UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RUBINEI PESKE ANGELO

Implicações do Estilo de Descrição de Códigos VHDL na Testabilidade

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

Prof. Dr. Marcelo Soares Lubaszewski
Orientador

Prof. Dra. Érika Cota
Co-orientadora

Porto Alegre, julho de 2005.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Angelo, Rubinei Peske

Implicações do Estilo de Descrição de Códigos VHDL na Testabilidade / Rubinei Peske Angelo – Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

67 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2005. Orientador: Marcelo Soares Lubaszewski; Co-orientadora: Érika Cota.

1.Teste. 2.VHDL 3.Projeto Digital. 4.Síntese visando Teste. 5.Projeto Visando Teste. I. Lubaszewski, Marcelo. II. Cota, Érika. III. Título.

AGRADECIMENTOS

Gostaria de agradecer aos meus avós e meus pais, Rudinei Angelo
e Lieda Peske Angelo, por todo o amor, carinho, dedicação,
incentivo e investimento em mim
durante toda a minha vida.

Gostaria de agradecer a todos aqueles que de alguma forma contribuiram com
o meu crescimento pessoal e profissional.

Aos meus amigos que nestes últimos dois anos sempre me apoiaram e motivaram,
por estarem ao meu lado durante os momentos difíceis e durante as jantas,
churrascos, redenção, sinuca e jogos de vôlei e futebol.

Ao CNPq pelo fomento manifestado através de uma bolsa de estudos.

Por fim, agradeço também aos meus orientadores Marcelo
e Érika, obrigado por acreditarem no meu trabalho
e pela inestimável orientação.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo Genético
ASIC	Application Specific Integrated Circuit
ATE	Automatic Test Equipment
ATPG	Automatic Test-Pattern Generation
BEHATE	Behavioral Test Environment
BDD	Binary Decision Diagram
BIST	Built-In Self-Test
CA	Cellular Automata
CAD	Computer Aided Design
CFG	Control Flow Graph
CMOS	Complementary Metal-Oxide Semicondutor
CUT	Circuit Under Test
DFT	Design For Testability
FAN	Fanout-oriented
HDL	Hardware Description Language
IP	Intelectual Property
LFSR	Linear Feedback Shift Register
PC	Parte de Controle
PO	Parte Operativa
PODEM	Path-Oriented Decision Making
ROM	Read-Only Memory
RTL	Register-Transfer Level
STG	State Transition Graph
VHDL	VHSIC Hardware Description Language
VHSIC	Very High-Speed Integrated Circuit
VLSI	Very Large Scale Integration
SA	Stuck-At
SCOAP	Sandia Controllability / Observability Analysis Program
TOPS	Topological Search

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Devido ao aumento da complexidade dos circuitos integrados atuais, os projetos são desenvolvidos utilizando linguagens de descrição de hardware (por exemplo, VHDL) e os circuitos são gerados automaticamente a partir das descrições em alto nível de abstração. Embora o projeto do circuito seja facilitado pela utilização de ferramentas de auxílio ao projeto, o teste do circuito resultante torna-se mais complicado com o aumento da complexidade dos circuitos. Isto traz a necessidade de considerar o teste do circuito durante sua descrição e não somente após a síntese. O objetivo deste trabalho é definir uma relação entre o estilo da descrição VHDL e a testabilidade do circuito resultante, identificando formas de descrição que geram circuitos mais testáveis. Como estudo de caso, diferentes descrições VHDL de um mesmo algoritmo foram utilizadas. Os resultados mostram que a utilização de diferentes descrições VHDL tem grande impacto nas medidas de testabilidade do circuito final e que características de algumas descrições podem ser utilizadas para modificar outras descrições e com isso aumentar a testabilidade do circuito resultante.

Palavras-Chave: Teste, VHDL, Projeto Digital, Síntese visando Teste, Projeto visando Teste.

Implications of the High-Level Design Style in the Testability

Abstract

Due to the increasing complexity of current circuits, designs are developed using high-level hardware description languages (VHDL, for example) and circuits are automatically synthesized from this high level description. Although the design is facilitated by the use of automatic synthesis tools, the test of the resulting circuit becomes more complicated with the increasing of the system complexity. Thus, it becomes necessary to consider the test of the circuit during its initial description and not just after the synthesis. This work has the goal of investing the relationship between the description style of a VHDL code and the testability of the resulting circuit, identifying description patterns that build circuits with better testability features. As case study, different VHDL descriptions of the same algorithm were used. Experimental results show that different high level descriptions have a significant impact on the testability measures of the synthesized circuit. As a result, we identify VHDL constructs that may be used to form a VHDL coding style directed to icrease circuit testability.

Keywords: Test, VHDL, Digital Design, Synthesis for Test, Design for Testability.

1 INTRODUÇÃO

1.1 Fluxo de projeto VLSI

Processos tecnológicos que utilizam tecnologias CMOS (*Complementary metal-oxide semicondutor*) estão seguindo a lei de Moore, a qual observa que o número de transistores dobra a cada 18 meses (MOORE, 1965). A busca de melhorias nos processos de fabricação e de novas formas ou técnicas de construção de circuitos integrados está proporcionando aos circuitos integrados manterem a tendência de possuírem cada vez mais transistores. A seguir, será apresentado o fluxo de projeto e teste de circuitos integrados CMOS, a qual é a tecnologia dominante na implementação de circuitos VLSI (*Very Large Scale Integration*). De acordo com Nicolici (2000) e Norwood (1997), o fluxo de um projeto VLSI é dividido em três passos: Especificação, Implementação e Fabricação, conforme mostrado na Figura 1.1. O teste, como se observa na figura, é considerado em todas etapas do fluxo de projeto.

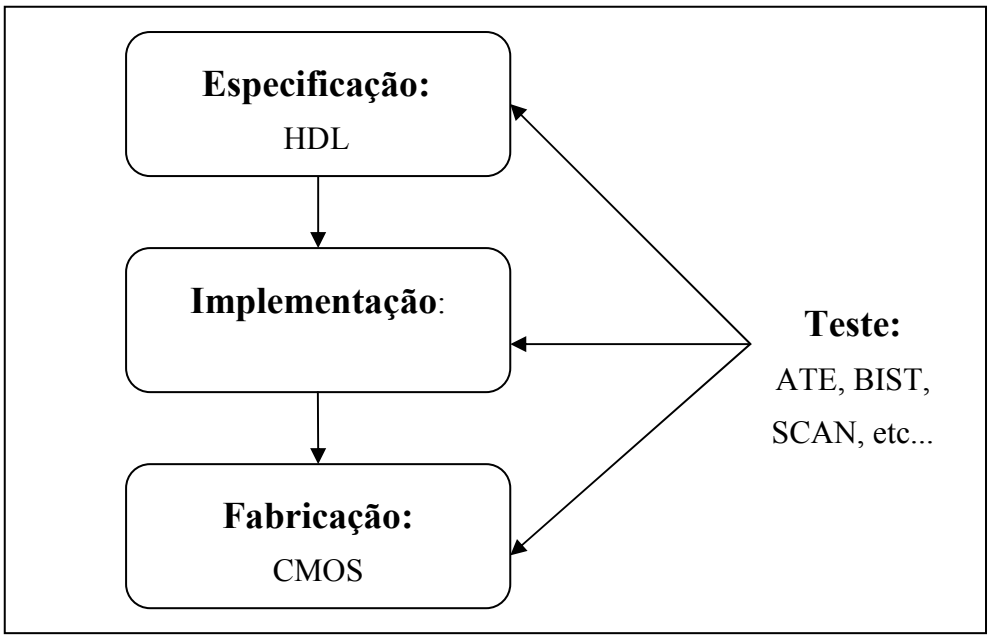

Figura 1.1: Fluxo de um projeto VLSI.

1.1.1 Especificação

A especificação é a etapa onde é descrita a funcionalidade do circuito integrado. A especificação é realizada em alguma linguagem de descrição de Hardware (HDL – *Hardware Description Language*) como VHDL (NAVABI, 1997), Verilog (THOMAS, 1998) ou SystemC (GRÖTKER, 2002), podendo ser descrita em dois domínios de projeto, o comportamental e o estrutural, e em vários níveis de abstração. Por exemplo, o nível de abstração **lógico** é a representação em forma de expressões de álgebra booleana no domínio comportamental, ou em interconexões de portas lógicas no domínio estrutural. Aumentando o nível de abstração chega-se ao nível de **transferência de registradores** (RTL – *Register-Transfer Level*), onde o circuito integrado é visto na forma de uma seqüência lógica, consistindo de registradores e unidades funcionais. O mais alto nível de abstração é o **algoritmo**, onde a especificação consiste em tarefas que descrevem abstratamente a funcionalidade do sistema (NICOLICI, 2000).

1.1.2 Implementação

A implementação é a etapa de geração de uma rede de componentes (*netlist*) que realizam as funções descritas na especificação. De acordo com a metodologia utilizada, a implementação pode ser *full-custom* ou *semi-custom* (REIS, 2002). A metodologia de projeto *full-custom* requer um grande esforço da equipe de projetistas, pois todas as características do circuito são detalhadamente otimizadas. Com a metodologia *semi-custom*, grande parte da implementação é realizada com a ajuda de ferramentas de auxílio ao projetista (CAD - *Computer Aided Design*). As ferramentas de CAD capturam a especificação inicial das HDLs e a transformam em uma implementação estrutural, para poder otimizar o *netlist* resultante, mapear o circuito em portas lógicas e rotear as conexões entre as portas lógicas (REIS, 2002).

1.1.3 Fabricação

O circuito gerado na implementação pode ser independente de alguma biblioteca tecnológica. Nesta fase, o *netlist* é convertido para uma descrição que leva em conta requisitos tecnológicos como área, energia ou velocidade. Após este mapeamento tecnológico, o circuito é convertido em uma representação geométrica chamada leiaute. O leiaute é criado com o auxílio de uma ferramenta de síntese física, convertendo cada componente do *netlist* com restrições tecnológicas (células, portas, transistores) em uma representação geométrica que atenda a função lógica do componente correspondente. Após a criação do leiaute, o projeto é enviado para uma fábrica onde os chips são construídos fisicamente (SHERWANI, 1998).

A fabricação é um dos passos finais do fluxo de projeto de um circuito integrado. Esta etapa resulta nos transistores conectados conforme a implementação, utilizando uma tecnologia de fabricação. Tecnologia de fabricação se refere ao processo utilizado para produzir o circuito integrado, o qual pode possuir características como: tipo de semicondutor (por exemplo, silício); o tipo de transistor (por exemplo, CMOS); e detalhes da tecnologia do transistor (por exemplo, 0.35 micron).

1.1.4 Teste

Em cada um dos passos do fluxo de projeto de um circuito podem ser realizadas etapas de teste. O teste assegura que a função desempenhada pelo circuito é correspondente a aquela especificada pelo projetista. Durante a especificação podem ser utilizadas técnicas de teste, tais como a análise de testabilidade, para encontrar áreas do circuito com problemas de testabilidade. Técnicas de teste como inserção de BIST (*Built-In Self_Test*) e cadeias de varredura podem ser utilizadas na etapa de implementação para melhorar as características de testabilidade do circuito. Na fabricação podem ser realizados testes para garantir que o circuito está sem defeitos que podem ocorrer durante o processo de fabricação. No Capítulo 2 serão explicadas mais detalhadamente as técnicas exemplificadas anteriormente. Esta dissertação tem por objetivo desenvolver metodologias para a etapa de especificação do projeto, para que, em um alto nível de abstração, modificações possam ser realizadas para refletir em uma melhor testabilidade do circuito após a fabricação.

1.2 Motivação

Nos dias atuais, com o grande avanço das tecnologias e processos de fabricação, os circuitos integrados podem conter milhões de transistores e são construídos com a ajuda de ferramentas de CAD, possibilitando gerar hardware a partir de um alto nível de abstração. Entretanto, os projetos destes circuitos complexos são mais propensos a falhas durante o fluxo do projeto. A necessidade de um menor tempo para lançamento do produto no mercado, aliado à necessidade de aumentar a confiabilidade do produto está levando o teste do circuito para as etapas iniciais do fluxo de projeto.

O teste dos milhões de transistores configura-se como o segundo maior custo do chip, só perdendo para o custo de fabricação, pois é necessário garantir que o circuito integrado não possui erros e que o mesmo está em condições de uso pelo consumidor final (CHEN, 2003a). O custo de fabricação é o maior devido a gastos com máscaras e aos processos físicos e químicos utilizados na fabricação. O custo do projeto pode ser reduzido se a etapa de teste for considerada nos estágios iniciais do projeto e, conseqüentemente, em níveis de abstração mais elevados, como em linguagens de descrição de hardware (CHEN, 2003a) (DEY, 1998).

O projeto de um circuito integrado pode ser desenvolvido por profissionais de várias áreas distintas, tais como: engenharia elétrica, engenharia eletrônica, engenharia de computação ou outras áreas afins. Estes projetistas geralmente podem utilizar técnicas diferentes de projeto e construir códigos VHDL de uma forma diferente entre elas, mas mesmo assim o circuito resultante precisa atender as especificações de: um correto funcionamento da função a ser realizada, restrição de área a ser ocupada, freqüência de operação. Os códigos são construídos para resolver um problema, mas normalmente não é garantido que a solução é a melhor possível em relação às restrições de área, freqüência, potência consumida ou testabilidade. A utilização de linguagens de descrição de hardware em alto nível de abstração permite a estes profissionais de áreas distintas realizarem seus projetos. Desenvolver um método de descrição de códigos que possa ser utilizada por todos projetistas, que visa conceber um circuito com melhores características de testabilidade, é o foco desta dissertação.

1.3 Objetivos

Como apresentado anteriormente, o teste está sendo conduzido para as etapas iniciais do fluxo de projeto e, com isso, novas técnicas precisam ser criadas para melhorar a testabilidade do circuito integrado, iniciando sua utilização em níveis de abstração mais altos.

Este trabalho tem como objetivo verificar a possibilidade de aumentar a testabilidade do circuito integrado apenas modificando o estilo de descrição dos códigos VHDL. Novas formas de aumentar a testabilidade de circuitos que iniciam seu fluxo de projeto com códigos VHDL são estudadas. Ao final da dissertação, são comparados os resultados obtidos com as utilizações de técnicas mais clássicas com estas novas formas de melhorar a testabilidade do circuito.

1.4 Organização do trabalho

Após esta breve introdução, no Capítulo 2 são mostrados os conceitos básicos de teste de circuitos e o que está sendo utilizado e estudado atualmente na comunidade científica e na indústria. No Capítulo 3 é apresentado um estudo de caso, onde diferentes códigos VHDL e seus impactos na testabilidade do circuito gerado são estudados. O Capítulo 4 apresenta os resultados de síntese e testabilidade gerados com as modificações propostas para as descrições VHDL. Finalmente, o Capítulo 5 apresenta as conclusões e define alguns trabalhos futuros.

2 ESTADO DA ARTE

Após a fabricação, os circuitos precisam ser testados a fim de garantir que os circuitos integrados estejam funcionando corretamente e que possam ser vendidos para o consumidor. Esse capítulo apresenta os conceitos básicos de teste de circuitos e o que está sendo utilizado e estudado atualmente na comunidade científica e na indústria.

2.1 Teste de circuitos integrados

2.1.1 Conceito de Defeito, Falha e Erro

Um defeito é uma imperfeição física ocorrida em um circuito. A Figura 2.1(a) ilustra um defeito de fabricação. Por exemplo, uma área de metal não desejada pode provocar um curto-circuito entre duas áreas de metal, modificando o comportamento do circuito (JANSCH-PORTO, 1989) (BUSHNELL, 2000).

Uma falha é uma abstração de um defeito. O defeito ilustrado na Figura 2.1(a) pode ser modelado como um curto-circuito no nível de transistor (Figura 2.1(b)), ou SA (*stuck-at*) 1 no nível de portas lógicas (Figura 2.1(c)) (BUSHNELL, 2000).

Um erro é o valor errado gerado por um sistema defeituoso. Por exemplo, na Figura 2.1(c) existe um erro quando os valores das portas A e B forem iguais ao nível lógico 1, pois o valor da saída será 1 quando deveria ser 0. Por outro lado, quando A ou B tiverem o valor 0, não há erro uma vez que o valor esperado será igual ao valor gerado (BUSHNELL, 2000).

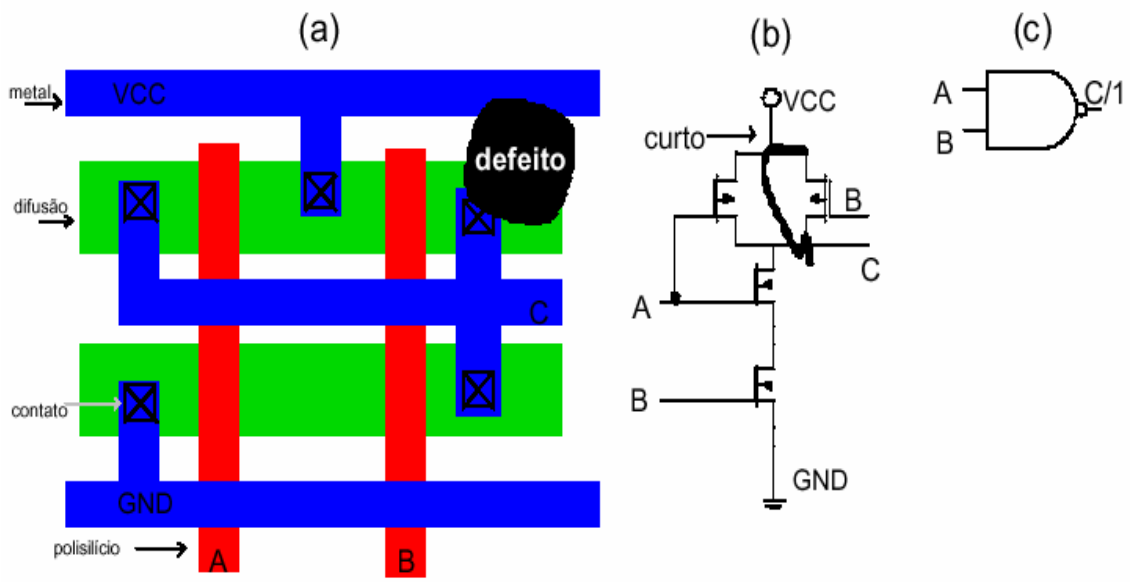

Figura 2.1: Definição de defeito e falha. (a) Nível de leiaute. (b) Nível de transistor. (c) Nível de porta lógica (AMORY 2002)

2.1.2 Domínios Físico e Lógico

A seguir serão apresentados os principais defeitos físicos que podem ocorrer em um processo de fabricação e os principais modelos de falhas que abstraem estes defeitos.

Domínio Físico – defeitos.

Os processos de fabricação para circuitos VLSI estão longe de serem perfeitos. Devido a diversas causas, defeitos são introduzidos nos circuitos. Exemplos de defeitos são apresentados a seguir: (ABRAMOVICI, 1990) (BUSHNELL, 2000)

- No dispositivo – curto-circuito entre *gate* e difusão, curto-circuito entre metal e polissilício, impurezas, avaria do dielétrico, defeitos em soldas.
- Na placa – falta de componentes, componente defeituoso, trilha quebrada, trilhas em curto e circuito aberto.
- Projeto Incorreto – defeitos funcionais.
- Defeitos ambientais – relativas à temperatura, alta umidade, vibrações, stress elétrico e radiações.

Domínio Lógico – Falhas (falhas estruturais)

A falha é o modelo que representa o efeito do defeito no sistema. Alguns exemplos são: (JERVAN, 2002)

- Falhas do tipo colagem (*Stuck-at*): Simples ou múltiplas.
- Falhas de Ponte (*Bridging faults*): AND, OR, sem realimentação ou com realimentação.
- Falhas de atrasos (*delay*): de porta lógica ou interconexão.

2.1.3 Tipos de teste

Entre os diversos tipos de teste, dois são de interesse desta dissertação: o teste funcional e o teste estrutural (NADEAU-DOSTIE, 1999) (BUSHNELL, 2000). Ambos verificam o comportamento do circuito aplicando valores na entrada do mesmo e comparando a saída com o valor esperado. Estes valores aplicados se chamam vetores de teste. Define-se como cobertura de falhas o percentual de falhas que são detectadas pelos vetores de teste durante a aplicação do teste, em relação ao total de falhas do circuito.

