*Análise Hierárquica de
Circuitos VLSI:
Projeto de um Pré-Processador*

Alceu Santucci Bravo

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA OBTENÇÃO DO GRAU DE MESTRE
EM
MATEMÁTICA APLICADA

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Siang Wun Song

-SÃO PAULO, Dezembro de 1986-

A meu pai

AGRADECIMENTOS

Quero agradecer, primeiramente, ao Professor Siang , não apenas por sua orientação dedicada e sensata, mas principalmente por seu incentivo e confiança, imprescindíveis para que este trabalho fosse levado adiante.

Na SID, agradeço ao "chefe", Antônio Fernando, e ao Alonso, por concordarem em que, nestes últimos meses, eu dedicasse uma parte substancial de meu horário de trabalho à elaboração da dissertação. Agradeço ainda ao Marcos e ao Adamastor, por assumirem uma carga adicional de trabalho e responsabilidade.

A Bete, à Mariana, e a(?) nenen que vem vindo, peço desculpas pelas inúmeras noites e fins de semana em que me ausentei, e expresso também meu desprazer por esse distanciamento.

A.S.B.

INDICE

1. INTRODUÇÃO	1
2. ANÁLISE DE PROJETOS DE CIRCUITOS	4
2.1 Necessidade de Ferramentas	4
2.2 Especificações Hierárquicas de Circuitos	6
2.3 Ferramentas Geométricas	11
2.4 Abordagens Convencionais	13
3. ABORDAGENS HIERÁRQUICAS	17
3.1 Dificuldades na Análise Hierárquica	17
3.2 O Tratamento de Especificações sem Sobreposições	20
3.3 Um Filtro para Verificação de Regras de Projeto	23
3.4 Um Método Geral	29
3.5 A Transformação Disjunta	36
4. UM PRÉ-PROCESSADOR PARA ANÁLISE HIERÁRQUICA	42
4.1 Abordagem	42
4.2 O Pré-Processador	43
4.3 Estratégia de Particionamento	46
4.4 Heurística de Seleção	54
4.5 Aspectos de Implementação	56
5. CONCLUSÃO	62
BIBLIOGRAFIA	65

CAPITULO 1

INTRODUÇÃO

A evolução da tecnologia em micro-eletrônica tem proporcionado um crescimento prodigioso na capacidade de integração de circuitos, possibilitando o desenvolvimento de pastilhas altamente complexas e especializadas. No entanto, o projeto de uma tal pastilha, de sua especificação funcional à produção do traçado final de suas máscaras, é um processo demorado e caro. Para que os progressos na tecnologia possam ser amplamente explorados, é imprescindível a utilização de sistemas que forneçam suporte computacional adequado ao projetista no desenvolvimento da pastilha. Tais sistemas facilitam o trabalho do projetista, reduzindo o tempo gasto nos projetos e viabilizando o desenvolvimento de projetos de maior envergadura.

Entre os sistemas de uso mais comum, distinguimos um grupo que designamos por ferramentas geométricas. Sua característica comum é o fato de trabalharem com a especificação das máscaras do circuito, que é o produto final do trabalho do projetista. De modo geral, as funções desses programas envolvem a procura de erros na especificação das máscaras, antes da mesma ser enviada à fabricação, ou a obtenção de informações diretamente do traçado

do circuito.

As especificações são produzidas em linguagens próprias para esse fim. Essas linguagens usualmente possuem um recurso análogo ao de macros em linguagens de montagem, através do qual uma composição de figuras geométricas pode ser definida como um símbolo, o qual pode então ser utilizado no lugar da composição. Esse recurso é empregado para tornar as especificações mais compactas e estruturadas.

Ferramentas geométricas convencionais começam expandindo as referências aos símbolos - isto é, substituindo o símbolo pela composição que ele define - de modo a produzir uma representação do circuito que só contém figuras geométricas. Com isso, o resto do programa manipula apenas figuras geométricas. Com a crescente integração, o número de figuras cresceu vertiginosamente, e os programas passaram a consumir recursos computacionais em demasia. Pesquisas tem sido então dedicadas à investigação de métodos que procuram aproveitar a informação presente na estrutura hierárquica da especificação para reduzir substancialmente o tempo e memória gastos na análise de máscaras de circuito.

Esses métodos, ditos hierárquicos, são o assunto de nossa dissertação. O problema é relativamente novo - as primeiras publicações datam do início da década - e não foi amplamente investigado. Em nosso trabalho, levantamos as principais questões e dificuldades envolvidas no tratamento da hierarquia das especificações. Estudamos os métodos existentes para atacar o problema, procurando analisar de modo crítico as soluções

propostas. Nesse processo, surgiram algumas idéias de como enriquecer os métodos estudados. Elaboramos então um projeto de um pré-processador de especificações de circuitos de uso geral, reunindo nele nossas propostas.

O capítulo 2 focaliza o problema estudado. São apresentados o ambiente de uso de ferramentas geométricas, as ferramentas mais importantes e os objetos por elas manipulados. Discute-se brevemente a maneira convencional de se implementar ferramentas geométricas, seus problemas, e coloca-se em perspectiva a utilização de abordagens hierárquicas.

O capítulo 3 descreve os trabalhos mais importantes sobre análise hierárquica de projetos de circuitos. Os problemas e dificuldades que devem ser enfrentados são apontados, bem como as alternativas básicas de solução. Os métodos são então discutidos, com ênfase naqueles em que baseamos nosso projeto.

O capítulo 4 apresenta nosso projeto. Nossas idéias são apresentadas e discutidas, junto com os principais problemas e questões associados à abordagem adotada. Todos os pontos relevantes para a implementação do pré-processador são apresentados.

O capítulo 5 retoma brevemente as principais questões discutidas no trabalho, encerrando a dissertação.

CAPITULO 2

ANÁLISE DE PROJETOS DE CIRCUITOS

2.1 NECESSIDADE DE FERRAMENTAS

Circuitos integrados são formados pela deposição de materiais diversos sobre uma pastilha de silício. No processo de fabricação, as regiões da pastilha em que os materiais serão depositados são selecionados por meio de um conjunto de máscaras que funcionam de maneira análoga a negativos fotográficos.

As máscaras podem ser montadas a partir de descrições em linguagens gráficas, também ditas geométricas, das quais CIF ("Caltech Intermediate Form"), que apresentamos adiante, é um exemplo. Projetistas de circuitos, por sua vez, utilizam essas mesmas linguagens para especificar como os circuitos devem ser fabricados. Desta maneira, linguagens gráficas servem de interface entre o projetista e o fabricante: a pastilha será produzida diretamente a partir da especificação do circuito em CIF desenvolvida pelo projetista.

Num ambiente convencional de desenvolvimento a especificação do circuito é feita num nível bastante baixo de abstração.

Geralmente com o uso de editores gráficos, o projetista, expressando-se sempre em termos de objetos geométricos nas diferentes camadas de material, compõe módulos progressivamente mais abstratos, até formar o circuito como um todo. O editor gera a especificação em linguagem gráfica que é enviada ao fabricante.

Esse processo dá margem a uma vasta gama de possíveis erros, que vão de erros em algoritmos a problemas na especificação das máscaras - como, por exemplo, uma ligação muito fina. Aqui, cabe apontar uma característica importante no desenvolvimento de circuitos integrados, que é a dificuldade de se tratar erros sobre a pastilha "física". De fato, cada vez que o projeto é alterado, levam-se semanas para produzir a pastilha correspondente; ademais, frequentemente o circuito produzido não implementa o projetado devido a falhas de fabricação, o que, claramente, dificulta muito os testes.

Com essas considerações, vemos que o projetista precisa contar com ferramentas que o ajudem a eliminar o maior número possível de erros antes de mandar o projeto para fabricação. Há diversos tipos de ferramentas. Neste trabalho, nos interessamos por aquelas que operam a partir da especificação do circuito em linguagem gráfica. Utilizaremos diversas formas para denominar tais programas: ferramentas para análise de máscaras, ferramentas geométricas, etc.

Antes de continuarmos, gostaríamos de observar que outros tipos de sistema, que proporcionam ambientes em que o projetista

trabalha em níveis mais elevados de abstração, têm sido propostos e desenvolvidos. Liesenberg[86] faz uma apresentação extensiva sobre o assunto.

2.2 ESPECIFICAÇÕES HIERÁRQUICAS DE CIRCUITOS

CIF - Caltech Intermediate Form - é uma linguagem para a especificação do traçado de circuitos integrados. Seu propósito é servir de interface padrão entre projetistas e fabricantes. Foi definida de maneira a ser independente da tecnologia de fabricação, de equipamentos e sistemas de automação, bem como para ser tratada principalmente de maneira automática. O capítulo 7 de Hon&Sequin[80] apresenta a linguagem de modo bastante completo. Aqui, vamos ilustrar alguns de seus aspectos, visando firmar idéias sobre os tipos de objeto que ferramentas geométricas manipulam. Salientamos ainda que CIF é bastante representativa das características comumente presentes em linguagens para especificação de circuitos.

A linguagem usa um sistema de coordenadas no plano. Coordenadas e tamanhos são dadas por números inteiros, em centésimos de micron. Rotações são dadas por meio de vetores de direção. A sintaxe permite uma certa flexibilidade na forma dos comandos, pois muitos caracteres, incluindo letras minúsculas, são considerados "brancos".

