PARALELIZANDO A FASE DE ROTEAMENTO DE CIRCUITOS BASEADOS EM FPGAs

Lucídio dos Anjos Formiga Cabral

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Nelson Maculan Filho, Ph.D.

Prof. Márcia Helena Costa Fampa, D.Sc.

Prof. Abílio Pereira de Lucena Filho, Ph.D.

Prof. Lúcia Maria de Assumpção Drummond, D.Sc.

Prof. Sérgio Bampi, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2001

CABRAL, LUCIDIO DOS ANJOS FORMIGA

Paralelizando a Fase de Roteamento de Circuitos baseados em FPGAs [Rio de janeiro] 2001

XIII, 149 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2001)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Roteamento de Circuitos baseados em FPGA.

2. Paralelismo.

3. Ávores de Steiner.

I. COPPE/UFRJ II. Título (série).

Aos meus pais, Antonio de Lisboa e Socorro.

A minha esposa, Ana Maria.

A minha filha, Ana Luíza.

Agradecimentos

O melhor da vitória
é compartilhá-la.

Gostaria de agradecer a todos que de alguma maneira estiveram presentes nos meus momentos de alegria e de angústia. Em particular, a minha esposa Ana Maria, que tem sido sempre o meu referencial de apoio e carinho.

Ao longo destes útimos quatro anos pude crescer como pessoa e como pesquisador, graças ao convívio enriquecedor com colegas, amigos e professores a quem agradeço sinceramente.

Em especial, aos meus orientadores, professor Nelson Maculan, a quem agradeço o exemplo, a amizade, a generosidade, o apoio e os auxílios para congressos e livros, e professor Júlio Salek Aude (in memorium), a quem serei eternamente grato pela amizade e pelas liçoes de vida transmitidas, naturalmente, no seu modo ímpar de ser e agir, quer fosse na maneira competentíssima e apaixonada com que se entregava ao trabalho, quer fosse no trato sempre gentil para com seus alunos e colegas, qualidades que se aliavam à sua integridade e que o tornaram um exemplo para todos aqueles que como eu tiveram o privilégio de desfrutar do seu convívio.

Aos meus velhos e novos amigos, sou grato pelo carinho, pelas conversas descontraídas, pela atenção e ... pelos ouvidos. Em particular aos amigos Flávio, Adriana, Prudente, Saula, Digna, Luiz, Conceição, Roberto, Ana, Ary e Carla, que tornaram mais leves e felizes estes quatro anos de trabalho.

Aos amigos Manoel Bezerra Campelo Neto e Marcone Jamilson Freitas Souza agradeço a amizade sincera, digna de irmãos, e os muitos momentos de apoio.

Às meninas da secretaria e demais funcionários do Programa de Engenharia de Sistemas e Computação, agradeço pelo trabalho competente.

Por fim, devo agradecer a CAPES, pela bolsa concedida, e ao Departamento de Estatística da Universidade Federal da Paraíba , pela liberação para o curso.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

PARALELIZANDO A FASE DE ROTEAMENTO DE CIRCUITOS BASEADOS EM FPGAs

Lucídio dos Anjos Formiga Cabral

Março/2001

Orientadores: Nelson Maculan Filho
Júlio Salek Aude

Programa: Engenharia de Sistemas e Computação

Este trabalho está focado na fase de roteamento de circuitos baseados em *Field Programmable Gate Arrays* (FPGAs). O roteamento é uma das etapas que mais consome tempo de toda a fase de projeto, justificando assim o desenvolvimento de uma abordagem paralela. Este trabalho se baseia na percepção de um desacoplamento do grafo de roteamento, que é decorrente da imposição de restrições sobre uma dada arquitetura de FPGA, particionando assim seus recursos de roteamento em domínios disjuntos de trilhas.

Usando-se o paradigma de memória distribuída foi desenvolvida uma aplicação paralela, onde cada processador recebe um conjunto diferente de redes de sinal (*nets*) e o roteia em sua partição do grafo de roteamento. Resultados computacionais são apresentados e comparações realizadas com os melhores resultados existentes na literatura. Também foi desenvolvida uma nova heurística para o problema da árvore de Steiner mínima, visando a seu uso no roteamento em FPGAs. Formulações de programação inteira para o problema de roteamento global e suas relaxações são estudadas.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

PARALLELING ROUTING PHASE OF CIRCUITS BASED ON FPGAs

Lucídio dos Anjos Formiga Cabral

March/2001

Advisors: Nelson Maculan Filho
Júlio Salek Aude

Department: Computing and Systems Engineering

This work is focused on the routing phase of integrated circuits based on Field Programmable Gate Arrays (FPGA's). It is one of the most time consuming phases of the complete design process, so this has motivated the development of parallel implementations. The present work is based on the perception of a disjunction of the routing graph under some architectural constraint imposed on a particular FPGA architecture model that divides its routing resources into track domains.

A parallel application based on the distributed memory paradigm has been developed. In this application, each processor receives a different set of nets and route it within its partition of the routing graph. Computational results are presented and compared with the best results reported in the literature. Also, a new heuristic for the minimum Steiner tree problem has been developed targeting to its use in FPGA routing. Integer programming formulations of global routing and its lagrangean relaxations are studied.

Sumário

Lista de Figuras

Lista de Tabelas

Introdução

Nos últimos anos, *Field Programmable Gate Arrays* (FPGAs) têm se tornado largamente aceitos como uma alternativa atraente para a implementação de circuitos digitais de dimensões consideráveis em um chip VLSI personalizado. Uma grande variedade de estilos FPGA estão comercialmente disponíveis e um dos tipos mais importantes é a arquitetura baseada em *arrays*, que consiste em linhas e colunas de blocos lógicos com canais horizontais e verticais de roteamento entre as colunas. Essa arquitetura foi inicialmente introduzida pela Xilinx, em [24] e mais tarde em [45] e [44]. Variações da arquitetura baseada em *arrays* são encontradas também em FPGAs produzidos pela AT&T [71] e QuickLogic [79]. FPGAs baseados em *arrays* estão disponíveis com capacidade lógica muito alta, como, por exemplo, a pastilha XC2V10000 que integra a nova família Virtex-II de FPGAs comerciais da Xilinx, cuja capacidade é de 10.000.000 de portas lógicas (uma porta lógica é normalmente definida como uma célula com quatro transistores, a exemplo de uma porta NAND), e usa tecnologia CMOS $0,12\mu m$ com 8 camadas de metal e alimentação de 1,5V. Com dispositivos tão grandes, o projeto de interconexão nos canais de roteamento tem um impacto crucial tanto no percentual da capacidade lógica do chip que pode ser efetivamente utilizado, como no desempenho dos circuitos implementados no FPGA, além de demandar um custo computacional cada vez mais alto.

Nos primeiros FPGAs baseados em *arrays* [24] e [45], interconectar compreendia, na maior parte das vezes, unir segmentos curtos de fios que cobriam o comprimento ou a largura de um único bloco lógico através de *switches* programáveis disponíveis somente para interconectar os segmentos curtos. Estas arquiteturas consideram apenas a utilização eficiente dos segmentos de fio em termos de área, implicando que longas conexões necessitem passar através de diversos *switches* de roteamento em série, degradando severamente o desempenho em termos de velocidade. Isto decorre

do fato de que os *switches* de roteamento têm resistência em série e capacitância parasítica significativas. Para superar esses problemas, arquiteturas recentes contêm canais de roteamento segmentados que compreendem uma mistura de segmentos de fios curtos e longos. Se ferramentas de CAD utilizarem cuidadosamente esses segmentos de tamanhos variáveis na implementação de circuitos, a introdução de canais de roteamento segmentados pode elevar bastante o desempenho de velocidade dos circuitos [57].

É óbvio que a implementação de qualquer circuito não trivial em um FPGA complexo requererá ferramentas de CAD sofisticadas. Um sistema de projeto típico [97], [77], [93] deverá incluir suporte para os seguintes passos: entrada do projeto inicial, otimização lógica, simulação, mapeamento tecnológico, posicionamento e roteamento. Este trabalho enfoca o estágio final do processo de CAD, investigando a maioria dos aspectos importantes associados com o roteamento em FPGAs baseados em *arrays*. Deste modo, questões como utilização efetiva dos recursos de interconexão disponíveis no FPGA e desempenho de velocidade, do ponto de vista de que todos os comprimentos de fios tendam para o mínimo comprimento, são discutidos.

A estratégia usual de roteamento é composta de dois passos: roteamento global (onde a topologia de cada rota é definida) seguido pelo roteamento detalhado. O roteador global aloca cada uma das conexões requeridas em um circuito a canais de roteamento específicos, e então o roteador detalhado aloca um segmento de fio do FPGA e *switch* de roteamento dentro dos canais para completar as conexões. O problema de roteamento é explosivamente combinatório podendo ser demonstrado ser NP-difícil pela sua equivalência ao problema de coloração de grafos [112]. Deste modo, algoritmos aproximativos têm sido desenvolvidos para este problema, em sua maioria basedos em heurísticas de busca por caminhos, e alguns poucos baseados em heurísticas para encontrar árvores de Steiner retilíneas.

A contribuição principal deste trabalho reside em apresentar uma estratégia de paralelização da fase de roteamento em FPGAs que é capaz de gerar bons *speedups*, muito superiores aos obtidos pela paralelização do roteador sequencial *Pathfnder* [25] e [26]. Além disso, esta estratégia de paralelização proposta possui um custo de comunicação extremamente baixo que permite a obtenção de excelentes *speedups* mesmo com o aumento do número de processadores, o que não ocorre com a pa-

ralelização do Pathfinder, cujo custo de comunicação cresce bastante, degradando o desempenho. Outro aspecto extremamente interessante desta proposta é que ela também manterá seu nível de desempenho independentemente do crescente aumento da capacidade lógica dos FPGAs, ao contrário da paralelização do *Pathfnder* em que o aumento do tamanho do grafo de roteamento produz um aumento no custo de comunicação para atualização dos custos de congestionamento de cada nó do grafo, degradando significativamente seu desempenho.

Primeiramente é apresentado um estudo das arquiteturas de FPGA, e a partir de características arquiteturais particulares é derivada uma aplicação paralela que visa reduzir o alto custo computacional exigido pela fase de roteamento durante o ciclo de projeto. Isto é obtido mediante a inédita percepção de que impondo-se algumas restrições arquiteturais sobre uma arquitetura 2-D FPGA, baseada em *arrays*, pode-se obter um desacoplamento do grafo de roteamento em termos de segmentos de fio (*domínios de trilhas*). Com isto pode-se particionar o conjunto de segmentos de fios entre os processadores e, por conseguinte, também dividir a tarefa de rotear o conjunto de redes (*nets*) dentre estes processadores, que recebem acesso exclusivo a partições de pinos de entrada de blocos lógicos.

A segunda contribuição consiste numa compilação dos principais trabalhos relacionados a fase de roteamento em FPGAs, que é mostrada no capítulo 3, resumindo um conjunto de idéias presentes nos muitos algoritmos sequenciais e nos parcos algoritmos paralelos.

A terceira é a apresentação de uma nova heurística para o problema da árvore de Steiner retilínea, tendo em mente sua aplicação futura ao problema de roteamento em FPGAs. Esta se adequa muito bem às características deste problema, onde as redes (*nets*) têm em média menos de cinco terminais a serem interligados. Sua principal vantagem reside no seu menor custo computacional, uma vez que ela possui complexidade $O(n \log n)$ enquanto o algoritmo baseado em caminhos mínimos tem complexidade $O(kn^2)$, onde $k = |T|$ corresponde ao número de terminais e $T \subset V$ e $n = |V|$.

Além disso algumas idéias relativas a formulações de programação inteira e suas possíveis relaxações Lagrangeanas são mostradas e discutidas. A obtenção de uma estimativa mais precisa sobre o número mínimo de trilhas necessárias para rotear um

dado circuito é crucial para que o roteamento sequencial busque o melhor roteamento experimentando apenas alguns poucos valores de número de trilhas. Isto faria com que a melhor iteração fosse alcançada mais rapidamente. Esta estimativa também é importante para que se possa definir o número inicial de trilhas em cada processador escravo na estratégia de paralelização proposta. Finalmente, apresentamos algumas idéias não consolidadas, que poderão nortear futuros trabalhos de pesquisa.

Capítulo 1

Uma Visão Geral do Projeto VLSI

Atualmente o projeto de circuitos integrados demanda grande esforço em desenvolvimento de ferramentas CAD para VLSI (*Very Large Scale Integration*), face ao patamar de complexidade alcançado. A tecnologia de VLSI tem sido usada com sucesso no desenvolvimento de microprocessadores, processadores de sinal, memórias de grande capacidade, controladores de memória e de entrada/saída (E/S), e muitos outros dispositivos. O atual nível de domínio tecnológico permite que hoje sejam construídos chips com centenas de milhões de transistores encapsulados.

Circuitos integrados neste nível de complexidade só são possíveis de serem desenvolvidos com a assistência de programas de computadores durante todas as fases do processo de projeto. Estes programas automatizam muitas das tarefas de projeto. O conjunto de programas e de ferramentas de software utilizado para auxiliar ou automatizar o processo de projeto de circuitos integrados é conhecido como Sistema de CAD (*Computer Aided Design*)[94].

Com o nível de integração praticamente dobrando a cada dois anos e com avanços tecnológicos paulatinos, o processo de fabricação incorpora naturalmente novos elementos, e portanto, diversos aspectos de projeto necessitam ser reavaliados, cujos reflexos devem estar presentes nas novas ferramentas de CAD. Como a complexidade de circuitos VLSI está na ordem de centenas de milhões de transistores, projetar um circuito VLSI é certamente uma tarefa muito complexa. Visando a reduzir tal complexidade, o processo de projeto é dividido em diversos níveis intermediários de abstração. A Figura 1.1 ilustra estes níveis relacionando-os aos seus respectivos passos de projeto.

1.1 Níveis de Abstração de Projeto

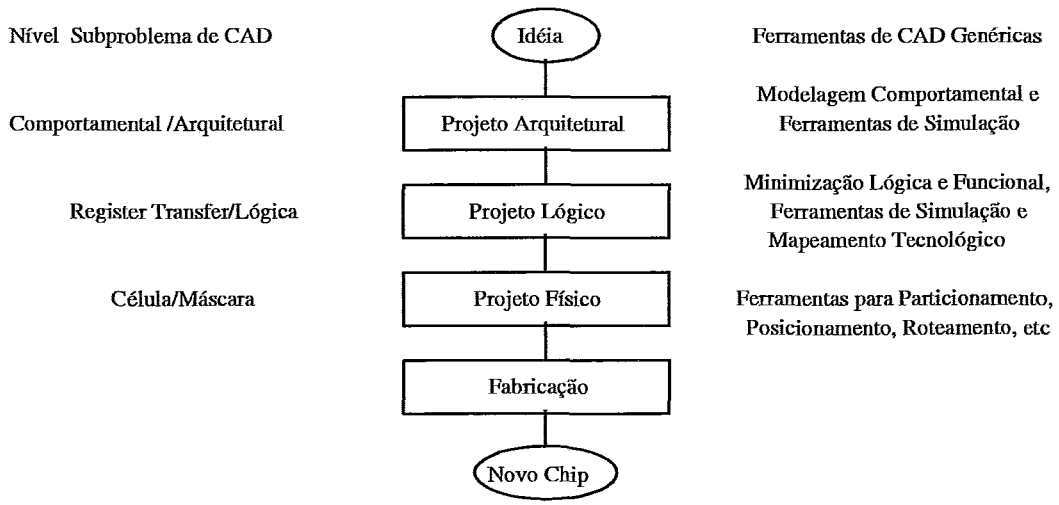

Figura 1.1: Níveis de abstração de projeto.

Como representado na Figura 1.1, cada passo de projeto é auxiliado por várias ferramentas de CAD, pois, na verdade, cada fase de projeto, ou seja, projeto arquitetural, lógico e físico, é composta de vários passos. Este trabalho está localizado na fase de projeto físico do circuito e investiga um dos seus passos, que diz respeito ao roteamento das redes entre os elementos de um circuito.