O objetivo do teste estrutural é descobrir alguma falha causada devido a defeitos de fabricação. É chamado de estrutural por depender da estrutura específica do circuito (portas lógicas, transistores, interconexões, *netlist*) e verificar estes elementos específicos dentro do circuito.

Para realizar o teste funcional, é necessário conhecer o comportamento da função desempenhada pelo circuito. Isso torna difícil a sua aplicação pelos engenheiros de teste que não participaram do desenvolvimento do restante do projeto. Além disso, a cobertura de falhas não é muito elevada quando aplicados os vetores de teste no circuito integrado. O teste estrutural, ao contrário, pode alcançar altas coberturas de falhas, pois os vetores são criados para detectar uma falha específica dentro da estrutura do circuito. (NADEAU-DOSTIE, 1999) (ABRAMOVICI, 1990).

2.1.4 Modelos de Falhas

De forma geral, os modelos de falhas estão associados aos diferentes níveis de abstração. Um modelo de falhas é uma representação abstrata dos defeitos físicos, de uma forma adequada para utilização nas atividades de simulação e geração de testes. A utilização de um modelo de falhas deve produzir, aproximadamente, o mesmo comportamento errôneo dos defeitos físicos reais modelados.

Com o propósito de representação de um circuito, visando a modelagem de falhas para utilização na geração de testes, os seguintes níveis de abstração são considerados (BUSHNELL, 2000):

- Nível Comportamental - A representação no nível comportamental é utilizada quando existe pouca ou nenhuma informação a respeito da organização do circuito a ser testado. Neste nível, não é óbvia a relação das falhas com os defeitos de fabricação.
- Nível Lógico - O nível lógico consiste de uma *netlist* de portas lógicas e as falhas de *stuck-at* são as mais comumente utilizadas como modelo de falhas em teste digital. Outros modelos de falhas como *bridging* e *delay* também são utilizados.
- Nível de transistor – Esse nível é utilizado quando existe conhecimento da formação dos componentes de um circuito digital, no nível de transistores ou outros níveis inferiores. As falhas nesse nível são caracterizadas por alterações de valores elétricos, tais como tensão, corrente ou resistência. Este nível pode incluir modelos de falhas do tipo *stuck-open* que são conhecidas como falhas dependentes da tecnologia. São principalmente modeladas em teste de circuitos analógicos.

Esta dissertação abordará somente o teste estrutural e o modelo de falhas utilizado no restante desse texto é o modelo de falhas do tipo *stuck-at* 0 ou *stuck-at* 1. A seguir serão mostrados os dois tipos principais de teste estruturais mais utilizados, o teste interno e o teste externo.

2.2 Teste Externo usando ATE

Atualmente, com o avanço da complexidade dos projetos de circuitos integrados, o teste destes circuitos, após serem fabricados, é automatizado. A Figura 2.2 mostra os princípios do teste externo usando um equipamento automático de teste (ATE - *Automatic Test Equipment*). Dois componentes básicos podem ser listados: o circuito sob teste (CUT – *Circuit Under Test*) e o equipamento automático de teste (ATE).

O CUT é o circuito integrado ou dispositivo que se deseja testar para encontrar defeitos de fabricação.

O ATE inclui algumas partes que executam o teste, tais como (BUSHNELL, 2000):

- Processador que controla o fluxo do teste e comunica para outros módulos do ATE se o CUT tem ou não defeitos;
- Módulos de tempo (*timing*) que definem os diferentes relógios necessários para cada pino do CUT;
- Um módulo que transforma as informações dos vetores de teste, como tempo e valor do sinal (nível lógico baixo ou alto), e controla a aplicação destes valores nos pinos do CUT;
- Módulo de alimentação, que provê a alimentação adequada ao CUT e é responsável pela precisão dos valores medidos de corrente e tensão.

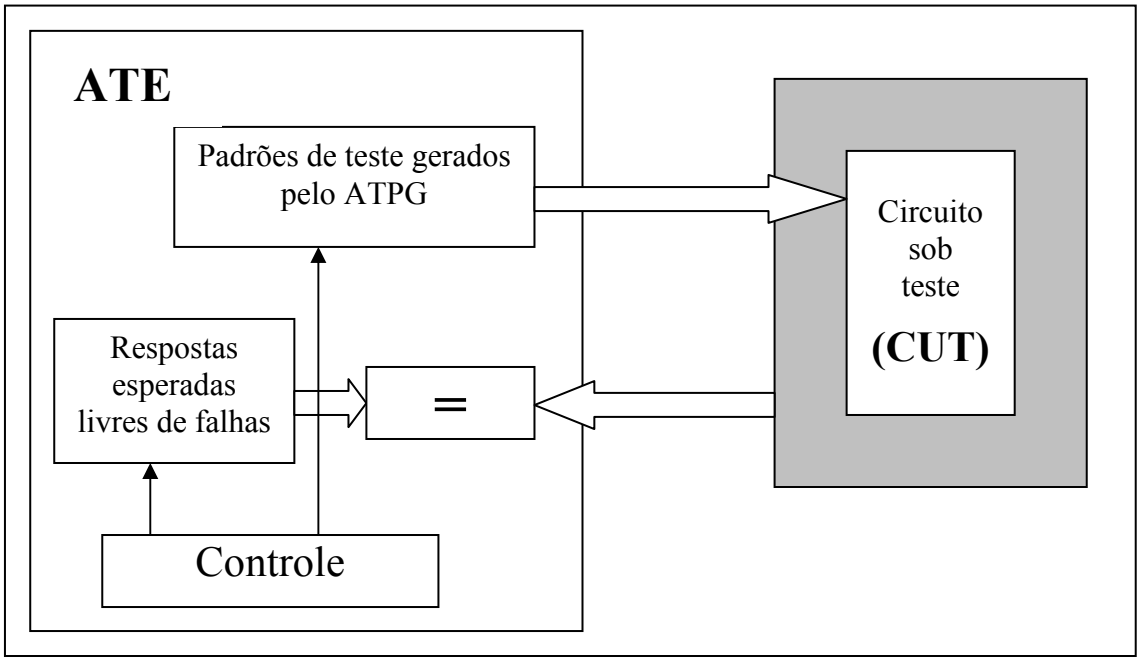

Figura 2.2: Princípio básico de teste externo usando ATE.

O ATE contém os vetores de teste e as respostas esperadas, que são comparadas com as respostas lidas pelo ATE durante o teste. Caso sejam diferentes, o CUT possui um defeito. Equipamentos de ATE podem medir respostas de tensão com milivolts de precisão e tempos com precisão de picosegundos (NEEDHAM, 1999).

Para testar um circuito usando um ATE necessitam-se de vetores de teste (grupos de 0s e 1s) que são aplicados nas entradas de um circuito para determinar se o circuito integrado está funcionando. Para descobrir quais vetores de teste utilizar, um gerador de padrões de teste é utilizado.

2.2.1 Geração Automática de Padrões de Teste - ATPG

Geração Automática de Padrões de Teste (ATPG - *Automatic test-pattern generation*) é o processo de gerar vetores de teste para um circuito. Através da simulação de falhas, pode-se determinar a cobertura de falhas alcançada pelos vetores gerados. Atualmente, os dois métodos para gerar padrões de teste mais utilizados são o aleatório e o determinístico (PATNAIK, 2002) (CUMANI, 2003).

Gerar padrões de teste de forma randômica é o processo de gerar vetores independente da estrutura do circuito. Entretanto, para alcançar uma alta cobertura de falhas necessita-se um de elevado número de vetores. Este método não leva em consideração a função ou estrutura interna do circuito a ser testado (CUMANI, 2003) (PATNAIK, 2002).

Por outro lado, uma ferramenta de ATPG que gera vetores de teste de forma determinística produz vetores de teste de acordo com o modelo do circuito. A geração determinística pode ser orientada a uma falha específica ou independente de falhas. A geração independente é realizada sem ter uma falha específica a ser ativada por um vetor. Algumas abordagens independentes usam o conceito de diferenças booleanas para a geração dos vetores de teste e são caracterizadas como algébricas, pois manipulam equações do circuito para gerar os vetores. Entretanto, para circuitos VLSI, esta geração algébrica é muito custosa em termos de tempo e as abordagens orientadas a falhas estão sendo utilizadas.

A geração determinística orientada a falhas é utilizada para gerar vetores de teste com o objetivo de detectar uma determinada falha dentro do circuito. Alguns dos mais conhecidos algoritmos de geração utilizados e desenvolvidos são o Algoritmo D (BOURICIUS, 1967), PODEM (*Path-Oriented Decision Making*) (GOEL, 1981), FAN (*Fanout-oriented*) (FUJIWARA, 1983) e TOPS (*Topological Search*) (KIRKLANDT, 1987). A geração é feita com o circuito modelado em nível lógico (*netlist* com portas lógicas) e estes algoritmos, a cada iteração, geram um vetor de teste para cada falha. Os algoritmos utilizam vários mecanismos para ativar a falha no ponto desejado e implicar o efeito da falha até uma saída. Além disso, é realizada a justificação da falha até uma entrada do circuito, assinalando valores lógicos baseados em grupos de condições necessárias para a propagação da falha (PATNAIK, 2002).

A Figura 2.3 mostra um fluxo de como é realizada a geração determinística orientada a falhas. Mais especificamente, existe uma lista contendo todas falhas possíveis no circuito. O procedimento seleciona uma destas falhas da lista e define um vetor para detectá-la. Alguns vetores de teste podem ser capazes de detectar mais de uma falha e isso pode ser observado na simulação de falhas (BUSHNELL, 2000).

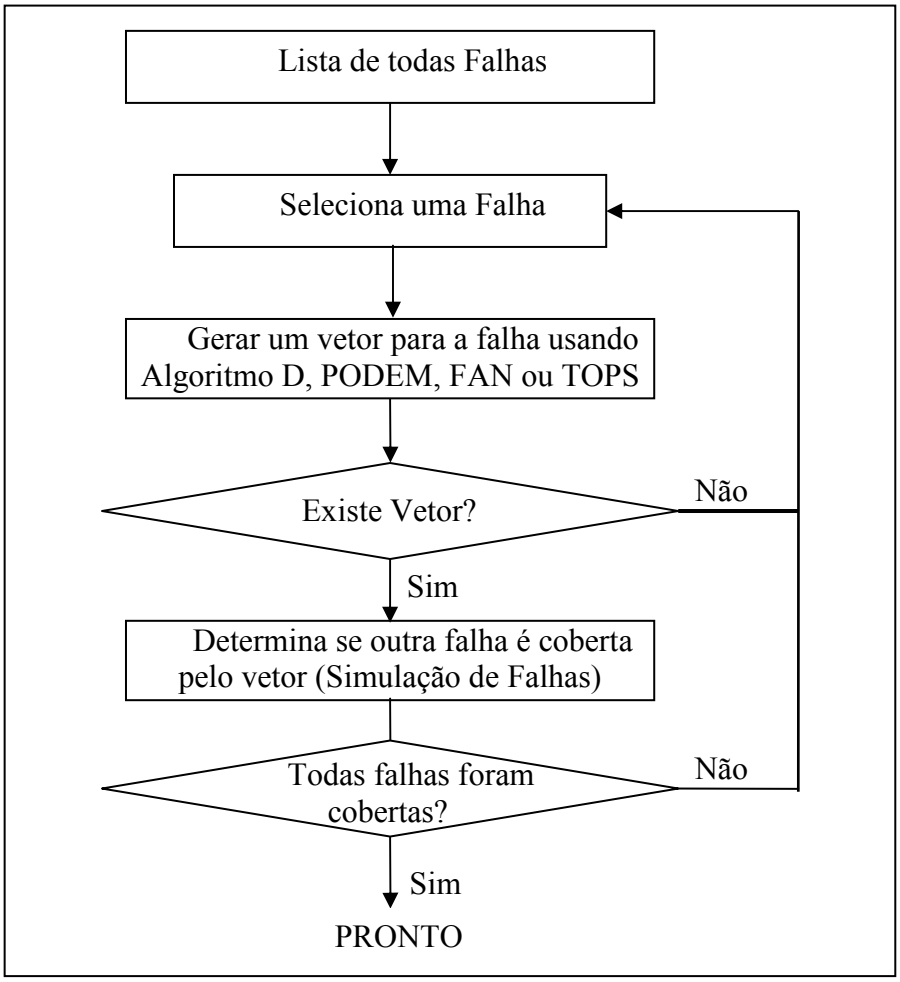

Figura 2.3: Fluxo da geração de vetores de teste.

Os ATPGs visam normalmente circuitos realizados em nível lógico (geralmente portas lógicas) o que pode acarretar, para circuitos seqüencias de tamanho e complexidade elevada, a necessidade de grande quantia de tempo para a sua execução (CUMANI, 2003). Além do tempo enorme gasto pelo ATPG, se o circuito está descrito em VHDL, ainda existe a necessidade de síntese do circuito para gerar a *netlist* com as portas lógicas. Por outro lado, atualmente existem trabalhos que estão buscando a execução de ATPGs a nível comportamental para reduzir o tempo gasto pelos ATPGs tradicionais. Alguns trabalhos que focam em ATPGs comportamentais são mostrados seguir.

2.2.2 ATPGs Comportamentais

A geração de padrões de teste em um nível de abstração mais alto possui a vantagem de utilizar apenas uma fração do tempo em relação à geração de padrões de teste em nível lógico. Outra vantagem é a independência dos padrões de teste gerados em relação a alguma ferramenta de síntese utilizada para sintetizar o circuito. A cobertura de falhas em alto nível de abstração pode ser utilizada para uma rápida exploração do espaço de projeto em relação a testabilidade, e isso pode resultar em um menor tempo de projeto.

Ferrandi et. al. (FERRANDI, 1998) (FERRANDI, 2002) propõem um gerador de padrões de teste comportamental independente da ferramenta de síntese. Como entrada

são utilizados códigos VHDL descritos na forma comportamental. A abordagem proposta, chamada de *bit-coverage*, é baseada na injeção de erros nas descrições VHDL e na comparação do comportamento da descrição livre de falhas com a descrição que possui erros. A ferramenta BEHATE (*BEHAvioral Test Environment*) gera padrões de teste considerando elementos de controle e de dados da especificação. Os erros são definidos nos diferentes elementos envolvidos na descrição VHDL, isto é, variáveis, portas, sinais, constantes, instruções de atribuição, instruções de desvio e instruções de laços de repetição. Os erros são do tipo *stuck-at* 0 ou 1 para as variáveis, sinais e portas e são do tipo *stuck-at true* ou *false* para os laços ou condições. A partir do código VHDL, a ferramenta cria um grafo de fluxo de controle (CFG – *Control Flow Graph*) que será utilizado para a tradução para uma árvore de decisão binária (BDD – *Binary Decision Diagram*). A ferramenta cria um BDD (AKERS, 1978) da descrição original e, para cada elemento da descrição, injeta um erro e cria outro BDD. Em seguida, é comparado o BDD original com o que possui falha e, se existe algum bit diferente nas saídas, conclui-se que a falha é testável. Caso contrário, a falha não é detectável.

A partir dos vetores de teste comportamentais podem ser gerados vetores de teste no nível lógico, utilizando-se informações de alocação e escalonamento da síntese. A Figura 2.4 mostra os blocos utilizados para realizar essa conversão de vetores de teste comportamentais em lógicos. A conversão é realizada pela simulação simultânea da descrição VHDL e o circuito no nível lógico. Para a simulação existe um gerador de relógios, um módulo comparador e um controle para a simulação, além das duas arquiteturas que representam o circuito em níveis de abstração diferentes. Inicialmente, é lido um vetor comportamental e as duas arquiteturas recebem a mesma entrada, correspondente ao vetor de teste, mas não o mesmo relógio. Enquanto a arquitetura em nível lógico recebe o sinal de relógio CK produzido pelo gerador de relógios, a arquitetura comportamental recebe um sinal de relógio diferente chamado OK, que é múltiplo de CK. O sinal OK só é ativado novamente quando a simulação produzir o mesmo valor de saída do circuito em nível lógico que o valor de saída do circuito comportamental. Após os resultados das saídas serem iguais, um novo OK é gerado e um novo vetor de teste é inserido na entrada dos circuitos. Com a realização deste procedimento identifica-se o número de vezes que cada vetor de teste comportamental necessita ser inserido na descrição lógica para realizar o teste. A abordagem mostrou-se interessante visto que é independente de uma ferramenta de síntese e é capaz de gerar padrões de teste a nível comportamental e lógico. Os resultados mostraram que a cobertura de falhas é superior e o tempo gasto na geração dos vetores de teste é inferior à alcançada por um ATPG comercial. Em contrapartida, o número de vetores gerados para o teste pelo ATPG comportamental foi muito maior que o gerado pelo ATPG comercial.

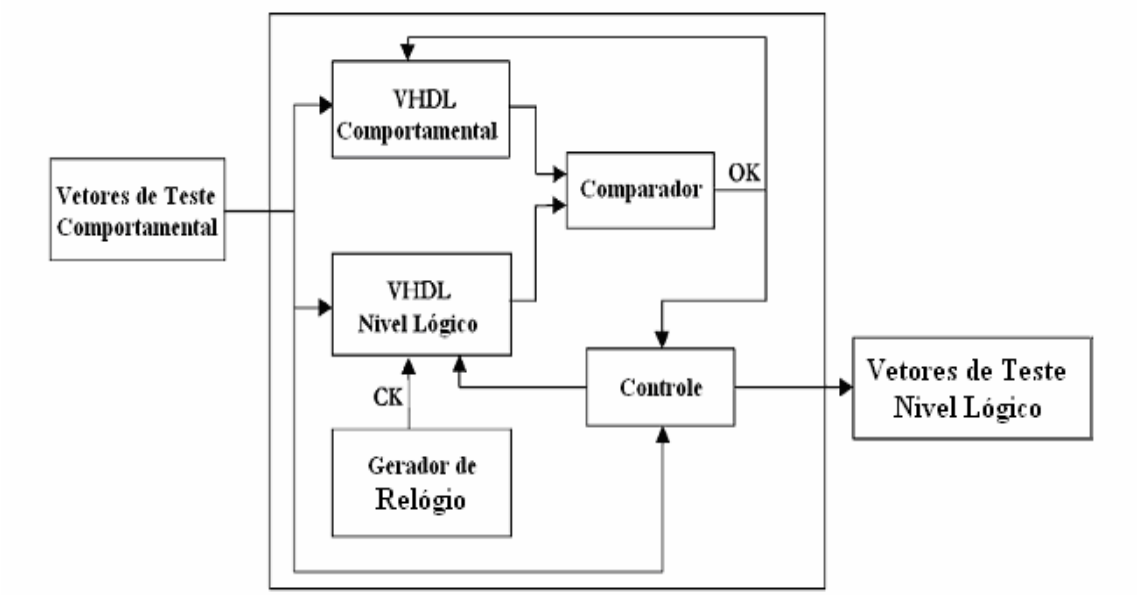

Figura 2.4: Blocos utilizados para geração de vetores de teste em nível lógico (FERRANDI, 2002).

Rudnick et. al. (RUDNICK, 1998) desenvolveram um gerador de vetores de teste que utiliza informações tanto da descrição VHDL como do circuito em nível de portas lógicas. Como entrada são utilizados códigos VHDL que podem conter múltiplos processos e todas as construções da linguagem. Em um primeiro passo, a técnica utiliza informações do código VHDL para gerar vetores de teste, utilizando a métrica de cobertura de declarações. Todas as declarações da descrição VHDL são exercitadas para gerar os padrões de teste iniciais. Para isso, a parte operativa e a parte de controle são identificadas e um grafo de transição de estado (STG - *State Transition Graph*) é gerado para a máquina de estados. Os padrões de teste iniciais são gerados levando em consideração todos os comportamentos possíveis dentro dos blocos de códigos para cada estado. Em um segundo momento, é utilizado um algoritmo genético (AG), para criar vetores de teste que consigam detectar um número maior de falhas com cada vetor. Para cada vetor de teste criado a partir das características do código VHDL, o AG explora várias alternativas de vetores por algumas gerações, e o melhor da espécie é inserido como o vetor de teste resultante. Este procedimento é repetido para todos os vetores gerados a partir do procedimento inicial. As informações do circuito no nível de portas lógicas são utilizadas pelo simulador de falhas PROOFS (HOLLAND, 1992) para avaliar o número de falhas detectadas por cada vetor de teste candidato. O vetor de teste que detecta o maior número de falhas, no simulador a nível lógico, entre o conjunto de indivíduos, é escolhido como o melhor da espécie. Os resultados mostraram que o número de falhas detectadas alcançado é semelhante, e às vezes superior, a duas ferramentas de geração de teste acadêmicas. Não foi feita uma comparação com ferramentas comerciais. As duas ferramentas acadêmicas utilizadas foram o HITEC (NIERMANN, 1991) e a GATEST (RUDNICK, 1994). O tempo gasto na geração dos vetores de teste foi inferior ao alcançado pelas outras duas ferramentas. Em contrapartida esta abordagem pode ser utilizada para gerar vetores de teste iniciais e deixar o gerador de vetores determinístico orientado a falhas para encontrar os vetores de teste para as falhas mais difíceis.