CIF dispõe de comandos para a especificação de figuras geométricas: retângulos, círculos, ligações e polígonos

arbitrários. Dispõe também de um comando para a especificação de camadas de material. A figura 2.1 apresenta um retângulo e sua especificação em CIF.

Box comprimento 3 largura 2 centro 4,3 direção 1,1

Figura 2.1 Um retângulo

Outro aspecto, de fundamental importância para o presente trabalho, é a possibilidade de se definir e chamar símbolos, num esquema análogo ao de macros em linguagens de montagem. Com este recurso, define-se um símbolo como uma certa composição de figuras geométricas. Ocorrências da composição podem então ser especificadas por meio de chamadas ao símbolo. Isto permite a confecção de especificações compactas, uma vez que os circuitos geralmente contém elementos que se repetem muitas vezes.

A definição de um símbolo é feita com um comando da forma

DS <número do símbolo> [<escala>]

<definição>

DF

<definição> pode conter, além de primitivas geométricas, chamadas a outros símbolos; as distâncias nela presentes são multiplicadas por <escala>. A chamada é feita com um comando da forma

C <número do símbolo> <transformações>

<transformações> especifica onde e como será colocada a composição de figuras associada ao símbolo. Além de translação, a composição pode sofrer operações de rotação e espelhamento em ambos os eixos.

Com o auxílio da figura 2.2, que ilustra o uso de símbolos, apresentamos a seguir alguns termos e conceitos importantes, que serão empregados no resto do texto. A figura 2.2(a) apresenta um par de retângulos numa certa composição, e o esboço do comando CIF que a define como sendo o símbolo 1. A figura 2.2(b) apresenta uma outra composição de retângulos, junto com o comando CIF que a define como o símbolo 2 por meio de duas chamadas ao símbolo 1. Dizemos que o símbolo 2 contém exemplares ou ocorrências do símbolo 1, e que este é sub-símbolo daquele.

(a)

DS 1

Box ...

Box ...

DF

(b)

DS 2

Call 1 ...

Call 1 ...

DF

(c)

Figura 2.2 Símbolos

Um polígono de fronteira de um símbolo S é um polígono que contém todas as figuras e exemplares de sub-símbolos de S , ou seja, que contém todo seu traçado. Polígonos de fronteira são utilizados por ferramentas geométricas para delimitar a área ocupada pelo símbolo. Usualmente empregam-se retângulos, ou então polígonos "manhattan", que são aqueles em que todos os ângulos são retos. Nas ilustrações que faremos, os símbolos em geral são representados por polígonos de fronteira, com sub-símbolos representados em linha tracejada. Assim, utilizando fronteiras retangulares, o símbolo 2 seria representado como na figura 2.2(c) (note-se que os retângulos tracejados representam os exemplares do símbolo 1).

Introduzimos, por fim, o conceito de regularidade. A regularidade é uma propriedade da especificação associada ao grau de utilização de seus símbolos. Uma primeira medida de regularidade foi definida por Lattin[79], visando avaliar a produtividade no desenvolvimento de pastilhas. A medida, no entanto, é difícil de computar, pois envolve o reconhecimento de transistores e memórias. Para uso em ferramentas de análise, outra medida, derivada da primeira, tem sido utilizada (Hon[83], Newell&Fitzpatrick[82]). Nesta, por nós adotada, a regularidade é definida pela razão entre o número de figuras geométricas do traçado e o número de figuras presentes na especificação. Assim, a regularidade da especificação do símbolo 2 é $4/2=2$, pois ela descreve 4 retângulos com o uso de apenas 2 (os retângulos da definição do símbolo 1).

2.3 FERRAMENTAS GEOMÉTRICAS

VERIFICAÇÃO DE REGRAS DE PROJETO

Para que um circuito integrado possa ser produzido corretamente é necessário que seu traçado satisfaça determinadas restrições, relacionadas a limitações do processo de fabricação. Estas restrições, chamadas regras de projeto, são dadas como um conjunto de valores mínimos para espessuras, espaçamentos e sobreposições de figuras geométricas nas camadas de material.

A figura 2.3 ilustra algumas regras apresentadas por Mead&Conway[80] para o processo nMOS. As dimensões estão dadas numa unidade de medida, lambda, que corresponde aproximadamente à resolução do processo de fabricação. Esta medida é comumente utilizada com o intuito de estender o período de validade de projetos, uma vez que avanços tecnológicos estão frequentemente diminuindo as dimensões físicas estabelecidas nas regras (no começo dos anos 80, lambda era cerca de 2 micra).

Figura 2.3 Exemplos de regras de projeto

A função de um programa verificador de regras de projetos é, dada a especificação do traçado de um circuito, determinar se ela está de acordo com as regras estabelecidas para um determinado processo de fabricação, produzindo uma relação das violações encontradas.

EXTRAÇÃO DE CIRCUITOS

Outra ferramenta bastante utilizada é um extrator de circuitos. Este programa monta, a partir de uma especificação geométrica, uma descrição do circuito a nível de chaves lógicas.

A descrição obtida é a entrada usual para um programa simulador, usado para se testar o funcionamento lógico do circuito. Um outro programa usa a mesma descrição para fazer uma série de verificações lógicas, como, por exemplo, se todos transistores podem potencialmente ser abertos e fechados, ou se há transistores que, caso fechados, possam causar curto-circuito.

OUTROS

Parâmetros elétricos, tais como capacitância e resistência de nós e conectores, também são obtidos do traçado do circuito. Eles são usados por simuladores para verificação do comportamento elétrico do circuito, o que inclui avaliação de desempenho e análise de temporização.

Por fim, traçadores gráficos ("plotters") utilizam a especificação geométrica para montar, em papel ou outro meio,

uma representação gráfica do circuito, geralmente usada para uma verificação visual do projeto.

2.4 ABORDAGENS CONVENCIONAIS

Programas convencionais de análise de circuitos operam a partir da descrição expandida do circuito: as chamadas a um símbolo são substituídas pelo conteúdo descrito na sua definição. Ao final do processo de expansão, a representação do circuito contém apenas objetos geométricos, e o programa trabalha assim com o conjunto de todas as figuras que compõem o circuito.

Um método bastante utilizado por tais programas é baseado em operações com polígonos. Por exemplo, um dos passos de um algoritmo usual de extração de circuitos (Ullman[84]) é encontrar todas intersecções entre retângulos de difusão e retângulos de polisilício, a fim de se determinar os transistores do circuito. Um outro algoritmo, para verificação de regras de projeto, começa com a união dos retângulos de uma mesma camada. Expandindo-se a seguir os polígonos resultantes em metade do espaçamento mínimo, pode-se verificar regras de espaçamento com operações de intersecção. O processo está ilustrado na figura 2.4. De modo análogo, a retração dos polígonos pode ser utilizada para a verificação de regras de espessura.

Figura 2.4 Verificação de espacejamento
com operações sobre polígonos

Os programas manipulam uma grande massa de dados. Em nMOS, por exemplo, uma porta lógica é constituída, a grosso modo, por 10 retângulos. Assim, circuitos complexos como os que vem sendo fabricados - que contam com 100 mil ou mais portas - são compostos por algo como 1 milhão de retângulos.

Esses números trazem problemas de consumo excessivo de tempo e espaço. Programas que implementam algoritmos simplistas apresentam tempo de execução $O(n^2)$, sendo inviável sua utilização mesmo com números menores. Em termos de espaço, uma conta rápida pode mostrar que um circuito como acima necessita de mais de uma dezena de megabytes de memória para ser armazenado.

Bastante trabalho tem sido realizado para atacar esses problemas, alguns dos quais tivemos oportunidade de conhecer no decorrer deste nosso estudo. Baird[78] e Lauther[78] elaboraram formas de reduzir o número de comparações entre figuras geométricas na verificação de regras de projeto. Bentley&Wood[79] e McCreight[80] desenvolveram algoritmos para

relatar intersecções entre retângulos, eficientes para entradas com grande volume de dados. Baker[80] propõe um método baseado numa varredura sobre uma representação do circuito em forma de "bitmap". Bentley,Haken&Hon[80] fazem um estudo estatístico de especificações de circuitos VLSI, apresentando um algoritmo baseado nos dados obtidos.

Os problemas no entanto não podem ser considerados resolvidos. A crescente complexidade dos circuitos - em poucos anos espera-se fabricar pastilhas com milhões de transistores - leva a crer que programas convencionais, ainda que baseados em algoritmos com tempo esperado de execução proporcional ao número de dispositivos, venham a gastar dias de CPU na realização de suas tarefas (Newell&Fitzpatrick[82]). As pesquisas com ferramentas hierárquicas surgem nesse contexto, como uma tentativa de se reduzir drasticamente o tempo gasto na análise de máscaras de circuito.

Na análise hierárquica, a idéia central é explorar a estrutura da descrição do circuito para evitar análises redundantes. De fato, num programa de verificação de regras de projeto, por exemplo, uma vez que um símbolo tenha sido verificado, não há a princípio necessidade de se ficar repetindo a mesma verificação em todos os lugares em que um exemplar do símbolo foi colocado. O símbolo pode ser verificado uma única vez, evitando-se assim um consumo desnecessário de tempo.