Na fase de projeto arquitetural é natural para o projetista pensar em termos de módulos maiores do circuito, tais como unidades aritméticas, unidades de memória, redes de interconexão e controladores. Por exemplo, suponha que um microprocessador esteja sendo projetado. Nesta fase são tomadas decisões relativas à arquitetura do microprocessador, tais como: Qual será o conjunto de instruções do processador? Quais serão os modos de endereçamento suportados? Haverá memória cache dentro do chip? Se sim, qual será o seu tamanho?

Estas são apenas algumas das muitas perguntas a serem respondidas nesta fase. Existem simuladores e ferramentas de medição de desempenho que ajudam a simular o comportamento da arquitetura especificada e a avaliar diferentes alternativas. Por exemplo, o tamanho da memória cache pode ser ajustado com o uso de uma ferramenta de simulação

Na fase de projeto lógico, ferramentas de síntese de alto nível e de síntese lógica

são utilizadas para transformar uma descrição comportamental do circuito em uma descrição estrutural. Ambas as descrições são, em geral, produzidas com o uso de uma linguagem de descrição de hardware (VHDL, Verilog, etc.). A descrição estrutural do circuito após a síntese lógica faz referência, na maioria dos ambientes de CAD, a elementos de uma biblioteca de módulos pré-projetados, chamados de macro células.

A fase de projeto físico é a fase que precede a fabricação de um circuito e inclui os seguintes passos: particionamento, geração da planta baixa (*floorplanning*), posicionamento (*placement*) e roteamento. O desempenho de um circuito, sua área, seu nível de produção e sua confiabilidade dependem sobremaneira do modo como o circuito é fisicamente implementado. Considere-se inicialmente o efeito do layout (posicionamento e roteamento) sobre o desempenho de um circuito. Num layout de um circuito integrado, metal e polisilício são usados para conectar dois pinos que sejam eletricamente equivalentes. Tanto as linhas de metal como as de polisilício introduzem impedância sobre os fios, incorrendo em atrasos que serão tanto maiores quanto maiores forem os tamanhos dos fios. Quando mais de uma camada de metal é usada no layout, existe uma outra fonte de impedância. Se uma conexão é implementada parcialmente na camada 1 de metal e parcialmente na camada 2 de metal, uma via é usada no ponto de mudança de camada. Do mesmo modo, se uma conexão é implementada parcialmente em poli e parcialmente em metal, um contato torna-se necessário para executar a mudança de camada. Contatos e vias introduzem uma quantidade significativa de impedância, contribuindo para um atraso ainda maior na propagação dos sinais.

O layout afeta de forma determinante a área de um circuito. Existem dois componentes para a área de um circuito integrado - a área funcional e a área de fiação. A área ocupada pelos elementos ativos (conjuntos de células lógicas) num circuito é chamada de área funcional. Já a área de fiação é aquela usada pelos fios que irão interconectar esses elementos ativos.

Um bom layout deve aproximar módulos com grande número de conexões entre eles, de modo a evitar tanto quanto possível o uso de fios longos, bem como minimizar o número de vias usadas. A área de um circuito tem influência direta sobre o desempenho e o rendimento do processo de fabricação, ou seja, no número de chips

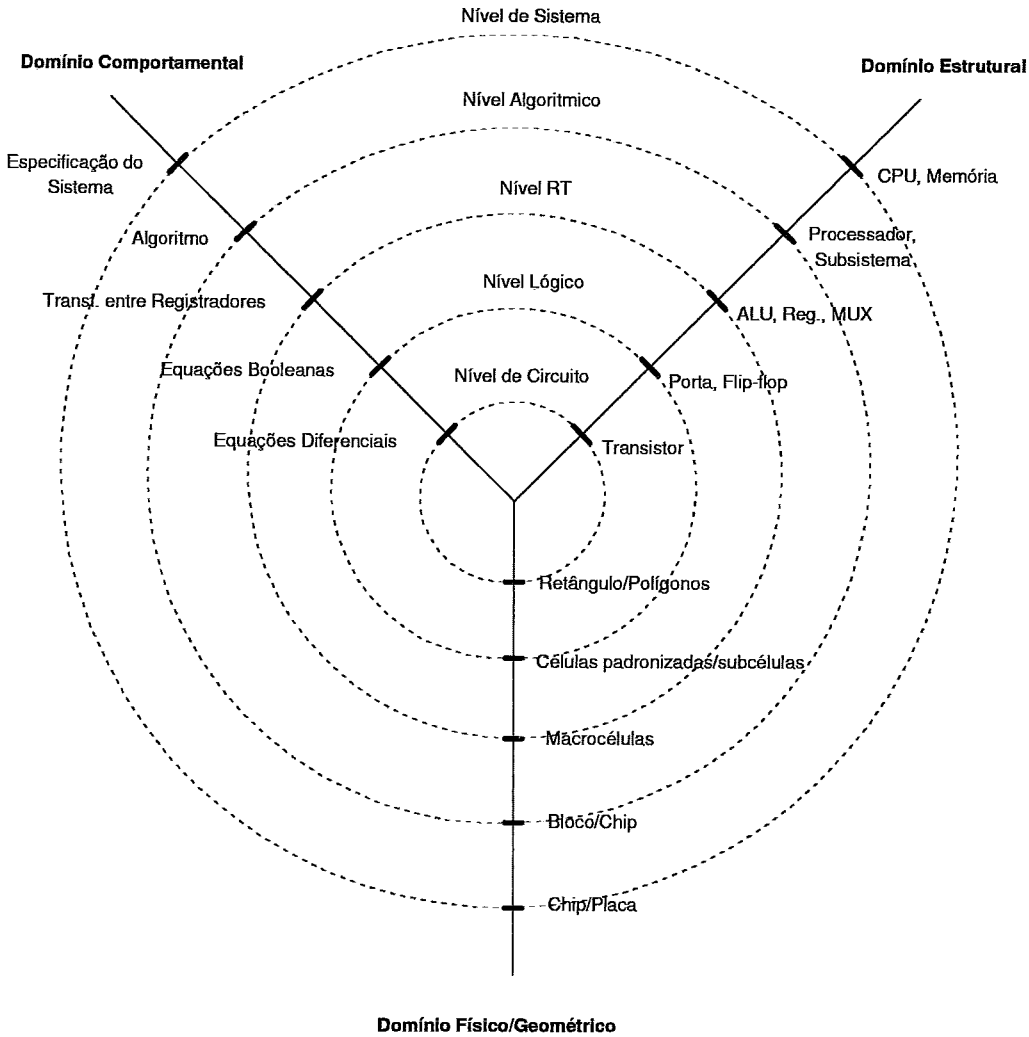

Figura 1.2: Domínios e níveis de abstração.

não defeituosos em um lote de chips produzidos. Quanto maior a área do chip, menor o rendimento. Um baixo rendimento significa um alto custo de produção, o que encarece o preço de venda do chip. A confiabilidade do chip também é influenciada pelo layout. Por exemplo, vias são fontes potenciais de imprecisão no momento da fabricação, portanto de erros. Assim, um layout que tenha um grande número de vias é mais suscetível a ter erros. Além disso, a largura de um fio de metal deve ser escolhida adequadamente pelo programa de layout, de modo a evitar o efeito de migração de metal.

Uma outra possibilidade de representação mais detalhada dos níveis de abstração de um projeto é dada pelo diagrama Y, introduzido inicialmente por Gajski e Kuhn [36], que corresponde à mais valorosa referência para o desenvolvimento de uma taxo-

nomia de automatização de projeto na área de sistemas eletrônicos. A idéia básica por trás do diagrama Y é de que cada elemento de um sistema eletrônico pode ser descrito dentro de três diferentes domínios. Esses domínios são os domínios comportamental, estrutural e físico/geométrico, onde o grau de detalhamento da implementação é crescente, nesta respectiva ordem. Os domínios são representados pelos três eixos de um "Y"convergindo para um ponto central comum. A Figura 1.2 ilustra este diagrama. Dentro de cada domínio, elementos podem ser descritos em vários níveis de abstração. Esses níveis são representados como pontos ao longo dos três eixos, com os níveis de mais alta abstração periféricos e os de mais baixa abstração próximos do centro do diagrama. O domínio comportamental descreve como um particular objeto responderia a um dado conjunto de entradas. Essa forma de representação caracteriza-se por não apresentar nenhuma noção da forma de implementação do sistema. O domínio comportamental preocupa-se em mostrar o que um sistema realiza, não importando a forma como este é implementado. Conforme podemos observar na Figura 1.2, os vários níveis de abstração deste domínio são descritos a partir de algoritmos, máquinas de estados finitos, equações lógicas ou booleanas e equações diferenciais. Para descrever comportamentalmente sistemas bastante complexos, existem linguagens de descrição de hardware, como VHDL (*Very high speed integrated circuit Hardware Description Language*), que permitem gerar uma descrição comportamental do circuito.

O domínio estrutural descreve a composição de circuitos a partir da descrição das interconexões dos seus componentes - abstrações de instâncias de elementos de circuito. Esse tipo de representação não especifica nada a respeito do comportamento do circuito, a não ser o que pode ser inferido a partir da descrição comportamental dos componentes que integram a estrutura. Os exemplos mais comuns de representações estruturais são diagramas de blocos e *netlists* (listas de conexões) de células lógicas.

O domínio físico/geométrico ignora, tanto quanto possível, a função que o projeto realiza, preocupando-se exclusivamente em fornecer uma representação espaço-geométrica da representação estrutural. O domínio físico/geométrico caracteriza-se por apresentar os detalhes e informações que serão utilizados na montagem ou fabricação física do sistema. Os elementos empregados na representação física são

polígonos, células e módulos, sendo que os dois últimos são utilizados para representar níveis de abstração mais elevados: células, compostas de polígonos e módulos constituídos por células. Na próxima seção, serão ilustrados alguns dos passos que compõem o projeto físico.

1.2 Passos do Projeto Físico

Uma subdivisão usual do projeto físico corresponde às tarefas de particionamento, atribuição, posicionamento e roteamento (este dividido em roteamento global seguido de roteamento detalhado), que são executadas obedecendo a esta ordem.

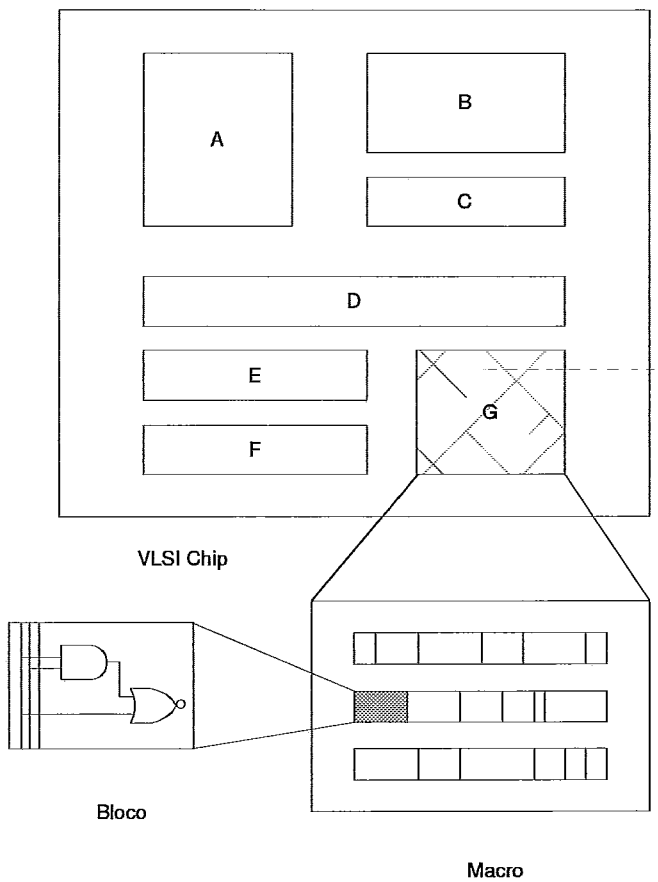

Figura 1.3: Hierarquia no projeto de chip VLSI.

O problema de particionamento pode ser visto como um processo no qual um módulo é construído a partir de submódulos em um dado nível hierárquico. Assim, o circuito sendo projetado deve ser particionado em submódulos menos complexos. Este particionamento é normalmente feito visando a obter-se submódulos com

alguma identidade funcional e com poucas ligações entre eles, resultando num agrupamento de elementos altamente interconectados entre si, num mesmo submódulo. A Figura 1.3 ilustra a idéia do particionamento.

O problema de atribuição consiste em construir um determinado circuito lógico a partir de módulos de circuito, cada qual com propriedades específicas, tais como tamanho, geometria e estrutura interna. O circuito lógico é geralmente descrito com o uso de blocos lógicos, tais como portas NAND, NOR ou inversores conectados por fios. Um módulo de circuito tem que satisfazer requisitos de desempenho especificados, como tempo de atraso, consumo de energia ou número de terminais de entrada/saída. No projeto *standard-cell* ou *gate-array*, que será visto em detalhes na próxima seção, os elementos lógicos fundamentais, que são os módulos de circuito, são projetados para fazer a interconexão entre transistores com diferentes características, de modo a possibilitar ao usuário selecionar um elemento que melhor se adeque às suas necessidades. Estes módulos de circuito são armazenados em uma biblioteca.

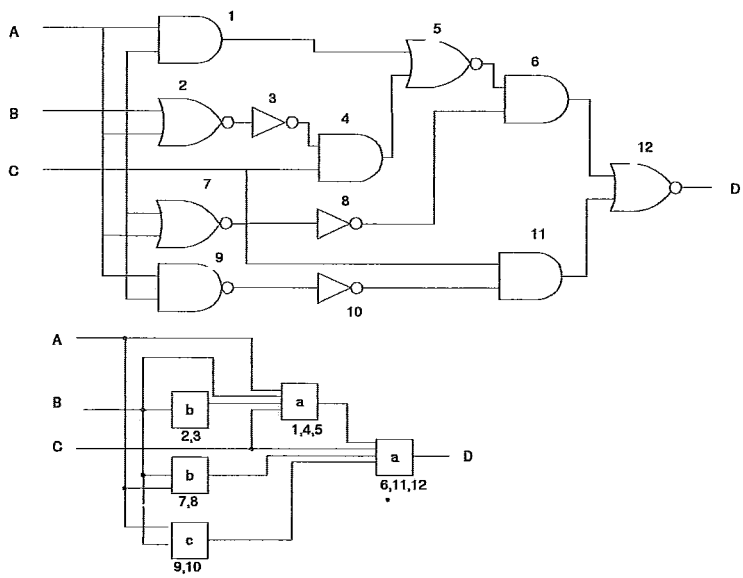

Figura 1.4: Atribuição de blocos lógicos a módulos de circuito.

A Figura 1.4 ilustra o problema de atribuição, onde dado o esquemático do circuito e um conjunto de módulos, conforme Figura 1.5, é feita uma atribuição de partes do circuito a módulos desta biblioteca. Para este exemplo, o circuito seria construído usando dois módulos a, dois módulos b e um módulo c.

O numero de módulos numa biblioteca tipicamente varia entre 100 e 200. Muitos

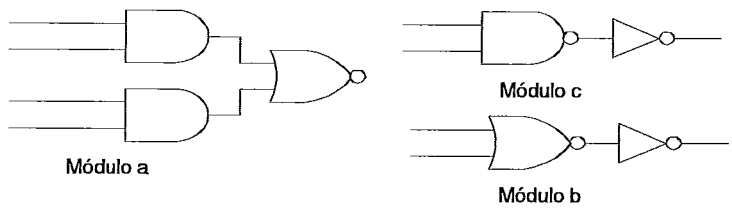

Figura 1.5: Exemplos de módulos de biblioteca.

blocos lógicos correspondem de forma biunívoca a circuitos na biblioteca. Outros blocos lógicos têm diversos módulos de circuito na biblioteca como opções, ou seja, são módulos com mesma função lógica, mas diferindo em tamanho, geometria, potência dissipada, posicionamento dos terminais e corrente de saída. Assim sendo, podem ser enunciados os dois passos que constituem o problema de atribuição:

- encontrar quais partes do circuito lógico correspondem a módulos na biblioteca que possuam a mesma função;
- determinar qual módulo da biblioteca deverá então ser selecionado segundo as especificações do usuário, por exemplo, em termos de tempo de atraso e geometria.