2.2.3 Projeto visando a Testabilidade

A geração e aplicação de vetores de teste podem ser mais eficientes se a testabilidade for considerada e melhorada durante a fase de projeto do circuito. O objetivo é melhorar a controlabilidade e a observabilidade do circuito com um mínimo incremento de área e performance, porém podem resultar em uma alta cobertura de falhas e baixo tempo de aplicação do teste, além de facilitar a geração de vetores. A controlabilidade e a observabilidade são os fatores mais importantes para determinar a complexidade da geração do teste. A controlabilidade é a facilidade de controlar (modificar) um determinado valor lógico em um determinado ponto do circuito, modificando valores nas entradas do circuito. Por outro lado, a observabilidade avalia a facilidade de se observar o valor lógico de um determinado ponto do circuito nas saídas primárias (ABRAMOVICI, 1990) (NORWOOD, 1997).

As técnicas de projeto visando a testabilidade (DFT - *Design for Testability*), têm como objetivo melhorar a testabilidade atuando durante a fase de projeto. Normalmente elas são utilizadas por projetistas que trabalham no nível de projeto lógico, podendo ser métodos *ad-hoc* ou estruturados. As técnicas de DFT *ad-hoc* levam em conta a experiência adquirida pelos projetistas e implicam uma boa prática de projeto. Alguns métodos de DFT *ad-hoc* são (BUSHNELL, 2000) (JERVAN, 2002):

- **Não permitir Realimentação Assíncrona** – as realimentações em lógica combinacional podem ocasionar oscilações com algumas entradas. A oscilação torna impossível a geração de vetores de teste por programas automáticos. Isso ocorre porque os algoritmos de geração de vetores de teste são construídos para circuitos combinacionais acíclicos (BUSHNELL, 2000).
- **Construir Registradores Inicializáveis** – a utilização de *flip-flops* com sinais de *reset* e *clear* facilita a controlabilidade por entradas primárias.
- **Não permitir Portas com *fan-in* elevado** – construir portas lógicas com *fan-in* elevado em suas entradas dificulta a controlabilidade da saída da porta lógica e a observabilidade das entradas.
- **Prover Controle de Teste para sinais difíceis de controlar** – sinais que requerem muitos ciclos de relógio para controlá-los incrementam o tamanho das seqüências de teste. Estas seqüências de teste muito grandes são difíceis de ser gerada. Para isso, normalmente, pinos de controle são inseridos nos circuitos melhorando a controlabilidade do mesmo.

Projetistas experientes conseguem utilizar as técnicas *ad-hoc* percebendo problemas em áreas do esquemático. Porém, utilizar técnicas *ad-hoc* para circuitos muito grandes nem sempre é possível. Além disso, é difícil encontrar problemas ou áreas difíceis de testar em um código VHDL, pois é necessário conhecer o comportamento da ferramenta de síntese.

Alguns trabalhos visam encontrar áreas difíceis de serem testadas usando algoritmos de definição de controlabilidade e observabilidade em alto nível (CHICKERMANE, 1994) (CHEN, 1994) (GU, 1997) (SESHADRI, 2002).

Como o tamanho e a complexidade dos sistemas digitais estão aumentando, as técnicas de DFT estruturadas estão ganhando popularidade. As técnicas de DFT estruturadas utilizam lógicas extras e sinais são inseridos nos circuitos para permitir o teste com alguns procedimentos pré-definidos. Os circuitos possuem, além de seu

estado de funcionamento normal, um ou mais modos de testes. Os métodos estruturados mais utilizados são o teste de varredura (*scan*) e o BIST (*built-in self-test*) (BUSHNELL, 2000). A seguir é apresentado o método de varredura, enquanto a técnica BIST é mostrada na seção 2.3.

2.2.3.1 Teste de varredura (*Scan*)

A idéia principal do projeto utilizando teste de varredura é obter controlabilidade e observabilidade dos registradores (*flip-flops*). Após a síntese, os *flip-flops* são modificados possibilitando a ligação destes para que, em modo teste, formem uma ou mais cadeias de deslocamento. Todos os *flip-flops* podem receber um determinado valor desejado deslocando os valores pela cadeia de deslocamento. Similarmente, os valores dos registradores também podem ser observados deslocando os valores dos registradores para a saída. Todos os *flip-flops* podem ser controlados e observados em um tempo (em termos de ciclos de relógio) igual ao número de *flip-flops* que compõem o registrador (cadeia) de deslocamento mais longo. (BUSHNELL, 2000).

A utilização da técnica de varredura possui algumas vantagens, tais como:

- Aumenta a controlabilidade e observabilidade, melhorando a cobertura de falhas.
- Pode ser totalmente automatizada utilizando ferramentas comerciais.

Entretanto a utilização do teste de varredura traz alguns custos na sua utilização, tais como (JERVAN, 2002) (NORWOOD, 1997) (BUSHNELL, 2000):

- Aumenta a área gasta pelo circuito, devido à modificação nos *flip-flops* para comporem os registradores de deslocamento.
- Requer pinos extras no circuito para a inserção, retirada e controle dos vetores de teste.
- Aumenta o atraso no circuito diminuindo a freqüência máxima de operação.
- Aumenta o consumo de energia durante o teste.
- Pode aumentar o tempo gasto no teste, devido à inserção serial de valores nos registradores de deslocamento.

Existem duas abordagens diferentes para teste de varredura: a varredura parcial e a varredura total.

A varredura parcial utiliza somente um subgrupo dos elementos de memória (*flip-flops*) para incluir nas cadeias de deslocamento. Por outro lado, a varredura total utiliza todos *flip-flops* nas cadeias de deslocamento. A principal vantagem do uso da varredura parcial é a redução dos custos de área e o aumento da velocidade do teste.

Esta seção apresentou os princípios básicos do teste externo usando ATE e os conceitos básicos da utilização do teste de varredura. Finalmente, nota-se que existem parâmetros de teste que mostram a qualidade do teste que utiliza varredura e ATE, que são: aumento da área, degradação da performance (freqüência de operação), eficiência do teste (cobertura de falhas), tempo de aplicação do teste e volume de dados do teste (vetores de teste).

2.3 Teste Interno usando BIST

A utilização do teste externo usando ATE possui o benefício de detectar os defeitos de fabricação. Por outro lado, ele possui dois problemas. O primeiro problema é que o ATE é extremamente caro e o preço deve aumentar cada vez mais, devido ao aumento do número de pinos dos circuitos no futuro. Outra desvantagem é que não é possível a inserção de um vetor de teste em um único ciclo de relógio quando se utiliza a técnica de varredura. Isso ocorre devido à necessidade de deslocamento dos dados pelas cadeias de deslocamento.

Estes problemas têm levado os projetistas a utilizar o auto teste integrado (BIST – *Built-In Self_Test*). BIST é uma técnica que utiliza partes do circuito para testar, internamente, o restante do circuito. Os vetores de teste não são gerados externamente como no ATE, mas são gerados internamente utilizando blocos internos ao circuito integrado. A arquitetura básica da DFT BIST, mostrada na Figura 2.5, necessita a adição de três blocos básicos ao circuito digital: um gerador de vetores de teste, um analisador de respostas e um bloco para o controle do teste. (AL-YAMANI, 2004) (TOUBA, 1996) (PATNAIK, 2002) (NADEAU-DOSTIE, 1999).

Existem alguns exemplos de geradores de vetores de teste para BIST: uma memória ROM (*Read Only Memory*) contendo os vetores armazenados; um CA (*Cellular Automata*) (CATTELL, 2005) ou um LFSR (*Linear Feedback Shift Register*) (CHEN, 2003b).

Tipicamente, o analisador de respostas é um comparador das respostas lidas durante o teste com as respostas armazenadas na memória ou é um LFSR usado como analisador de respostas (PATNAIK, 2002). O controle do teste é responsável pelo controle das tarefas do teste. Ele indica o início do teste e é responsável por indicar quando deve ser realizada a análise das respostas e informar se o circuito está com defeitos ou não.

O teste interno usando BIST traz algumas vantagens na sua utilização (AL-YAMANI, 2004). A seguir são mostradas estas vantagens:

- BIST geralmente possui um custo menor em relação ao teste externo usando ATE.
- Com teste interno usando BIST é possível aplicar o teste na velocidade de execução normal do circuito (teste *at-speed*) e isso ajuda na detecção de defeitos em tempos de teste menores.
- É possível testar o circuito sem tirar o circuito de operação (teste *online*).

Entretanto, a utilização de BIST possui algumas desvantagens:

- Pode acarretar em um grande aumento da área do circuito.
- Em alguns casos, para alcançar uma alta cobertura de falhas, podem ser necessários muitos vetores de teste, tornando proibitivo o teste (AL-YAMANI, 2004).

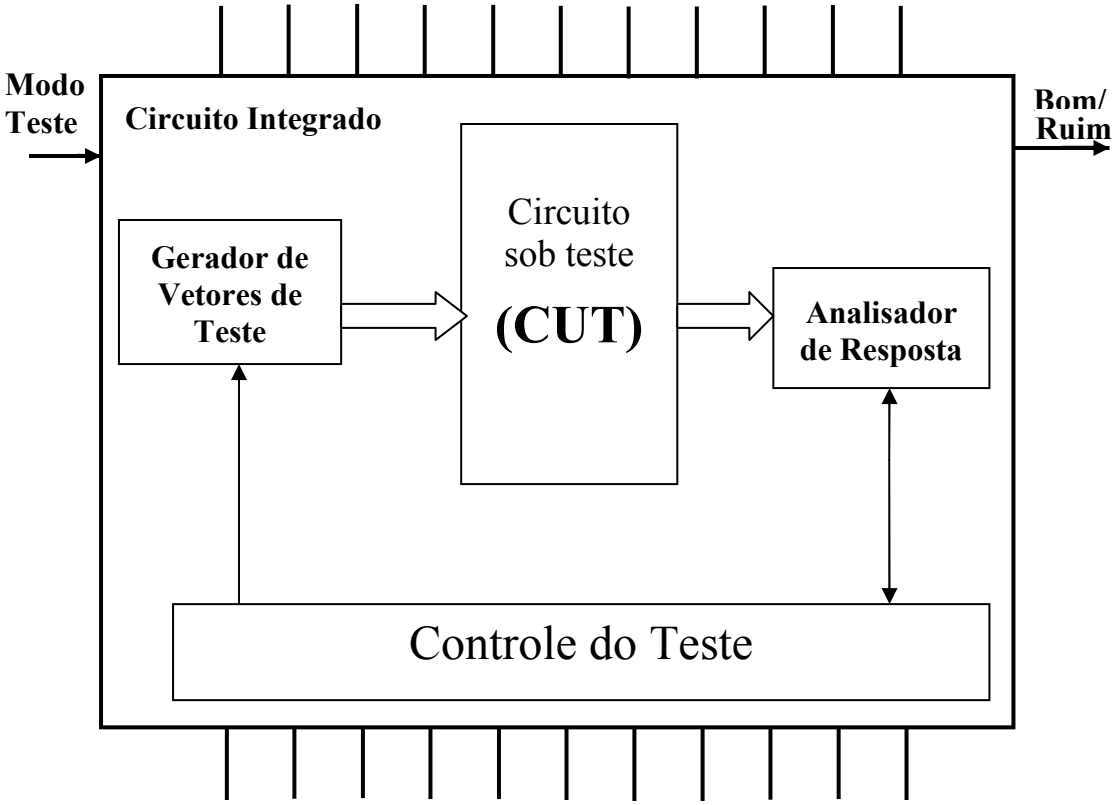

Figura 2.5: Princípio básico de teste interno usando BIST. (Figura adaptada de (NICOLICI, 2000)).

2.4 Abordagens para Aumentar Testabilidade

Além das técnicas de teste *ad-hoc* e estruturadas (varredura e BIST) apresentadas anteriormente, existem outras alternativas para buscar um aumento na testabilidade de descrições VHDL. Por exemplo, a análise da testabilidade do circuito resultante para encontrar gargalos para a testabilidade. Veremos alguns trabalhos que focam no aumento da testabilidade a seguir.

2.4.1 Análise de Testabilidade

A análise de testabilidade, originalmente, utilizava os circuitos no nível lógico e procurava mostrar a controlabilidade e observabilidade dos sinais do circuito. Em 1979, Goldstein (BUSHNELL, 2000) propôs um algoritmo para determinar a dificuldade para controlar ou observar sinais em um circuito digital. Foi construído um algoritmo eficiente para computar estas medidas, o qual foi chamado de SCOAP (*Sandia Controllability / Observability Analysis Program*). O SCOAP consiste de seis medidas para cada sinal do circuito: *Combinational* 0-*controllability; Combinational* 1-*controllability; Combinational observability; sequential* 0-*controllability; sequential* 1-*controllability; sequential obervability.*

As três medidas combinacionais são relativas ao número de sinais que devem ser manipulados para controlar ou observar o sinal do circuito. As três medidas seqüenciais são relativas ao número de ciclos de relógio necessários para controlar ou observar o sinal. A medida é realizada com o circuito em nível lógico. O cálculo da

controlabilidade é medido através da dificuldade de controlar o sinal específico, onde se está medindo a controlabilidade, a partir das entradas do circuito. O valor da controlabilidade do sinal específico é a máxima distância (em número de portas lógicas) deste ponto até as entradas. A cada porta lógica até a profundidade do circuito é somado o valor 1 na controlabilidade. Após todas as controlabilidades serem estabelecidas, é necessário realizar o cálculo das observabilidades. O cálculo é o inverso, pois inicia dos pinos de saída e o algoritmo é realizado em direção as entradas.

Além desta medida de testabilidade que utiliza o circuito em nível lógico, atualmente existem trabalhos que geram medidas de testabilidade com o circuito em nível comportamental (LEE, 1993) (VISHAKANTAIAH, 1993a) (CHICKERMANE, 1994) (GU, 1994) (CHEN, 1994) (NOURANI, 1997) (CORNO, 1997) (GU, 1997) (HSU, 1998) (SESHADRI, 2002).

A abordagem proposta por Gu (1994) utiliza descrições VHDL e transforma esta especificação em uma representação de redes de *petri* estendida. A análise de testabilidade é definida em relação às medidas de controlabilidade e observabilidade. As medidas de controlabilidade se referem ao custo para implicar um valor em uma linha que conecta duas unidades funcionais (registradores, somadores, multiplexadores ou multiplicadores) e as medidas de observabilidade são referentes à dificuldade de observar estes valores nas saídas. Os custos são relativos ao tempo para encontrar os vetores para justificar a falha em uma linha ou para observar um valor em uma saída do circuito e o custo para alcançar uma cobertura de falhas elevada. Após obter a análise de testabilidade do circuito, duas técnicas são usadas para melhorar a testabilidade do circuito tendo por base estes resultados. Se existem registradores responsáveis por áreas difíceis de serem testadas, estes registradores são transformados para fazer parte da cadeia de varredura. A segunda estratégia insere células de teste (*T-Cell*) nas linhas que conectam as unidades funcionais que possuem problemas de testabilidade.

Em outro trabalho, Gu (1997) utiliza a especificação do circuito para construir o fluxo de controle, utilizando redes de petri (PETERSON, 1983). Em seguida são analisadas as redes de petri para encontrar estados da parte de controle que são difíceis de serem alcançados e, conseqüentemente, possuem problemas de testabilidade. Depois de descobertos estes problemas, são inseridos pinos ou modificados os registradores para formarem as cadeias de deslocamento para o teste de varredura. Estes dois tipos de modificações são utilizados para inicializar os registradores ou facilitar o controle dos laços de repetição ou o controle dos desvios. Esta abordagem melhora a cobertura de falhas, mas com um aumento de área e aumento no número de pinos do circuito.

Seshadri (2002) utiliza a análise de testabilidade para encontrar pontos do circuito que são difíceis de testar. Ele utiliza o fluxo de dados e o fluxo de controle obtido a partir dos códigos VHDL comportamentais para determinar a controlabilidade e observabilidade de todos os operandos dos códigos. A partir desta análise é possível inserir pinos de teste nos pontos do circuito que são difíceis de observar e inserir cadeia de varredura parcial nos registradores que são identificados como difíceis de testar. Esta técnica mostra resultados obtidos com alta cobertura de falhas e que não possuem um aumento de área tão significativo quanto com a utilização de varredura total.

Outro trabalho que utiliza análise da testabilidade em códigos VHDL para identificar áreas dificilmente testáveis no projeto foi proposto por Chickermane (1994). A técnica analisa o número de ciclos de relógio necessários para controlar e observar nodos da descrição. Após esta análise, podem ser inseridos caminhos de varredura parciais, o que

melhora a testabilidade sem comprometer tanto a área do circuito resultante quanto a utilização da técnica de varredura total. Outro trabalho, proposto por Chen (1994), analisa as descrições comportamentais e detecta as áreas difíceis de serem testadas. Após a análise são inseridos pontos de teste ou caminhos de varredura parcial para melhorar a controlabilidade e a observabilidade dos circuitos.

Outra técnica que aumenta a controlabilidade de descrições VHDL comportamentais foi proposta por Hsu (1998) e foca no fluxo de controle das descrições. Para isso, alterações nas descrições VHDL são realizadas, o que permite ao projetista ter uma visão preliminar da testabilidade antes da síntese do circuito. Os autores propõem alterar o fluxo de controle da execução inserindo pontos de controle em laços e desvios. O nodo de decisão (desvio ou laço) é dito k-controlável se pode ser controlado direta ou indiretamente por uma entrada primária com k ciclos de relógio. Se o valor de k é elevado, então este nodo é candidato a receber um dos três tipos de alterações em laços e desvios. A controlabilidade é alcançada inserindo pinos de teste com funções lógicas AND, OR ou XOR para forçar os nodos de decisão para TRUE ou FALSE. O trabalho mostrou que com a inserção de poucos pinos de teste foi alcançada uma melhora na cobertura de falhas das descrições utilizadas.

Outra forma empregada para alcançar uma alta cobertura de falhas é utilizar o conhecimento da funcionalidade do circuito (BHATTACHARYA, 1996) para reduzir o acréscimo de área da inserção de cadeias de varredura. A técnica observa no circuito alguns registradores que formam caminhos entre as entradas e as saídas. Estes registradores que formam caminhos são utilizados para criar caminhos de teste paralelos às cadeias de varredura que ligam os outros registradores do circuito. Após todos os caminhos funcionais serem inseridos, se mesmo assim houver a necessidade de inserção de cadeia de varredura, poucos registradores são modificados para formar as cadeias de varredura complementares. Com isso, a testabilidade é aumentada em relação à descrição original do circuito e é reduzido o acréscimo de área em relação à inserção de varredura total, pois nem todos registradores necessitam ser modificados.

2.4.2 Análise do Circuito Sintetizado

A geração de vetores de teste para circuitos seqüenciais no nível lógico pode ser muito demorada devido ao enorme espaço de busca, o que pode tornar a geração de teste impraticável. Existem trabalhos (VISHAKANTAIAH, 1993a) que utilizam o conhecimento do comportamento dos módulos para reduzir o espaço de busca do ATPG. A Figura 2.6 mostra um exemplo de módulos conectados formando um módulo maior e criando uma série de restrições ao ATPG. Por exemplo, para detectar uma falha no módulo O a partir das entradas, é necessário levar em consideração os módulos X, Z, P e Q. Estes módulos são responsáveis pela condução dos valores entre os módulos e são necessários para o ATPG encontrar um vetor de teste para o módulo O. As ferramentas de ATPG podem não levar em consideração restrições entre os módulos e isso pode tornar o procedimento de geração de vetores de teste muito caro em relação ao tempo gasto. A ferramenta ATKET (VISHAKANTAIAH, 1992) extrai o conhecimento da ligação entre os blocos dos circuitos descritos na linguagem VHDL, que serão utilizados pela ferramenta AMBIANT (VISHAKANTAIAH, 1993b) para identificar caminhos de propagação e justificação das falhas internas aos módulos. A ferramenta armazena o conhecimento dos valores necessários para guiar a justificação e propagação das falhas dos módulos que são formados por vários módulos e isso é utilizado para reduzir o espaço de busca e com isso diminuir o tempo de execução do ATPG. Gargalos

para a testabilidade são identificados quando as restrições entre os módulos não são realizadas, isto é, quando não é encontrado um vetor de teste para aquele caminho de teste.

