)
10: else if v' <= v_1 then
11:    return αβCompF1(v',v_0,m_1,w_1)
12: end if
13: if (v_1 < v') ∧ (v' ≤ 0) then
14:    return αβ(CompF1(v',v_0,m_1,w_1)+CompF1(v',v_1,m_2,w_0))
15: end if
16: temp =CompVln(v_0,m_1,w_1)−CompVln(v_1,m_2,w_0)
17: temp = temp + v_1 − v_0
18: if (0 < v') ∧ (v' ≤ v_2) then
19:    return αβ(CompVln(v',m_1,w_0)−CompVln(v',m_2,w_1)+temp)
20: end if
21: temp =CompVln(v_2,m_1,w_0)−v_2 + temp
22: return αβ(−CompVln(v',m_2,w_1)+v' + temp)
```

Figura D.41: Computa o valor do PD baseado da equação D.17

```
CompPDCase5(v',v,w,α,β)
1:
2: v_0 = v(0) v_1 = v(1) v_2 = v(2) v_3 = v(3)
3: w_0 = w(0) w_1 = w(1) w_2 = w(2) w_3 = w(3)
4: m_1 = (w_1 − w_0)/(v_2 − v_0)
5: m_2 = (w_1 − w_0)/(v_3 − v_1)
6: if v' <= v_0 then
7:    return 0
8: else if (v' >= v_3) then
9:    return αβ(w_1 − w_0)(w_3 − w_2)
10: else if v' <= v_1 then
11:    return αβCompF1(v',v_0,m_2,w_0)
12: end if
13: if (v_1 < v') ∧ (v' ≤ 0) then
14:    return αβ(CompF1(v',v_0,m_2,w_0)+CompF1(v',v_1,m_1,w_1))
15: end if
16: temp =CompVln(v_1,m_1,w_1)−CompVln(v_0,m_2,w_0)
17: temp = temp + v_0 − v_1
18: if (0 < v') ∧ (v' ≤ v_2) then
19:    return αβ(CompVln(v',m_1,w_0)−CompVln(v',m_2,w_1)+temp)
20: end if
21: temp = v_2−CompVln(v_2,m_1,w_0)+temp
22: return αβ(CompVln(v',m_2,w_1)−v' + temp)
```

Figura D.42: Computa o valor do PD baseado da equação D.17

```
PDofMultVarVar(v',IntervalA,IntervalB)
a = IntervalA.LowLimit b = IntervalA.HighLimit α = IntervalA.Density
c = IntervalB.LowLimit d = IntervalB.HighLimit β = IntervalB.Density
w(0) = c w(1) = d w(2) = a w(3) = b
v(0) = ac v(1) = ad v(2) = bc v(3) = bd
SortL2H(v)
if (a = 0) ∨ (b = 0) then
  if (c < 0) ∧ (0 < d) then
    return CompPDCase23(v',v,w,α,β)
  else
    return CompPDCase1(v',v,w,α,β)
  end if
end if
if (a < 0) ∧ (0 < b) then
  if (c = 0) ∨ (d = 0) then
    w(0) = a w(1) = b w(2) = c w(3) = d
    return CompPDCase23(v',v,w,α,β)
  else if (c < 0) ∧ (0 < d) then
    if bc < ad then
      w(0) = a w(1) = b w(2) = c w(3) = d
    end if
    if bd < ac then
      return CompPDCase5(v',v,w,α,β)
    else
      return CompPDCase4(v',v,w,α,β)
    end if
  end if
end if
if 0 < a then
  if (c = 0) ∨ (d = 0) then
    w(0) = a w(1) = b w(2) = c w(3) = d
  else if (0 < c) ∧ (¬((d/c) < (b/a))) then
    w(0) = a w(1) = b w(2) = c w(3) = d
  else if (d < 0) ∧ (ac < bd) then
    w(0) = a w(1) = b w(2) = c w(3) = d
  else if (c < 0) ∧ (0 < d) then
    w(0) = a w(1) = b w(2) = c w(3) = d
  end if
else if b < 0 then
  if (c = 0) ∨ (d = 0) then
    w(0) = a w(1) = b w(2) = c w(3) = d
  else if (0 < c) ∧ (ac < bd) then
    w(0) = a w(1) = b w(2) = c w(3) = d
  else if (d < 0) ∧ (¬(bc < ad)) then
    w(0) = a w(1) = b w(2) = c w(3) = d
  else if (c < 0) ∧ (0 < d) then
    w(0) = a w(1) = b w(2) = c w(3) = d
  end if
end if
return CompPDCase1(v',v,w,α,β)
```

Figura D.43: Computa o PD baseado da equação D.17

Referências

1 KEUTZER, K.; NEWTON, A. R.; RABAEY, J. M.; SANGIOVANNI-VINCENTELLI, A. L. System-level design: orthogonalization of concerns andplatform-based design. IEEE Trans. on CAD of Integrated Circuits and Systems, v. 19, n. 12, p. 1523–1543, 2000.