O problema de posicionamento (*placement*) consiste em encontrar posições físicas adequadas para cada módulo sobre o layout. Assim, dado um conjunto de módulos com portas (entradas, saídas, pinos de alimentação e de terra) sobre as bordas, as dimensões dessas células (altura, largura, etc.) e um conjunto de redes (as quais correspondem o conjunto de pinos que devem ser interligados), o posicionamento encontrará localizações físicas para cada célula, visando a minimizar alguma função objetivo, sujeita a certas restrições impostas pelo projetista, pelo processo de implementação ou pela estratégia de layout e estilo de projeto. Exemplos de restrições incluem prevenção de superposição de células no layout e exigência de que as células devam se ajustar às dimensões das localizações escolhidas. Existe ainda uma série de requisitos (geralmente conflitantes) que devem ser minimizados no projeto de um circuito: o comprimento da fiação, ruído, dissipação de calor, área, etc. Entretanto, acima de tudo, o posicionamento utilizado deve prover fácil roteamento dos sinais para formar as redes. Como é impossível incorporar todos os objetivos de projeto

em uma única função objetivo, uma escolha usual é minimizar o somatório dos comprimentos das redes, visando a reduzir o tempo máximo de atraso do circuito. O posicionamento das células determina aproximadamente o comprimento dos fios de interconexão.

Vale ressaltar que a métrica normalmente adotada para medir o custo de um posicionamento é a distância retilínea ou distância Manhattan, isto é, se (x_1, y_1) e (x_2, y_2) são dois pontos no plano, a distância retilínea d é dada por: $d = |x_1 - x_2| + |y_1 - y_2|$.

Seja o seguinte exemplo composto por três redes, $S_1 = A, B, C, E, H$, $S_2 = B, D, E, G$ e $S_3 = B, D, F$, e oito módulos A, B,..., H. Aqui, por razões didáticas, as redes são definidas como conexões entre módulos e não entre pinos. A figura 1.6 mostra duas alternativas para o posicionamento dos módulos e a diferença resultante na função custo.

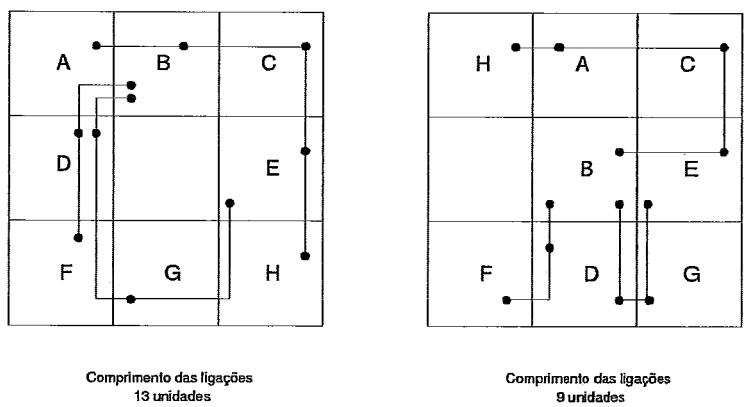

Figura 1.6: Diferença na função objetivo resultante de duas alternativas de posicionamento.

O posicionamento é normalmente guiado por funções de custo que, ou estimam o comprimento total da fiação, usada para conectar cada uma das redes como mostrado na Figura 1.6, ou estimam o congestionamento, ou seja, a interação entre as redes, os componentes e a área de layout. Em ambos os objetivos, busca-se a minimização, contudo estes são geralmente antagônicos, no sentido de que quanto menor o congestionamento, maior o comprimento total de fio utilizado e vice-versa. O *placement* sob a forma de um problema de otimização é NP-difícil, não existindo, portanto, algoritmos para solucioná-lo em tempo polinomial [58]. Assim sendo, várias heurísticas têm sido desenvolvidas para a solução deste problema.

Após o posicionamento, segue-se o roteamento, que é o processo de atribuir fisicamente trilhas (canais de fios) para os fios que conectarão as portas. Um posicionamento tem como objetivo principal gerar uma configuração na qual todas as conexões possam ser estabelecidas, ou seja, roteadas. Se isto não é possível e se o número de trilhas não pode ser aumentado, um novo posicionamento se faz necessário. Neste ponto, percebe-se claramente a intrínseca interdependência entre o posicionamento e o roteamento. Isto sugere que a melhor solução seria alcançada resolvendo-se simultaneamente os dois problemas, entretanto, devido a enorme complexidade desta tarefa, isto não é usualmente feito.

Uma dificuldade inerente a esta metodologia de posicionamento seguido pelo roteamento é que, por vezes, ao buscar o melhor posicionamento, através de alguma função objetivo, como por exemplo minimizar o comprimento total da fiação, chega-se a uma impossibilidade de concluir o roteamento. Esta situação é ilustrada na Figura 1.7, que mostra uma visão de um canal de roteamento num layout *standard-cell.*

Pode ser observado que a redução do comprimento total de fio pode levar a uma falha no roteamento. Isto demonstra que um *placement* visando a minimizar o comprimento de fiação pode, em alguns casos, resultar num circuito com mais área do que um *placement* que busque minimizar o congestionamento.

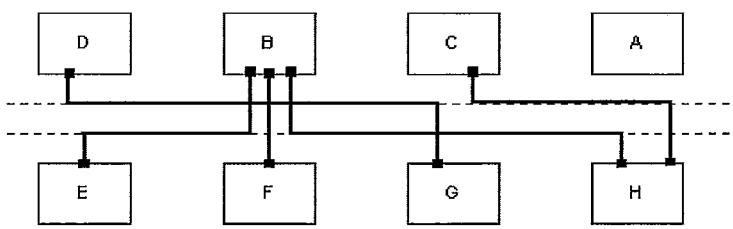

Figura 1.7: Duas trilhas requeridas. Todas as conexões roteadas.

Nas Figuras 1.7 e 1.8 temos as seguintes redes de 2 pontos: (B,E), (B,F), (B,H), (C,H) e (D,G). Para o segundo posicionamento, a rede (D,G) não tem como ser roteada, pois as duas trilhas já estão sendo usadas na região onde D poderia se conectar. Neste caso, para que se possa concluir o roteamento é necessário adicionar uma nova trilha, o que implica um aumento de área.

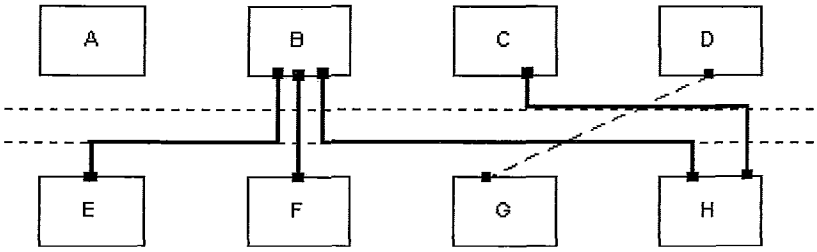

Figura 1.8: Comprimento de fio mais curto. Três trilhas são necessárias para concluir o roteamento.

1.3 Estilos de Layout

Existem diversas técnicas de layout que são usadas para gerar representações físicas dos circuitos. Estas técnicas diferem essencialmente nas restrições estruturais que elas impõem sobre os elementos de layout e também sobre a superfície de layout. As várias técnicas de layout pertencem a duas classes gerais que são:

- a) Layout *Full-Custom*, onde elementos de layout são todos projetados a mão e podem ser posicionados em qualquer lugar sobre a superfície de layout.
- b) Layout *Semi-Custom*, onde alguma estrutura é imposta sobre os elementos de layout e superfície, de modo a reduzir a complexidade das tarefas de layout.

A seguir estão enumerados os estilos atuais de layout que serão discutidos nas próximas subseções, com especial enfoque em FPGA.

- 1. *Full-Custom*
- 2. Estilo de Projeto *Gate-array*
- 3. Estilo de Projeto *Standard-cell*
- 4. Macrocélula
- 5. PLA (*programmable logic array*)
- 6. FPGA (*field programmable gate-array*)

1.3.1 Layout *Full-custom*

Este estilo de layout refere-se ao projeto de layout manual, onde um projetista experiente usa um editor de layout para gerar a descrição física do circuito. Tal estilo de projeto demanda grande esforço e consome muito tempo, devido à necessidade do projetista ter que se preocupar com todos os detalhes desta tarefa árdua.

Em contrapartida, este estilo de projeto de layout dá ao projetista completo controle no momento de decidir pelas localizações dos blocos do circuito e de que forma interconectá-los. Assim, um projetista experiente pode, geralmente, alcançar um alto grau de otimização, tanto em termos de área como em desempenho do circuito.

Devido ao longo tempo de projeto que tal abordagem necessita, seu uso se restringe a circuitos onde a obtenção de máximo desempenho é vital e onde haja expectativa de produção em larga escala, quando então o custo de projeto será amortizado pelo grande número de cópias produzidas. Quando o número de cópias de um circuito a serem produzidas é pequeno, o custo de projeto *full-custom* é proibitivo. Isto se aplica a alguns Circuitos Integrados de Aplicação Específica (ASICs), circuitos que executam uma função específica em uma aplicação, em que o aspecto mais importante é o tempo gasto para se colocar o produto no mercado, que corresponde ao tempo total de projeto, fabricação e teste. Para tanto, é necessário um alto grau de automatização, que só pode ser alcançado pela padronização dos processos de projeto e de teste. Portanto, arquiteturas de layout padronizadas são usadas para reduzir o tempo de projeto. Como exemplos destas temos: *gate-arrays, sea-of-gates, standard-cells, programmable logic arrays* e *field programmable gate arrays.*

1.3.2 Layout *Gate-array*

Um *gate-array* consiste em um grande número de transistores, já pré-fabricados sobre uma pastilha (*wafer*) na forma de uma matriz bidimensional. Inicialmente os transistores em uma matriz não são conectados uns aos outros. Então, para que se monte o circuito sobre um *gate-array*, é necessário que se façam as conexões, usando metal, através de um processo de construção de máscaras (*masking*). Este processo de adicionar fios de metal a um gate-array é chamado de personalização do array. *Gate-arrays* são também chamados *Mask Programmable Gate-Arrays* (MPGAs). O

custo de produção associado a um chip *gate-array* é baixo devido ao alto percentual, por lote, de chips produzidos sem erros de fabricação.

A personalização envolve dois tipos de interconexões - fiação dentro das células e fiação inter células. Uma célula é um pequeno módulo de circuito, tal como uma porta NAND com duas entradas, que pode ser implementada conectando um grupo de transistores em uma vizinhança local do *gate-array*.

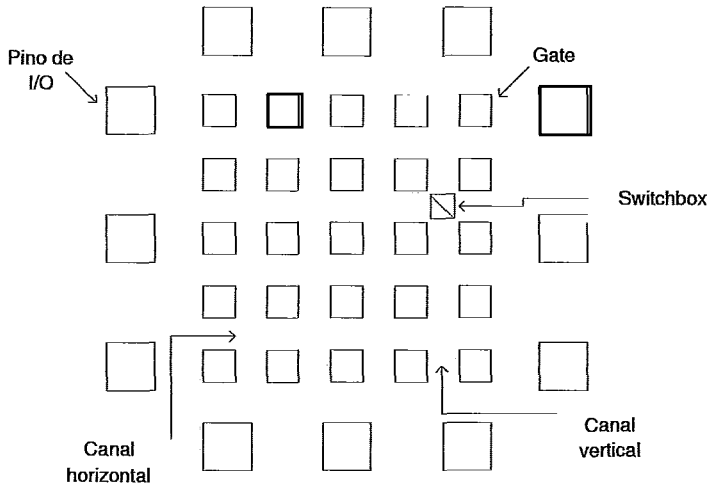

Figura 1.9: Um *floorplan* para *gate-array*.

De modo a estabelecer as conexões entre as células de uma forma sistematizada, o *gate-array* é estruturado como uma matriz regular de células básicas, como ilustrado na figura 1.9. Cada quadrado na figura representa uma célula e engloba um grupo de transistores no *gate-array*. Os fios que interconectam as células são postos dentro de regiões chamadas canais. A planta baixa (*floorplan*) do chip *gate-array* pode ser comparada à estrutura de uma cidade, que tem um dado número de edificações (células) e ruas (canais) para que o tráfego (fios) possa fluir de um local (uma edificação) para outro. Um cruzamento de ruas, ou seja, a área onde um canal horizontal intercepta um canal vertical, é chamado de *switchbox*. Cada canal horizontal (vertical) pode acomodar um número limitado de fios, que corresponde à densidade de trilha do canal.

Uma dificuldade inerente a este estilo de projeto é o fato de que, ao se tentar evitar longos fios entre as células, através do posicionamento de células fortemente conectadas o mais próximo possível, pode-se atingir um nível de congestionamento local que torne o layout não roteável.

1.3.3 Layout *Standard-cell*

Um *standard-cell* é um bloco lógico que executa uma função padrão. Exemplos de *standard-cells* são: porta NAND de duas entradas, porta XOR de duas entradas, flip-flop D, multiplexador com duas entradas e outras. Uma biblioteca de células é um banco contendo informações sobre as células padronizadas (*standard-cells*). A informação relevante sobre uma célula consiste no seu nome, sua funcionalidade, sua estrutura de pinos e um layout para a célula em uma determinada tecnologia.

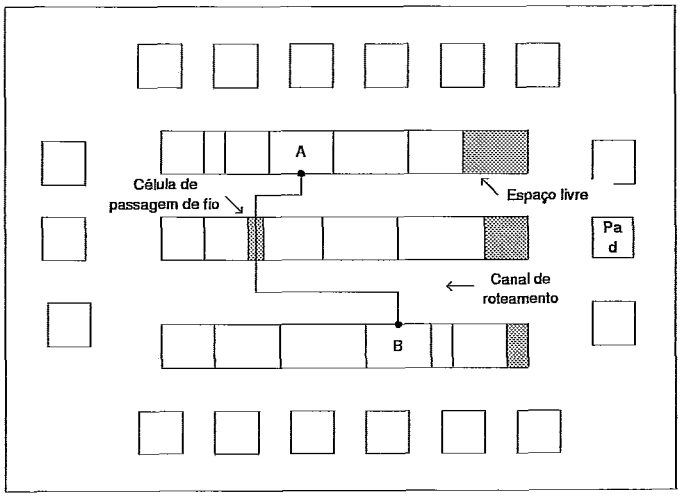

Figura 1.10: *Floorplan* de um layout *standard-cell.*

A vantagem de projetar um circuito usando uma biblioteca de células é a grande redução obtida no tempo de projeto. Como os layouts de células já estão disponíveis, a síntese física irá somente tratar com os seguintes aspectos: determinar a localização de cada célula e a interconexão das mesmas.

Estes aspectos correspondem a tarefas difíceis, que são objetos dos problemas de posicionamento (*placement*) e roteamento, respectivamente. Para reduzir a complexidade destas tarefas é utilizada uma planta baixa (*floorplan*) padrão, conforme Figura 1.10. O layout é dividido em diversas linhas. Uma linha consiste em células posicionadas próximas umas das outras. Pode ser observado que a altura de cada linha é a mesma, bem como a altura de qualquer célula na linha, uma vez que todas as células são pré-projetadas para terem a mesma altura.

Linhas são separadas por canais de roteamento horizontal. Células dentro da mesma linha, ou células de duas linhas vizinhas podem ser interconectadas por fios

usando o canal adjacente, ou seja, o canal que separa estas linhas. Se duas células em linhas não adjacentes tem que ser conectadas, uma técnica mais elaborada é usada para esta finalidade, como ilustrado na figura 1.10. Aqui a célula A na linha 1 tem uma conexão para a célula B na linha 3. Um tipo especial de célula, chamada de célula de passagem de fio (*feedthrough cell*), é colocada na linha 2.