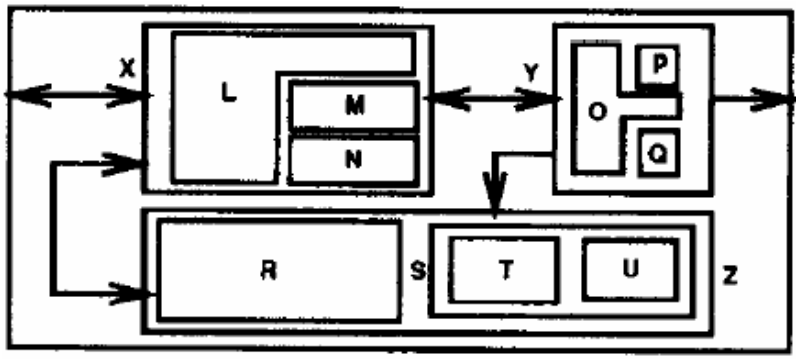

Figura 2.6: Conexões entre módulos (VISHAKANTAIAH, 1993b).

As técnicas utilizadas nestas ferramentas auxiliam os projetistas a considerar características de testabilidade nos estágios iniciais do fluxo de projeto. As restrições entre módulos que não são realizadas pela ferramenta ATKET, podem ser um gargalo para o teste global. A ferramenta AMBIANT gera sugestões para o projetista quando a ferramenta ATKET não realizou as restrições. As sugestões são aceitas ou não pelo projetista para gerar um projeto mais testável. Os resultados são interessantes, mas a ferramenta possui uma pequena quantidade de sugestões para o projetista, tais como inserção de pinos de controlabilidade e a utilização de re-síntese do circuito com as modificações sugeridas.

2.4.3 Síntese visando Teste

Normalmente, as ferramentas de síntese são capazes de gerar um circuito otimizado para ocupar uma área menor ou atingir freqüência de operação superior. Porém, estas ferramentas nem sempre produzem circuitos que são otimizados para testabilidade. Atualmente existem propostas, a partir de descrições VHDL, para que os processos de síntese possam objetivar uma melhoria nas características de testabilidade do circuito resultante após a síntese. Alguns trabalhos têm como objetivo reduzir os laços dentro do circuito (*feedback*), aumentar a controlabilidade e observabilidade dos registradores ou gerar circuitos considerando medidas de testabilidade durante as alocações dos registradores (SAFARI, 2003). Estas técnicas necessitam ter acesso aos processos e algoritmos de síntese, o que nem sempre é possível.

Como visto anteriormente, existem várias técnicas para aumentar a testabilidade. Algumas abordagens necessitam o controle do processo de síntese (FLOTTES, 1995) (VISHAKANTAIAH, 1993b), outras modificam os registradores para construir as cadeias de varredura, ou ainda são baseadas na inclusão de pontos de controle / observabilidade nos circuitos. A primeira abordagem é difícil de ser utilizada devido à necessidade de ter controle do processo de síntese. A segunda abordagem geralmente é utilizada com o circuito no nível lógico e acarreta em um aumento de área. A terceira abordagem requer a análise de testabilidade para descobrir onde inserir os pinos de controle / observação no circuito. Estas abordagens podem afetar os custos de projeto como área e performance.

Este trabalho vai verificar a possibilidade de aumentar a testabilidade do circuito apenas modificando o estilo das descrições VHDL. Para isso não vai ser necessário modificar o processo de síntese e nem aumentar o número de pinos no circuito, pois apenas irá ser modificada a forma de descrever o código VHDL. Os próximos capítulos irão mostrar o desenvolvimento da metodologia utilizada. Em um primeiro momento, foram construídos vários códigos VHDL que descrevem um mesmo algoritmo e isso vai expor o fato de que diferentes estilos de descrição causam diferenças em relação às medidas de testabilidade do circuito final. Depois, estas descrições recebem técnicas de teste *ad-hoc* visando aumentar a testabilidade. As descrições com as melhores características de testabilidade são comparadas com as demais para levantar hipóteses do porque estas são melhores para o teste. A seguir, as descrições com características de testabilidade inferiores são modificadas para alcançar um circuito final com melhor testabilidade.

3 ESTUDO DE CASO: CÁLCULO DA RAIZ QUADRADA

Esta seção abordará o estudo de caso para a dissertação mostrando a utilização de códigos de descrição de hardware em alto nível de abstração e seus impactos na testabilidade do circuito gerado. Para o estudo são utilizados diferentes estilos de descrição de um código VHDL que implementa o algoritmo da raiz quadrada. O objetivo é mostrar que há uma relação estreita entre a forma de descrição em alto nível do circuito e as medidas de testabilidade verificadas após a síntese.

Os estilos de descrição de códigos que implementam circuitos integrados são os estilos comportamental e estrutural. Uma descrição comportamental é uma especificação da funcionalidade do projeto em forma de algoritmo, podendo conter pouca ou nenhuma informação sobre a estrutura e comportamento ciclo-a-ciclo da implementação. Descrições estruturais, por outro lado, contêm informações do comportamento ciclo-a-ciclo da implementação e interconexões de blocos (tais como: unidades funcionais, registradores, multiplexadores, barramentos, blocos de memória, etc.) descritos com baixo nível de abstração, ou seja, mais próximo da implementação física do circuito (DEY, 1998).

O estudo de caso descrito neste capítulo mostra vários códigos VHDL que implementam um mesmo algoritmo, produzindo circuitos com diferentes propriedades de área e freqüência de operação. O estudo também expõe o fato de diferentes estilos de descrição causarem diferenças em relação às medidas de testabilidade do circuito final.

Os projetos normalmente passam por etapas de teste no seu fluxo de desenvolvimento como explanado no capítulo anterior. A possibilidade de mapear características de códigos VHDL que produzem um circuito final mais testável pode ser de grande valia para a comunidade científica e para a indústria. Tal mapeamento pode auxiliar os projetistas a descrever códigos que produzam circuitos mais testáveis. A utilização das características em descrições de alto nível traz vantagens, pois é mais fácil modificar um circuito descrito em várias linhas de código VHDL do que modificar o projeto descrito em milhares de transistores.

A seguir é apresentado o algoritmo que implementa a raiz quadrada (algoritmo extraído de (CARRO, 2001)) e as diversas descrições que o implementam. Os resultados de síntese e testabilidade também são mostrados e discutidos. Ao final da seção são apresentadas algumas conclusões.

3.1 Algoritmo da Raiz quadrada

O algoritmo, mostrado na Figura 3.1, realiza o cálculo da raiz quadrada de um número inteiro. O resultado, também inteiro, é um arredondamento para baixo da

verdadeira raiz. Desta maneira, a raiz de $I = 42$ (6,48074069) vai ser arredondada para 6 pelo algoritmo. Na figura, I é o número inteiro do qual se quer extrair a raiz, e r é o resultado ao final da execução.

```
t = 1
r = 1
d = 2
s = 4

WHILE ( t ) {
      r = r + 1
      d = d + 2
      s = s + d + 1
      t = sgn ( s – I )
}
Sgn = 1 se s ≤ I, senão sgn = 0
```

Figura 3.1: Algoritmo da raiz quadrada.

É importante ressaltar as dependências de dados existentes entre as variáveis no algoritmo. A primeira é entre *s* e *d*, ou seja, o novo valor de *s* só pode ser calculado depois de o valor de *d* ter sido calculado. Da mesma forma, existe uma dependência entre *s* e *t*. A dependência de dados influencia a implementação do algoritmo em hardware, visto que existe uma barreira temporal entre os dados que deve ser levada em conta. A não observação desta dependência pode acarretar a utilização do valor anterior (portanto incorreto) na iteração do algoritmo.

Normalmente, na implementação do algoritmo como um circuito, as variáveis são transformadas em registradores e a dependência de dados é controlada através da construção de uma máquina de estados que controla o tempo de atualização dos mesmos. A seguir são mostradas as características e as diversas possibilidades de implementação do algoritmo.

3.2 Implementações do Algoritmo

A implementação do algoritmo de cálculo da raiz quadrada em hardware pode ser feita através da descrição do mesmo em VHDL e posterior síntese para um determinado substrato (FPGA, *Standard Cell*, etc.). A descrição do algoritmo em VHDL, por outro lado, pode ser feita de diversas maneiras produzindo circuitos com diferentes áreas e freqüências de operação. Algumas destas possibilidades são, por exemplo:

- Descrever na forma estrutural ou comportamental.
- Descrever a Parte de Controle (PC) e a Parte Operativa (PO) separadas ou juntas.
- Utilizar máquina de estados explícita ou descrever em forma de um algoritmo.

- A comunicação entre a PC e a PO pode ser feita através do estado onde se encontra a máquina de controle ou através de sinais de controle explícitos ou, ainda, através de micro-programação.

Sabendo-se das inúmeras possibilidades de descrição disponíveis para um projetista, e sabendo-se que a síntese será diferente para cada tipo de descrição, o primeiro objetivo deste estudo de caso é definir o grau de variabilidade da testabilidade do circuito gerado a partir de cada modelo em alto nível. A partir deste estudo pretende-se definir estilos de descrição que auxiliem o teste do circuito final.

Foram utilizadas seis descrições na linguagem VHDL do algoritmo da raiz quadrada, sendo quatro estruturais e duas comportamentais. As descrições foram projetadas por alunos da disciplina CMP117 da Pós-Graduação em Computação da UFRGS nos anos de 1998 e 2003. As formações das pessoas que geraram os diversos códigos variam de Engenharia Elétrica, Ciência da Computação e Engenharia da Computação. As descrições foram inicialmente modificadas para possuírem o mesmo número de pinos de entrada e saída. Todas as seis versões possuem onze pinos de entrada e cinco pinos de saída, como mostra a Figura 3.2. Os pinos de entrada são: um pino para relógio do sistema, um pino para reset, um pino para iniciar o cálculo e oito pinos para entrada de dados. Os pinos de saída são: um pino para indicar que o cálculo está pronto e quatro pinos para saída de dados.

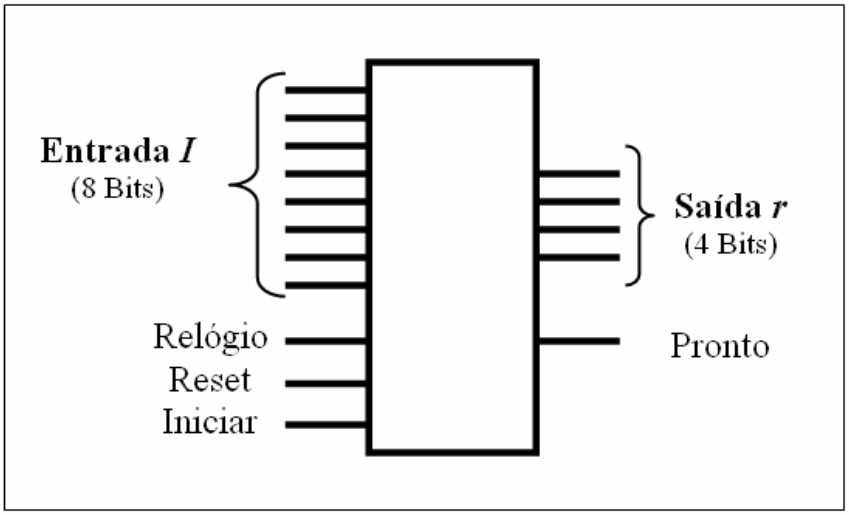

Figura 3.2: Pinos de entrada e saída dos circuitos que implemantam o algoritmo.

As versões transformam cada expressão do algoritmo em um equivalente em hardware. Ou seja, sendo *r*, *d* e *s* variáveis do algoritmo, as mesmas são simplesmente implementadas em registradores, cada uma com o seu próprio operador. Uma simples máquina de estados pode ser implementada para controlar a dependência de dados discutida anteriormente ou pode-se descrever em forma de um algoritmo.

As características de cada versão são mostradas a seguir:

- Versão E1 - descrita na forma estrutural com registradores declarados como sinais, possui PC e PO separadas, utiliza máquina de estados (descrito com a construção IF THEN ELSE) e a comunicação entre a PC e a PO é realizada através do estado em que a máquina de estados se encontra. Não utiliza funções descritas pelo projetista e os somadores utilizam as bibliotecas da ferramenta, pois são decritos com o símbolo +.

- Versão E2 - descrita na forma estrutural com registradores declarados como sinais, possui PC e PO separadas, a comunicação é realizada por código micro-programado, ou seja, a parte de controle envia um vetor de controle para a parte operativa. Não utiliza funções descritas pelo projetista e os somadores utilizam as bibliotecas da ferramenta, pois são decritas com o símbolo +.
- Versão E3 - descrita na forma estrutural com registradores declarados como sinais, possui PC e PO juntas e menos estados na descrição da máquina de estados. Não utiliza funções descritas pelo projetista e os somadores utilizam as bibliotecas da ferramenta, pois são decritas com o símbolo +.
- Versão E4 - descrita na forma estrutural com PC e PO separadas, utiliza máquina de estados (descrito com a construção CASE). A comunicação entre PC e PO é realizada através do estado em que a máquina de estados se encontra. Não utiliza funções descritas pelo projetista e os somadores utilizam as bibliotecas da ferramenta, pois são decritas com o símbolo + da linguagem VHDL.
- Versão C1 - descrita na forma comportamental com registradores declarados como variáveis. Não utiliza funções e não possui máquina de estados explícita. Entretanto, a descrição apresenta uma única construção estrutural: a comparação entre os sinais s e I é implementada através de um subtrator e a comparação do bit mais significativo do resultado da subtração, ao invés de utilizar o símbolo < da linguagem.
- Versão C2 - descrita na forma comportamental. Utiliza variáveis, funções e não possui máquina de estados explícita. Contudo é usada uma função para implementar a comparação entre os sinais s e I do algoritmo. Esta função é uma construção permitida pela linguagem e é declarada fora do processo principal que controla as operações. Os somadores utilizam as bibliotecas da ferramenta, pois são decritos com o símbolo + da linguagem VHDL.

A seguir são apresentados dois exemplos de descrição VHDL que implementam o algoritmo da raiz quadrada. A Figura 3.3 mostra a descrição estrutural E1 e a Figura 3.4 apresenta a descrição comportamental C1. As demais descrições podem ser encontradas no Anexo.

```
-- Descrição Estrutural E1

ENTITY raizq IS

PORT
(
    x:      IN  INTEGER range 0 to 255;  -- valor de entrada
    clk:    IN  BIT;  -- relógio
    start:  IN  BIT;
    reset:  IN  BIT;
    pronto: OUT BIT;    -- sinal de resultado em y
    y:      OUT INTEGER range 0 to 15  -- raiz_quadrada de x
);

END raizq;
```

```
ARCHITECTURE comportamento OF raizq IS
TYPE STATE_TYPE IS (estado_0, estado_1, estado_2, estado_3,
    estado_4, estado_5);
SIGNAL state : STATE_TYPE;
SIGNAL r,d,s : INTEGER range 0 to 256;
SIGNAL t : BIT;

BEGIN

    -- Descricao Parte Controle
PROCESS (clk,reset)

BEGIN-- inicio do processo

IF (reset = '1')then
    state <= estado_0;
ELSIF (clk'EVENT AND clk = '1') THEN
    CASE state IS
      WHEN estado_0 =>
           if start = '1' then
                state <= estado_1;
           else
                state <= estado_0;
           end if;
      WHEN estado_1 =>
           IF (t = '1')then
                state <= estado_2;
           ELSE
                state <= estado_5;
           END IF;
      WHEN estado_2 =>
           state <= estado_3;
      WHEN estado_3 =>
           state <= estado_4;
      WHEN estado_4 =>
           state <= estado_1;
      WHEN estado_5 =>
           state <= estado_0;
      WHEN OTHERS =>
           state <= estado_0;
    END CASE;
END IF;
END PROCESS;  -- Fim descricao PC


PROCESS (state,clk) -- Descricao Parte Operativa

BEGIN

IF (clk'EVENT AND clk = '0') THEN
    IF (state = estado_0) THEN
      r <= 1;
      d <= 2;
      s <= 4;
      t <= '1';
      pronto <= '0';
      y <= 0;
ELSIF (state = estado_2) THEN
      r <= r + 1;
      d <= d + 2;
```

```
   ELSIF (state = estado_3) THEN
          s <= s + d + 1;
   ELSIF (state = estado_4) THEN
        IF (s <= x)then
          t <= '1';
        ELSE
          t <= '0';
        END IF;
   ELSIF (state = estado_5) THEN
        y <= r;
        pronto <= '1';
   END IF;
   END IF;
   END PROCESS;   -- Fim descricao PO
END comportamento;
```

Figura 3.3: Descrição estrutural E1.

```
   -- Descrição Comportamental C1

   library ieee;
   use ieee.std_logic_1164.all;
   use ieee.std_logic_unsigned.all;

   entity raiz is

   port (
        clk:    in  std_logic;
        reset:  in  std_logic;
        start:  in  std_logic;
        i:      in  std_logic_vector ( 7 downto 0);
        pronto: out std_logic;
        result: out std_logic_vector(3 downto 0)
   );
   end raiz;

   architecture a_raiz of raiz is

   begin
   process (clk, reset)

   variable r : std_logic_vector(3 downto 0);
   variable d : std_logic_vector(4 downto 0);
   variable s,temp : std_logic_vector(8 downto 0);

   begin
   if (reset='1') then
        r := "0000";
        d := "00000";
        s := "000000000";
        pronto <= '0';
   elsif (clk'event and clk='1') then
        if start = '1' then
             r := "0001";
             d := "00010";
             s := "000000100";
             pronto <= '0';
        else
             temp := i-s;
             if (temp(8)='0') then
```

```
                r := r + 1;
                d := d + 2;
                s := s + d + 1;
           else
                pronto <= '1';
                result <= r;
           end if;
      end if;
 end if;
 end process;
 end a_raiz;
```

Figura 3.4: Descrição comportamental C1.

3.3 Resultados de Síntese e Testabilidade

Nesta seção serão mostrados os resultados práticos da síntese para ASIC (*Application Specific Integrated Circuit*) e as medidas de testabilidade das diversas formas de descrição do algoritmo da raiz quadrada. Serão discutidas ainda as implicações do estilo de descrição na testabilidade gerada pelo circuito. A implementação para FPGA (*Field Programmable Gate Array*) está fora do escopo desta dissertação.

Para a obtenção dos resultados de síntese e testabilidade, o fluxo mostrado na Figura 3.5 foi usado. A ferramenta Leonardo Spectrum™ da Mentor Graphics (MENTOR, 1999) foi usada para sintetizar as descrições VHDL e a ferramenta Flextest™ (MENTOR, 1999) foi utilizada para obter as características de teste dos diversos circuitos sintetizados.

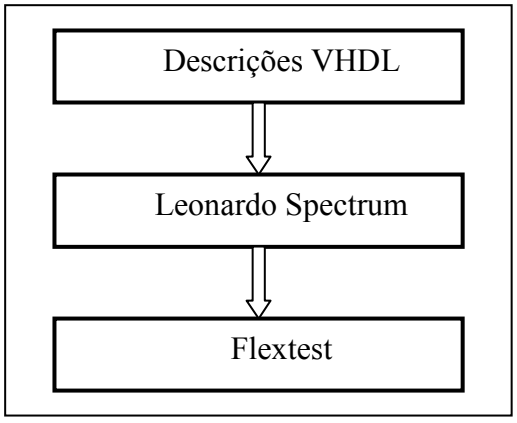

Figura 3.5: Fluxo para obtenção dos resultados de síntese e testabilidade.