Devemos também notar que há uma tendência, como um meio de se viabilizar o desenvolvimento de sistemas complexos, de se

utilizar metodologias de projeto estruturado que resultam no desenvolvimento de circuitos com alto grau de hierarquia e regularidade. Cada vez mais estrutura está efetivamente presente nos circuitos projetados, o que favorece a utilização de enfoques hierárquicos.

Outro ponto favorável se destaca quando observamos que, num programa convencional, a presença de um erro em um símbolo faz com que esse erro seja relatado em todos os lugares em que o símbolo é chamado. Se há muitas ocorrências do símbolo, o resultado é um relatório em que outros erros podem se perder em meio à massa de mensagens produzidas. Um programa com tratamento hierárquico pode relatar o erro apenas uma vez.

A análise hierárquica amplia a perspectiva de se resolver os problemas apresentados. Ela no entanto apresenta seus próprios problemas, alguns dos quais ainda não resolvidos de modo bastante satisfatório.

CAPITULO 3

ABORDAGENS HIERÁRQUICAS

3.1 DIFICULDADES NA ANÁLISE HIERÁRQUICA

Bentley e Ottmann[81] estudaram a complexidade de diversos problemas de operação com retângulos, quando estes são definidos numa linguagem hierárquica. Os problemas estudados, em sua maior parte, tem aplicação na análise de circuitos VLSI, e os resultados obtidos são negativos: entre os problemas, alguns são NP-completos e outros necessitam tempo exponencial.

A razão para esses resultados pode ser entendida quando se verifica que uma descrição em CIF pode especificar uma quantidade muito grande de objetos, como é o caso da descrição apresentada na figura 3.1, cujo n -ésimo símbolo descreve 2^{*n} retângulos. Bentley e Ottmann utilizaram essa construção para mostrar que o problema de determinar os pares de retângulos que se interceptam pode requerer tempo exponencial no tamanho da descrição.

```

DS 1
    Box ...
    Box ...
DF

DS i
    Call i-1
    Call i-1
DF                                , para  $2 \leq i \leq n$ 

```

Figura 3.1

Os resultados desestimulam a procura de algoritmos para análise hierárquica que sejam eficientes em todas instâncias de problemas. Nessas circunstâncias, um caminho usual é a busca de algoritmos que funcionem bem com entradas que obedeçam determinadas restrições. Bentley e Ottmann estudaram, no mesmo trabalho, uma versão da linguagem hierárquica na qual não pode haver, na definição de um símbolo, intersecções entre os retângulos de fronteira dos símbolos nele chamados, nem entre estes e figuras. Com essa restrição, muitos dos problemas investigados deixam de ser difíceis, passando a ter soluções lineares ou mesmo triviais. Este último resultado é, de certa forma, esperado, uma vez que o problema básico ao se fazer análise hierárquica reside no tratamento de sobreposições de símbolos.

Sobreposições podem tornar a análise de um exemplar de um símbolo irrelevante para a análise de um outro exemplar do mesmo símbolo. Consideremos a verificação de regras de projeto. Uma sobreposição pode introduzir erros em um símbolo ou então validar construções que seriam incorretas na ausência da sobreposição. Como exemplo, tomemos o símbolo S da figura 3.1(a) e suponhamos que os retângulos representados não estejam suficientemente separados. Se a um exemplar de S se sobrepuser uma figura que conecte os retângulos, como na figura 3.1(b), não há violação da regra de espaçamento. Já se não houver uma tal sobreposição, um erro deve ser relatado. Desta maneira, de nada adianta analisar S isoladamente. O mesmo acontece na extração de circuitos, onde sobreposições podem mudar radicalmente o circuito representado: transistores podem ser criados ou removidos, ligações podem ser quebradas ou estabelecidas.

Figura 3.2 O efeito de sobreposições

A imposição de restrições, que proíbem a sobreposição de símbolos, proporciona o uso de algoritmos eficientes. Entretanto, por motivos de ordem prática, nem sempre é viável sua implementação. O problema aqui é que projetistas de

circuitos podem ser avessos a restrições. Além disso, há muitos sistemas de auxílio ao projeto em uso, gerando especificações sem nenhum tipo de restrição. Com isso, outras soluções têm sido pesquisadas, em busca de métodos que apresentem bom desempenho em ocorrências esperadas do problema.

Por fim, vale salientar que, na análise da complexidade, o tamanho da entrada nos problemas estudados é o tamanho da especificação CIF. Não se pode assim relacionar os resultados acima descritos com o tempo de execução dos algoritmos utilizados em abordagens convencionais, pois, para estes, o tamanho da entrada considerado é o número de retângulos da especificação expandida.

3.2 O TRATAMENTO DE ESPECIFICAÇÕES SEM SOBREPOSIÇÕES

Uma das maneiras de se lidar com sobreposições é impedir sua ocorrência. Alguns trabalhos tem sido realizados com esse enfoque, restringindo o formato das especificações. A restrição principal consiste em não permitir sobreposições. Isso significa, mais precisamente, que: 1) exemplares de símbolos não podem se sobrepor a outros exemplares; e 2) exemplares de símbolos não podem se sobrepor a figuras geométricas.

Scheffer[81] discute um método que consiste basicamente em forçar a restrição acima, sem colocar, entretanto, restrições sobre o formato dos símbolos. Dessa forma, os símbolos podem ter polígonos de fronteira arbitrários, o que possibilita um bom

aproveitamento da área da pastilha. Wagner[84] apresenta um sistema composto por diversas ferramentas que, tendo sido exaustivamente utilizado em projetos de pastilhas na Intel Corporation, foi reescrito para aproveitar a estrutura hierárquica das especificações. O sistema é bastante conservador: além de não permitir sobreposições, as fronteiras dos símbolos devem ser retangulares. Tanto Scheffer quanto Wagner exigem que elementos ativos do circuito (um transistor, por exemplo) fiquem inteiramente contidos em um símbolo. Newell e Fitzpatrick[82], no entanto, descrevem um programa para extração de circuitos que trata símbolos com elementos ativos incompletos.

A ausência de sobreposições possibilita a implementação de ferramentas de análise hierárquica eficientes e relativamente simples. De modo geral, as análises são feitas recursivamente. Símbolos que só contém figuras geométricas são analisados isoladamente, armazenando-se informações relevantes para a análise do contexto em que exemplares do símbolo serão colocados. A análise de um símbolo que contém exemplares de outros símbolos envolve a análise de cada um dos sub-símbolos e a composição das informações sobre eles.

Vejamos como pode ser feita a verificação de regras de projeto. Como não há sobreposições, o interior dos símbolos pode ser analisado isoladamente. A análise da região fronteira, no entanto, pode ser afetada pelo contexto em que os exemplares do símbolo são colocados. Tomemos o exemplo da figura 3.1, e suponhamos que o retângulo A não tenha a espessura mínima necessária. Não podemos, pelo simples exame de S, afirmar que

isso constitui um erro, pois os exemplares de S podem ser colocados em lugares em que outras figuras complementem a espessura necessária. Já no retângulo B um erro legítimo pode ser apontado, uma vez que ele não pode ser afetado por elementos que porventura venham a ser adjacentes aos exemplares de S.

Figura 3.3 Erros internos e erros potenciais

Quando um símbolo é analisado, armazenamos sua região de fronteira, para que a possamos examinar nos contextos em que exemplares do símbolo sejam colocados. Para verificar um símbolo, substituímos os exemplares de sub-símbolos pelas respectivas regiões de fronteira e aplicamos um algoritmo convencional de verificação de regras, desprezando, entretanto, os erros encontrados a uma certa distância da fronteira. Com esse método, a região de fronteira é examinada nos contextos em que o símbolo é chamado e apenas os erros legítimos são relatados.

As idéias para se implementar um algoritmo de extração de circuitos são similares. Cada símbolo, uma vez analisado, fica descrito por um circuito interno e um conjunto de pontos de conexão. A extração do circuito associado a um dado símbolo envolve a extração do circuito de cada um de seus sub-símbolos, seguida da composição desses sub-circuitos mais as figuras do símbolo. É importante notar que, aqui também, a ausência de sobreposições desempenha um papel fundamental, uma vez que o circuito interno de um símbolo não é afetado por elementos externos a ele. Além disso, os símbolos se comunicam somente através de suas fronteiras, de modo que as conexões entre eles podem ser estabelecidas sem dificuldade.

3.3 UM FILTRO PARA VERIFICAÇÃO DE REGRAS DE PROJETO

Whitney[81] desenvolveu um programa que serve como um pré-processador para um verificador de regras de projeto convencional. O programa, atuando como um filtro, procura remover informações geométricas que sejam redundantes para a verificação das regras. Para um projeto com estruturas que se repetem muitas vezes, o filtro produz uma descrição que contém uma parte reduzida do traçado presente no projeto original. A descrição, que é totalmente expandida (sem hierarquia), é então submetida a um verificador de regras convencional. Whitney sustenta que os erros presentes no projeto original estarão também na especificação produzida pelo filtro. A figura 3.4, extraída do artigo citado, dá uma boa idéia da atuação do programa, mostrando o traçado original de um circuito (figura

3.4(a)) e o traçado que é submetido à verificação, após o filtro (figura 3.4(b)).