Comparado ao estilo *gate-array*, o layout *standard-cell* oferece mais flexibilidade. Em um chip *standard-cell*, o espaço de fiação não é fixado a priori, ou seja, em tempo de projeto pode-se modificar a largura de um canal de roteamento. Além disso, as células podem ter larguras variáveis. A desvantagem do *standard-cell* em relação ao *gate-array* é que todos os passos de fabricação são necessários para produzir o chip.

1.3.4 Layout Macrocélula

Tanto o projeto *gate-array* como o projeto *standard-cell* impõem restrições sobre as células que são usadas para projetar o circuito. Por exemplo, as células em um layout *standard-cell* devem ter a mesma altura. Se esta restrição é removida, as células não podem ser posicionadas em uma planta baixa(*floorplan*) baseada em linhas. Observe-se o um exemplo, na figura 1.11, onde dois arranjos diferentes para um mesmo conjunto de macrocélulas são mostrados. Na figura 1.11(b) temos que a área ocupada é menor, devido ao arranjo mais compacto.

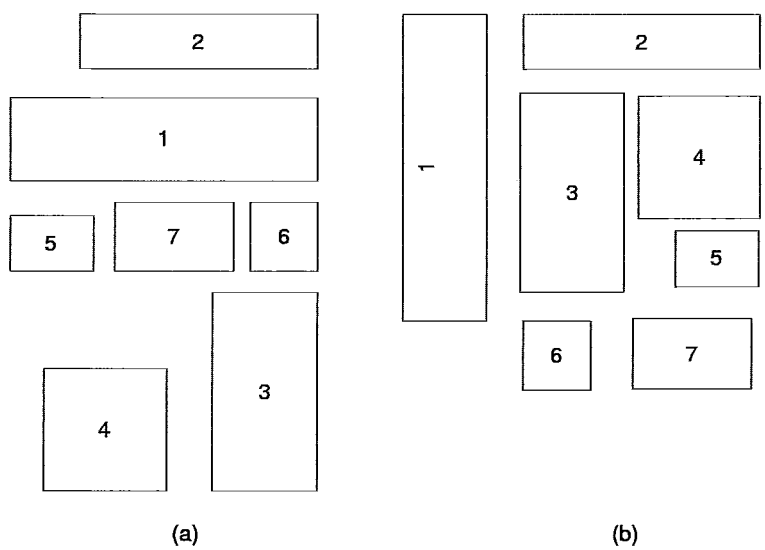

Figura 1.11: (a) Células de diferentes alturas e larguras posicionadas em um *floorplan* baseado em linhas. (b) Um *floorplan* mais compacto do mesmo circuito.

Assim, o estilo de projeto macrocélula caracteriza-se por permitir que cada célula possa variar em ambas as dimensões. A vantagem disto é que células mais complexas podem ser disponibilizadas numa biblioteca, como por exemplo: registradores, memórias e unidades lógicas e aritméticas. A desvantagem fica por conta da dificuldade maior do problema a ser tratado pelas ferramentas de síntese física de CAD, face à ausência de uma planta baixa (*floorplan*) padrão.

1.3.5 PLA (*Programmable Logic Arrays*)

Um *programmable logic array* (PLA) é uma forma conveniente de implementar expressões soma de produtos (SOP) de dois níveis. Na figura 1.12 observa-se exemplos de duas funções deste tipo, Z e Y, ambas definidas com três entradas A_0, A_1, A_2. Os produtos (termos AND) são formados no primeiro nível e suas somas (o termo OR) são calculadas no segundo nível. Assim, um PLA consiste em um plano AND para implementar os termos produto e um plano OR para implementar as somas. Considere-se as seguintes funções: $Z = A_0 * A_1 + A_0 * A_2 + A_1 * A_2$ e $Y = A_0 * A_1 + \bar{A}_0 * \bar{A}_2$. Essas duas funções podem ser implementadas usando a PLA mostrada na Figura 1.12. O plano AND tem duas linhas verticais para cada entrada - uma correspondendo diretamente à entrada e outra para a forma invertida da entrada. Existem tantas linhas horizontais na PLA quantos sejam os termos produto.

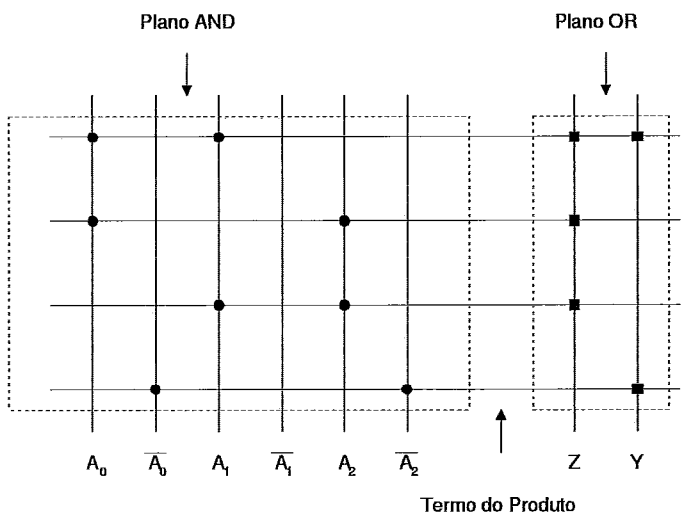

Figura 1.12: Um PLA para implementar as funções Z e Y.

Observando as equações das funções Y e Z, existem apenas quatro termos produto

diferentes. Um contacto é colocado no plano AND, na interseção de uma linha i com uma coluna j se o termo produto i contem a entrada j. Por exemplo, na linha $A_0 * A_1$, existem dois contactos onde a linha intercepta as colunas A_0 e A_1. O plano OR contém tantas linha verticais quantas sejam as saídas, neste caso duas. Um ponto de conexão é colocado no plano OR, na interseção de uma linha i com uma coluna j se a saída correspondente à coluna j contem o produto i. Por exemplo, um cruzamento é colocado onde a coluna Y intercepta a linha $A_0 * A_1$.

1.3.6 Layout FPGA

FPGAs (*Field Programmable Gate Arrays*) são matrizes de blocos lógicos e segmentos de fio pré-fabricados, sendo que tanto os recursos lógicos como os de roteamento são programáveis pelo usuário [93], [105]. Em face ao seu baixo custo de projeto, FPGAs têm sido cada vez mais usados para implementar ou prototipar circuitos integrados de aplicação específica (ASICs). As tecnologias FPGA são normalmente classificadas em três grandes classes: (a) Look-Up-Table (LUT), baseadas em SRAM (por exemplo, Xilinx [24], [44]); (b) Multiplexador, baseadas em anti-fusíveis (por exemplo, Actel [50]); (c) PLD, baseadas em EPROM (por exemplo, Altera [51]);

Aqui novamente existem as mesmas etapas anteriormente discutidas, como: posicionamento e roteamento. O posicionamento consiste em determinar em qual bloco lógico (célula da matriz FPGA) irá localizar-se cada parte funcional do circuito. Isto claramente afeta sensivelmente a consecução da tarefa de roteamento do circuito. Esta última etapa consiste em interligar os pinos de uma mesma rede de sinal através dos recursos de interconexão oferecidos pela arquitetura FPGA. Na próxima seção é detalhada a arquitetura alvo deste trabalho, e definidos alguns termos que serão largamente mencionados doravante.

Capítulo 2

Fundamentos sobre FPGA

Este capítulo provê informação básica sobre arquieturas FPGA (*Field Programmable Gate Arrays*) e também descreve o ciclo de projeto baseado no uso de ferreamentas de CAD, usado para mapear automaticamente circuitos dentro de pastilhas FPGA.

2.1 Arquiteturas de FPGAs

Todos os FPGAs são compostos de três componentes fundamentais: blocos lógicos (L), blocos de entrada/saída (E/S) e roteamento programável. Um circuito é implementado em um FPGA, programando-se cada bloco lógico para implementar uma pequena parte da lógica requerida pelo circuito e cada bloco de E/S para agir como um pino (*pad*) de entrada ou de saída. O roteamento programável é configurado para fazer todas as conexões necessárias entre blocos lógicos e entre blocos lógicos e blocos de E/S (*pads*).

A arquitetura de FPGA investigada neste trabalho é mostrada na Figura 2.1. Ela consiste em um array bidimensional de células lógicas e recursos de roteamento. Cada bloco (célula lógica) é marcado com L e pode ser configurado para ser uma LUT, um flip-flop, ...,etc. Segmentos de fio, para conectar pinos de blocos, passam entre os blocos através de canais verticais e horizontais e são alinhados em trilhas. Portanto, um canal vertical (horizontal) é um conjunto de trilhas entre duas colunas (linhas) consecutivas de células. No exemplo da figura 2.1 existem quatro trilhas por canal e cada bloco lógico tem quatro pinos sobre cada um dos seus lados.

Como segmentos de fio são pré-fabricados, a personalização dos recursos de roteamento é alcançada através da programação adequada de *switches*. Estes últimos são agrupados dentro de blocos de *switches* (S) e de conexão (C) (Figuras 2.1(a) e

2.1(b)). Os blocos C contêm *switches* de roteamento que podem ser programados para conectar pinos de células lógicas a segmentos de fio. Os pontos terminais de segmentos de fio estão dentro dos limites de blocos S e estes contêm *switches* que permitem que um segmento de fio seja conectado a um outro segmento de fio. A arquitetura aqui apresentada permite somente segmentos de fio que tenham tamanho simples (ou seja, todo segmento de fio cobre somente um bloco C, ou ainda, que seus pontos extremos estejam em blocos S vizinhos). Doravante, esta arquitetura será denominada um FPGA 2-D array (*island style*).

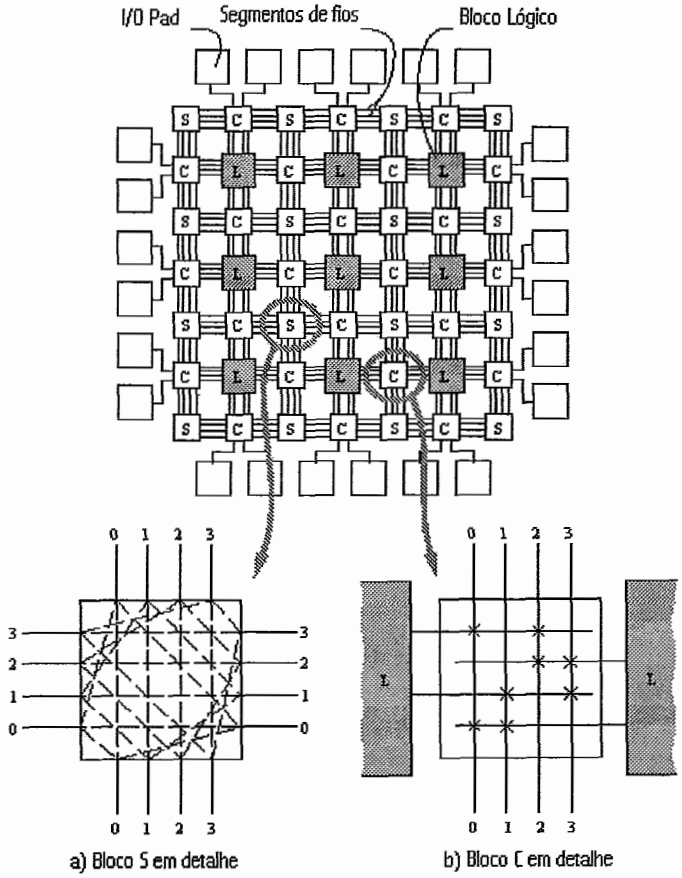

Figura 2.1: Modelo geral de um FPGA baseado em array.

2.1.1 Tecnologias de Programação de FPGAs

Existem três diferentes técnicas que permitem fabricar FPGAs: (a) Look-Up-Table (LUT), baseada em SRAM (Xilinx [113]), (b) multiplexador, baseada em anti-

fusíveis (Actel [50]) e (c) PLD, baseada em EPROM (Altera [51]). Neste trabalho a arquitetura alvo usa a tecnologia (a).

Atualmente a tecnologia mais popular usa SRAM (Synchronous RAM) células para controlar os *transistores de passagem, multiplexadores* e *buffers tri-state* permitindo assim configurar-se todo o roteamento programável e blocos lógicos segundo a necessidade de cada circuito. A figura 2.2 mostra esses três tipos de *switches* baseados em células SRAM. Muitos FPGAs da Xilinx [113], as maiores pastilhas da Altera [51], os últimos FPGAs da Actel [50], todos os FPGAs da Lucent Tecnologies [103] e os VF1 FPGAs da Vantis [32] são baseados em células SRAM.

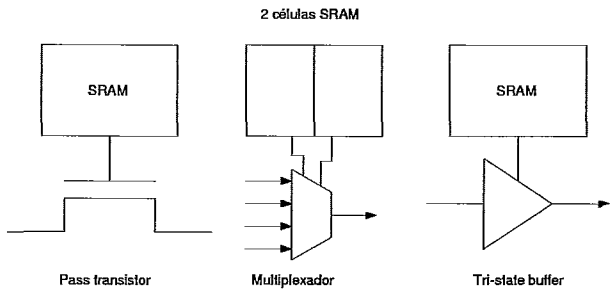

Figura 2.2: Três tipos de *switches* programáveis usados em FPGAs baseados em células SRAM .

2.1.2 Arquiteturas do Bloco Lógico de FPGAs

O bloco lógico usado em um FPGA influencia fortemente a velocidade e a eficiência em área do FPGA. Enquanto muitos diferentes tipos de blocos lógicos têm sido usados em FPGAs [93], a maioria dos FPGAs comercialmente disponíveis usa blocos lógicos baseados em *Look-Up-Tables* (LUTs).

A figura 2.3 mostra como uma LUT de 2 entradas pode ser implementada em um FPGA baseado em células SRAM - uma LUT de k-entradas requer 2^k células SRAM e um multiplexador com 2^k-entradas. Uma LUT de k-entradas pode implementar qualquer função de k-entradas simplesmente programando as 2^k células SRAM de modo a obter-se a tabela verdade da função desejada.

Trabalhos anteriores têm demonstrado que LUTs com 4-entradas levam a FPGAs com a mais alta eficiência em área [85] e a maioria dos FPGAs comerciais são baseados nestas LUTs. Muitos FPGAs modernos têm seus blocos lógicos compostos não apenas por uma única LUT, mas sim por um grupo de LUTs e registradores com

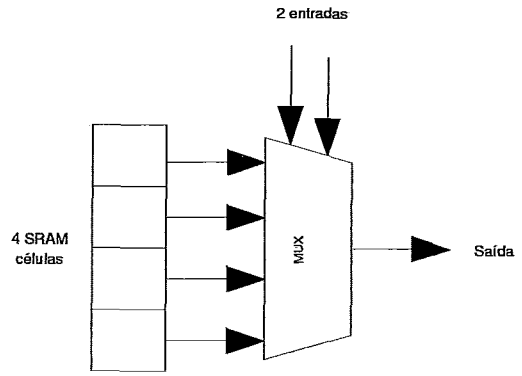

Figura 2.3: Uma LUT 2-entradas implementada em um FPGA baseado em células SRAM.

alguma interconexão local entre eles. A figura 2.4 mostra um bloco lógico composto de duas LUTs de quatro entradas.

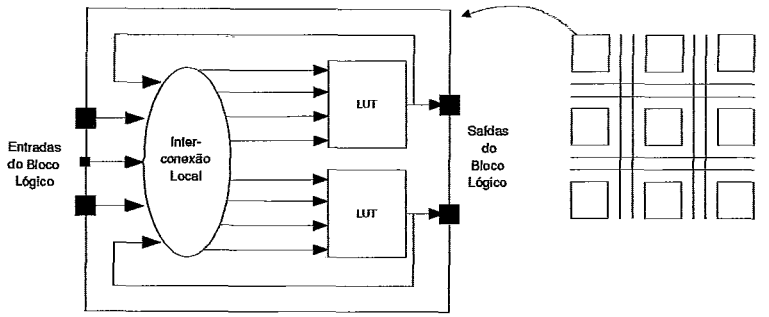

Figura 2.4: Exemplo de *cluster* lógico contendo duas 4-LUTs.