A Tabela 3.1 mostra os resultados de síntese como número de portas lógicas, freqüência de operação, e número de *flip-flops*. Além disso, mostra os resultados de testabilidade das várias versões da raiz, tais como: número total de falhas, número de falhas detectadas por simulação, número de falhas não testáveis, número de falhas não testáveis pelo ATPG, número de falhas não observáveis, número de falhas não controláveis, cobertura de teste e o número de padrões gerados são apresentados para facilitar a avaliação dos resultados.

Os resultados de síntese foram obtidos sintetizando as descrições para ASIC utilizando a tecnologia 0.35um da biblioteca ADK e configurando a ferramenta

Leonardo Spectrum™ para manter a hierarquia dos blocos da descrição e tendo como objetivo obter uma menor área do circuito. A partir da Tabela 3.1 pode-se notar que existem grandes diferenças em relação às características de síntese. As versões estruturais possuem freqüência de operação maior que as versões comportamentais, chegando a versão estrutural E4 a possuir uma freqüência de operação 73,7% maior que a freqüência da versão comportamental C1. As freqüências de operação das descrições comportamentais são menores devido à forma de descrição que não implementa uma máquina de estados explícita, o que implica a necessidade da ferramenta de síntese escalonar as tarefas. Como os algoritmos utilizados pelas ferramentas de síntese nem sempre garantem um desempenho adequado, ou seja, podem não produzir circuitos sintetizados com características melhores que os estruturais, a maioria dos projetos atuais ainda é descrita na forma estrutural.

Tabela 3.1: Resultados de síntese e testabilidade.

	E1	E2	E3	E4	C1	C2
Número de Portas	351	360	315	329	427	259
Freqüência de Operação (MHz)	250.9	254.7	226.2	288.7	166.2	212.7
Flip-Flops	34	33	26	33	23	24
Total de falhas	1132	1268	1070	1033	1836	958
Não Testáveis pelo ATPG (%)	7	2,2	0,7	4,8	0,6	1,4
Não Observáveis (%)	11,5	7,6	9,5	8,3	39,9	16,4
Não Controláveis (%)	8,7	0	0,7	2,2	0	0,6
Cobertura de Falhas (%)	72,8	90,2	89,1	84,70	59,5	81,6
N° de vetores Gerados	300	8521	359	2597	232	136

Em relação à área gasta por cada versão, pode-se notar que a menor (C2) e a maior (C1) área consumida são das versões comportamentais, possuindo uma diferença de 64,8% no tamanho entre elas. Procurou-se descobrir o porquê da diferença ser tão grande entre duas descrições do mesmo estilo (comportamental) e descobriu-se que os algoritmos de síntese tiveram dificuldades no escalonamento da construção estrutural contida na descrição C1. A construção é estrutural devido ao seu detalhamento ser maior, como visto na Figura 3.6(a), que uma simples comparação entre dois registradores (Figura 3.6 (b)). A versão C1 sofreu uma modificação em sua estrutura trocando esta construção estrutural pelo seu correspondente código comportamental e os algoritmos de síntese geraram um circuito com características de síntese e testabilidade semelhantes a da descrição C2. Com isso, a área ocupada pelo circuito gerado a partir da descrição C1 modificada diminuiu para 274 portas, com freqüência de operação de 207,7 MHz e cobertura de falhas de 76,8% gerados com 81 vetores de teste.

temp := i-s; if (temp(8)='0')	if (s <= i)
(a)	(b)

Figura 3.6: (a) Construção estrutural. (b) Modificação para comportamental.

Os resultados de testabilidade da Tabela 3.1 foram obtidos deixando a ferramenta Flextest rodar o ATPG com as configurações padrão. Dentre os diversos resultados, cabe salientar que: as falhas classificadas como não testáveis pelo ATPG podem, na realidade, serem testáveis. Porém, o ATPG não conseguiu encontrar um vetor de teste por restrições ou limitações da própria ferramenta. As restrições do ATPG são necessárias para restringir o número de tentativas que o gerador de vetores de teste realiza antes de abortar a geração para a falha específica a qual está sendo rodada. São três maneiras de abortar a geração de vetores. A primeira maneira é configurar o ATPG para executar até um determinado valor máximo de conflitos do gerador de vetores (*backtracking*). Os conflitos ocorrem quando o ATPG não consegue assinalar valores lógicos baseados nas condições necessárias para a propagação da falha. Quando isso ocorre, o ATPG volta para a condição anterior e tenta alcançar uma saída através desta nova condição. O valor para *backtracking* é o número de tentativas de utilizar outra condição que o ATPG pode utilizar para cada falha do circuito. Outra maneira para abortar o ATPG é o número máximo de ciclos de relógio utilizado na geração de vetores de cada falha. A terceira maneira é definir o tempo máximo (em segundos) que o ATPG pode executar antes de abortar a geração para a falha específica. Para os experimentos apresentados nesta dissertação, a restrição imposta ao ATPG foi de que a ferramenta deveria gerar padrões de teste utilizando um número máximo de *backtracking* igual a 3000, número máximo de ciclos igual a 5000 e limite de tempo gasto pelo ATPG igual a 5000s. As falhas não observáveis são aquelas para as quais não se pode propagar o comportamento errôneo do circuito a um ponto observável. As falhas não controláveis são aquelas para as quais o procedimento de geração de padrões não consegue encontrar valores de entrada que imponham o valor oposto ao da falha no nodo de interesse (MENTOR, 1999). Somando o percentual de falhas não testáveis pelo ATPG com os percentuais de falhas não observáveis, não controláveis e o percentual de cobertura de falhas, obtense os 100% de falhas do circuito.

Além das diferenças de síntese entre todos seis circuitos, pode-se notar que ocorreram diferenças em relação à testabilidade dos mesmos. As versões possuem diferenças em relação à cobertura de falhas e número de vetores de teste. A versão estrutural E2 possui cobertura de falhas 51,5% maior que a versão comportamental C1. A Figura 3.7 mostra as máximas diferenças entre todas as seis versões em forma de gráfico, indicando que os circuitos resultantes podem possuir grandes diferenças não só nas características de síntese (área = 64,8% e freqüência = 73,7%), mas também na cobertura de falhas (51,5%) do circuito gerado.

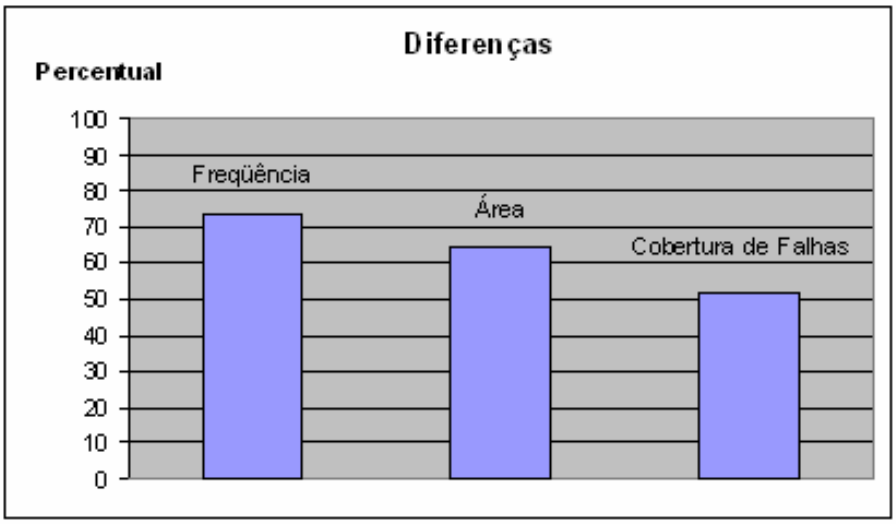

Figura 3.7: Máximas diferenças em relação à freqüência, área e testabilidade entre todas as versões.

Considerando somente as descrições estruturais E1, E2, E3 e E4, pode-se observar na Tabela 3.1 que a variação de área é de, no máximo, 14% (diferença entre as versões E2 e E3), como mostra a Figura 3.8 (a). A Figura 3.8 (b) mostra que a freqüência de operação possui uma variação maior que a variação de área, pois chega a ter uma diferença de 27% entre as versões estruturais E3 (226.2 MHz) e E4 (288.7 MHz).

As comparações entre as características de testabilidade mostradas na Tabela 3.1 não são muito justas, visto que a cobertura de falhas é obtida com um número livre de vetores. O número de vetores de teste diferentes implica um tamanho de memória de teste e tempo de execução do teste diferente entre as versões. Para uma comparação mais justa foi realizada uma alteração na configuração do Flextest™, indicando que a ferramenta deve parar o ATPG quando o número de padrões gerados for igual a 136. Este valor foi escolhido por ser o menor número de vetores utilizado entre todas versões. Conforme vimos na Tabela 3.1, a versão C2 utiliza somente 136 vetores para obter sua cobertura de falhas. A Tabela 3.2 mostra as estatísticas da execução do Flextest™ com esta alteração.

Tabela 3.2: Testabilidade das soluções com número máximo de padrões igual a 136.

	E1	E2	E3	E4	C1	C2
Total de falhas	1132	1268	1070	1033	1836	958
Não Testáveis pelo ATPG (%)	0	2,2	0,5	0	0,05	1,4
Não Observáveis (%)	31	29,7	14,1	31,4	45,95	16,4
Não Controláveis (%)	11,8	4	5,3	7	0	0,6
Cobertura de Falhas (%)	57,2	64,0	80,1	61,6	54	81,6
N° de vetores Gerados	136	136	136	136	136	136

Para um mesmo número de vetores de teste e comparando apenas os resultados entre as quatro versões estruturais, embora a variação de área e freqüência de operação dos circuitos seja no máximo 14% e 27% respectivamente, a variação na cobertura de falhas é bem maior, chegando a 40% de diferença entre as versões de maior (E3) e

menor (E1) cobertura de falhas, para um mesmo número de vetores de teste. Isto indica que a síntese gera circuitos com características de acessibilidade bastante distintas. A Figura 3.8 apresenta as diferenças máximas entre as versões estruturais, indicando que a característica de testabilidade (cobertura de falhas, Figura 3.8(c)) possui uma variação maior que as outras características de síntese. Isso mostra que mesmo utilizando o mesmo nível de descrição podem-se alcançar diferentes circuitos com grandes variações na testabilidade.

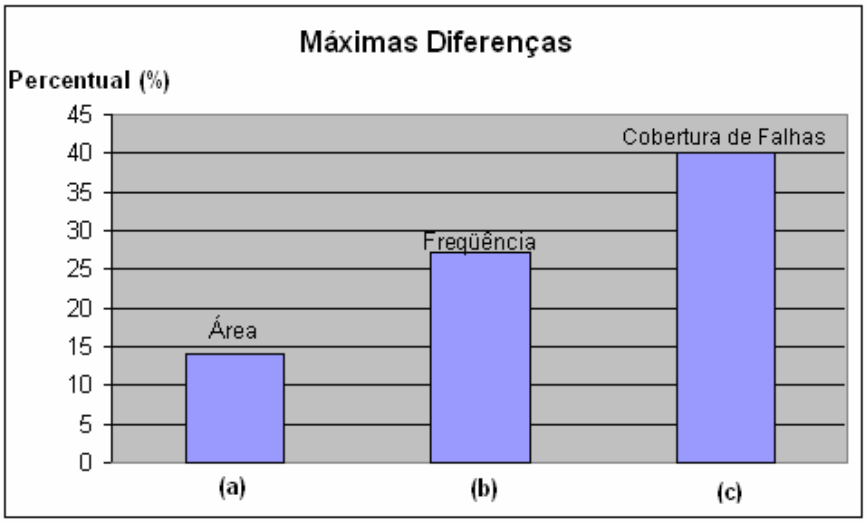

Figura 3.8: Máximas diferenças em relação à área, freqüência e testabilidade entre as versões estruturais.

Para um mesmo número de vetores, que implica um mesmo custo de teste, podemos notar que outras características de testabilidade possuem diferenças. Note por exemplo, que o percentual de falhas não observáveis na versão E3 (14%) é bem menor que nas versões E1 (31%), E2 (29%), e E4 (31%), como mostra a Figura 3.9. Isso mostra que a versão estrutural E3 possui uma característica no seu código que produz um circuito com falhas mais fáceis de serem observáveis.

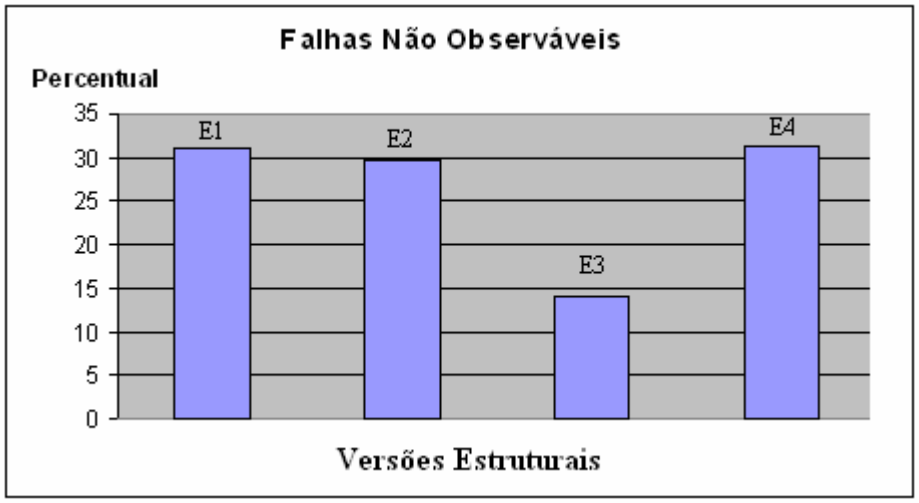

Figura 3.9: Percentual de falhas não observáveis nas versões estruturais.

Outra característica que possui variação entre os códigos é o número de falhas não controláveis, como mostra a Figura 3.10. A versão estrutural E2 possui 4% de falhas desta classe, enquanto a versão E1 chega a possuir 11,8%. A versão E2 possui quase três vezes menos falhas não controláveis que a versão E1, mostrando que o código da versão E2 possui alguma característica que faz o circuito gerado possuir esta vantagem.

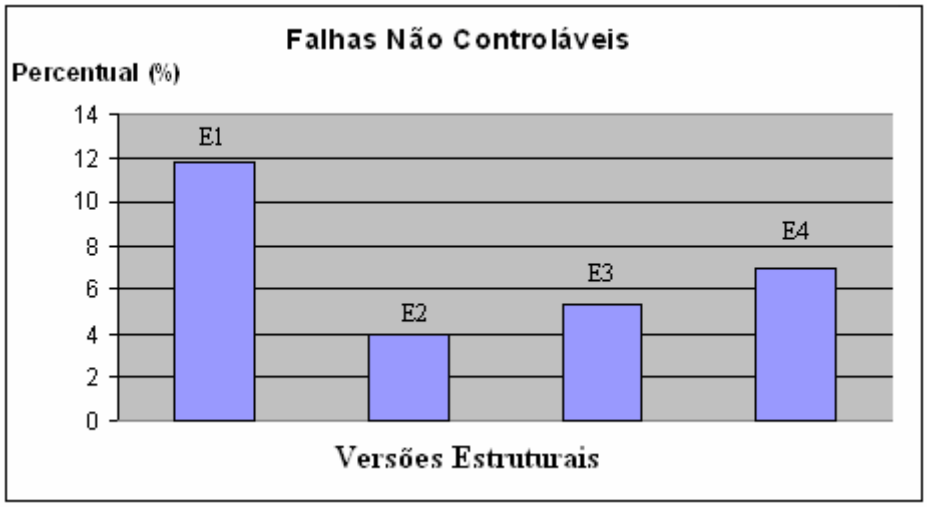

Figura 3.10: Percentual de falhas não controláveis nas versões estruturais.

Comparando-se as duas versões comportamentais, versão C1 e C2, nota-se também uma enorme diferença na área (55%), cobertura de falhas (51%) e freqüência de operação (28%).

Observe na Tabela 3.2 que a porcentagem de falhas não observáveis da versão C1 (46%) é muito maior que na versão C2 (16%). Isso mostra que alguma característica da primeira versão gera dificuldade a observabilidade das falhas do circuito.

Comparando descrições comportamentais e estruturais, a mesma diversidade de soluções de teste pode ser observada. O trabalho está focado no estudo das razões destas diferenças e o que é necessário realizar para contornar estes problemas, e conseguir um projeto com maior testabilidade.

Este estudo de caso inicial mostrou que para um mesmo algoritmo ou problema, existem diversas maneiras de descrevê-lo em uma linguagem de descrição de hardware. A utilização de diferentes estilos de descrição (estrutural e comportamental) traz conseqüências nas características finais do circuito gerado. Nota-se na Tabela 3.1, que se forem consideradas somente a área ocupada e freqüência de operação como restrições de projeto, as descrições estruturais possuem melhores resultados que as descrições comportamentais. Como os projetistas normalmente buscam soluções para o gargalo do projeto, a versão estrutural E4 seria escolhida como o melhor compromisso entre área ocupada e freqüência de operação. Entretanto, nota-se na Tabela 3.2 que considerando as características de testabilidade, a versão E4 não é a melhor escolha. A versão comportamental C2 possui a maior cobertura de falhas para um mesmo número de vetores de teste (um importante custo para o teste) e seria escolhida se a restrição do projeto fosse testabilidade.

3.4 Regras Clássicas de Teste

Procurando na literatura específica, algumas regras clássicas de projeto foram estudadas e suas viabilidades de emprego foram levantadas. Algumas regras são utilizadas por projetistas experientes que notam problemas em áreas de esquemáticos lógicos, mas suas aplicações em códigos VHDL não são possíveis devido ao nível de abstração. Um exemplo disso seria utilizar a regra de não permitir portas com *fan-in* elevado. Não é possível utilizar esta regra em um alto nível de abstração, porque não se tem controle ao nível de transistores, pois o responsável pela conversão da descrição VHDL em transistores é a maquina de síntese, ficando mascarado ao projetista que descreve na linguagem HDL.

Entretanto, outras regras clássicas de teste podem ser aplicadas a códigos VHDL, tais como a construção de registradores inicializáveis e a inclusão de controle para sinais difíceis de controlar (BUSHNELL, 2000). A construção de registradores com *reset* facilita a controlabilidade por entradas primárias e, conseqüentemente, melhora a utilização do ATPG. A inserção de pontos para controlar sinais difíceis de controlar pode ser utilizada em descrições de alto nível. Porém, é necessário saber onde inserir estes pontos para modificar o código. Alguns trabalhos desenvolvidos utilizam informações a partir de códigos VHDL. Chickermane (CHICKERMANE, 1994) tem como objetivo em seu trabalho procurar pontos difíceis de testar nos códigos VHDL, enquanto Hsu (HSU, 1998) insere pinos de controle em descrições VHDL para aumentar a testabilidade. Esta dissertação não aborda o estudo de pontos difíceis de testar, mas descobrir uma maneira de descrever um código que produza um circuito final mais testável.

O estudo de caso anterior mostrou que existe uma grande variabilidade nos resultados de testabilidade mesmo para circuitos que desempenham a mesma função lógica (cálculo da raiz quadrada). A partir do estudo anterior procurou-se descobrir uma forma de descrever os códigos para facilitar a testabilidade. Em um primeiro momento, comparando os códigos das descrições, buscou-se descobrir as diferenças entre os mesmos e foi descoberto que algumas descrições possuíam elementos de memória não necessários para o correto funcionamento, ou seja, possuíam registradores com bits desnecessários. Alguns códigos para o cálculo da raiz quadrada (número com tamanho de 8 bits) possuíam os registradores *r* e *d* com tamanhos maiores que os 4 bits e 5 bits, respectivamente necessários para realizar o cálculo com entrada de tamando de 8 bits.

Todos os códigos foram revisados e modificados quando necessário para possuir somente o tamanho de memória necessário. Além disso, todos os códigos possuem os registradores inicializáveis por um sinal externo. Se o pino de entrada *reset* for acionado, os registradores são inicializados para um valor inicial zero. A Tabela 3.3 mostra com símbolo √ os códigos que foram modificados. Por exemplo, as descrições E1, E2, E4 e C2 tiveram seus registradores modificados enquanto as descrições E1, E4 e C2 tiveram a inserção de um reset que, quando acionado, zera os registradores. As descrições não marcadas com o símbolo √ não necessitaram modificações.