2 CHANG, H.; COOKE, L.; HUNT, M.; MARTIN, G.; MCNELLY, A.; TODD, L. Surviving the soc revolution. Boston: Springer, 1999. 256 p.

3 MACHINES, I. B. Coreconnect bus architecture. In: _____. [s.n.], 2010. Disponível em: <https://www-01.ibm.com/chips/techlib.nsf/products/CoreConnect>.

4 ARM. Amba system ip and design tools. In: _____. [s.n.], 2010. Disponível em: <www.arm.com/products/system-ip/amba/>.

5 OCP-IP. Open core protocol specification. In: _____. [s.n.], 2010. Disponível em: <http://www.ocpip.org/home.php>.

6 BERGERON, J. Writing Testbenches - Functional Verification of HDL Models. Boston: Kluwer Academic Publishers, 2003.

7 WILE, B.; GROSS, J.; ROESNER, W. Comprehensive Functional Verification. San Francisco: Morgan Kaufmann, 2005. 702 p.

8 PIZIALI, A. Functional Verification Coverage Mesasuremnt and Analysis. Boston: Kluwer Academic Publishers, 2004. 213 p.

9 VERMA, S.; HARRIS, I.; RAMINENI, K. Automatic generation of functional coverage models from behavioral verilog descriptions. In: Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07. [S.l.: s.n.], 2007. p. 1 –6.

10 FALLAH, F.; DEVADAS, S.; KUETZER, K. Occom: efficient computation of observability-based code coverage metrics for functional verification. In: Design Automation Conference, 1998. Proceedings. [S.l.: s.n.], 1998. p. 152 – 157.

11 I., C. C.; STRUM, M.; CHAU, W. J. Automatic generation of parameter-domain-based functional input coverage model. In: Test Workshop, 20010. LATW '10. 11th Latin American. [S.l.: s.n.], 2010. p. 1 – 6.

12 JERINIC, V.; MüLLER, D. Safe integration of parameterized ip. Integration, the VLSI Journal, v. 37, n. 4, p. 193 – 221, 2004. ISSN 0167-9260. IP and Design Reuse. Disponível em: <http://www.sciencedirect.com/science/article/pii/S016792600300107X>.

13 BUSHNELL, M.; AGRAWAL, V. Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits. New York: Springer, 2000. 712 p.

14 DRECHSLER, R. Advanced Formal Verification. Norwell, MA, USA: Kluwer Academic Publishers, 2004. ISBN 1402077211.

15 DEVADAS, S.; GHOSH, A.; KEUTZER, K. An observability-based code coverage metric for functional simulation. In: Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on. [S.l.: s.n.], 1996. p. 418 –425.

16 ABARBANEL, Y.; BEER, I.; GLUHOVSKY, L.; KEIDAR, S.; WOLFSTHAL, Y. Focs: Automatic generation of simulation checkers from formal specifications. In: CAV. [S.l.: s.n.], 2000. p. 538–542.

17 BEER, I.; BEN-DAVID, S.; LANDVER, A. On-the-fly model checking of rctl formulas. In: CAV. [S.l.: s.n.], 1998. p. 184–194.

18 CLARKE, E.; GRUMBERG, O.; PELED, D. A. Model Checking. Cambridge, Massachusetts, USA: The MIT Press, 1999.

19 TAYLOR, S.; QUINN, M.; BROWN, D.; DOHM, N.; HILDEBRANDT, S.; HUGGINS, J.; RAMEY, C. Functional verification of a multiple-issue, out-of-order, superscalar alpha processor-the dec alpha 21264 microprocessor. In: Proceedings of the 35th annual Design Automation Conference. New York, NY, USA: ACM, 1998. (DAC '98), p. 638–643. ISBN 0-89791-964-5. Disponível em: <http://doi.acm.org/10.1145/277044.277208>.

20 FOSTER, H.; LACEY, D.; KROLNIK, A. Assertion-Based Design. 2. ed. Norwell, MA, USA: Kluwer Academic Publishers, 2003. ISBN 1402074980.

21 FUJITA, M.; GHOSH, I.; PRASAD, M. Verification Techniques for System-Level Design. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007. ISBN 0123706165, 9780123706164.