Em [30], Chung e Rose verificaram que para um bloco lógico composto de quatro LUTs de 4 entradas conectadas em uma topologia específica de árvore consegue-se um FPGA 7% mais denso e 28% mais rápido do que um FPGA tendo seu bloco lógico composto de uma única LUT de 4 entradas. Outros resultados para *clusters* de LUTs são descritos em [15] e [69].

2.1.3 Arquiteturas de Roteamento de FPGAs

FPGA comerciais podem ser classificados em três grandes grupos, baseados em sua arquitetura de roteamento. Os FPGAs da Xilinx, Lucent e Vantis são baseados em arrays (*island-style,*) enquanto os FPGAs da Actel são baseados em linhas e os FPGAs da Altera são hierárquicos. Neste trabalho, somente a arquitetura baseada em arrays (*island-style*) é enfocada.

Será admitido que todos os canais horizontais e verticais têm o mesmo número de trilhas, denotado por W. As trilhas, em cada canal horizontal, são numeradas de baixo para cima e, em cada canal vertical, da esquerda para a direita. Um número associado a uma trilha corresponde ao identificador da trilha. A flexibilidade de um bloco C, F_C é definida como o número de trilhas ao qual um pino lógico pode se conectar. A figura 2.1(b) ilustra um bloco C. As trilhas passam através dele e podem ser conectadas aos pinos de blocos lógicos via um conjunto de *switches*. Na figura 2.1(b), uma opção de roteamento é representada por um X , de modo que cada pino pode ser conectado para duas trilhas verticais, ou seja, $F_C = 2$. Já no exemplo da Figura 2.5(c), existem três pinos lógicos no bloco C e cada um destes pode se conectar a qualquer uma das quatro trilhas. Logo, a flexibilidade deste bloco C é 4.

Um bloco S é uma caixa retangular de *switches* (chaves programáveis) que conecta segmentos de fios em dois canais distintos. Dependendo da topologia, cada segmento de fio sobre um lado de um bloco S pode ser conectado para um ou para todos, ou para alguma fração dos segmentos de fios sobre cada um dos outros lados de um bloco S. A flexibilidade do bloco S é dada pelo parâmetro F_S , que define o número de outros segmentos de fio aos quais um segmento de fio chegando em um bloco S pode se conectar.

Um exemplo de bloco S aparece na Figura 2.5(c), onde cada linha tracejada representa um *switch* de roteamento programável - nesta figura, $F_S = 3$. A flexibilidade e a topologia adotada nos blocos S têm considerável impacto na roteabilidade das redes. A flexibilidade de um bloco S, F_S, é definida como o número de segmentos de fios ao qual cada segmento de fio pode conectar-se. Por exemplo, a Figura 2.5(c) mostra todas as possibilidades de conexões para o segmento de fio do lado direito, que está sobre a trilha 1. Ele pode se conectar ao segmento de fio na parte superior sobre a trilha 4, ou na parte de baixo sobre a trilha 1, ou ainda no lado esquerdo sobre a trilha 3. Suponha que todos os outros segmentos de fio adjacentes a este bloco S possam somente conectar-se a três outros segmentos de fios. Assim, sua flexibilidade é 3.

Suponha que um bloco S tenha $F_S = 3$, e adicionalmente, que cada segmento de fio possa se conectar a exatamente um segmento de fio em cada um dos canais restantes adjacentes a este bloco S. A Figura 2.1(c) ilustra um exemplo de um bloco

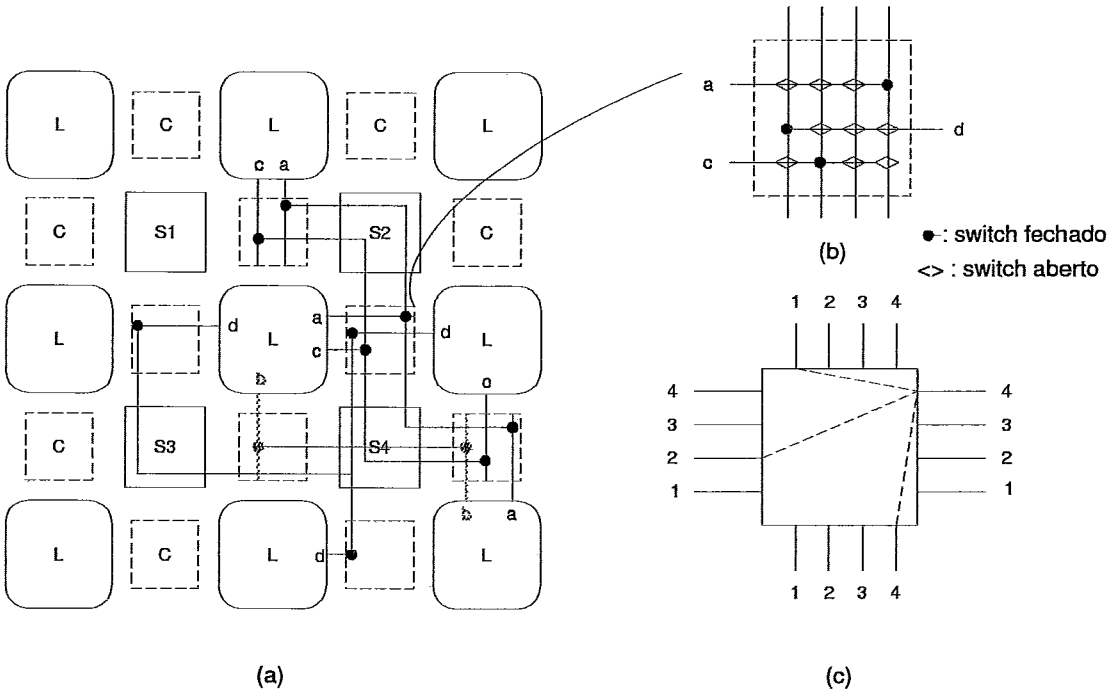

Figura 2.5: (a) Exemplo de roteamento. (b) Um bloco C. (c) Um bloco S.

S com flexibilidade básica. Um outro interessante bloco S é o bloco S identidade, que permite que todos os segmentos com o mesmo identificador de trilha sejam conectados e possui flexibilidade básica. O bloco S diagonal (Figura 2.1(a)) usado no FPGA XC4000 é deste tipo.

Em [84] foi feito um estudo detalhado da flexibilidade dos blocos S e C, onde foi observado que a melhor arquitetura em termos de eficiência em área usa $F_S = 3$ ou 4 e F_C variando entre 0.7W e 0.9W, onde W é o número de trilhas em cada canal. Uma alta flexibilidade significa um maior número de escolhas e uma baixa flexibilidade implica poucas escolhas. Se os blocos são mais flexíveis do que é realmente necessário para se obter maior grau de roteabilidade, poderá haver *switches* sobrando, acarretando um FPGA desnecessariamente lento e grande.

Uma característica arquitetural importante é como o bloco C é implementado. Se cada X, na Figura 2.1(b), é simplesmente um transistor de passagem, então dois ou mais *switches* sobre um pino podem ser ligados, de modo a permitir um roteamento *dogleg*, que consiste na possibilidade de um terminal em um bloco L conectar-se a duas ou mais trilhas. Entretanto, se os Xs ao longo de um pino de entrada são implementados como um (de)multiplexador, somente uma conexão

para as trilhas pode ser feita. Nestes casos, *doglegs* não são possíveis. No entanto, FPGAs comerciais tais como Xilinx XC4000 [113] e Lucent ORCA FPGAs [106] não permitem pinos de entrada *doglegs*.

Muitos trabalhos têm devotado especial atenção a FPGAs onde todos os fios (segmentos) cobrem (varrem) somente um bloco lógico antes de terminar em um bloco de *switch* e estes trabalhos têm comparado arquiteturas somente sobre o aspecto de eficiência em área. Aqui assume-se que todas as trilhas de roteamento consistem em somente um segmento de fio curto contido no interior de um bloco C, como mostrado na Figura 2.1. Esta hipótese é feita para facilitar a comparação de resultados alcançados por diversos algoritmos de roteamento para FPGA recentemente produzidos e baseados somente em segmentos de fios curtos (roteamento não-segmentado). A métrica eficiência em área usualmente considerada nesses trabalhos tem sido avaliar o número de *switches* programáveis utilizados no roteamento. No entanto alguns trabalhos têm considerado arquiteturas de roteamento que incluem fios de tamanhos diferentes, como ilustrado na figura 2.6. Em [19] Brown et al investigaram diferentes distribuições de tamanhos de fios.

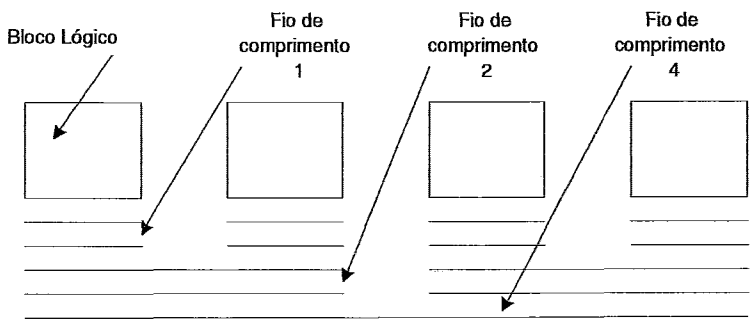

Figura 2.6: (a) Exemplo de segmentação de canal.

Uma rede é um conjunto de pinos associados a um mesmo sinal e que devem ser interconectados por segmentos de fios. Uma rota detalhada de uma rede é um conjunto de segmentos de fio e *switches* programados apropriadamente para conectar pinos e segmentos da rede. Uma rota detalhada de uma rede é simples se todo pino da rede conecta-se a exatamente uma trilha. Uma rota detalhada de uma rede é uma rota *dogleg* se existe um pino da rede, o qual conecta-se a duas ou mais trilhas.

Um roteador global decide, para cada rede, por quais blocos S e blocos C a rota detalhada desta rede irá passar. Um roteador detalhado determina uma rota

detalhada para cada rede, respeitando a topologia da rota global da rede. Assim, para cada rede, um roteador detalhado aloca segmentos de fios seguindo a topologia especificada pelo roteador global, de modo que nenhuma superposição ocorra entre rotas detalhadas de diferentes redes. Quando um pino está conectado a um segmento de fio em um bloco C, este segmento de fio não pode ser acessado por qualquer outro pino no mesmo bloco C, de modo que podemos considerar que uma rota global se origina e termina em blocos C. Então, uma rota global de uma rede pode ser vista como um conjunto de blocos C e blocos S.

Dado um FPGA 2-D e as rotas globais de todas as redes, estas são ditas roteáveis sem *doglegs* em um FPGA, se todas elas têm rotas detalhadas simples, não incorrem em superposição uma com as outras, seguem suas respectivas rotas globais especificadas e o número de trilhas requeridas não excede W. As redes são ditas roteáveis com *doglegs* em W trilhas se todas as trilhas tem rotas detalhadas não superpondo umas com as outras, sendo cada uma delas uma rota detalhada simples ou *dogleg*, seguindo a rota global especificada.

A densidade de canal resultante de um roteamento global, W_g , é o número máximo de rotas globais que atravessam juntas qualquer canal. Uma densidade de canal de um roteamento detalhado, W_d , é o número mínimo de trilhas W necessárias para que as topologias de redes oriundas do roteamento global sejam roteadas. Observe que o número de trilhas W é fixo para um dado chip e W_d depende do roteamento global. Se $W_d > W$, então o roteamento não pode ser concluído. Um taxa de mapeamento é o valor de W_d/W_g .

Um segmento de fio x em um bloco C é dito ser *conectável* a um segmento y em outro bloco C, se existe uma seqüência $< x = s_0, s_1, s_2, ..., s_n = y >$ de segmentos de fio tal que para todo $i, 0 < i \leq n$, um segmento de fio s_{i-1} é conectado ao segmento de fio s_i, dentro de algum bloco S. Definiremos um *domínio de trilha* como um conjunto maximal de segmentos de fio tal que quaisquer dois segmentos do conjunto são conectáveis. As estruturas de roteamento com blocos S de flexibilidade básica podem ser divididas em duas classes principais: *arquiteturas disjunta e sobreposta.* Uma *arquitetura de roteamento disjunta* é uma estrutura em que todos os segmentos de fio que compõem todo o chip são particionados (i.e. separados de forma disjunta) em idênticos W domínios de trilhas, de tal forma que cada domínio de trilha contenha

exatamente um segmento de fio em cada bloco C. Se isto não ocorre, então tem-se uma estrutura que corresponde a *arquitetura de roteamento sobreposta.*

A principal vantagem advinda do modelo de FPGA descrito acima é a sua generalidade, a qual suporta um vasto espectro de arquiteturas de roteamento, permitindo alterar o número de trilhas por canal e o conteúdo dos blocos C e S. Estudos anteriores examinaram os efeitos dos parâmetros F_C e F_S [84], [106]. Com base nesses estudos anteriores, serão usados os valores $F_S = 3$ e $F_C = W$, onde W é o número de trilhas por canal, para todos os experimentos neste trabalho. Vale ressaltar que essas mesmas hipóteses são usadas também em recentes publicações sobre algoritmos de roteamento [4], [28], [63], [107], [3] e são geralmente aceitas como sendo razoáveis.

2.2 CAD para FPGAs

Implementar um circuito em um FPGA moderno requer que centenas de milhares ou mesmo milhões de *switches* programáveis e bits de configuração sejam configurados adequadamente, ou seja, ligados ou desligados. Claramente, esta é uma tarefa que somente é possível com o uso de ferramentas de CAD. Usuários de FPGAs descrevem um circuito em um alto nível de abstração, tipicamente usando uma linguagem de descrição de hardware (tal como VHDL) ou entrada de esquemático. Ferramentas de CAD então convertem esta descrição em alto nível em um arquivo de programação especificando o estado (ligado ou desligado) de cada *switch* programável dentro do FPGA. Devido à complexidade desta tarefa são necessários diversos passos, mostrados na figura 2.7. Nas prómimas subseções descrevemos brevemente os passos de Síntese, *Placement* e Roteamento.

2.2.1 Síntese e Empacotamento de Blocos Lógicos

O primeiro estágio converte a descrição inicial do circuito, em geral feita através de uma linguagem de descrição de hardware ou de esquemático, numa *netlist* de portas básicas. Então o processo de síntese lógica converte esta *netlist* de portas básicas em uma *netlist* de blocos lógicos do FPGA de tal maneira que o número de blocos lógicos necessários seja minimizado e/ou a velocidade do circuito seja maximizada. A tarefa de síntese lógica é suficientemente complexa, tanto que é geralmente dividida em dois ou mais subproblemas. A figura 2.8 mostra um exemplo de decomposição do

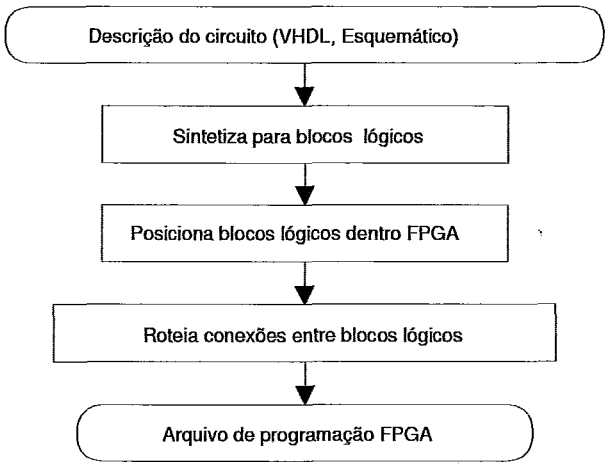

Figura 2.7: Fluxo de CAD para FPGA.

procedimento de síntese em três subproblemas, os quais são descritos rapidamente nesta seção.