Tabela 3.3: Modificações com regras clássicas nas descrições.

Modificações	E1	E2	E3	E4	C1	C2
Registradores	√	√		√		√
Reset	√			√		√

A seguir, serão apresentados os resultados de síntese e testabilidade dos circuitos gerados a partir das modificações mostradas na Tabela 3.3.

3.4.1 Comparações entre os circuitos com e sem utilização das Regras Clássicas.

A Tabela 3.4 apresenta os resultados de síntese e testabilidade para os seis códigos com as modificações explanadas anteriormente, ou seja, com *reset* que inicializa os registradores e com os elementos de memória (registradores) com tamanhos padronizados para possuírem o valor necessário para o correto funcionamento.

Tabela 3.4: Resultados de síntese e testabilidade dos circuitos com regras clássicas de teste.

	E1	E2	E3	E4	C1	C2
Número de Portas	303	316	315	293	427	263
Freqüência de Operação (MHz)	270.5	264.8	226.2	290.3	166.2	216.9
Flip-Flops	31	30	26	30	23	23
Total de falhas	950	1102	1070	893	1836	896
Não Testáveis pelo ATPG (%)	7,2	2,5	0,7	3,8	0,6	1,5
Não Observáveis (%)	12,8	19,8	9,5	26,7	39,9	22,4
Não Controláveis (%)	1,8	0	0,7	1	0	1,3
Cobertura de Falhas (%)	78,2	77,7	89,1	68,5	59.5	74.8
N° de vetores Gerados	277	759	359	171	232	87

Os resultados foram obtidos deixando a ferramenta Flextest rodar o ATPG com as configurações padrão, produzindo um número livre de vetores de teste. Os resultados de síntese e testabilidade dos circuitos se modificaram para as descrições que possuíam registradores com tamanho maior que o necessário (códigos E1, E2, E4 e C2) e para as descrições que necessitaram de um reset que zera os registradores.

A Figura 3.11 apresenta as comparações entre as áreas dos circuitos utilizando as regras clássicas (colunas cinza escuro) e as versões originais apresentadas anteriormente (colunas cinza claro). A área ocupada pelos circuitos E1, E2 e E4 diminuiu, como esperado, em relação às áreas ocupadas pelos circuitos utilizados na seção anterior. A inserção de um *reset* que zera todos registradores altera a área também, mas não acarreta em um aumento muito significativo. As versões já possuíam reset, mas nem todas zeravam os registradores e sim colocavam em um estado inicial do algoritmo (r = 1, d = 2, s = 4), e a inicialização dos registradores com zero implicou um aumento de área. As versões E3 e C1 não necessitaram alterações no tamanho de seus registradores. A versão estrutural E3 e comportamental C1 não modificaram sua área, pois já possuíam o *reset* que zerava os registradores e os tamanhos dos registradores corretos, enquanto a versão comportamental C2 necessitou a regra clássica do reset e a redução de apenas 1 bit do tamanho de um registrador.

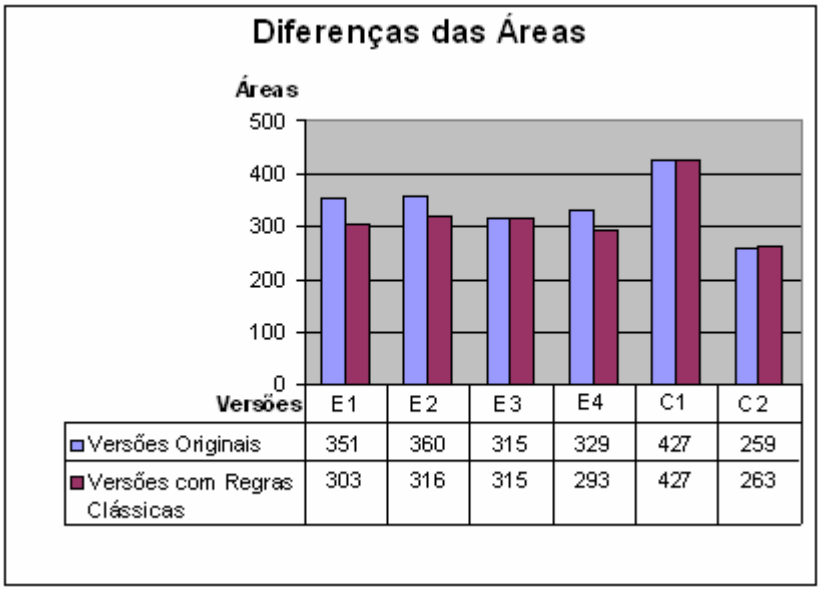

Figura 3.11: Diferenças de áreas entre os circuitos com e sem regras clássicas de teste.

A utilização da regra de inicializar os registradores e usar apenas o tamanho de memória necessário aumentou as freqüências de operação dos circuitos, como mostra a Figura 3.12. Isto mostra que, ao diminuir o tamanho de registradores, (e conseqüentemente dos somadores necessário para realizar as operações do algoritmo), torna-se menor o caminho crítico do circuito e, portanto, aumenta a freqüência de operação do circuito. O aumento da freqüência de operação ocorreu somente devido à redução do tamanho dos registradores e não por causa da modificação para inicialização dos registradores com zero, pois a alteração para inicializar (*reset*) os registradores não é o estado que consome mais tempo na execução da aplicação e, portanto não influencia a freqüência.

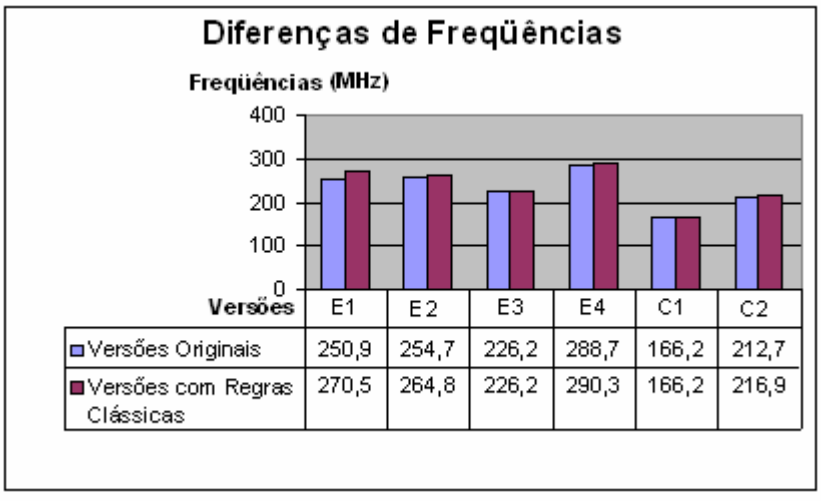

Figura 3.12: Diferenças de freqüência entre os circuitos com e sem regras clássicas de teste.

Para a comparação das características de testabilidade entre as versões com e sem a utilização das regras clássicas de teste ser realizada, procurou-se utilizar o mesmo

número de padrões de teste gerado. A Tabela 3.5 mostra os resultados de testabilidade para no máximo 136 padrões de teste. Contudo, duas descrições (E2 e C2) não alcançaram o número de vetores desejados. As restrições do ATPG (número de *backtrackings*, número máximo de ciclos e limite de tempo) não permitiram alcançar o número de 136 vetores para estes dois circuitos.

Tabela 3.5: Resultados de testabilidade dos circuitos com regras clássicas de teste com número máximo de padrões igual a 136.

	E1	E2	E3	E4	C1	C2
Total de falhas	950	1102	1070	893	1836	896
Não Testáveis pelo ATPG (%)	6,5	2,5	0,5	3	0,05	0,8
Não Observáveis (%)	21,6	32,9	14,1	27,1	45,95	23,1
Não Controláveis (%)	1,8	0,5	5,3	1	0	1,3
Cobertura de Falhas (%)	70,1	64	80,1	68,9	54	74,8
N° de vetores Gerados	136	133	136	136	136	87

A Figura 3.13 mostra um gráfico com as diferenças de cobertura de falhas entre os circuitos com e sem utilização das regras clássicas. Note que dois circuitos (E3 e C1) não necessitaram modificações e, portanto, não tiveram seus resultados alterados. Entre as quatro outras aplicações restantes, duas delas (E1 e E4) aumentaram sua cobertura de falhas, mostrando que projetar utilizando estas regras clássicas melhora a cobertura de falhas. Além disso, o circuito E2 teve a mesma cobertura de falhas, mas necessitou três vetores de teste a menos que a versão E2 sem as modificações. A comparação entre as coberturas de falhas das versões C2 não são muito justas devido ao número de vetores de teste serem diferentes. Embora a redução de cobertura tenha passado de 81,6% para 74,8%, o decremento de vetores foi bem maior, de 136 para 87 vetores de teste.

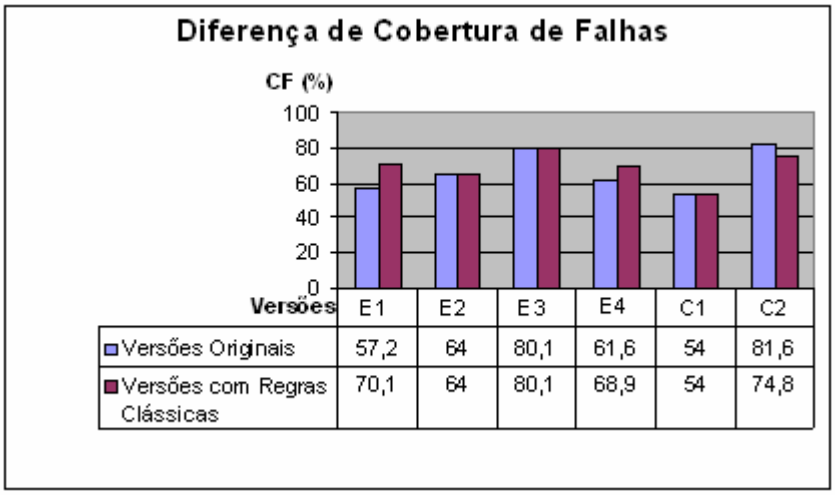

Figura 3.13: Diferenças de cobertura de falhas entre os circuitos com e sem regras clássicas.

A utilização das regras clássicas de teste melhora as características de testabilidade, além de melhorar as características de síntese. A cobertura de falhas de dois circuitos foi aumentada (E1 e E4) e E2 necessitou menos vetores de teste para alcançar a cobertura

de teste da versão original. Outros dois circuitos (E3 e C1) já utilizavam as regras e não tiveram seus resultados de testabilidade modificados. Além disso, a versão C2 diminuiu a cobertura de falhas com a utilização das regras clássicas de teste, mas a comparação não é muito justa devido ao fato do número de vetores de teste ser muito diferente (136 e 87).

A utilização das regras clássicas de teste comprovou a melhoria nas características de testabilidade, mas como visto nos resultados não é alcançada uma alta cobertura de falhas dos circutos. Visando melhorar as características de testabilidade dos circuitos, iremos modificar as descrições VHDL para produzir circuitos mais testáveis. O próximo capítulo utiliza diferenças entre os códigos dos circuitos para modificar os códigos visando a produção de circuitos mais testáveis.

4 MODIFICAÇÕES VISANDO AUMENTO DE TESTABILIDADE

As comparações no capítulo anterior foram realizadas entre as versões utilizando as regras clássicas de teste e as versões originais. A partir de agora, começaremos a mapear as características e diferenças entre as descrições VHDL. Após, modificações nos códigos são implementadas buscando melhorar a testabilidade do circuito gerado. O objetivo das modificações nos códigos é mapear características de descrição de um código VHDL que produzam, ao final da síntese, um circuito com melhores características de testabilidade.

Para uma comparação entre as versões, o ATPG foi configurado para produzir 87 vetores de teste para cada circuito, pois é o menor número de vetores gerados por uma das descrições (versão C2 na Tabela 3.4). A Tabela 4.1 mostra os resultados de testabilidade dos circuitos que utilizam regras clássicas de teste. Note que as descrições E2 e E3 não alcançaram 87 vetores, mas isso não impede que façamos comparações entre todas versões para levantar hipóteses para tentar explicar as diferenças de testabilidade.

Tabela 4.1: Resultados de testabilidade dos circuitos com regras clássicas de teste com número máximo de padrões igual a 87.

	E1	E2	E3	E4	C1	C2
Total de falhas	950	1102	1070	893	1836	896
Não Testáveis pelo ATPG (%)	0	2,5	0,5	3,8	0	0,8
Não Observáveis (%)	51,7	60,2	16,9	37,3	51,1	23,1
Não Controláveis (%)	4	2,4	7,4	1,7	0	1,3
Cobertura de Falhas (%)	44,3	34,9	75,2	57,2	48.9	74,8
N° de vetores Gerados	87	83	83	87	87	87

Podemos notar que os circuitos E3 e C2 possuem as maiores coberturas de falhas. O passo seguinte é descobrir quais características do código geram um circuito mais testável e utilizar este conhecimento para construir códigos mais testáveis. Algumas diferenças entre os códigos foram mapeadas para tentar descobrir quais características melhoram, ou não, a testabilidade. Estas características são algumas hipóteses levantadas para explicar as diferenças de testabilidade e serão explanadas a seguir.

4.1.1 Mapeamento de características que influenciam o circuito gerado.

A partir da Tabela 3.4, da Tabela 3.5 e da Tabela 4.1, algumas características que podem influenciar o circuito gerado foram mapeadas nos códigos VHDL:

- a versão estrutural E3 possui a maior cobertura de falhas entre todas versões. Como característica que a diferencia das outras, podemos citar que E3 possui menos estados na parte de controle.
- a versão estrutural E2 possui uma descrição onde a comunicação entre PO e PC é realizada por código micro-programado. Isto pode explicar o porque deste circuito possuir menos falhas não controláveis (Tabela 3.4).
- a versão estrutural E3 possui o menor número de falhas não observáveis entre as versões. Isto pode ter ocorrido porque esta descrição possui menos estados, o que facilita a geração de vetores de teste.
- a versão comportamental C1 possui uma construção estrutural em seu código. Isso pode explicar o fato deste circuito possuir um percentual de falhas não observáveis quase três vezes maior que a versão C2.

A partir das suposições acima, na Seção 4.2 serão estudadas as implicações das características das descrições nos circuitos gerados.

4.2 Modificações nas Descrições

Como visto anteriormente, as descrições comportamentais possuem vantagens e desvantagens em relação aos circuitos gerados por descrições estruturais. Como desvantagem nota-se que as freqüências de operação são menores. Em contrapartida, as áreas ocupadas são menores que as ocupadas pelos circuitos gerados pelas descrições estruturais. Outra vantagem que se concluiu é que, para um mesmo número de vetores de teste, as descrições comportamentais possuem características de testabilidade melhores.

Atualmente existem mais projetos estruturais sendo reusados como núcleos IP (Propriedade Intelectual) do que projetos comportamentais. Pode-se imaginar que os projetistas estão mais interessados em aumentar a testabilidade destes núcleos estruturais para facilitar o teste do sistema onde estes núcleos serão utilizados.

Desta forma, nesta dissertação serão modificadas as descrições estruturais do algoritmo incluindo nelas características das descrições comportamentais e com isso melhorar as características de testabilidade do circuito final.

A seguir serão apresentadas as modificações realizadas nos códigos VHDL. Algumas diferenças entre os códigos foram mapeadas na seção anterior e servirão como hipóteses para as modificações, pois características das descrições comportamentais serão utilizadas nas modificações das três descrições estruturais (E1, E2, E4) para aumentar a cobertura de falhas. Estas três descrições foram escolhidas por possuírem as menores coberturas de falhas entre as descrições estruturais, como mostrado na Tabela 4.1.

Existem duas diferenças básicas entre as descrições estruturais (E1, E2, E4) e as descrições comportamentais: enquanto as estruturais utilizam uma estrutura para a parte operativa e outra estrutura para parte de controle, C2 define parte operativa e parte de

controle juntas em uma mesma estrutura. A segunda diferença é que as descrições comportamentais não possuem uma máquina de estados explícita e o seu código é implementado em forma de um algoritmo, enquanto as descrições arquiteturais possuem uma máquina de estados explícita implementada pelo projetista. A partir destas diferenças, foram realizadas três modificações independentes nas descrições estruturais:

- Modificação M1 - definir a parte de controle e parte operativa juntas em uma mesma estrutura na descrição original.
- Modificação M2 - construir a parte de controle com menor número de estados na máquina de controle, deixando a parte de controle e parte operativa separadas.
- Modificação M3 - construir a parte operativa e a parte de controle juntas com a máquina de controle possuindo menor número de estados.

Note que as modificações inseridas não mudam a característica de código estrutural, pois seu comportamento continua sendo ciclo-a-ciclo.

Os resultados de síntese e testabilidade para as descrições modificadas foram obtidos para uma síntese para ASIC utilizando a tecnologia 0.35um da biblioteca ADK e configurando a ferramenta Leonardo Spectrum™ para manter a hierarquia dos blocos da descrição e tendo como objetivo obter uma menor área do circuito. Os resultados de teste foram obtidos com a ferramenta Flextest ™ configurada para rodar o ATPG até o número máximo de 87 vetores de teste gerados. Todas as descrições modificadas podem ser encontradas no Anexo ao final desta dissertação.

4.2.1 Modificações na descrição Estrutural E1

Primeiramente, vamos modificar a descrição estrutural E1 inserindo características das versões comportamentais. A Tabela 4.2 mostra os resultados de síntese e testabilidade para as três modificações na versão estrutural E1. Na tabela, a segunda coluna apresenta os resultados para a versão E1 original sem modificação. A terceira coluna é a descrição original com a modificação M1. A quarta coluna é a modificação M2, com o número de estados reduzido de seis para quatro estados. Finalmente, na quinta coluna é mostrada a versão E1 com a modificação M3.

Tabela 4.2: Resultados de síntese e testabilidade das modificações em E1.

	E1	E1_M1	E1_M2	E1_M3
Número de Portas	303	312	310	302
Freqüência de Operação (MHz)	270.5	255.8	214.7	232
Flip-Flops	31	30	26	25
Total de falhas	950	1012	1063	1055
Não Testáveis pelo ATPG (%)	0	0	1,1	0
Não Observáveis (%)	51,7	45	31,6	17,5
Não Controláveis (%)	4	3,5	2,9	0,7
Cobertura de Falhas (%)	44,3	51,5	64,4	81,8
N° de vetores Gerados	87	87	87	87

E1_M1 – Versão E1 com PC e PO juntas.
E1_M2 – Versão E1 com 4 estados na máquina de estados.
E1_M3 – Versão E1 com PC e PO juntas e 4 estados.

Com a primeira modificação (E1_M1), nota-se que a área aumentou de 303 para 312 portas lógicas e a freqüência de operação diminuiu como mostrou a Tabela 4.2. A modificação simplifica a comunicação entre a parte de controle e a parte operativa. A comunicação é a maior dificuldade do teste, pois para acessar um ponto da PO é necessário interferir na PC. Pode-se observar que todas as medidas de testabilidade do circuito foram melhoradas com a modificação. Além da cobertura de falhas que foi elevada, o número de falhas não controláveis e número de falhas não observáveis foram reduzidas quando comparadas com a descrição E1 original.

A segunda modificação (E1_M2) aumentou a área de 303 portas na versão original para 310 portas com a modificação no código. A freqüência de operação sofreu uma redução em relação à versão original, visto que a descrição possui menos estados para realizar o mesmo número de operações, e conseqüentemente foi necessário re-escalonar as atividades colocando mais tarefas em um mesmo estado o que diminui a freqüência de operação. A modificação propiciou uma redução no número de falhas não controláveis e não observáveis como esperado, pois facilita o processo de geração de vetores. Por exemplo, para gerar um vetor para testar uma falha em um ponto específico, pode-se necessitar executar várias iterações na máquina de controle. Logo, com menor múmero de estados é mais fácil controlar as iterações e alcançar o ponto necessário, facilitando a justificação da falha e a controlabilidade do circuito. Da mesma forma, a implicação da falha para a saída é facilitada, melhorando a observabilidade do circuito.