Figura 3.4 Efeito do filtro de Whitney

O programa elimina informações redundantes de duas maneiras: 1) verificando cada símbolo uma única vez e 2) verificando as interações entre pares de elementos do circuito também uma única vez. Aqui, um elemento é um símbolo ou um objeto geométrico. Uma interação entre dois elementos existe quando o seu posicionamento relativo pode causar uma violação das regras de projeto; a interação é caracterizada não só pelo par de elementos como também pelo seu posicionamento relativo. Com

esses conceitos, o algoritmo utilizado pelo programa está descrito na figura 3.5.

verifica-símbolo(s):

```
    se s já foi verificado, retorna
    se s só contém figuras
        imprime-traçado(s)
    senão
        para cada exemplar t chamado em s
            verifica-símbolo(t)
        para cada par de elementos e1,e2 que interagem
            verifica-interação(e1,e2)
    marca s como verificado
```

verifica-interação(e1,e2):

```
    se a interação e1,e2 já foi verificada, retorna
    se e1 e e2 são ambos figuras
        imprime-traçado(e1,e2)
    senão
        seja e1 símbolo
        para cada elem. e1' de e1 que interage com e2
            verifica-interação(e1',e2)
    marca interação e1,e2 como verificada
```

Figura 3.5 O algoritmo do filtro

Como um exemplo (extraído de Hon[83]), consideremos um símbolo S composto por 4 exemplares de um símbolo A e 2 exemplares de um símbolo B , como na figura 3.6(a). Para verificar S , o programa irá verificar cada um dos exemplares de símbolos de S . Assim, a primeira vez em que os símbolos A e B forem encontrados (exemplares $A1$ e $B1$) eles serão processados; supondo-se que contenham apenas figuras geométricas, estas serão impressas. Os outros exemplares de A e B não serão efetivamente verificados. O programa irá então tratar as interações entre os elementos de S . De todas interações, ele irá verificar apenas as interações $(A1,A2)$, $(A1,B1)$ e $(B1,A1)$. É interessante notar que, neste processo, algumas figuras de $A1$ e $B1$ serão impressas novamente, causando assim uma duplicação de figuras (Whitney usa um pós-processador para eliminar cópias de figuras do arquivo de saída). As outras interações serão desprezadas. A saída do programa seria assim a mostrada na figura 3.6(b).

(a)

(b)

(c)

Figura 3.6 Exemplo da atuação do filtro sobre um símbolo

O método apresenta uma série de qualidades. O fato de o filtro ser um programa em separado faz com que ele possa ser rodado como pré-processador para qualquer verificador de regras já em uso, sem necessidade de qualquer modificação. O ganho em tempo de processamento pode ser significativo, o que, entretanto, depende da especificação examinada. Whitney relata resultados experimentais com duas pastilhas: numa delas, bastante estruturada, o tempo gasto na verificação das regras caiu de 17h01m (verificação convencional) para 3h18m (verificação com filtro); na outra, irregular, só o filtro gastou mais de 72h, sendo que a análise sem o filtro pode ser feita em menos de 12h. O resultado é considerado razoavelmente satisfatório, pois especificações bem estruturadas podem ter seu tempo de processamento bastante reduzido.

Por outro lado, o método não é geral e tampouco se generaliza diretamente para outras ferramentas. Outro problema, apontado em Hon[83], é uma falha no algoritmo, que se deve ao fato deste examinar pares de elementos. Tomemos o exemplo da figura 3.6. A saída do programa não contém uma interação entre dois exemplares do símbolo A sem a ocorrência de B - que é o caso da interação entre os exemplares A3 e A4. O traçado de A e B podem ser tais que hajam violações de regras na interação entre os exemplares de A que sejam ocultadas pela presença do exemplar de B - a figura 3.6(c) é um exemplo disso, se considerarmos que os retângulos dos exemplares do símbolo A não estão suficientemente espaçados. Tais violações passarão despercebidas.

3.4 UM MÉTODO GERAL

Hon[83] apresenta um trabalho no qual foi desenvolvido um método de processar especificações hierárquicas irrestritas e que pode ser aplicado em diversas ferramentas de análise. As ferramentas são implementadas com dois módulos. O primeiro deles, dito frontal ("front-end"), reúne uma série de funções necessárias ao tratamento hierárquico da especificação, sendo independente da tarefa a ser realizada e, portanto, comum a todas elas. O outro módulo, final ("back-end"), realiza funções específicas da ferramenta em questão. A ênfase de Hon é para o módulo frontal. Ele procurou concentrar funções aí, no intuito de facilitar o desenvolvimento dos diversos módulos finais.

O módulo frontal conduz o processamento do programa. Sua função principal é particionar símbolos em sub-símbolos que não se interceptam. Ele procura reconhecer a ocorrência de símbolos iguais, para evitar a execução de análises redundantes, e estabelece a ordem em que os sub-símbolos serão agregados de forma a completar o exame do símbolo original. Módulos finais devem prover duas funções principais: 1) analisar símbolos que contém apenas objetos geométricos e 2) analisar a justaposição de dois símbolos anteriormente verificados, combinando-os em um único símbolo. Como o módulo frontal elimina sobreposições, módulos finais podem ser implementados de maneira similar à descrita na seção 3.2. O algoritmo está representado na figura 3.7. Os procedimentos verifica-símbolo, já-verificado e particiona compõem o módulo frontal, enquanto analisa-traçado e combina formam o final. Pode-se observar que o algoritmo é uma

generalização do filtro de Whitney (figura 3.5), o qual foi estendido de forma a possibilitar a realização de diferentes análises.

```
verifica-símbolo(s):  
    se já-verificado(s), retorna  
    se s só contém figuras  
        analisa-traçado(s)  
    senão  
        particiona(s)  
        para cada exemplar t de s  
            verifica-símbolo(t)  
    u <-- 0  
    para cada exemplar t de s  
        u <-- combina(u,t)
```

Figura 3.7 O algoritmo de Hon

Como exemplo, suponhamos que a verifica-símbolo seja dado um símbolo S, composto por figuras e exemplares de símbolos que se interceptam, algo como na figura 3.8(a). O programa associa a cada símbolo um retângulo de fronteira em relação ao qual são verificadas as intersecções; assim, um par de exemplares de símbolos se interceptam se seus retângulos de fronteira se interceptam. De alguma forma, que veremos adiante, S é particionado em exemplares que não se interceptam (figura 3.8(b)). Cada um destes exemplares é, por sua vez, examinado; aqui, supondo-se que T2 e T3 sejam iguais, somente um deles será efetivamente verificado. Depois, será analisada a justaposição

de cada um desses exemplares com o aglomerado que estará sendo montado (os símbolos são agregados na ordem lexicográfica em (x,y) de seus pontos esquerdos inferiores). A figura 3.8(c) representa um passo desse processo, o da combinação de T3 com o aglomerado U obtido da combinação de T1 e T2.

Figura 3.8 Alguns passos do algoritmo de Hon

É interessante observar que os símbolos obtidos no particionamento são possivelmente novos, criados pelo programa. Assim, para saber se um símbolo já foi anteriormente verificado, uma marca não funciona. No programa de Hon, isso é determinado comparando-se o conteúdo do símbolo com o daqueles que já foram analisados (o programa emprega uma tabela de espalhamento ("hashing") para agilizar a busca).

Vejamos agora como é feito o particionamento de um símbolo. Hon descreve duas estratégias de particionamento, por ele denominadas Heurística e Disjunta. Esta última foi

primeiramente apresentada em Newell&Fitzpatrick[82] e iremos apresentá-la quando tratarmos desse trabalho.

O passo principal da estratégia Heurística de particionamento é a obtenção, por meio de uma série de expansões e agrupamentos de símbolos, de um conjunto de símbolos que não se interceptam. Na expansão de um exemplar de um símbolo, o exemplar é substituído pelo seu conteúdo - figuras e exemplares de símbolos que o compõem. A expansão é uma operação que possibilita a remoção de sobreposições de símbolos. O agrupamento é uma operação de sentido inverso ao da expansão. Com ele, um conjunto de símbolos e figuras é reunido em um outro símbolo. O agrupamento tem o papel de hierarquizar o particionamento. Podemos ver o processo realizado por uma sequência de expansões e agrupamentos como uma reestruturação do modo como o símbolo é definido. Obtido um conjunto de símbolos que não se interceptam, o algoritmo, tomando por base tal conjunto, conclui o particionamento com uma operação de "fatiamiento". A figura 3.9 ilustra as operações descritas.

O algoritmo do particionamento está apresentado na figura 3.10. O procedimento agrupa define como grupo um conjunto de símbolos que se interceptam. Entretanto, um grupo só será transformado em um símbolo se: 1) ele não se intercepta com outros grupos ou símbolos; e 2) seu retângulo de fronteira é menor que o do símbolo original. Essas regras asseguram que o particionamento será efetivo, pois só se criam símbolos menores que o original e

que satisfaçam a condição, de não sobreposição com outros símbolos, que está sendo buscada. Ainda, ao se formar um símbolo, as porções das figuras geométricas do símbolo original que o interceptam são passadas para ele. Isso garante que não haverá sobreposições entre símbolos e figuras.