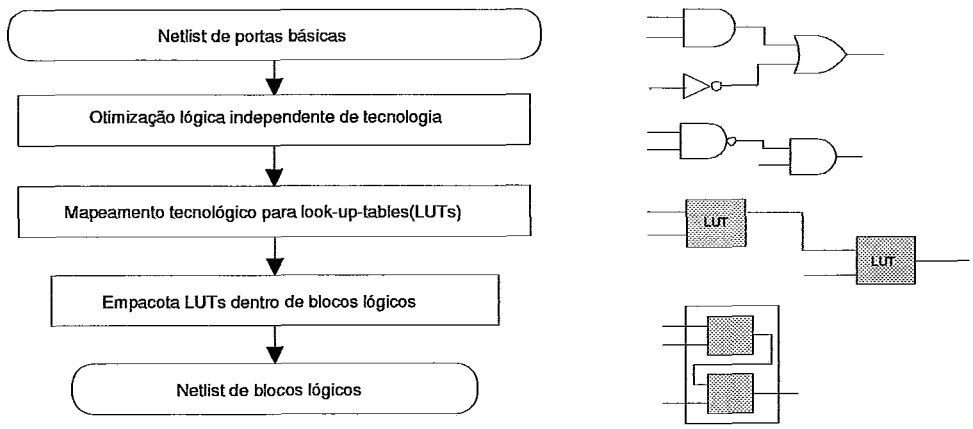

Figura 2.8: Detalhes do procedimento de síntese.

A otimização lógica independente de tecnologia remove lógica redundante e efetua simplificações lógicas sempre que possíveis. A *netlist* otimizada de portas lógicas é então mapeada para *look-up-tables*. O terceiro passo do processo de síntese é necessário sempre que um bloco lógico do FPGA contem mais de uma LUT. Este passo agrupa diversas LUTs e registradores dentro de um bloco lógico, respeitando limitações tais como o número de LUTs, o número de sinais de entrada distintos e *clocks* que um bloco lógico pode conter. As metas de otimização nesta fase são empacotar tão próximo quanto possível LUTs conectadas, de modo a minimizar o número de sinais a serem roteados entre bloco lógicos e a tentar preencher cada

bloco lógico em toda a sua capacidade, de forma a minimizar o número de blocos lógicos usado.

2.2.2 *Placement*

Algoritmos de *placement* determinam qual bloco lógico dentro de um FPGA deverá implementar cada um dos blocos lógicos requeridos pelo circuito. As metas de otimização são: (i) posicionar, de forma aglutinada, blocos lógicos conectados minimizando assim o comprimento total de fio requerido (*wirelength-driven placement*); (ii) posicionar blocos lógicos de modo a balancear a densidade de fiação ao longo dos canais do FPGA (*routability-driven placement*), ou (iii) maximizar a velocidade do circuito (*timing-driven placement*).

Atualmente as três principais classes de posicionadores (*placers*) em uso são min-cut (partitioning-based) [86] [47], analítico [5] [82] e baseados em *simulated annealing* [14] [101]. No caso desta última classe, tem-se a vantagem de ser possível adicionar facilmente novas metas de otimização ou restrições ao posicionador.

Uma vez que a quantidade de recursos de roteamento em FPGAs é limitada e configurada pelo fabricante quando a pastilha FPGA é produzida, algumas ferramentas de *placement* tentam otimizar não somente o comprimento total de fio, mas também sua roteabilidade. Em [70], Ebeling et al descrevem um posicionador usando *simulated annealing*, que enfoca o Triptych FPGA desenvolvido pela Universidade de Washington. Sua função de custo incorpora não somente o termo correspondente ao semi-perímetro do menor retângulo que engloba os terminais da rede (*net*), mas também um termo de *porosidade* que monitora a fração de blocos lógicos em uma área local que estão sendo usados. Desde que o Triptych FPGA é geralmente não roteável em regiões onde os blocos lógicos são completamente usados, manter-se uma *porosidade* em torno de 50% através do FPGA é essencial. Em [74] e [73], Nag e Rutenbar descrevem um posicionador baseado em *simulated annealing* que executa *placement* e roteamento simultaneamente em um único passo combinado. Esta ferramenta atinge resultados de alta qualidade, mas o tempo de CPU requerido é extremamente alto. Por exemplo, para um circuito contendo somente 461 blocos lógicos Xilinx 4000 foram gastas 11 horas de CPU para posicionar e rotear. A complexidade deste algoritmo é $O(n^3)$, onde n é o número de blocos lógicos em um

circuito.

2.2.3 Roteamento

Após as localizações (posições) para todos os blocos lógicos terem sido definidas, um roteador determina quais *switches* programáveis devem ser ligados para conectar todos os pinos de entrada e saída de blocos lógicos, bem como os pinos de entrada e saída do circuito. É usual representar a arquitetura de roteamento de um FPGA através de um grafo $G_A = (N, D)$ orientado (dirigido) [74] [34]. Cada fio e cada pino de bloco lógico torna-se um nó $n \in N$ neste grafo de recursos de roteamento e cada *switch* torna-se um arco orientado $a \in D$ (para *switches* unidirecionais, tais como *buffers*) ou um par de arcos orientados (para *switches* bidirecionais, tais como *transistores de passagem*) entre dois nós apropriados. A figura 2.9 mostra o grafo de roteamento de recursos correspondendo a uma porção de um FPGA cujos blocos lógicos contém uma única LUT com 2-entradas e 1-saída. Alguns trabalhos têm representado FPGAs como grafos não orientados [1], mas uma representação de grafo orientado é necessária se *switches* direcionais, tipo *buffers tri-state* e multiplexadores, devem ser modelados corretamente.

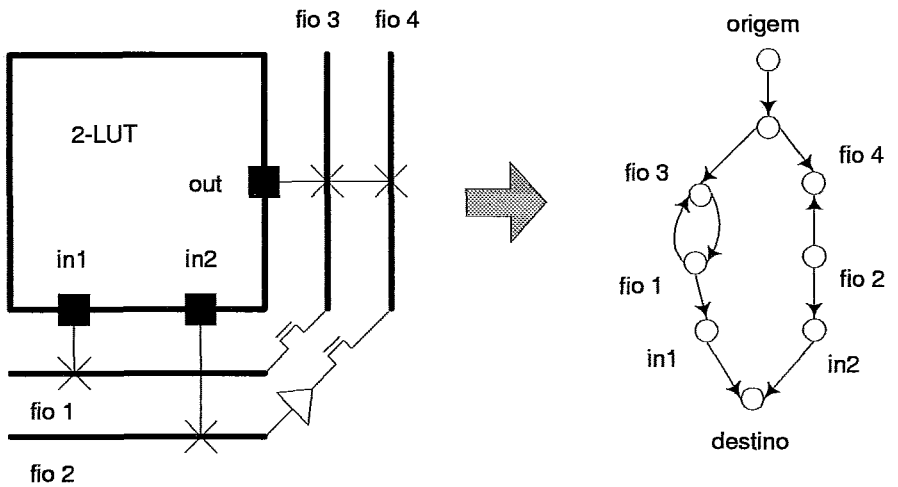

Figura 2.9: Modelando roteamento detalhado FPGA como um grafo orientado.

Frequentemente blocos lógicos FPGA têm pinos logicamente equivalentes. Por exemplo, todos os pinos de entrada de uma LUT são logicamente equivalentes. Isto significa que um roteador pode completar uma dada conexão usando qualquer um dos pinos de entrada de uma LUT. Esta equivalência lógica é modelada no grafo de

roteamento de recursos adicionando-se nós fontes (*source*) nos quais todas as redes começam e nós destinos (*sink*) nos quais todos os terminais de rede terminam. Existe um nó fonte para cada conjunto de pinos de saída logicamente equivalentes e existe um arco do nó fonte para cada um destes pinos de saída. De forma similar, existe um nó destino para cada conjunto de pinos de entrada logicamente equivalentes e um arco de cada um desses pinos de entrada para um nó destino. Portanto, cada rede de sinal consiste em um único nó de G_A, nó origem do sinal, junto com um subconjunto de nós correspondente ao seus destinos (*sinks*).

Rotear uma rede de sinal (*net*) corresponde a encontrar uma árvore neste grafo de recursos de roteamento entre os nós representando os pinos de blocos lógicos a serem conectados, tal que todos os nós destinos possam ser alcançados a partir do nó origem . E desde que o número de fios (recursos de roteamento) de um FPGA é limitado deve-se buscar por árvores as mais curtas possíveis. Além disso é importante que o roteamento de uma rede (*net*) não use recursos de roteamento que outre rede (*net*) necessite. Assim, a maioria dos roteadores de FPGA têm algum tipo de esquema de prevenção de congestionamento para resolver a disputa por recursos de roteamento.

Uma meta adicional de otimização é construir redes sobre ou próximas do caminho crítico, roteando-as usando caminhos mais curtos e recursos de roteamento mais rápidos. Roteadores que tentam otimizar desempenho desta maneira são chamados *timing-driven* e desde que grande parte do atraso num FPGA se deve ao roteamento programável, torna-se essencial aplicar-se um roteador deste tipo para que se obtenha circuitos com bons desempenhos.

Os roteadores de FPGA podem ser divididos em dois grupos. Roteadores *global-detalhado combinado* determinam totalmente um caminho de roteamento em um único passo, enquanto os algoritmos de roteamento em dois passos primeiramente executam o roteamento global para determinar quais pinos de blocos lógicos e canais de roteamento cada *net* irá usar, e então, executam o roteamento detalhado para determinar os fios que cada rede (*net*) irá usar dentro de cada um dos canais especificados. A tarefa de um roteador detalhado para FPGA é frequentemente difícil ou impossível porque o roteamento FPGA tem flexibilidade limitada e o roteador detalhado é altamente restringido pelas decisões do roteador global sobre quais canais (

seqüência de blocos de conexão C) cada *net* deve usar. Roteadores global-detalhado combinado têm maiores chances de otimizar o roteamento, uma vez que eles estão livres de tais restrições.

Paradigma de Roteamento baseado em Negociação

O paradigma de roteamento baseado em negociação tem sido usado com sucesso em diversos roteadores de FPGA. Este paradigma foi originalmente desenvolvido por McMurchie e Ebeling [70] para uma arquitetura FPGA padronizada, mas tem sido usado também com muito sucesso para o roteamento de uma variedade de arquiteturas FPGAs.

Quando o roteamento das redes (*nets*) é feito de forma sequencial, a ordem na qual as redes são roteadas pode ser crítica, desde que alguns recursos de roteamento absolutamente necessários para o roteamento de uma dada rede já estejam sendo ocupados por redes anteriormente roteadas. A principal idéia do paradigma de negociação é inicialmente permitir compartilhamento ilimitado dos recursos de roteamento e então, repetidamente, rerotear as redes, até que não exista mais nenhum recurso sendo compartilhado. Atribuindo custos de congestionamento aos recursos compartilhados e incrementando estes custos a cada iteração, tão logo as redes (*nets*) tenham sido roteadas, o algoritmo de roteamento encoraja a busca por rotas alternativas até que todos os conflitos tenham sido resolvidos. Figura 2.10 mostra o pseudocódigo para um roteador NC(*Negotiation-Based Router*), onde $RT(i)$ denota o conjunto de nós no roteamento corrente da rede (i).

O custo de um recurso de roteamento n tem três termos

$$c_n = (d_n + h_n) * (p_n + 1)$$

O custo base é d_n, que pode ser usado para refletir o atraso de um recurso de roteamento. O termo de congestionamento de primeira ordem p_n é o número de redes que correntemente compartilham o recurso de roteamento. Ele vale 1 se a utilização do recurso de roteamento n não incorrer em nenhum compartilhamento (i.e., nó já ocupado) e é incrementado pelo número excedente de uso (quantidade de redes compartilhando o nó menos a sua capacidade). p_n é também dado em função

do número de iterações executadas. Nas iterações iniciais p_n cresce lentamente e para forçar a convergência do algoritmo, o mesmo cresce rapidamente nas últimas iterações.

O termo de congestionamento de segunda ordem h_n é o congestionamento histórico do recurso de roteamento e cresce monotonicamente a cada iteração na qual o mesmo é compartilhado. De fato h_n é incrementado por um dado valor fixo cada vez que uma rede é roteada através de um nó já utilizado.

Figuras 2.11, 2.12, 2.13 e 2.14 ilustram alguns aspectos descritos nesta Seção, o grafo de roteamento de um FPGA, uma visão detalhada deste, um exemplo de roteamento e uma visão detalhada do roteamento, respectivamente.

Neste trabalho, é usado o algoritmo VPR [14], o qual é baseado neste paradigma, para efetuar o roteamento de um subconjunto de redes (*nets*) em cada processador. No proxímo capítulo, enfocaremos detalhadamente as diversas idéias sobre roteamento de FPGA através do resumo de alguns dos principais trabalhos existentes na literatura.

```
NC(netlist, arquitetura do FPGA, RotearTudo/RotearCompatilhado );
1  enquanto existir recursos compartilhados faça
2      Para  toda rede S_i faça
3          se RotearTudo ou S_i tem recursos compartilhados então
4              desfaça roteamento da rede S_i ;
5          fim-se;
5          RT_i ← n_s (nó origem de S_i);
6          enquanto existir nós destino de S_i ainda não conectados a n_s  faça
7              P_j ← encontre-caminho-minimo(RT_i,n_s);
8              adicione P_j para RT_i;
9          fim-enquanto;
10         atualize os custos de primeira e segunda ordem dos nós de RT_i ;
12     fim-para
13 fim-enquanto;


encontre-caminho-minimo(RT,n) {
    Para  todo nó n em RT faça
        coloque n sobre a fila Q com chave igual a 0 ;
    fim-para
    enquanto um novo nó destino não for alcançado faça
        retire o nó m, com menor valor de chave, da fila Q ;
        se m ainda não foi anteriormente retirado da fila então
            Para  todo vizinho n de m faça
                coloque n sobre a fila Q com custo de n mais a chave de m ;
            fim-para
        fim-se
    fim-enquanto
    backtrace a partir do nó destino alcançado até que um nó de RT seja alcançado;
    retorne este caminho;
}
```

Figura 2.10: O algoritmo NC.

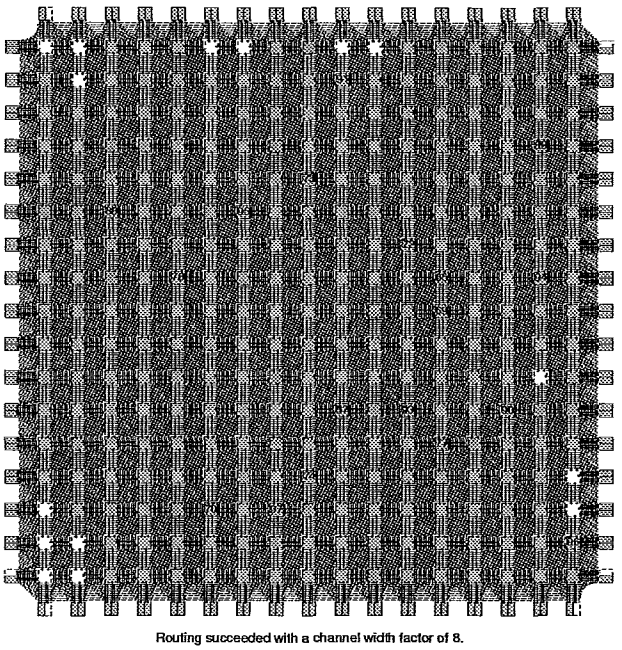

Figura 2.11: FPGA completo.