Finalmente, a terceira modificação (E1_M3) gerou um circuito com a área um pouco menor e a freqüência de operação também menor, visto que a descrição possui menor número de estados para realizar o mesmo número de operações, e com isso a profundidade lógica aumentou devido à necessidade de realizar mais operações nos estados. Por outro lado, a modifiação trouxe uma grande melhoria nas características de testabilidade do circuito resultante, quase dobrando a cobertura de falhas. A versão E1_M3 alcançou uma cobertura de falhas superior a todas outras descrições apresentadas na Tabela 4.1. Note na Tabela 4.1 que as versões E3 (75,25%) e C2 (74,8%) eram as descrições que possuíam as maiores coberturas de falhas e a versão E1 com a terceira modificação alcançou 81,8% de cobertura de falhas.

A Figura 4.1 mostra a evolução do percentual de falhas não observáveis e a Figura 4.2 mostra a evolução do percentual de falhas não controláveis conforme foram aplicadas as modificações na descrição estrutural E1. Note que tanto o número de falhas não controláveis como não observáveis foram diminuindo. A modificação de juntar PC e PO melhora as medidas de testabilidade, mas diminuir o número de estados melhora ainda mais tais medidas. Juntando as duas modificações na mesma descrição obtém-se os melhores resultados.

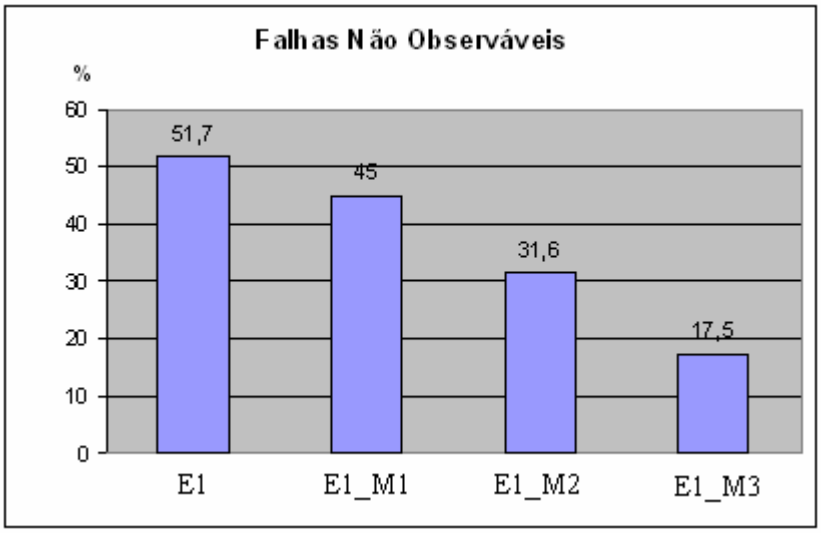

Figura 4.1: Percentual de falhas não observáveis nas descrições E1 modificadas.

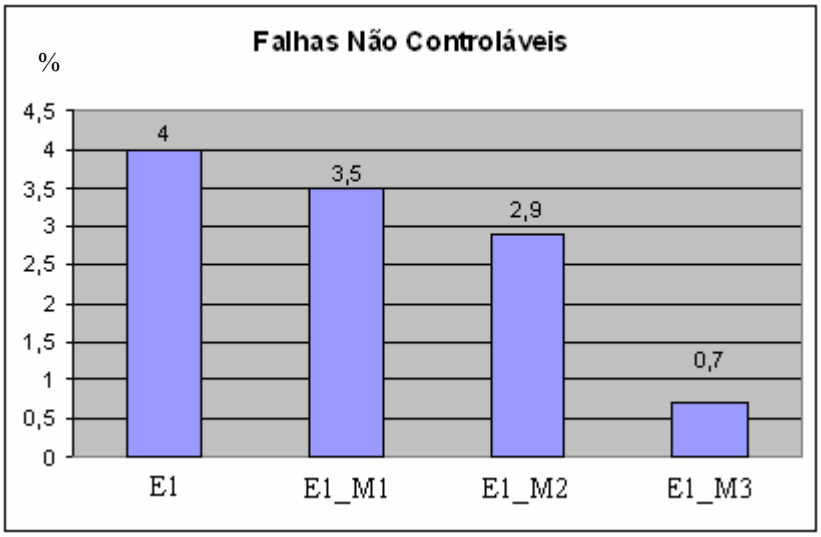

Figura 4.2: Percentual de falhas não controláveis nas descrições E1 modificadas.

A Figura 4.3 mostra as coberturas de falhas da versão original e das versões modificadas de E1.

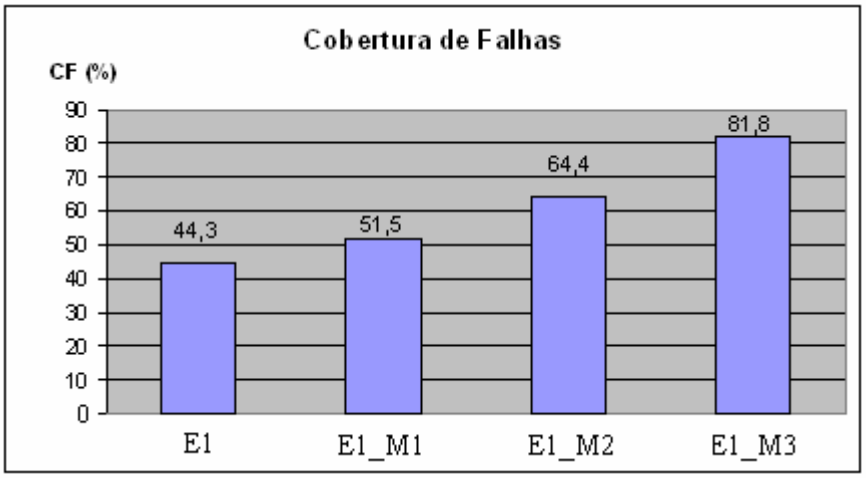

Figura 4.3: Percentual da cobertura de falhas nas descrições E1 modificadas.

4.2.2 Modificações na descrição Estrutural E2

As modificações M1 e M3 não se aplicam a descrição E2, pois a comunicação entre a parte de controle e a parte operativa é feita por código micro-programado e não pelo estado da máquina de controle, como em E1.

A modificação M2 foi implementada na descrição E2. O número de estados da máquina de controle foi reduzido de seis para quatro estados. A tabela 4.3 mostra os resultados de síntese e testabilidade para esta modificação.

Tabela 4.3: Resultados de síntese e testabilidade da modificação em E2.

	E2	E2_M2
Número de Portas	316	229
Freqüência de Operação (MHz)	264.8	213.3
Flip-Flops	30	24
Total de falhas	1102	948
Não Testáveis pelo ATPG (%)	2,5	0,1
Não Observáveis (%)	60,2	27,8
Não Controláveis (%)	2,4	3,1
Cobertura de Falhas (%)	34,9	69
N° de vetores Gerados	83	87

E2_M2 – Versão E2 com 4 estados na máquina de estados.

Utilizando a segunda modificação (E2_M2), a área do circuito original é 38% maior que a área gerada pelo circuito com a modificação, devido à nova máquina de estados possuir menos estados. Por outro lado, a freqüência de operação diminuiu 19%, devido à necessidade de mais tarefas terem de ser realizadas no mesmo estado,. Note que o número de falhas não controláveis aumentou menos de um ponto percentual. Mas, em contrapartida, o número de falhas não observáveis diminuiu mais de 32 pontos percentuais (de 60,2% para 27,8%).

4.2.3 Modificações na descrição Estrutural E4

A Tabela 4.4 mostra os resultados de síntese e testabilidade para as três modificações na versão estrutural E4 deixando a ferramenta executar sem restrições de número de vetores de teste. Na tabela, a segunda coluna apresenta os resultados para a versão E4 original sem modificação. A terceira coluna é a descrição original com a modificação de parte de controle e parte operativa juntas. A quarta coluna é a modificação de diminuir o número de estados de seis para três. Finalmente, na quinta coluna é mostrada a versão E4 com a parte de controle e parte operativa juntas com três estados na máquina de controle.

Tabela 4.4: Resultados de síntese e testabilidade das modificações em E4.

	E4	E4_M1	E4_M2	E4_M3
Número de Portas	293	254	237	236
Freqüência de Operação (MHz)	290.3	215,4	174,2	174.2
Flip-Flops	30	30	24	24
Total de falhas	893	984	997	991
Não Testáveis pelo ATPG (%)	3,8	0	0,3	0
Não Observáveis (%)	37,3	37,4	31,2	26,9
Não Controláveis (%)	1,7	3,2	3,4	2,4
Cobertura de Falhas (%)	57,2	59,4	65,1	70,7
N° de vetores Gerados	87	87	87	87

E4_M1 – Versão E4 com PC e PO juntas.
E4_M2 – Versão E4 com 3 estados na máquina de estados.
E4_M3 – Versão E4 com PC e PO juntas e 3 estados.

Com a primeira modificação (E4_M1), a área ocupada pelo circuito diminuiu, mas a freqüência de operação também. Pode-se observar que o percentual de falhas não controláveis e não observáveis ficaram um pouco maiores. Contudo, o circuito produzido pela modificação não produziu nenhuma falha da classe das não testáveis pelo ATPG e a cobertura de falhas do circuito final foi incrementada de 57,2% para 59,4%.

Com a segunda modificação (E4_M2), a área sofreu um decremento de 19,2%, devido à nova máquina de estados possuir menos estados. Contudo, a freqüência de operação sofreu uma redução em relação à versão original, devido a mais tarefas serem realizadas em cada estado. Com a modificação foi reduzido o número de falhas não observáveis, como esperado. A cobertura de falhas aumentou de 57,2% (E4) para 65,1% com a modificação.

Finalmente, a terceira modificação (E4_M3) proporcionou novamente uma grande melhoria nas características de testabilidade do circuito resultante, aumentando a cobertura de falhas para 70,7%. A área consumida foi menor e a freqüência de operação também diminuiu. Note na Tabela 4.4 que a cobertura de falhas da versão com as modificações é 23% maior que a versão original utilizando 87 vetores.

4.2.4 Comparações entre os resultados das modificações.

A Tabela 4.5 apresenta alguns resultados de testabilidade para as três descrições estruturais originais e modificadas. Resultados como cobertura de falhas e número de vetores de teste gerados são mostrados. A ferramenta de teste executou o ATPG com as configurações padrões com o objetivo de observar a máxima cobertura de falhas alcançada. As versões modificadas das descrições E1 e E4, mostradas na tabela, são as versões com a terceira modificação. A versão modificada de E2 é a versão com menos estados em relação à versão original (segunda modificação).

Note que nos três casos a cobertura de falhas aumentou bastante, mas em contrapartida o número de vetores de teste necessários para alcançar esta cobertura de falhas elevada foi muito maior. Por outro lado, para o mesmo custo de teste (87 vetores), as versões modificadas também apresentaram melhores resultados de teste.

Tabela 4.5: Comparação entre os resultados das modificações.

Versões	Original		Modificada	
	CF (%)	VT	CF (%)	VT
E1	78,2	277	94,1	1397
E2	77,7	759	93,3	1933
E4	68,5	171	92,9	1553

CF – Cobertura de Falhas
VT – número de Vetores de Teste gerados

4.3 Diferentes formas de síntese

Visando mostrar que as modificações são independentes da forma de síntese, as descrições foram sintetizadas de duas formas diferentes. Uma com o objetivo de diminuir a área e outra com o objetivo de aumentar a freqüência de operação do circuito.

As Tabelas 4.6, 4.7 e 4.8 mostram os resultados de síntese e testabilidade para as três descrições estruturais originais e para as modificadas. A ferramenta Flextest executou com a restrição de o número de vetores de teste gerado pela descrição modificada ser igual ao número de vetores gerado pela versão original. Nas tabelas, a segunda coluna apresenta os resultados para as descrições originais e com a ferramenta Leonardo executando com o objetivo de gerar a menor área. A terceira coluna é a descrição modificada sendo sintetizada com o mesmo objetivo. A quarta coluna é a descrição original sendo sintetizada com o objetivo de aumentar a freqüência de operação. Finalmente, na quinta coluna é mostrada a descrição modificada sendo sintetizada com o objetivo de aumentar a freqüência de operação.

A Tabela 4.6 apresenta os resultados para a descrição E1 original e modificada. Note que os resultados de síntese da descrição E1 original foram os esperados, ou seja, em relação à área ocupada, a descrição E1 possuiu menor área (303 portas) quando o objetivo da síntese era este. Quando o objetivo da síntese era possuir uma freqüência de operação superior, isso foi alcançado (306.3MHz).

Por outro lado, a versão com a terceira modificação (E1_M3) alcançou uma menor área ocupada (302 portas) quando era este o objetivo da ferramenta. Contudo, a ferramenta não conseguiu gerar um circuito mais rápido quando este era o seu objetivo.

Note ainda que houve melhorias em relação à cobertura de falhas em ambos os tipos de síntese. Com a ferramenta de síntese otimizando área, as modificações realizadas na descrição E1 obtiveram uma cobertura de falhas maior que a versão original (78,2%), alcançando 89,4%. Quando a síntese objetivava freqüência, com as modificações, a cobertura de falhas obtida foi de 80% contra os 73,1% da versão E1 original.

Tabela 4.6: Diferentes formas de sintese na descrição E1 original e E1 modificada.

	Objetivo de Área		Objetivo de Freqüência	
	E1	E1_M3	E1	E1_M3
Número de Portas	303	302	330	326
Freqüência de Operação (MHz)	270.5	232	306.3	230.2
Flip-Flops	31	25	31	25
Total de falhas	950	1055	1145	1200
Não Testáveis pelo ATPG (%)	7,2	0,8	6,6	1,3
Não Observáveis (%)	12,8	9,2	17,1	18
Não Controláveis (%)	1,8	0,6	3,2	0,7
Cobertura de Falhas (%)	78,2	89.4	73.1	80
N° de vetores Gerados	277	277	176	176

E1 – Descrição Estrutural E1 original.
E1_M3 – Descrição E1 com PC e PO juntas e 4 estados.

A Tabela 4.7 apresenta os resultados das duas formas de síntese para as descrições E2 original e modificada. Os resultados de síntese da descrição E2 original e modificada foram os esperados. Note que novamente houve melhorias em relação à cobertura de falhas em ambas as formas de síntese. Com a ferramenta de síntese objetivando área, a modificação realizada na descrição E2 obteve uma cobertura de falhas maior que a versão original (77,7%), alcançando 91,9% de cobertura de falhas com a modificação. E quando a síntese objetivava freqüência, a modificação gerou um circuito com cobertura de falhas de 93,6% contra os 62,4% de cobertura de falhas obtido com a versão E2 original.

Tabela 4.7: Diferentes formas de sintese na descrição E2 original e E2 modificada.

	Objetivo de Área		Objetivo de Freqüência	
	E2	E2_M1	E2	E2_M1
Número de Portas	316	229	355	252
Freqüência de Operação (MHz)	264.8	213.3	299.4	291.3
Flip-Flops	30	24	30	24
Total de falhas	1102	948	1360	1106
Não Testáveis pelo ATPG (%)	2,5	0,5	2,3	3,2
Não Observáveis (%)	19,8	6,1	35,3	3,2
Não Controláveis (%)	0	1,5	0	0
Cobertura de Falhas (%)	77,7	91,9	62,4	93,6
N° de vetores Gerados	759	754	1783	1798

E2 – Descrição Estrutural E2 original.
E2_M1 – Descrição E2 com 4 estados na máquina de estados.

Da mesma forma, os resultados das duas formas de síntese para as descrições E4 original e modificada foram os esperados, como mostra a Tabela 4.8. A ferramenta otimizando área, alcançou uma área menor (E4 = 293 portas e E4_M3 = 236 portas) em relação à área alcançada pela outra forma de síntese. A ferramenta com o objetivo de elevar a freqüência gerou os melhores resultados em relação à freqüência de operação como esperado (E4 = 351.1 MHz e E4_M3 = 207.9 MHz).

Cabe salientar que novamente houve melhorias em relação à cobertura de falhas em ambas as formas de síntese. Com a ferramenta de síntese objetivando área, as modificações realizadas na descrição E4 obtiveram uma cobertura de falhas maior que a versão original (68,5%), alcançando 81,5% de cobertura. Com a ferramenta objetivando freqüência, as modificações geraram um circuito com cobertura de falhas de 92,4% contra os 82,7% da versão E4 original.

Tabela 4.8: Diferentes formas de síntese na descrição E4 original e E4 modificada.

	Objetivo de Área		Objetivo de Freqüência	
	E4	E4_M3	E4	E4_M3
Número de Portas	293	236	309	255
Freqüência de Operação (MHz)	290.3	174.2	351.1	207.9
Flip-Flops	30	24	30	24
Total de falhas	893	991	1070	1129
Não Testáveis pelo ATPG (%)	3,8	0,6	3,3	1,4
Não Observáveis (%)	26,7	16,7	13,4	0,7
Não Controláveis (%)	1	1,2	0,6	5,5
Cobertura de Falhas (%)	68,5	81.5	82,7	92,4
N° de vetores Gerados	171	170	399	395

E4 – Descrição Estrutural E4 original.
E4_M3 – Descrição E4 com PC e PO juntas e 3 estados.

4.4 Inserção de cadeias de varredura

Visando descobrir quais os impactos das modificações nas descrições, quando cadeias de varredura são utilizadas, foram inseridas estas cadeias nas descrições originais e modificadas. Nesta seção, serão chamadas de descrições originais as descrições que utilizam regras clássicas de teste.

4.4.1 Inserção de cadeias de varredura nas descrições originais

Primeiramente, é realizada uma comparação entre os custos da utilização das cadeias de varredura nas descrições originais com as regras clássicas de teste. A Tabela 4.9 mostra os resultados de síntese e testabilidade destas descrições após a inserção de cadeias de varredura. Para inserir as cadeias de varredura foi utilizada a ferramenta Dftadvisor^{TM} da Mentor Graphic (MENTOR, 1999) e para obter os resultados de testabilidade foi utilizada a ferramenta FastScan^{TM} (MENTOR, 1999).

Tabela 4.9: Resultados da inserção de cadeias de varredura nas descrições originais.

	E1	E2	E3	E4	C1	C2
Número de Portas	348	365	363	334	466	308
Freqüência de Operação (MHz)	244.2	246.3	195.5	262	164.6	201.2
Cobertura de Falhas (%)	99.59	98.73	99.92	100	99.95	99.19
N° de vetores Gerados	50	83	62	47	271	50

A partir das Tabelas 3.4 e 4.9, algumas comparações podem ser feitas entre as descrições utilizando cadeias de varredura e as descrições originais com as regras clássicas de teste.

A Figura 4.4 apresenta o percentual de incremento da área ocupada com a utilização das cadeias de varredura nas descrições originais. Em média as áreas ocupadas das descrições aumentaram 14.9% devido à troca dos registradores do circuito por registradores modificados para permitir a ligação das cadeias de varredura.

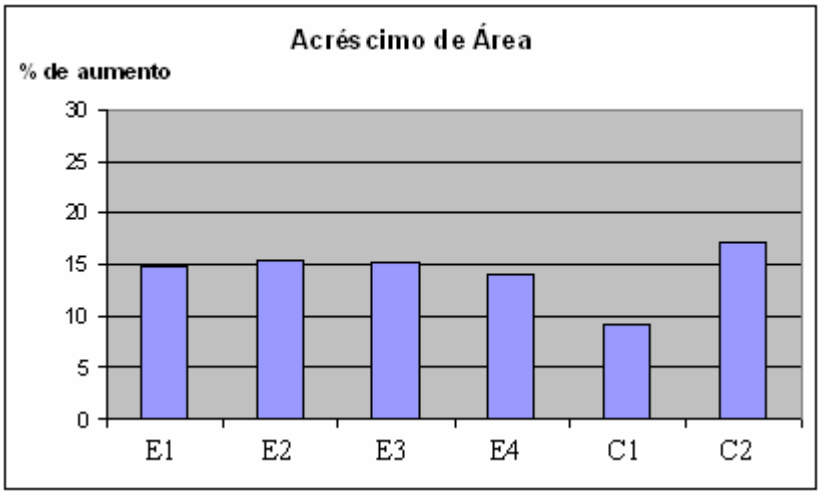

Figura 4.4: Acréscimo de área com a utilização de cadeias de varredura.

A Figura 4.5 mostra o percentual de decréscimo da freqüência de operação das descrições que utilizam as cadeias de varredura. Em média a freqüência das descrições foi reduzida em 8% com a utilização das cadeias. A redução ocorre devido à troca dos registradores por outros que possuem um atraso de propagação maior, diminuindo a freqüência de operação.