Figura 3.9 Operações utilizadas no particionamento

```

particiona(s):
    para todo exemplar t de s com fronteira igual à de s
        expande(t)
    agrupa(s)
    enquanto houver sobreposições de símbolos
        t <-- seleciona(s)
        expande(t)
        agrupa(s)
    fatia(s)

```

Figura 3.10 Algoritmo do particionamento Heurístico

Outro ponto importante do algoritmo é o que se refere à escolha de exemplares de símbolos para expansão. O algoritmo começa com a expansão de eventuais símbolos que tenham o retângulo de fronteira igual ao do símbolo original, pois estes terão que ser necessariamente expandidos. A partir daí, a escolha é tarefa do procedimento seleciona. O programa utiliza uma série de heurísticas para realizar essa escolha. As heurísticas foram elaboradas por Hon de modo "ad hoc", pelo exame de estatísticas e também pela observação do andamento do programa. A estratégia empregada pode ser descrita como: "retardar a expansão de 'arrays' de símbolos até que seja inevitável, expandindo então os menos complexos em primeiro lugar". Hon procura efetivá-la com a utilização das seguintes heurísticas, em ordem:

- 1) procure um símbolo que represente o menor número de figuras caso seus exemplares sejam completamente expandidos; se há exemplares do símbolo envolvidos em intersecções, selecione o que estiver envolvido no maior número delas.

2) procure o exemplar de símbolo envolvido no maior número de intersecções; selecione-o se o símbolo for chamado menos que 4 vezes.

3) se há um exemplar de um símbolo chamado menos que 4 vezes envolvido em alguma intersecção, selecione-o.

4) selecione o exemplar envolvido no maior número de intersecções.

Hon experimentou seu método numa série de projetos reais, obtidos de fontes diversas. Em suas análises, ele procura comparar as estratégias Heurística e Disjunta (a implementação de Hon desta última difere da de Newell e Fitzpatrick no fato de que, para Hon, os símbolos tem fronteiras retangulares, e não "manhattan" poligonais). Para isso, ele coletou não só medidas de tempo como também algumas outras medidas visando caracterizar as partições geradas. A figura 3.11 apresenta uma tabela extraída de seu trabalho, mostrando o tempo gasto na extração de circuitos com um algoritmo convencional e, com seu algoritmo, utilizando as duas estratégias de particionamento.

Os experimentos indicam que, de modo geral, o método apresenta um desempenho melhor que programas convencionais. Contudo, na expectativa de ganhos de várias ordens de magnitude, Hon fica um pouco desapontado com os resultados, principalmente em relação à estratégia Heurística, com a qual não foi possível, em 5 horas de processamento, concluir o particionamento de um dos projetos mais regulares (circuito RAM). A estratégia Disjunta revela-se melhor em termos do tempo total de execução. Entretanto, outros dados levantados por Hon mostram que a partição obtida com a

estratégia Heurística é, em geral, mais rapidamente processada pelo módulo final de extração de circuitos. Poderíamos dizer que o método Heurístico produz uma partição melhor, mas que gasta mais tempo para fazê-lo.

Circuito		Tempos (em minutos)		
Nome	#dispositivos	Convencional	Heurístico	Disjunto
Cherry	881	0.98	1.31	1.27
ALU	2944	3.73	3.25	3.95
Synth	3659	4.24	2.64	2.20
DChip	4884	7.04	4.84	4.33
Root	6630	10.65	4.26	4.11
FIFO	8269	10.66	12.69	1.61
SChip2	9473	14.05	10.97	11.54
Filters80	11987	97.88	5.56	6.72
CFast	12034	14.04	21.03	16.67
FIR	14421	18.77	3.34	3.34
Testram	20480	23.09	26.53	1.09
RAM	29138	33.25	-	8.84
RISCb	42084	65.26	191.80	27.67

Figura 3.11 Experimentos com extração de circuitos

3.5 A TRANSFORMAÇÃO DISJUNTA

Newell e Fitzpatrick[82] utilizam uma abordagem que não difere muito da proposta por Hon, discutida na seção anterior. Como lá, a análise de um projeto é feita em dois passos, em que o

primeiro é comum a todas as ferramentas e o segundo implementa alguma análise em particular. Mas a distribuição de funções entre os passos é diferente. No método de Newell e Fitzpatrick, o primeiro passo, por eles denominado transformação disjunta, se restringe a transformar a especificação fornecida, que pode conter sobreposições, numa outra especificação, livre de sobreposições.

A transformação disjunta utiliza um método de particionamento que foi chamado Disjunto por Hon. O particionamento de um símbolo é feito tomando por base regiões em que exemplares de sub-símbolos se interceptam. Um novo símbolo é associado a cada região, sendo reconhecidas as repetições de símbolos anteriormente definidos. A figura 3.12(a) apresenta a definição original de um símbolo S. A figura 3.12(b) mostra as regiões utilizadas no particionamento, e a figura 3.12(c) mostra a definição resultante de S, feita em termos de novos símbolos, criados pelo programa.

O programa associa a cada símbolo uma janela "manhattan" poligonal fora da qual quaisquer porções de exemplares são ignoradas. Na inicialização, o programa define a janela dos símbolos de entrada como o seu retângulo de fronteira mínimo. O particionamento é implementado com um par de procedimentos mutuamente recursivos, divide e reúne. A função do procedimento divide é definir as regiões disjuntas mostradas na figura 3.12(b), enquanto que a função de reúne é criar os novos símbolos apresentados na figura 3.12(c), reconhecendo repetições.

(a)

(b)

(c)

Figura 3.12 Particionamento Disjunto

O procedimento divide recebe um símbolo como parâmetro. Ele divide a área do símbolo em faixas, cujos limites são dados pelas arestas horizontais das janelas dos exemplares que compõem o símbolo. A figura 3.13(a) mostra as faixas F1, F2, F3 e F4 definidas pelo procedimento para o exemplo da figura 3.12. A partir da faixa inferior e em direção às superiores, o

procedimento varre cada uma das faixas, da esquerda para a direita. Nessa varredura, o programa calcula o conjunto de exemplares presentes entre cada par de arestas verticais das janelas. Com a numeração de exemplares descritas na figura 3.13(a), a faixa F1 apresenta a sequência de conjuntos {T1}, {T2}. Na faixa F2, a sequência é formada pelos conjuntos {T1}, {T1,U1}, {T1}, {T2}, {T2,U2}, {T2}. Cada um desses conjuntos irá definir uma das regiões apresentadas na figura 3.12(b). Durante a varredura, toda vez que um novo conjunto é encontrado uma nova região é definida, compreendendo a área ocupada pelo conjunto na faixa que está sendo examinada. Quando um conjunto já existente é encontrado, a região associada é estendida com a área ocupada pelo conjunto na faixa em questão. As porções das figuras do símbolo original que interceptam a região são incorporadas na sua definição. A figura 3.13(b) apresenta a evolução da região associada ao conjunto {T1} durante a varredura.

Uma vez completada a varredura das faixas, divide passa as regiões obtidas ao procedimento reúne. A função deste é reconhecer regiões iguais e formar novos símbolos. Para verificar a igualdade de regiões, o procedimento utiliza o conjunto de exemplares que as compõem e também seu posicionamento relativo. Desta forma, o procedimento irá reconhecer, por exemplo, que as regiões definidas pelos conjuntos {T1,U1} e {T2,U2} são idênticas, dando origem ao símbolo B da figura 3.12(c). Na formação de um novo símbolo, o programa substitui os exemplares que compõem a região pelos seus respectivos conteúdos (a operação de expansão apresentada na

seção anterior). Por fim, o procedimento divide é aplicado a cada um dos novos símbolos criados, num processo recursivo que termina quando é encontrado um símbolo que não contém sub-símbolos.

(a)

(b)

Figura 3.13 Definição das regiões disjuntas

Além da transformação disjunta, os autores implementaram um extrator de circuitos hierárquico, que trabalha a partir da especificação sem sobreposições produzida na transformação. Eles experimentaram o programa com algumas pastilhas, obtendo os resultados apresentados na tabela da figura 3.14, extraída de seu trabalho. A regularidade mostrada é a da especificação produzida na transformação. Os resultados mostram que o método pode trazer benefícios mesmo com circuitos pequenos. Mostram ainda que o desempenho do programa de extração está relacionado à regularidade da especificação examinada, acompanhando seu crescimento.

Circuito			Tempos (min:seg)	
Nome	#transistores	regularidade	convencional	hierárquica
powtrans	2	5.7	1:33	0:44
adder	848	4.4	4:38	0:39
cherry	881	2.8	2:37	3:01
raluchip	1853	3.0	10:35	1:53
fifo	8082	14.2	26:54	3:00
testram	20480	457.7	62:48	5:46

Figura 3.14 Experimentos com
extração de circuitos

CAPITULO 4

UM PRÉ-PROCESSADOR PARA ANÁLISE HIERÁRQUICA

No início deste nosso estudo, tencionávamos implementar um verificador de regras de projeto hierárquico. No entanto, a medida em que aprofundamos nossa pesquisa, a proposta foi alterada. O interesse passou a residir, não numa ferramenta específica, mas em métodos gerais de tratamento de hierarquia. Este capítulo apresenta nossas reflexões sobre o assunto, na forma de um projeto de uma ferramenta hierárquica de uso geral, contendo propostas de nossa autoria.