22 KATROWITZ, M.; NOACK, L. M. I'm done simulating; now what? verification coverage analysis and correctness checking of the dec chip 21164 alpha microprocessor. In: Proceedings of the 33rd annual Design Automation Conference. New York, NY, USA: ACM, 1996. (DAC '96), p. 325–330. ISBN 0-89791-779-0. Disponível em: <http://doi.acm.org/10.1145/240518.240580>.

23 FINE, S.; ZIV, A. Coverage directed test generation for functional verification using bayesian networks. In: Proceedings of the 40th annual Design Automation Conference. New York, NY, USA: ACM, 2003. (DAC '03), p. 286–291. ISBN 1-58113-688-9. Disponível em: <http://doi.acm.org/10.1145/775832.775907>.

24 TOBAR, E. L. R.; R., A.; CHAU, S. M. . W. J. A genetic approach to automatic bias generation for biased random instruction generation. In: Proceedings on International conference Very Large Scale Integration VLSI-SoC 2009. [S.l.: s.n.], 2009.

25 JERINIC, V.; LANGER, J.; HEINKEL, U.; MIILLER, D. New methods and coverage metrics for functional verification. In: Design, Automation and Test in Europe, 2006. DATE '06. Proceedings. [S.l.: s.n.], 2006. v. 1, p. 1 –6.

26 JERENIC, V. Paragraph - Test Parameters for Intellectual Property. Tese (Doutorado) — Fakultät Elektrotechnik und Informationstechnik, Technischen Universität Chemnitz, 2005.

27 CASTRO, C. Análise da Influência do Uso de Domínios de Parâmetros sobre a Eficiência da Verificação Funcional Baseada em Estimulação Aleatória. Dissertação (Mestrado) — Electrical Deparment, Universidade de São Paulo, São Paulo, 2009.

28 MARQUEZ, C. I. C.; STRUM, M.; CHAU, W. J. A pd-based methodology to enhance efficiency in testbenches with random stimulation. In: SBCCI '09: Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design. New York, NY, USA: ACM, 2009. p. 1–6. ISBN 978-1-60558-705-9.

29 MISHRA, P.; DUTT, N. Functional coverage driven test generation for validation of pipelined processors. In: Design, Automation and Test in Europe, 2005. Proceedings. [S.l.: s.n.], 2005. p. 678 – 683 Vol. 2. ISSN 1530-1591.

30 HENNESSY, J. L.; PATTERSON, D. A. Computer Architecture; A Quantitative Approach. 1st. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992. ISBN 1558600698.

31 GAISLER, A. Leon2 processor. In: _____. [s.n.], 2010. Disponível em: <http://www.gaisler.com/cms/>.

32 ROTH, J. P. Diagnosis of automata failures: A calculus and a method. IBM Journal of Research and Development, v. 10, n. 4, p. 278 –291, july 1966. ISSN 0018-8646.

33 TASIRAN, S.; FALLAH, F.; CHINNERY, D.; WEBER, S.; KEUTZER, K. A functional validation technique: biased-random simulation guided by observability-based coverage. In: Computer Design, 2001. ICCD 2001. Proceedings. 2001 International Conference on. [S.l.: s.n.], 2001. p. 82 –88.

34 MARQUEZ, C. I. C.; STRUM, M.; CHAU, W. J. A pd-based methodology to enhance efficiency in testbenches with random stimulation. In: SBCCI. [S.l.: s.n.], 2009.

35 MONTGOMERY, D. C.; RUNGER, G. C. Applied Statistics and Probability for Engineers. [S.l.]: John Wiley and Sons, Inc., 2003.

36 PAPOULIS, A. Probability, Random Variables, and Stochastic Processes. 3rd. ed. [S.l.: s.n.], 1991.

37 JAMES, B. R. Probabilidade: Um Curso Em Nivel Intermediario. [S.l.]: IMPA, 1981.

38 INITIATIVE, O. S. Systemc reference manual. In: _____. [s.n.], 2010. Disponível em: <www.systemc.org/>.

39 GAJSKI, D. D.; RAMACHANDRAN, L. Introduction to high-level synthesis. IEEE Design & Test of Computers, v. 11, n. 4, p. 44–54, 1994.

40 VAHID, F.; GIVARGIS, T. Embedded system design - a unified hardware / software introduction. [S.l.]: Wiley-VCH, 2002. I-XXI, 1-324 p. ISBN 978-0-471-45303-1.

41 CASSANDRAS, C. G.; LAFORTUNE, S. Introduction to Discrete Event Systems. [S.l.]: Springer, 2008.