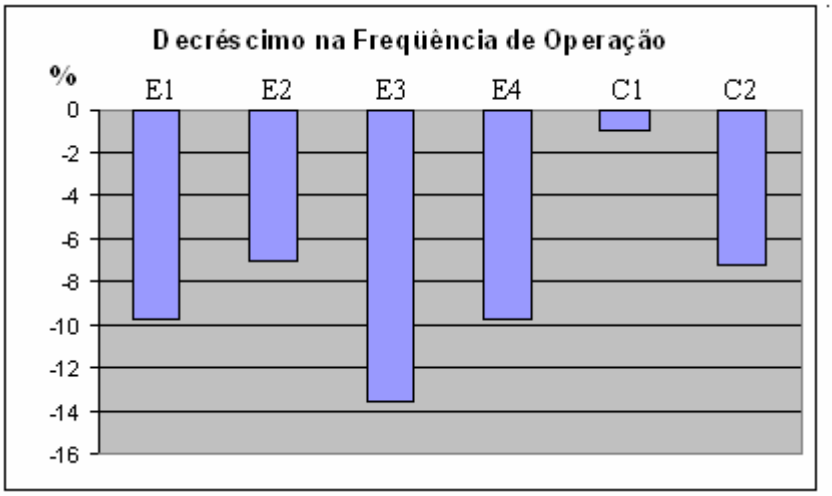

Figura 4.5: Decréscimo de freqüência de operação com a utilização de cadeias de varredura.

A utilização das cadeias de varredura aumentou a cobertura de falhas de todas descrições, porém esta técnica traz algumas desvantagens na sua utilização, como mostrado a seguir:

- É necessário um acréscimo de pinos no circuito, sendo um pino para a entrada de dados da cadeia, um para a saída e outro para controlar.
- A utilização das cadeias produziu um acréscimo de 14,9% em média da área ocupada pelo circuito.
- A freqüência de operação reduziu em média 8% com a utilização desta técnica.
- O tempo necessário para o teste pode ser elevado, devido à inserção serial dos vetores de teste.

4.4.2 Inserção de cadeias de varredura nas descrições modificadas

A seguir são mostrados os resultados de síntese e testabilidade da inserção de cadeias de varredura nas três descrições modificadas (E1_M3, E2_M1, E4_M3). Note na Tabela 4.10 que as freqüências de operação diminuíram em média 11%, as áreas ocupadas aumentaram em média 58% e a cobertura de falhas chegou próximo dos 100%.

Tabela 4.10: Resultados da inserção de cadeias de varredura nas descrições modificadas.

	E1_M3	E2_M1	E4_M3
Número de Portas	341	419	429
Freqüência de Operação (MHz)	208.4	193.3	156.8
Cobertura de Falhas (%)	100	99.91	99.51
N° de vetores Gerados	62	59	54

A Tabela 4.11 apresenta os resultados de síntese e testabilidade para as descrições originais e modificadas, com ambas possuindo cadeias de varredura. Esta comparação é interessante, visto que podemos comparar os reflexos das modificações realizadas nas descrições na utilização das cadeias. Note que a descrição modificada de E1 possui uma área ocupada menor que a área ocupada pela descrição E1 original, enquanto que as outras duas versões modificadas possuem uma área maior que as originais. Por outro lado, as freqüências de operação das descrições modificadas são inferiores as freqüências das descrições originais como esperado, pois as descrições modificadas possuem menos estados na máquina de controle. As coberturas de falhas alcançadas são similares, mas os números de vetores de teste gerados são diferentes. A utilização de cadeias de varredura nas descrições modificadas mostrou que, em relação ao número de vetores de teste, acarretou uma melhoria na minoria dos casos, pois somente a descrição modificada E2_M1 produziu menos vetores que a sua descrição original.

Tabela 4.11: Resultados da inserção de cadeias de varredura nas descrições originais e modificadas.

	E1	E1_M3	E2	E2_M1	E4	E4_M3
Número de Portas	348	341	365	419	334	429
Freqüência de Operação (MHz)	274.1	208.4	246.3	193.3	262	156.8
Cobertura de Falhas (%)	99.59	100	98.73	99.91	100	99.51
N° de vetores Gerados	50	62	83	59	47	54

Pode-se realizar comparações entre as descrições modificadas (sem cadeias de varredura) e as descrições originais contendo cadeias de varredura. Nota-se que as regras propostas geram um circuito com a cobertura de falhas em torno de 94% para as descrições modificadas. A área ocupada dos circuitos gerados com a utilização das regras propostas são em média 38% menores que as geradas pelas descrições originais com cadeias de varredura. A utilização de cadeias de varredura nas descrições originais ainda aumentou em três o número de pinos dos circuitos (entrada, saída e controle de teste). Contudo, as freqüências de operação das descrições utilizando cadeias de varredura são em média 11% maior que as freqüências das descrições modificadas com as regras propostas.

5 CONCLUSÕES

Este trabalho apresentou algumas metodologias utilizadas para aumentar a testabilidade de circuitos integrados. O trabalho foi focado em identificar técnicas de descrição para incrementar a testabilidade de circuitos que são especificados na linguagem VHDL.

O estudo de caso inicial apresentou seis descrições VDHL que especificam o algoritmo da raiz quadrada, expondo que, para um mesmo algoritmo, existem inúmeras formas de descrevê-lo na linguagem VHDL. Este estudo mostrou que existe uma grande variabilidade nos resultados de síntese e testabilidade do circuito resultante utilizando estas diferentes formas de descrição.

Inicialmente foram utilizadas regras clássicas de teste nas descrições VHDL tais como: colocar os registradores em um valor inicial igual a zero e utilizar somente o tamanho necessário dos elementos de memória. Com isso, dados iniciais de testabilidade foram levantados para servir como parâmetros para futuras modificações.

Em seguida, foram levantadas características das descrições VHDL que produziram, ao final da síntese, um circuito com melhores características de testabilidade. As diferenças entre os códigos que descrevem os circuitos foram utilizadas para aumentar a testabilidade de descrições VHDL estruturais. Foram identificadas duas regras para aumentar a testabilidade dos códigos VHDL estruturais. Uma regra propõe reduzir o número de estados da máquina de controle e a outra propõe construir a parte de controle e a parte operativa juntas em uma mesma estrutura.

As três descrições estruturais com menor cobertura de falhas foram modificadas utilizando as duas regras propostas e os resultados obtidos mostraram que as coberturas de falhas foram aumentadas significativamente e as áreas ocupadas diminuíram. Em contrapartida, ocorreu uma redução da freqüência de operação dos circuitos.

Para mostrar que as modificações são independentes da forma de síntese, as descrições modificadas foram sintetizadas de duas formas diferentes. Uma com o objetivo de diminuir a área e outra com o objetivo de aumentar a freqüência de operação do circuito. As três descrições modificadas foram sintetizadas das duas maneiras e em todos casos os circuitos gerados com as descrições modificadas sempre obtiveram melhores coberturas de falhas que as descrições originais.

O trabalho foi realizado para gerar um circuito mais testável sem necessitar inserir cadeias de varredura. Contudo foram realizadas comparações entre as descrições modificadas (com e sem cadeias de varredura) e as descrições originais contendo

cadeias de varredura. Nota-se que a utilização de cadeias de varredura nas descrições modificadas produziu melhorias somente na minoria dos casos, melhorando em relação à área ocupada na descrição E1_M3 e reduzindo o número de vetores de teste na descrição E2_M1.

5.1 Trabalhos Futuros

Embora os resultados obtidos tenham sido satisfatórios, eles se referem a um único circuito de complexidade baixa e para uma única ferramenta de síntese. Desta forma, é importante que as técnicas avaliadas nessa dissertação sejam verificadas para outros circuitos de complexidade mais elevada e com diferentes algoritmos de síntese. Dessa forma, as seguintes atividades constituem os trabalhos imediatos para continuação desta dissertação:

- Descobrir o impacto na síntese e na testabilidade da utilização destas duas regras propostas em projetos mais complexos.
- Avaliar os impactos na energia consumida com a utilização destas modificações.
- Definir regras para tentar reduzir o impacto na freqüência de operação das modificações propostas.
- Avaliar as modificações nas descrições VHDL de circuitos com *pipeline*, *software-pipeline* e combinacionais.

REFERÊNCIAS

ABRAMOVICI, M.; BREUER, M.; FRIEDMAN, A. **Digital Systems Testing and Testable Design**. Piscataway, NJ: IEEE Press, 1990. 652p.

AKERS, S. Binary decision diagrams. **IEEE Transactions on Computers**, [S.l.], v. C-27, n.6, p. 509–516, 1978.

AL-YAMANI, A. **Deterministic Built-In Self Test for Digital Circuits**. 2004. 136 f. Dissertation (Degree Of Doctor Of Philosophy) - Stanford University, San Francisco, USA.

AMORY, A. De M. **Integração de Técnicas de Teste Baseado em Software no Fluxo de Projeto de SOCS**. 2002. 113f. Dissertação (Mestrado em Ciência da Computação) - Pontifícia Universidade Católica do Rio Grande do Sul, PUCRS, Porto Alegre.

BHATTACHARYA, S.; DEY, S. H-SCAN: A High-Level Alternative to Full-Scan Testing With Reduced Area and Test Application Overheads. In: VLSI TEST SYMPOSIUM, VTS, 1996. **Proceedings…** Princeton, NJ, USA: IEEE Computer Society, 1996. p. 74-80.

BOURUCIUS, W.; ROTH, J.; SCHNEIDER, P. Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits. **IEEE Transactions on Computers**, [S.l.], v.16, n.10, p. 567- 580, Oct. 1967.

BUSHNELL, M.; AGRAWAL, V. **Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits.** Boston: Kluwer Academic Publishers, 2000.

CARRO, L. **Projeto e Prototipação de Sistemas Digitais**. Porto Alegre: Ed. da UFRGS, 2001. 176p.

CATTELL, K. et al. One-Dimensional Linear Hybrid Cellular Automata: Their Synthesis, Properties, and Applications in VLSI Testing. **Tutorial Paper.** Disponível em: <http://www.cs.uvic.ca/~mserra/CATumain.pdf >. Acesso em: jan. 2005.

CHEN, C.-I.H.; KARNICK, T.; SAAB, D.G. Structural and behavioral synthesis for testability techniques. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,** [S. l.], v.13, n.6, p. 777-785, June 1994.

CHEN, C.-I.H. Behavioral Test Generation / Fault Simulation. **IEEE Potentials**, [S.l], v.22, n.1, p. 27-32, Feb. 2003.

CHEN, X.; HSIAO, M. Energy-Efficient Logic BIST Based on State Correlation Analysis. In: IEEE VLSI TEST SYMPOSIUM, VTS, 2003. **Proceedings…** Napa Valley, CA, USA: IEEE Computer Society, 2003. p. 267–272.

CHICKERMANE, V.; LEE, J.; PATEL, J.H. Addressing design for testability at the architectural level. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,** [S. l.], v.13, n.7, p.920-934, July 1994.

CORNO, F.; PRINETTO, P.; REORDA, M. Testability analysis and ATPG on Behavioral RT-level VHDL. In: INTERNATIONAL TEST CONFERENCE, ITC, 1997. **Proceedings…** Washington, DC: IEEE Computer Society, 1997. p. 753-759.

CUMANI, G. **High-level Test of Electronic systems**. 2003. 88f. Tese (Doutorado em Engenharia de Computação e Sistemas) – Universidade Politécnica de Torino, Torino, Itália.

DEY, S.; RAGHUNATHAN, A.; ROY, R. Considering Testability During High-Level Design (Embedded Tutorial). In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASP-DAC, 1998. **Proceedings…** Yokohama, Japan: [s.n.], 1998. p. 205-210.

FERRANDI, F.; FUMMI, F.; SCIUTO, D. Implicit Test Generation for Behavioral VHDL Models. In: INTERNATIONAL TEST CONFERENCE, ITC, 1998. **Proceedings…** Washington, DC, USA: IEEE Computer Society, 1998. p. 587-596.

FERRANDI, F.; FUMMI, F.; SCIUTO, D. Test Generation and Testability Alternatives Exploration of Critical Algorithms for Embedded Applications. **IEEE Transactions on Computers,** [S.l.], v. 51, n.2, p. 200–215, Feb. 2002.

FLOTTES, M.L.; HAMMAD, D.; ROUZEYRE, B. High-Level Synthesis for Easy Testability. In: EUROPE DESIGN & TEST CONFERENCE, ED&TC, 1995. **Proceedings…** Paris, France: [s.n.], 1995. p. 198-206.

FUJIWARA, H.; SHIMONO, T. On the acceleration of test generation algorithms. **IEEE Transactions on Computers,** [S.l.], v.32, n.12, p. 1137–1144, Dec. 1983.

GOEL, P. An implicit enumeration algorithm to generate tests for combinational logic circuits. **IEEE Transactions on Computers**, [S.l], v. 30, n.3, p. 215-222, Mar. 1981.

GRÖTKER, T.; Et al. **System Design with SystemC**. [S.l.]: Springer, 2002. 240p.

GU, X.; KUCHINSKI, K.; PENG, Z. Testability Analysis and Improvement from VHDL Behavioral Specifications. In: EUROPEAN DESIGN AUTOMATION CONFERENCE, EDAC, 1994. **Proceedings…** Los Alamitos: IEEE Computer Society, 1994. p. 644-649.

GU, X.; LARSSON, E.; KUCHINSKI, K.; PENG, Z. A controller Testability Analysis and Enhancement Technique. In: EUROPEAN DESIGN AND TEST CONFERENCE, 1997. **Proceedings…** [S.l.:s.n.], 1997. p.153-157.

HOLLAND, J.; CHENG, W.; PATEL, J. PROOFS: A Fast, memory-efficient sequential circuit Fault Simulator. **IEEE Transactions on Computer-Aided Design**, [S.l.], v.11, n.2, p.198–207, Feb. 1992.

HSU, F.; PATEL, J. High-level variable selection for partial-scan implementation. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN, ICCAD, 1998. **Proceedings…**New York: ACM SIGDA, 1998. p. 79-84.

JANSCH-PORTO, I. E. J. **Testabilidade:** Mini curso. Rio de Janeiro: PUC/RJ, 1989. p.115-166. Minicurso apresentado no 3. Simpósio Brasileiro de Computadores Tolerantes a Falhas, 1989.

JERVAN, G. **High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems**. 2002. 112f. PhD Thesis (degree of Licentiate of Engineering) - Department of computer and Information Science Limköpings Universitet, Suécia.

KIRKLANDT, T.; MERCER, M. A topological search algorithm for ATPG. In: IEEE DESIGN AUTOMATION CONFERENCE, DAC, 24., 1987, Miami Beach**. Proceedings…** New York: ACM, 1987. p. 502–508.

LEE, J.; PATEL, J. Testability Analysis Based on Structural and Behavioral Information. In: IEEE VLSI TEST SYMPOSIUM, VTS, 1993. **Proceedings…** [S.l.:s.n.], 1993. p. 139-145.

MENTOR GRAPHICS CORPORATION. **Scan and ATPG process Guide:** Manual Software Version V8.6_4. [S.l.], 1999.

MOORE, G. E. Gramming More Components Onto Integrated Circuits. **Electronic Magazine**, [S.l.], v.38, n.8, p.114-117, Apr. 1965.

NADEAU-DOSTIE, B. **Design for at-Speed Test, Diagnosis and Measurement**. Boston: Kluwer Academic Publishers, 1999. 264p.

NAVABI, Z. **VHDL:** Analysis and Modeling of Digital Systems. 2nd ed. [S.l.]: McGraw-Hill Professional, 1997. 656p.

NEEDHAM, W.M. Nanometer technology challenges for test and test equipment. **Computer**, [S.l.], v.32, n.11, p. 52–57, Nov. 1999.

NICOLICI, N. **Power Minimisation Techniques for Testing Low Power VLSI Circuits.** 2000. 268f. PhD Thesis (degree of Doctor of Philosophy) - University of Southampton, England.

NIERMANN, T.M.; PATEL, J.H. HITEC: A test generation package for sequential circuits. In: EUROPEAN DESIGN AUTOMATION CONFERENCE, EDAC,1991. **Proceedings…**Amsterdam, Netherlands: IEEE Computer Society, 1991. p.214-218.

NOURANI, M.; PAPACHRISTOU, C. Structural BIST Insertion Using Behavioral Test Analysis. In: EUROPEAN DESIGN AND TEST CONFERENCE, ED&TC, 1997. **Proceedings…** Paris, France: [s.n.], 1997. p.64-68.

NORWOOD, R.B. **Synthesis-for-Scan: Reducing Scan Overhead with High Level Synthesis.** 1997. 67f. Dissertation (Partial Fulfillment Of The Requirements for the Degree of Doctor of Philosophy), Stanford University, San Francisco, USA.

PATNAIK, L.M.; JAMADAGNI, H.S.; AGRAWAL, V.K.; VARAPRASAD, B. The state of VLSI testing. **IEEE Potentials**, [S.l.], v.21, n.3, p.12-16, 2002.

PETERSON, J.L. **Petri-Net Theory and the Modeling of System**. [S.l]: Prentice Hall, 1983.

REIS, R. et al. **Concepção de circuitos integrados**. 2 ed. Porto Alegre: Sagra Luzzatto, 2002. 272p.

RUDNICK, E.; PATEL, J.H.; GREENSTEIN, G.S.; NIERMANN, T.M. Sequential Circuit Test Generation in a genetic algorithm framework. In: DESIGN AUTOMATION CONFERENCE, DAC, 1994. **Proceedings…** San Diego, California, USA: ACM Press, 1994. p.698-704.

RUDNICK, E.; VIETTI, R.; ELLIS, A.; CORNO, F.; PRINETTO, P.; REORDA, M. Fast Sequential Circuit Test Generation Using High-Level and Gate-Level Techniques. In: DESIGN AUTOMATION AND TEST IN EUROPE, DATE, 1998. **Proceedings…** Los Alamitos: IEEE Computer Society, 1998. p. 570-576.

SAFARI, S.; ESMAEILZADEH, H.; JAHANGIR, A. A Novel Improvement Technique For High-Level Test Synthesis. In: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2003. **Proceedings…** [S.l.: s.n.], 2003. p.609-612.

SESHADRI, S.; HSIAO, M. Behavioral-Level DFT via Formal Operator Testability Measures. **Journal of Electronic Testing: Theory and Applications,** [S.l.], v.18, n.6, p.595-611, 2002.

SHERWANI, N. **Algorithms for VLSI Physical Design Automation**. 3rd ed. USA: Kluwer Academic Publishers, 1998. 608p.

THOMAS, D.; MOORBY, P. **The Verilog Hardware Description Language**. 4th ed. USA: Kluwer Academic Publishers, 1998. 376p.

TOUBA, A. **Synthesis Techniques for Pseudo-Random Built-In Self-Test**. 1996. 33f. Dissertation (Partial Fulfillment Of The Requirements for the Degree of Doctor of Philosophy), Stanford University, San Francisco, USA.

VISHAKANTAIAH, P.; ABRAHAM, J.; ABADIR, M. ATKET – Automatic Test Knowledge Extraction From VHDL. In: DESIGN AUTOMATION CONFERENCE, DAC, 1992. **Proceedings…** Los Alamitos: IEEE Computer Society, 1992. p.273–278.

VISHAKANTAIAH, P.; ABRAHAM, J. High Level Testability Analysis Using VHDL Descriptions. In: EUROPEAN CONFERENCE ON DESIGN AUTOMATION, 1993. **Proceedings…** Los Alamitos: IEEE Computer Society, 1993. p.170-174 .

VISHAKANTAIAH, P.; THOMAS, T.; ABRAHAM, J.; ABADIR, M. AMBIANT: Automatic Generation of Behavioral Modifications For Testability. In: IEEE INTERNATIONAL CONFERENCE ON VLSI IN COMPUTERS AND PROCESSORS, 1993. **Proceedings…** Los Alamitos: IEEE, 1993. p. 63–66.

ANEXO DESCRIÇÕES VHDL

Este anexo apresenta todas as descrições VHDL que implementam o algoritmo para o cálculo da raiz quadrada. No Capítulo 3 foram apresentadas duas descrições: uma estrutural E1 e outra comportamental C1. Em anexo a esta dissertação é incluído um CD contendo as descrições VHDL originais e as descrições modificadas, além de conter todos relatórios e *scripts* utilizados nesta dissertação.