4.1 ABORDAGEM

Especificações hierárquicas sem restrições podem ser difíceis de processar. Vimos, no capítulo 3, discussões nesse sentido, e também alguns experimentos com casos de desempenho bastante insatisfatório. Além disso, alguns dos métodos genéricos vistos produzem uma nova hierarquia de símbolos, diferente da especificada pelo projetista, o que pode dificultar a utilização das ferramentas. Por exemplo, um verificador de regras de projeto pode apontar erros em símbolos criados pelo sistema,

desconhecidos do projetista, dificultando sua localização.

Abordagens restritivas não apresentam esses problemas. Elas possibilitam o uso de ferramentas de bom desempenho e trabalham com a hierarquia especificada pelo usuário. Por outro lado, elas podem ser inaceitáveis, pois impõem ao projetista um estilo particular de projeto, ou dele exigem o fornecimento de informações complementares.

Adotamos no nosso projeto um método não restritivo, em que procuramos dar atenção aos problemas acima apontados, tentando amenizá-los. O método implementa a análise de um projeto em dois passos. No primeiro, um pré-processador transforma a especificação do circuito numa especificação que não contém sobreposições. Esta, no segundo passo, é submetida a alguma ferramenta particular de análise. O projeto diz respeito apenas ao primeiro passo, o pré-processador. Análises subsequentes podem ser realizadas da maneira descrita na seção 3.2 .

4.2 O PRÉ-PROCESSADOR

O pré-processador tem como entrada uma especificação de um circuito em CIF, sem qualquer tipo de restrição. Ele produz uma outra especificação, do mesmo circuito, também em CIF. Nesta, cada símbolo é delimitado por uma fronteira "manhattan" poligonal que não se intercepta com a de nenhum outro símbolo, nem com figuras geométricas que não façam parte do próprio símbolo. Como vimos na seção 3.2, uma especificação CIF com

essas características pode ser processada sem dificuldade por ferramentas adequadas de análise hierárquica.

Há uma série de vantagens na utilização deste esquema. Qualquer especificação CIF pode ser tratada, não importando quais metodologias e sistemas de desenvolvimento foram utilizados na sua produção. Há uma total independência entre o pré-processador e os programas de análise. A saída do pré-processador pode ser submetida a ferramentas que manejem apenas especificações sem sobreposições, ampliando assim o universo de uso dessas ferramentas, que passam indiretamente a tratar projetos quaisquer. Por outro lado, os programas de análise não ficam amarrados ao pré-processador, podendo ser aplicados diretamente a especificações CIF (este não é o caso, por exemplo, do método de Hon, que tem uma interface particular entre o primeiro e o segundo passos).

Para o programa, um símbolo é definido pelos exemplares e figuras que o compõem e também por um polígono de fronteira que ele próprio estabelece. Esse polígono define para o programa a região ocupada pelo símbolo. Dessa maneira, o programa considera a intersecção entre dois símbolos como sendo a intersecção entre seus polígonos de fronteira. Temos em mente uma implementação que trabalha apenas com polígonos "manhattan". Inicialmente, o programa estabelece o polígono de fronteira de cada símbolo como sendo o menor retângulo que o contém. Assim, símbolos originais tem fronteiras retangulares. Símbolos criados pelo programa, entretanto, tem polígonos de fronteira "manhattan" arbitrários.

O programa principal está apresentado na figura 4.1. No início, ele compila a especificação CIF do circuito, gerando uma representação interna que reflete diretamente a estrutura de símbolos da especificação. Como em CIF não há um "símbolo principal", os exemplares e figuras do nível mais alto da hierarquia são reunidos em símbolo interno, denotado por 50 na figura, que representa o circuito como um todo. 50 é então submetido ao procedimento transforma. Concluída sua execução, a especificação de saída estará pronta, restando apenas imprimi-la.

```
principal():  
    inicializa()  
    transforma(50)  
    imprime()
```

Figura 4.1 Programa principal

O algoritmo de transformação está apresentado na figura 4.2. O cerne do processo reside no procedimento particiona. Ele toma um símbolo S, definido em termos de exemplares de símbolos que possivelmente se interceptam, e o reestrutura, obtendo uma definição de S que não mais contém sobreposições. O processo porém não está completo, pois os símbolos que agora compõem S, embora não se interceptem entre si, podem eles próprios conter sobreposições. Aplicando-se o procedimento de transformação recursivamente a cada um deles, obtém-se, como resultado final, uma especificação de S em que nenhum símbolo da hierarquia contém sobreposições - que é o resultado desejado. O símbolo é

marcado para evitar a repetição do procedimento com o mesmo símbolo.

```
transforma(s):  
    se s já foi transformado, retorna  
    particiona(s)  
    para cada exemplar t de s  
        transforma(t)  
    marca S como transformado
```

Figura 4.2 Algoritmo de transformação

4.3 ESTRATÉGIA DE PARTICIONAMENTO

Numa abordagem em dois passos como a que adotamos, a estratégia de particionamento de símbolos tem um papel muito importante. É ela que determina a hierarquia produzida pelo pré-processador, e a qualidade desta, por sua vez, influencia fortemente o desempenho dos programas de análise.

Não se sabe muito a respeito de como caracterizar especificações hierárquicas. A qualidade não é uma propriedade intrínseca da especificação, dependendo antes da tarefa a ser realizada, de modo que especificações com determinadas características favorecem determinados tipos de análise. Não obstante, propriedades como regularidade e hierarquização são importantes na qualificação de descrições. Os testes realizados por Newell e Fitzpatrick[82] indicam que o desempenho de seu extrator de

circuitos cresce junto com a regularidade da especificação analisada. Hon[83] afirma que, as demais condições mantidas iguais, tanto o aumento da regularidade quanto o da hierarquização facilitam a realização de análises hierárquicas.

A regularidade está relacionada à repetição de símbolos. Na medida em que se analisa cada símbolo distinto uma única vez, é natural associar regularidade e desempenho. A hierarquização está relacionada à estrutura da descrição. Um aumento na hierarquização leva a uma redução no número de interfaces entre símbolos que se deve analisar e, conseqüentemente, no tempo de execução. Consideremos um símbolo S, composto por oito exemplares de um símbolo A; podemos especificá-lo chamando A oito vezes (figura 4.3(a)) ou, de modo mais hierárquico, agrupando dois exemplares de A num símbolo B, dois de B num símbolo C, e dois de C para formar S (figura 4.3(b)). Enquanto sete interfaces devem ser analisadas na primeira especificação, a segunda requer a de apenas três.

Outro aspecto que se deve considerar na avaliação da descrição produzida pelo programa é a sua similaridade com a hierarquia original. A semelhança entre ambas pode facilitar o uso subsequente de ferramentas de análise. Além disso, consideramos que, em geral, a especificação desenvolvida pelo projetista é bem estruturada. Assumindo esse ponto de vista, preservar a hierarquia original significa assegurar a geração de uma especificação de qualidade.

(a)

(b)

Figura 4.3 Especificações de um símbolo com diferentes graus de hierarquização

A estratégia de partição utilizada no projeto é baseada no método Heurístico de Hon[83]. A diferença com este é que optamos por trabalhar com fronteiras de símbolos "manhattan" poligonais, e não retangulares como no método original.

A motivação para a introdução dessa—modificação pode ser entendida com o seguinte exemplo. Com o método de Hon, uma situação como a da figura 4.4(a) exige que um dos exemplares A, B ou C seja expandido, pois o retângulo de fronteira do grupo formado por A e B intercepta C (figura 4.4(b)). Trabalhando com fronteiras "manhattan" poligonais não há conflito, A e B podem

ser agrupados em um símbolo que não tem interferência com C (figura 4.4(c)). Gostaríamos de acrescentar que o exemplo não é artificial. Para que o projetista possa aproveitar ao máximo a área da pastilha, construções como a ilustrada são necessárias, uma vez que dificilmente se consegue combinar retângulos de diferentes tamanhos em um outro retângulo, sem que haja perda de área.

Figura 4.4 Forma da fronteira
e particionamento

A diferença entre as hierarquias produzidas pelos dois métodos tem origem nas expansões adicionais que são realizadas com o método original. Em primeiro lugar, não há razão estratégica para a realização dessas expansões; trabalhando-se com polígonos quaisquer, elas se tornam arbitrárias. Além disso, expansões modificam a estrutura presente na descrição de partida. Evitá-las é uma maneira de buscar a preservação da hierarquia original. Por fim, parece-nos que expansões tendem a reduzir o grau de hierarquização da especificação. Uma desvantagem da estratégia proposta é que é mais fácil escrever programas que

trabalhem apenas com retângulos. Os trabalhos de Scheffer[81] e de Newell&Fitzpatrick[82] mostram, contudo, que não há maiores problemas em se desenvolver ferramentas de análise que manipulem polígonos "manhattan" arbitrários.

O ALGORITMO

Dado um símbolo S , unindo-se os polígonos de fronteira de cada um dos exemplares de símbolos que compõem S , obtém-se um conjunto de regiões poligonais conexas que não se interceptam entre si. Vamos chamar de grupo o conjunto de exemplares que compõem uma tal região. A cada uma dessas regiões podemos associar um símbolo, definido pelos exemplares que a formam mais as porções das figuras especificadas em S que a interceptam. Com isso, S pode ser definido por meio de exemplares desses símbolos, mais as porções de suas figuras não incorporadas em nenhum deles. Essa especificação de S satisfaz as condições desejadas: 1) não há sobreposição entre exemplares de símbolos, pois seus polígonos de fronteira são regiões que não se interceptam; e 2) não há sobreposição de figuras e exemplares, pois as porções das figuras que interceptavam os exemplares foram incorporadas nas definições dos símbolos.