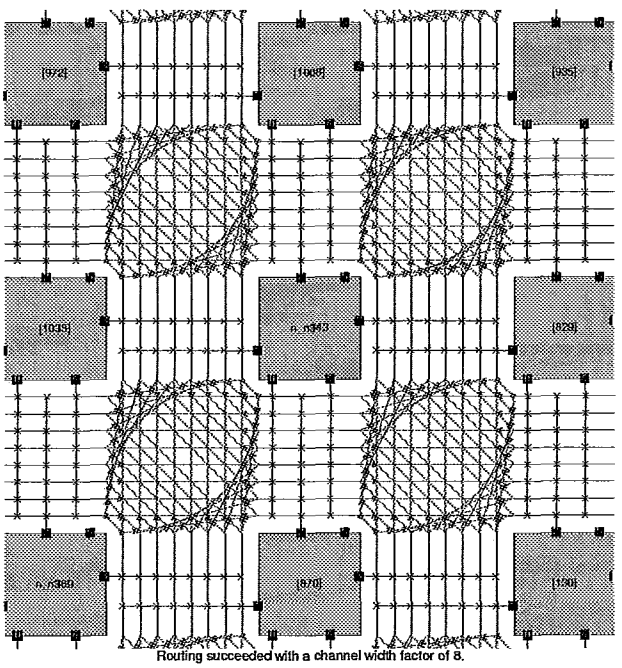

Figura 2.12: Visão gráfica de um exemplo de arquitetura de FPGA; o bloco lógico é uma LUT de 4 entradas + registrador.

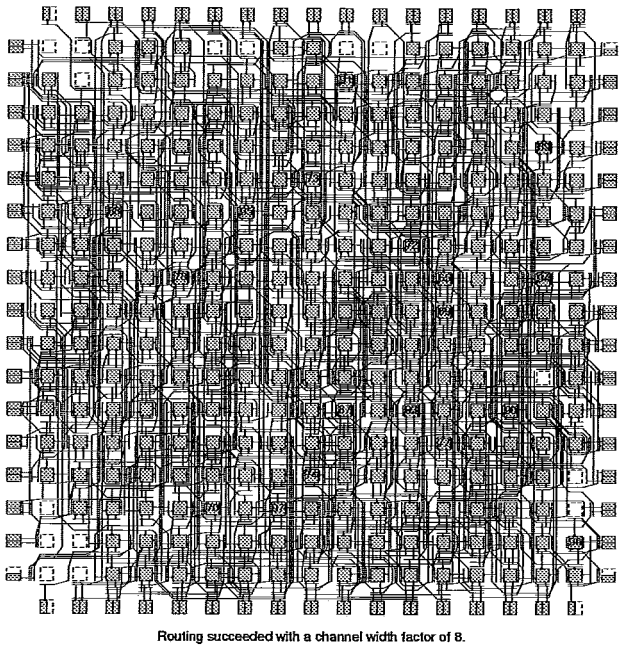

Figura 2.13: Exemplo de roteamento de FPGA.

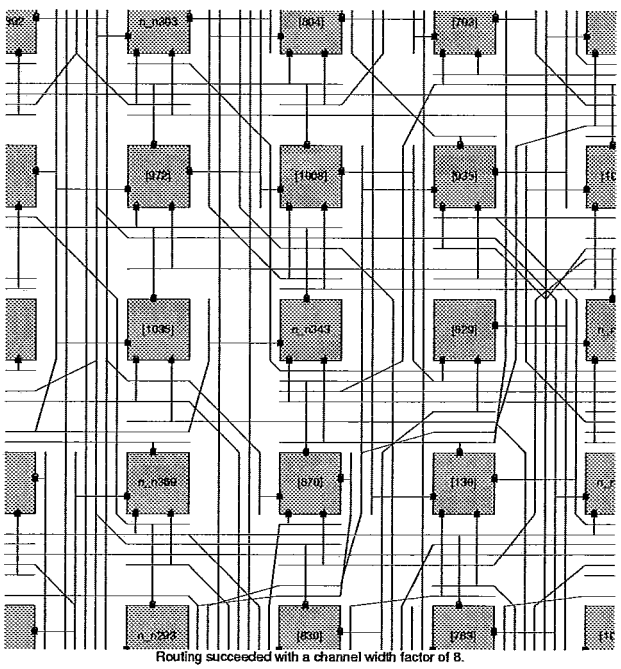

Figura 2.14: Visão ampliada de um exemplo de roteamento de FPGA.

Capítulo 3

Trabalhos Anteriores

Como já foi mencionado, nesta seção será feita uma revisão bibliográfica relativa a algoritmos propostos na literatura para resolver o problema de roteamento FPGA.

Antes, porém, uma visão geral dessas ferramentas e algoritmos é apresentada através de uma breve descrição da abordagem utilizada por eles na solução dos problemas de *placement*, roteamento global e roteamento detalhado.

3.1 Posicionamento de Blocos Lógicos e de Pinos de E/S

Muitos estudos de roteamento de FPGA têm usado as *netlists* do *benchmark* originalmente gerada para avaliação da ferramenta CGE/SEGA. O posicionamento (*placement*) desses *benchmarks* foi gerado por ALTOR [86], uma ferramenta que originalmente se propunha ao posicionamento de circuitos do tipo *standard cell*. ALTOR usa uma estratégia de bi-particionamento min-cut recursivo, repetidamente particionando nas direções horizontal e vertical.

Ferramentas recentes chamadas FPR [3], SPLACE [107] e VPR [14] incluem algoritmos de *placement* que visam ao uso específico para FPGAs. FPR usa um técnica de particionamento recursivo que é similar à empregada em ALTOR, mas, em cada passo, usa *simulated annealing* para dividir a *netlist* em uma grade m x n, para algum valor pequeno de m e n fixos. Antes de cada passo recursivo, FPR também executa roteamento global. Esta estratégia simultânea de *placement* e roteamento global é única entre todas as ferramentas de FPGA consideradas neste trabalho. Em comparação, SPLACE e VPR usam algoritmos *simulated annealing* para *placement*. VPR provê um tratamento mais eficiente de redes com alto *fanout*

(número de ligações a partir de um único terminal) e é capaz de tratar um número maior de movimentos do que SPLACE em uma dada quantidade de tempo de CPU.

3.2 Roteadores Globais para FPGA

O roteador global LocusRoute [83] foi originalmente desenvolvido para aplicações *standard cell.* Ele aceita um *placement* e uma *netlist* multiponto (multi-terminal) como entradas e quebra as redes em redes de dois pontos (terminais). Cada rede de dois pontos é roteada com duas ou menos curvas de noventa graus (*bend*) com o objetivo de minimizar a densidade de canal. Um custo por *bend* pode ser adicionado, de modo a minimizar uso de *bends* [106]. A saída é um grafo composto de partes relativamente grandes ou sintetizadas (*coarse graph*) de cada conexão consistindo em uma série de segmentos de canais adjacentes para guiá-la através do *array* FPGA. A qualidade da atribuição de canal do LocusRoute é mensurada pela máxima densidade do canal, D_{max} , que é o maior número de sinais distintos ocupando um único segmento de canal.

O passo de roteamento global do VPR [14] usa um roteador de labirinto sobre redes multipontos em uma maneira similar a [70]. Todas as redes são roteadas, e se necessário são removidas e re-roteadas diversas vezes. Após cada iteração, ele agrega um custo histórico para segmentos de canal com densidade maior do que a densidade alvo, D_{alvo}. Assim o roteamento das redes subsequentes tendem a evitar canais congestionados, a não ser que não haja outra alternativa. VPR encontra a densidade mínima possível D_{alvo} que sucessivamene roteia um circuito com $D_{max} \leq D_{alvo}$.

3.3 Roteadores Detalhados para FPGA

O algoritmo CGE (*Coarse Graph Expansion*) [18] foi desenvolvido especificamente para pesquisa de roteamento FPGA. Ele expande todas as redes de dois pontos ao longo de suas rotas globais dentro de um pequeno número de caminhos distintos, cuidadosamente podando o espaço de busca. Recursos de fio para o caminho de mais baixo custo são concedidos até que o circuito seja roteado. Uma estratégia de remoção (*rip-up*) é empregada se necessário. Nesse caso, menos poda de possíveis

escolhas é feita em áreas difíceis de rotear.

O sucessor do CGE, SEGment Allocator (SEGA) [64], usa uma estrutura de função de custo diferente para fazer uso de segmentos de fios longos. SEGA também se baseia na hipótese de que uma rede pode ser completamente expandida dentro de todos os caminhos possíveis ao longo da rota global. Consequentemente, SEGA não re-expande uma rede quando seus caminhos são exauridos. Ao invés disto, a função custo aumenta a prioridade da rede tanto quanto diminuam as suas escolhas. Esta técnica alcança bons resultados, fazendo com que o estilo CGE *rip-up* seja desnecessário para o algoritmo.

Desde a data da publicação original de SEGA, diferentes funções de custo têm sido exploradas para investigar a roteabilidade e o desempenho de velocidade [112]. A função custo usada para produzir os resultados deste trabalho, chamada *Area*, tem sido de longe a mais profícua no uso de menos trilhas de fiação. O custo *Area* faz com que SEGA primeiro identifique as redes que têm o menor número de caminhos restantes. Dentre essas redes, o caminho com o menor custo *Demand* (função custo do CGE) é escolhido.

3.4 Roteadores Gobal e Detalhado Combinados para FPGA

O algoritmo Bin-Packing Guloso (GBP) [110] combina roteamento global e detalhado em um único passo. Fazendo as hipóteses de que $F_S = 3$, $F_C = W$ e que blocos S de topologia disjunta são usados, um FPGA é roteado por tratar todo domínio de trilha como um *bin* e de forma gulosa ir preenchendo este *bin* com redes até que não haja mais espaço disponível para qualquer outra rede. GBP então passa para o próximo domínio de trilha e repete o processo. Observações de que GBP não empacota densamente os últimos domínios de trilha levam para o algoritmo *Orthogonal Greedy Coupling* (OGC) [111]. Mudando de um algoritmo guloso para outro (que tem uma meta de otimização diferente) após alguns domínios de trilhas já terem sido empacotados, os últimos domínios de trilha foram mais densamente empacotados e poucas trilhas de roteamento foram usadas.

Uma série de algoritmos de roteamento de um único passo foi apresentada em [4]. Nesses algoritmos, redes multipontos são roteadas uma rede por vez. Se uma rede

falha ao ser roteada, ela é movida da cabeça da fila de redes ordenadas e o roteamento é reiniciado. Os recursos de roteamento do FPGA são representados por um grafo, que encolhe à medida que as redes vão sendo roteadas. O algoritmo difere pelo modo como eles roteiam uma rede multiponto através do grafo remanescente. Cinco algoritmos básicos diferentes foram apresentados, três dos quais foram aperfeiçoados usando iteração. Desses oito algoritmos, quatro minimizam o comprimento dos fios por resolver o problema da Árvore de Steiner em redes e quatro minimizam a distância da origem para o destino usando algoritmos de caminho mínimo.

O algoritmo de *Placement* e Roteamento FPGA (FPR) [3] usa a mesma estratégia de roteamento de rede descrita acima. Ele também usa o algoritmo IKMB para executar roteamento detalhado. Entretanto, antes de cada passo de particionamento recursivo, FPR seleciona gulosamente uma rota global parcial para cada rede baseada sobre arborescências de Steiner retilínea e atribui redes a blocos S específicos. Isto permite ao FPR balancear o congestionamento através de cada corte e fixar a entrada do sinal ou pontos de saída sobre cada lado antes de cortar (particionar) cada subpartição.

O algoritmo TRACER-fpga [28] [63] também executa roteamento combinado de redes multipontos. Ele usa um roteador de labirinto propagando da origem para todos os destinos de modo a rotear cada rede. Inicialmente, todas as redes são roteadas permitindo compartilhamento de fio. Em seguida, técnica semelhante à *simulated annealing* escolhe redes para remoção (*rip up*) e reroteamento; redes compartilhando recursos são propícias a serem removidas. Durante o reroteamento um alto custo é usado para desencorajar futuro compartilhamento. Quando nenhum compartilhamento mais ocorre, uma solução é encontrada. Ele também usa folgas (disponibilidade de trilhas num canal) para ordenar redes durante o roteamento inicial e para seleção de redes durante a remoção. Utilizando as folgas, ele dá prioridade às redes longas para conexões diretas e permite que redes curtas sejam roteadas em torno do congestionamento.

O algoritmo SROUTE [107] seqüencialmente emprega roteamento de labirinto para cada rede multiponto buscando pelo destino mais próximo. Se um caminho para uma rede não pode ser encontrado, ela é movida do início da fila ordenada das redes e o roteamento é reiniciado. Para reduzir o espaço de busca do roteamento de

labirinto, ele inicialmente segue caminhos que avançam em direção ao destino mais próximo.

3.5 Flexibilidade das Estruturas de Interconexão para FPGAs

Em [84] os autores enfocam o projeto da estrutura de interconexão e estudam o efeito de sua flexibilidade sobre a propriedade da roteabilidade de um FPGA, bem como sobre os requisitos de recursos de fiação. A propriedade de roteabilidade se traduz no percentual do número total de conexões que são efetivamente roteadas.

Assim posto, este trabalho envolve algumas questões: 1) Qual o efeito da flexibilidade do bloco de conexão C, denotada por F_C , sobre a taxa de sucesso de roteamento?; 2) Qual o efeito da flexibilidade do bloco de interconexão S, denotada por F_S , sobre a taxa de sucesso de roteamento?; 3) De que maneira interagem F_C e F_S ?; 4) Qual o efeito de F_C e F_S sobre o número de trilhas por canal requeridas para alcançar 100% de roteamento?.

De modo a tentar responder às questões (1) (2) (3) e (4) foi implementado um conjunto de circuitos industriais sobre uma variedade de estruturas de interconexão e foram medidos os percentuais de sucesso de roteamento e o número requerido de trilhas por canal para cada circuito. Para isso foi adotado o seguinte procedimento de implementação: i) Executar o mapeamento tecnológico do circuito original no bloco lógico; o resultado deste passo é uma nova *netlist* que interconecta somente blocos lógicos e é funcionalmente equivalente ao circuito original, ii) Executar o posicionamento (*placement*) da *netlists* resultantes, usando ALTOR [86], iii) Executar o roteamento global do circuito, que determina o caminho, consistindo em uma seqüência de canais pelos quais cada fio deverá passar. Isto fornece o número de trilhas que serão necessárias em cada canal, supondo que os blocos de interconexão S e de conexão C têm flexibilidade total; iv) Executar o roteamento detalhado do circuito. Para cada caminho global (rota global), determine os fios exatos nos blocos de interconexão S e de conexão C.

Ao final tem-se como saída duas possibilidades: o percentual de redes que foram roteadas com sucesso dado um número particular W de trilhas por canal; ou o número de trilhas por canal necessário para atingir 100% das conexões para um

dado F_S e uma dada razão de $\frac{F_C}{W}$.

Com base nos resultados em [84], as principais conclusões do trabalho foram que os blocos de conexão C devem ter alta flexibilidade para alcançar altos percentuais de roteamento, e que os blocos de interconexão S requerem flexibilidade limitada.

3.6 Análise Detalhada de Trabalhos Anteriores sobre Roteadores FPGA Sequenciais

Nas proximas subseções são revistas algumas das principais idéias contidas em alguns trabalhos mais relevantes que têm norteado a pesquisa em roteamento FPGA.

3.6.1 Roteador Detalhado CGE

O CGE (*Coarse Graph Expansion*) [18] foi o primeiro algoritmo desenvolvido especificamente para FPGAs. Este adotou uma metodologia nova para o problema de roteamento que permitia seu uso em uma vasta gama de diferentes arquiteturas de roteamento FPGA.

Como o roteamento FPGA é um problema combinatório complexo, a tradicional metodologia de dois estágios permite a separação de dois problemas distintos: balanceamento das densidades de todos os canais de roteamento e atribuição de segmentos de fios específicos a cada conexão. O roteador global divide redes multipontos em redes de dois pontos e as roteia em caminhos de mínima distância. Sua principal meta é distribuir as conexões entre os canais de tal maneira que as densidades do canais sejam balanceadas. O roteador global define uma rota de curso para cada conexão por associá-la a uma seqüência de segmentos de canal. A figura 3.1 mostra uma rota global marcada num FPGA.