A figura 4.5 ilustra o processo. Um símbolo S é, na sua definição original, composto por alguns exemplares e figuras geométricas (estas estão representadas com traço irregular). Há duas regiões conexas disjuntas, uma formada pelos exemplares S_1 e S_2 e a outra por S_3 , S_4 e S_5 .

(a) definição original

(b) definição produzida

Figura 4.5 Particionamento de um símbolo

Cada um desses dois grupos de exemplares, junto com as porções das figuras que os interceptam, irá compor a definição de um novo símbolo, T1 e T2 respectivamente. S passa então a ser definido com exemplares desses símbolos e com as porções remanescentes de suas figuras.

O processo que descrevemos é uma maneira de obter uma especificação de um símbolo em que não há sobreposições. Suponhamos, porém, que o símbolo dado contenha uma única região conexa, como é o caso do símbolo T1 da figura 4.5(b). O procedimento irá montar um símbolo T1', que é o mesmo que T1, e redefinir T1 como sendo um exemplar de T1'. Mais à frente, o mesmo irá acontecer com T1' e, assim por diante, o procedimento irá indefinidamente redefinir o símbolo por meio dele próprio. O processo não é efetivo.

O algoritmo apresentado na figura 4.6 implementa o particionamento do modo descrito, garantindo porém que a transformação seja efetiva. Inicialmente, são expandidos os exemplares de símbolos "grandes", isto é, aqueles que têm fronteira igual à do símbolo S dado. Estes exemplares terão que ser necessariamente expandidos, uma vez que interceptam todos os outros, formando sempre uma única região. O procedimento grupos calcula os grupos de exemplares que compõem cada uma das regiões formadas pela união dos exemplares de sub-símbolos de S, retornando o número de grupos obtidos. Se houver uma única região, o programa seleciona um exemplar de símbolo para ser expandido, realiza a expansão e volta a calcular os grupos. Esse processo para quando: 1) houver duas ou mais regiões; ou 2)

não houver nenhuma região - isto é, S contém apenas figuras (note-se que o procedimento grupos retorna zero neste caso). O procedimento redefine associa então um símbolo a cada uma das regiões, contendo os exemplares que compõem a região e as porções das figuras de S que a interceptam. O programa verifica se já existe um símbolo com esse conteúdo. Se existir, ele é usado; senão, um novo símbolo é criado. Esses símbolos são então utilizados na redefinição de S . A figura 4.7 ilustra o algoritmo.

```

particiona(s):
    para cada exemplar t de s com fronteira igual à de s
        expande(t)
    enquanto grupos(s) = 1
        t ←-- seleciona(s)
        expande(t)
    redefine(s)

```

-Figura 4.6 Algoritmo de particionamento

Notemos que, com esse algoritmo, o programa se torna efetivo. De fato, as hierarquias dos símbolos que compõem a nova definição de S são menores que a de S , pois são subconjuntos desta e há pelo menos duas delas. Isso conduz à obtenção de símbolos com hierarquias cada vez menores, até chegar àqueles com hierarquia vazia (sem sub-símbolos), quando então termina a recursão do processo de transformação (veja figura 4.2).

Figura 4.7 Passos do particionamento

4.4 HEURISTICA DE SELEÇÃO

O procedimento de seleção deve escolher um dos exemplares que compõem um dado símbolo para expansão. Como fazer essa escolha? Que características dos exemplares devem ser levadas em conta? Em seu trabalho, Hon[83] apresenta uma série de heurísticas, descritas na seção 3.4, para realizar a seleção. No conjunto, elas procuram implementar uma estratégia que consiste em retardar a expansão de matrizes de símbolos e, uma vez que isso

tenha que ser feito, expandir primeiro os menos complexos.

Propomos aqui a utilização do grau de regularidade dos símbolos nos critérios de seleção, basicamente dando preferência à expansão de exemplares de símbolos de menor regularidade. A regularidade é uma propriedade que reflete a estruturação do símbolo como um todo. Selecionando os símbolos menos regulares, a heurística procura preservar as estruturas mais importantes da hierarquia original.

Há diversas situações em que a heurística apresenta um comportamento que nos parece adequado. Por exemplo, se o exemplar de um símbolo que contém sub-símbolos intercepta um que só contém figuras, será expandido este último. O novo símbolo que daí resulta terá basicamente a estrutura do primeiro, sendo apenas acrescido de parte do traçado do exemplar expandido. De modo geral, matrizes tendem a ser preservadas, uma vez que apresentam um bom grau de regularidade. Entre duas matrizes, será escolhida para expansão a que tem menos elementos.

O procedimento de seleção utiliza apenas a seguinte heurística:

selecione o exemplar de símbolo de menor regularidade envolvido no maior número de sobreposições.

Observemos inicialmente que todos os sub-símbolos do símbolo dado estão envolvidos em alguma intersecção, uma vez que há um único grupo de símbolos. Assim, todos são candidatos à seleção. Toma-se então o símbolo menos regular. Se houver mais de um símbolo com a regularidade mínima, ou se houver mais de um

exemplar do símbolo escolhido, seleciona-se o exemplar envolvido no maior número de intersecções. Este critério é baseado nas heurísticas de Hon. Se ainda houver dois ou mais exemplares passíveis de escolha, deve ser utilizado um critério uniforme de desempate, de modo a selecionar, em situações idênticas, sempre o mesmo exemplar. Por exemplo, entre exemplares de símbolos distintos pode-se escolher o que tem menos figuras, ou mesmo o de nome menor. Entre exemplares de um mesmo símbolo, pode-se utilizar sua posição relativa - selecionando-se o exemplar mais à esquerda, por exemplo.

4.5 ASPECTOS DE IMPLEMENTAÇÃO

Consideremos o exemplo da figura 4.8. Um símbolo S, originalmente composto por dois exemplares de um símbolo T e um exemplar de V, é particionado de forma que passa a ser especificado por um exemplar de T e um exemplar de N, um novo símbolo criado pelo programa, formado por uma ocorrência de T e uma de V (figura 4.8(a)). T é então submetido ao processo de transformação. Suponhamos que, de uma especificação inicial com exemplares T1, T2 e T3, se chegue a uma especificação com exemplares de símbolos novos N1 e N2, como mostra a figura 4.8(b). Chega a vez de N ser transformado e aqui surge nossa questão. Suponhamos que temos guardado ambas as descrições de T, a original e a final. Qual delas utilizar no processo de transformação de N? Observemos que o uso de uma ou outra conduz a situações diferentes; por exemplo, a figura 4.8(c) mostra o resultado da expansão de T utilizando-se cada uma das descrições.

(a) particionamento de S

(b) transformação de T

(c) expansão de T na transformação de N

Figura 4.8 Descrição original x descrição final

Passamos a discutir uma implementação que obtém uma série de benefícios do uso das especificações originais dos símbolos. Para começar, gostaríamos de mostrar um fato importante para esta nossa abordagem, relacionado ao algoritmo de particionamento:

se a hierarquia completa de um símbolo S contém apenas símbolos originais, a especificação dos símbolos criados pelo procedimento particiona(S) também só referencia símbolos originais.

Consideremos o algoritmo, apresentado na figura 4.6. O programa realiza uma série de expansões. Uma expansão simplesmente troca o exemplar de um símbolo pelos exemplares e figuras que o compõem. Assim sendo, após as expansões a hierarquia que define o símbolo ainda contém apenas símbolos originais. S é então redefinido e aqui, eventualmente, alguns novos símbolos serão criados. Estes símbolos são definidos por um grupo de exemplares de sub-símbolos de S e algumas figuras; como a hierarquia que define S ainda contém apenas símbolos originais, a especificação dos símbolos criados também será composta apenas por símbolos originais.

Este fato nos permite fazer uma implementação como segue. Temos sempre armazenada a descrição original de cada símbolo. Para os símbolos originais, essa descrição é a de entrada. Para os criados pelo programa, é a descrição com que foram definidos na criação. Fazendo com que o procedimento de particionamento trabalhe sempre com a hierarquia dada pela descrição original de cada símbolo, podemos concluir, por indução e pelo que vimos

acima, que, também para os símbolos criados pelo programa, essa descrição só referencia símbolos originais. Quando, no processo de particionamento, um símbolo é redefinido em termos de exemplares de sub-símbolos que não se sobrepõe, esta nova especificação é armazenada na descrição disjunta do símbolo. Esta é a descrição que se deseja do símbolo, a que não contém sobreposições. Ela é utilizada quando o procedimento de transformação é recursivamente chamado para cada exemplar que compõe o símbolo, e também para a impressão da especificação de saída do programa.

Assim, retomando o exemplo da figura 4.8, a especificação de saída do programa apresentará T como sendo composto por exemplares dos símbolos N1 e N2 (sua descrição disjunta) e estes são os símbolos que, na transformação de T, terão sofrido o processo de transformação recursivo. Já no particionamento de N, de cuja descrição inicial faz parte um exemplar de T, terá sido utilizada a descrição original de T - aquela formada por T1, T2 e T3.

A utilização desse esquema traz uma série de benefícios, não só para a própria implementação como também na funcionalidade do pré-processador. Um primeiro e maior benefício diz respeito à definição dos símbolos gerados pelo programa. Como apontamos no início deste capítulo, a produção de uma hierarquia diferente da especificada pelo projetista pode dificultar a utilização das ferramentas de análise. Com esta nossa implementação, o pré-processador pode produzir uma documentação relacionando os símbolos criados pelo programa com símbolos definidos pelo

projetista, bastando para isso apresentar a descrição original dos símbolos criados.

O programa pode utilizar a descrição original para verificar se já existe um símbolo com um determinado conteúdo, o que permite uma identificação mais abrangente de símbolos iguais. Suponhamos que os símbolos apresentados na figura 4.8 façam parte de uma especificação mais ampla. Podemos imaginar que exemplares de T e V componham a definição de algum outro símbolo, na mesma justaposição em que aparecem em S. A utilização da descrição original nos permite reconhecer essa composição como sendo o símbolo N. Se não a utilizássemos, essa identificação não seria possível, uma vez que N, depois de transformado, não mais estaria definido em termos de T e V. Além disso, a implementação de uma técnica de "hashing" para agilizar a busca seria mais trabalhosa, pois a função de "hashing", sendo baseada no conteúdo do símbolo, mudaria quando ele fosse redefinido.

O esquema permite ainda que a implementação trabalhe, de fato, apenas com retângulos. As fronteiras dos símbolos originais são retângulos (estabelecidas na inicialização). Como as descrições iniciais contém apenas símbolos originais, o procedimento de particionamento terá que manipular apenas fronteiras retangulares. Isso pode ser visto na figura 4.8(c). Com a utilização da descrição original, a expansão de T resulta em símbolos com fronteira retangular. Com a descrição final, o particionamento teria que manipular fronteiras não retangulares - no caso, a de N2. Um raciocínio análogo vale em relação ao

cálculo da regularidade, que é utilizada como critério de seleção de símbolos a expandir. Os símbolos candidatos a expansões são sempre originais. Assim, pode-se calcular a regularidade apenas para eles, no início do programa.

CAPITULO 5

CONCLUSÃO

Estudamos neste trabalho o problema da análise hierárquica de projetos de circuitos integrados. Visando proporcionar aos projetistas de circuitos ferramentas geométricas de melhor qualidade, diversos trabalhos tem sido realizados no sentido de explorar a estrutura hierárquica das especificações. Vimos que o problema central na análise hierárquica é o tratamento de sobreposições. Como é muito difícil manipular sobreposições, uma maneira de abordar o problema consiste em, primeiro, eliminá-las, para então realizar as análises desejadas. Seguindo essa linha, elaboramos o projeto de um pré-processador cuja função consiste precisamente em transformar uma especificação qualquer de um circuito numa especificação livre de sobreposições.

O projeto apresenta algumas idéias elaboradas por nós:

- . a especificação produzida pelo pré-processador está na mesma linguagem (CIF) da especificação de entrada, o que proporciona uma maior independência entre o pré-processador e os programas de análise subsequentes

- . o algoritmo de particionamento, baseado na estratégia Heurística de Hon[83], trata fronteiras "manhattan" poligonais, o que, nos parece, deve conduzir à produção de especificações de melhor qualidade
- . a heurística de seleção de símbolos para expansão é baseada na regularidade dos símbolos; ela é simples, de fácil computação e tem uma base conceitual na sua formulação
- . o pré-processador preserva a descrição original dos símbolos, o que lhe possibilita produzir uma documentação relacionando os símbolos criados pelo programa com símbolos conhecidos do projetista; outros benefícios, para a própria implementação, são também obtidos

O objetivo principal do estudo de métodos hierárquicos é o de reduzir o consumo de recursos computacionais empregados na análise do traçado de circuitos. Os trabalhos que estudamos nos mostram que os resultados que se poderia imaginar ainda não foram alcançados. Não obstante, alguns métodos apresentaram desempenhos bastante bons, revelando claramente sua superioridade em relação a métodos convencionais. Os resultados obtidos mostram ainda que, de modo geral, especificações bem hierarquizadas e regulares são mais eficientemente processadas. Isso constitui mais um argumento a favor da utilização de metodologias de projeto estruturado.

Um possível caminho para a continuidade deste trabalho é realizar um estudo comparativo das estratégias de particionamento apresentadas, implementando os algoritmos e experimentando-os com projetos de circuitos reais. Pode-se

também tentar desenvolver novas estratégias de particionamento. Nesse sentido, uma idéia que tivemos e não exploramos consiste em, quando dois símbolos se interceptam, realizar o particionamento de modo a retirar de um deles a sua porção que está na intersecção, incorporando-a ao outro símbolo. As porções dos símbolos menos regulares seriam incorporadas aos símbolos mais regulares. Uma outra linha de pesquisa, mais ambiciosa, consiste em procurar desenvolver um modelo estatístico de especificações de circuitos, projetando um algoritmo que apresente um bom desempenho nos casos esperados.

Por fim, gostaríamos de colocar nossa expectativa sobre a continuidade de uso de ferramentas geométricas, face ao desenvolvimento de compiladores de silício. Tem havido um esforço muito grande de pesquisa em busca de sistemas que, a partir da especificação de um circuito numa linguagem de alto nível, produzam uma especificação correta de seu traçado. Em princípio, a utilização de tais sistemas elimina a necessidade de ferramentas geométricas. Acreditamos porém que, pelo menos para determinados tipos de projetos - como, por exemplo, o de pastilhas que serão produzidas em muito larga escala - ferramentas geométricas serão úteis ainda por bastante tempo.

BIBLIOGRAFIA

Baird, H.S. [78]

Fast Algorithms for LSI Artwork Analysis

Journal of Design Automation and Fault-Tolerant Computing,
vol. 2, n. 2, 1978, pp. 179-209

Baker, C.M. [80]

Artwork Analysis Tools for VLSI Circuits

Master's Thesis, Massachusetts Institute of Technology,
Laboratory for Computer Science, 1980

Bentley, J.L., Haken, D. e Hon, R.W. [80]

Statistics on VLSI Designs

Technical Report CMU-CS-80-111, Carnegie-Mellon University,
Computer Science Department, 1980

Bentley, J.L. e Ottmann, T. [81]

The Complexity of Manipulating Hierarchically Defined Sets
of Rectangles

Technical Report CMU-CS-81-109, Carnegie-Mellon University,
Computer Science Department, 1981

Bentley, J.L. e Wood, D. [79]

An Optimal Worst-Case Algorithm for Reporting Intersections
of Rectangles

Technical Report CMU-CS-79-122, Carnegie-Mellon University,
Computer Science Department, 1979

Chao, S., Huang, Y. e Yam, L.M. [80]

A Hierarchical Approach for Layout versus Circuit
Consistency Check

Proceedings of the 17th Design Automation Conference, 1980,
pp. 270-276

Hon, R.W. [83]

The Hierarchical Analysis of VLSI Designs

Ph.D. Thesis, Columbia University, Department of Computer
Science, 1983

Hon, R.W. e Séquin, C. [80]

A Guide to LSI Implementation

2nd Edition, Xerox PARC Technical Report SSL-79-7, 1980

Johnson, S.C. [82]

Hierarchical Design Validation Based on Rectangles

Proceedings of 1982 Conference on Advanced Research in VLSI,
MIT, 1982, pp. 97-100

Lattin, W. [79]

VLSI Design Methodology: The Problem of the 80's for
Microprocessor Design
Proceedings of Caltech Conference on VLSI, 1979, pp. 247-252

Lauther, U. [78]

4-Dimensional Binary Search Trees as a Means to Speed up
Associative Searches in Design Rule Verification of
Integrated Circuits
Journal of Design Automation and Fault-Tolerant Computing,
vol. 2, n. 3, 1978, pp. 241-247

Liesenber, H.K.E. [86]

Projetos VLSI e seu Suporte Computacional
V Escola de Computação, Belo Horizonte, 1986

McCreight, E.M. [80]

Efficient Algorithms for Enumerating Intersecting Intervals
and Rectangles
Xerox PARC Technical Report CSL-80-9, 1980

Mead, C.A. e Conway, L.A. [80]

Introduction to VLSI Systems
Addison-Wesley Publishing Company, 1980

Newell, M.E. e Fitzpatrick, D.T. [82]

Exploiting Structure in Integrated Circuit Design Analysis
Proceedings of 1982 Conference on Advanced Research in VLSI,
MIT, 1982, pp. 84-92

Scheffer, L.K. [81]

A Methodology for Improved Verification of VLSI Designs
without Loss of Area

Proceedings of Caltech Conference on VLSI, 1981, pp. 299-309

Scheffer, L.K. e Soetarmen, R. [85]

Hierarchical Analysis of IC Artwork with User Defined
Abstraction Rules

Proceedings of IEEE 22nd Design Automation Conference, 1985,
pp. 293-298

Ullmann, J.D. [84]

Computational Aspects of VLSI
Computer Science Press, 1984

Wagner, T.J. [84]

Hierarchical Layout Verification

Proceedings of IEEE 21st Design Automation Conference, 1984,
pp. 484-489

Whitney, T. [81]

A Hierarchical Design-Rule Checking Algorithm

Lambda Magazine, First Quarter 1981, pp. 40-43