A rota global é representada por um grafo sintético (*coarse graph*), denotado $G(V, A)$, onde o bloco L na posição (2,2) é dito ser a raiz do grafo e o bloco L na posição (4,4) é chamado folha. Deste modo, $G(V, A)$ sintetiza todos os caminhos possíveis para estabelecer uma conexão entre os blocos lógicos em (2,2) e (4,4). Após o roteamento global, o problema é transformado no seguinte: para cada conexão de dois pontos, o roteador detalhado deve escolher segmentos de fios específicos para implementar os segmentos de canal atribuídos durante o roteamento global.

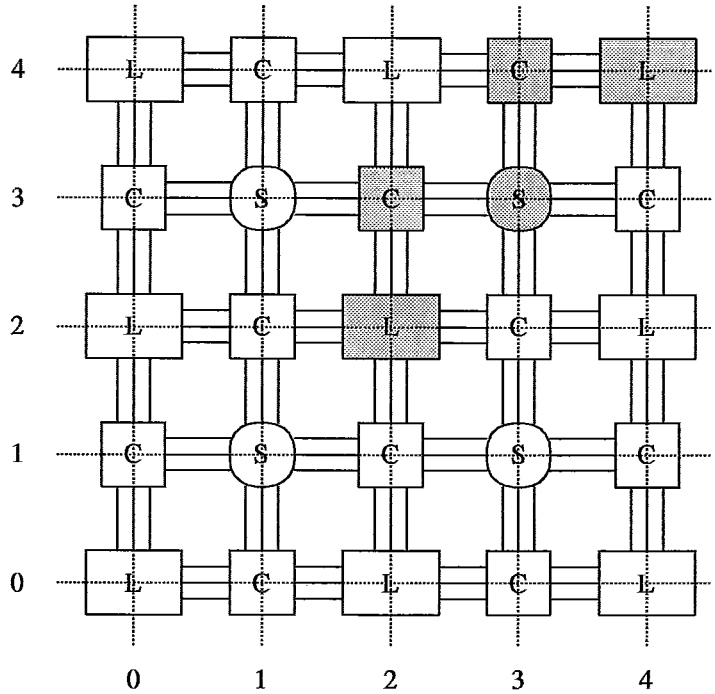

Figura 3.1: Uma rota global.

Fases do CGE

Considere-se o grafo $G_F(V, A)$ associado a um típico FPGA, onde V é o conjunto de todos os blocos contidos no FPGA, i.e., blocos L, C e S, e o conjunto de arestas A contem um arco $(i, j), i \in V, j \in V$, se houver algum arco correspondente no FPGA. De modo a ilustrar tal grafo, imagine-se o FPGA da Figura 3.1 com densidade de canal igual a 1 i.e., $W = 1$. Assim define-se um grafo sintético (*coarse graph*) como um subgrafo de $G_F(V, A)$.

O algoritmo básico possui duas fases: na primeira, ele enumera um número de alternativas para a rota detalhada de cada grafo sintético e, então, na segunda fase, observando todas as alternativas de um só vez, ele faz escolhas específicas para cada conexão.

A . Fase 1: A Expansão dos Grafos Sintéticos (*Coarse Graphs*)

Durante a fase 1, CGE expande cada grafo sintético e grava um subconjunto das possíveis maneiras de implementação da conexão. Para cada $G(V, A)$, a fase de expansão produz um grafo expandido, chamado $D(N, E)$. O conjunto N denota o conjunto de vértices de D e E o seu conjunto de arcos, com cada arco referindo-se a um específico segmento de fio em um FPGA. Os arcos são rotulados com um número

que se refere ao respectivo segmento de fio.

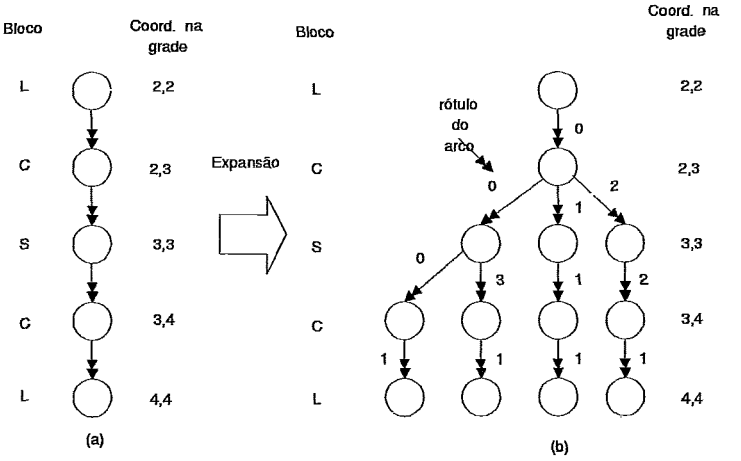

Figura 3.2: (a) Um grafo sintético típico e (b) seu grafo expandido.

Funções tipo caixa preta são usadas para representar a topologia de interconexão dos blocos L e C. A função caixa preta usada para um bloco C é denotada como $f_C([d_1, d_2, l], d_3)$ e para um bloco S como $f_S([d_1, d_2, l], d_3)$. Os parâmetros entre colchetes definem um arco que conecta o vértice d_1 ao vértice d_2, usando um segmento de fio rotulado por l. Tal arco mais adiante será referido como e, onde $e = (d_1, d_2, l)$. O parâmetro d_3 é o vértice sucessor de d_2. O resultado da expansão de um grafo é ilustrado na Figura 3.2. A tarefa da chamada da função pode ser descrita como: se um segmento de fio rotulado por l é usado para conectar d_1 a d_2, quais são os segmentos de fio que podem ser usados para alcançar d_3 a partir de d_2?. A função chamada retorna o conjunto de arcos que respondem a esta questão. Vale ressaltar aqui que, ao usar estas funções do tipo caixa preta, torna-se o algoritmo independente com respeito à arquitetura de roteamento FPGA.

O processo de expansão do grafo para cada grafo sintético segue os seguintes passos:

CRIE D e dê a ele a mesma raiz que G. Torne o sucessor imediato do nó raiz de D o mesmo que para a raiz de G.

ENQUANTO fazendo uma busca em profundidade em D, enumere os caminhos que se originam em cada vértice de acordo com:

EXPANDA um vértice C em D chamando $f_C(e_C, n) = Z$, e_C é o arco em D que já foi escolhido para conectar C a partir do seu predecessor. n é o vértice sucessor requerido para C (em G) e Z é o conjunto de arcos retornados por $f_C(.)$. A chamada para $f_C(.)$ adiciona $|Z|$ arcos para D.

EXPANDA um vértice S em D chamando $f_S(e_S, n) = Z$, e_S é o arco em D que já foi escolhido para conectar S a partir do seu predecessor . n é o vértice sucessor requerido para S (em G) e Z é o conjunto de arcos retornados por $f_S(.)$. A chamada para $f_S(.)$ adiciona $|Z|$ arcos para D.

FIM-ENQUANTO

B . Fase 2: Encadeamento da Rede

Após a expansão, cada $D(N, E)$ pode conter um certo número de caminhos alternativos. CGE coloca todos os caminhos oriundos de todos os grafos expandidos dentro de uma única lista de caminhos. Então, baseado em uma função de custo, o roteador seleciona caminhos a partir desta lista; cada caminho selecionado define a rota de sua correspondente conexão.

Cada arco no grafo expandido possui um custo composto de duas partes: $c_f(e)$ mede a competição entre diferentes redes pelos mesmos segmentos de fios e $c_t(e)$ é um número que reflete o atraso de roteamento associado com o segmento de fio. Cada caminho tem um custo que é a soma dos custos dos seus arcos. CGE seleciona caminhos baseados sobre o custo c_t somente se o caminho corresponde a uma conexão com tempo crítico associado.

O custo c_f possui dois objetivos:

- Selecionar um caminho que tenha um efeito negativo relativamente pequeno sobre as conexões restantes, em termos de roteabilidade. O custo determina a seleção de caminhos que contém segmentos de fios que têm grande demanda.

- Ele é usado para identificar um caminho que é *essencial* para uma conexão. Tal conexão possui somente um caminho remanescente no FPGA, devido à eliminação de seus outros possíveis caminhos pelos caminhos selecionados anteriormente.

Assim, a fase 2 prossegue da seguinte maneira:

> PONHA todos os caminhos existentes nos grafos expandidos dentro de uma lista de caminhos.
>
> ENQUANTO a lista de caminhos não estiver vazia
>
> SE existirem caminhos na lista de caminhos que são sabidos serem essenciais.
>
> SELECIONE o caminho essencial que tenha o menor custo c_f
>
> SENÃO SE existirem caminhos na lista de caminhos que correspondam a conexões com tempos críticos associados.
>
> SELECIONE o caminho crítico que tenha o menor custo c_t
>
> SENÃO
>
> SELECIONE o caminho com o menor custo c_f
>
> MARQUE o grafo correspondente ao caminho selecionado como roteado - remova todos os caminhos deste grafo que estão na lista de caminhos.
>
> ENCONTRE todos os caminhos em que há conflito com o caminho selecionado e os remova da lista de caminhos. Se uma conexão perde todos os seus caminhos alternativos, expanda novamente seu grafo sintético - se isto resultar em nenhum novo caminho, a conexão é considerada não roteável.
>
> ATUALIZE os custos de todos os caminhos afetados.
>
> FIM-ENQUANTO

Para determinar se um arco e possui grande demanda, o roteador poderá simplesmente contar o número de ocorrências de e que estão em grafos expandidos de outras redes. Entretanto, algumas ocorrências de e são menos essenciais do que outras, devido à existência de outras alternativas para o respectivo grafo expandido. Então, o custo de um arco e que tem j outras ocorrências $(e_1, e_2, ..., e_j)$ é definido como

$$c_f(e) = \sum_j \frac{1}{alt(e_j)}$$

onde $alt(e_j)$ é o número de arcos em paralelo com e_j.

Para o caso especial onde $alt(e_j)$ é 0, e_j é um arco que é essencial para a conexão associada porque não existem mais outras alternativas. Neste caso qualquer caminho em um grafo que use e_j é identificado como essencial. Quando o cálculo de um custo revela que um caminho é essencial, CGE atribui a tal caminho a mais alta prioridade para roteamento.

Complexidade

Embora a definição do grafo de expansão implique que todos os caminhos possíveis em um FPGA sejam armazenados durante a expansão, isto não é prático porque o número de caminhos pode ser extremamente grande.

Então o número de caminhos nos grafos expandidos é reduzido por um mecanismo de poda que atua durante a expansão. O algoritmo que implementa o mecanismo de poda é parametrizado de tal forma que a quantidade de espaço de memória requerido seja controlada e que ainda assim os grafos expandidos contenham um número suficiente de alternativas por conexão.

Resultados

CGE foi usado para rotear diversos circuitos industriais implementados em FPGAs, tendo seus resultados comparados com um algoritmo de roteamento estilo labirinto (*maze router*). Estes mostraram que um roteador de labirinto necessita, em média, de 60% mais trilhas do que o CGE. Além disso, o número de trilhas W que o CGE necessitou para rotear cada um dos circuitos foi muito próximo do valor do mínimo obtido pelo roteador global, portanto com excelentes resultados. O tempo gasto de CPU em segundos variou de 25 a 215, para os circuitos BUSC, DMA, BNRE, DESM e Z03 [18]. Destes circuitos, o último e maior, tem 586 blocos e 2135 conexões.

3.6.2 Roteador detalhado SEGA

O algoritmo de roteamento detalhado SEGA (SEGment Allocator), em [64], sucessor do CGE, buscou enfocar não apenas a questão de área, mas também o aspecto de desempenho de velocidade, dentro de um outro modelo de arquitetura de FPGA, possuindo canais segmentados. De fato, isto representa um novo tipo de problema de roteamento.

Para que se possa entender este novo problema, observe-se a figura 3.3, que mostra três visões de uma seção de um canal de roteamento em FPGA baseado em *arrays*. Em cada visão, a figura ilustra as opções de roteamento disponíveis naquele canal para três diferentes conexões, denominadas A,B e C. Na figura 3.3, um segmento de fio no canal é mostrado como uma linha horizontal contínua, e um segmento de fio, que seja passível de uso por uma determinada conexão, é posto em

negrito. Um *switch* de roteamento que une dois segmentos horizontais é desenhado como um linha tracejada e um *switch* que une um segmento horizontal a um pino em um bloco lógico (L) é mostrado como um X. Finalmente, as ligações a pinos de blocos lógicos são desenhadas como linhas verticais. Como pode ser observado na Figura 3.3, a arquitetura de roteamento neste FPGA tem três trilhas e os *switches* de roteamento são distribuídos tais que somente as trilhas 2 e 3 podem conectar os pinos de blocos lógicos para a conexão A e somente as trilhas 1 e 2 podem ser usadas para conexões B e C. A seguir considere-se o problema de roteamento, inicialmente sob a perspectiva de completar todas as três conexões, considerando o uso de segmentos de fio de acordo com seus tamanhos.

Suponha-se que o roteador complete a conexão A primeiro. Se ele escolhe rotear a conexão A sobre a trilha 2, então uma das conexões B ou C irá falhar, pois ambas competem por uma única alternativa, que é a trilha 1. Por outro lado, se o roteador tivesse escolhido a trilha 3 para A, então B poderia usar a trilha 1 e C a trilha 2, ou vice-versa. Pode-se desta forma observar que decisões de roteamento tomadas para uma conexão podem desnecessariamente bloquear outras.

A solução de roteamento acima satisfaz a meta de completar todas as três conexões, mas somente uma das duas escolhas faz o melhor uso dos segmentos de fios disponíveis. Examinando os canais de roteamento, evidencia-se o fato de que a conexão B deve ser atribuída à trilha 2, uma vez que o segmento de fio existente na mesma se adequa perfeitamente ao tamanho da conexão. Isto também leva à melhor solução para a conexão C, uma vez que ela requer somente um segmento de fio na trilha 1 mas iria necessitar de 2 segmentos na trilha 2. Assim posto, conclui-se que algoritmos de roteamento para FPGA não devem considerar apenas o estabelecimento com sucesso de todas as conexões, mas também considerar o número de segmentos de fios alocados por conexão.

Todos os trabalhos anteriores ao do SEGA consideravam apenas a questão de maximizar a roteabilidade sujeita a restrições de área. Este trabalho é, portanto, o primeiro a adicionar uma nova restrição ao problema que é a restrição de atraso, fazendo com que o número de segmentos alocados a uma conexão seja reduzido.

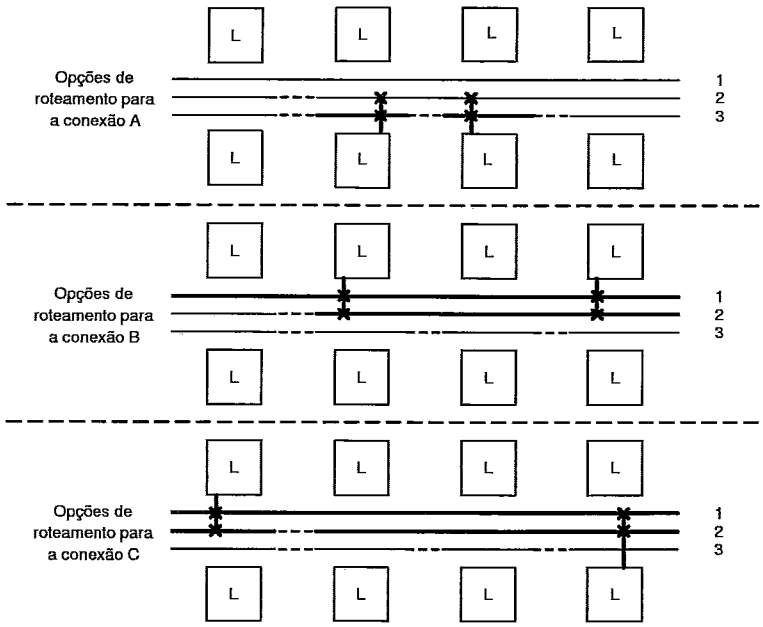

Figura 3.3: Um exemplo de um problema de roteamento de FPGA usando segmentos de fio de tamanhos distintos.

Procedimento de Implementação

Para implementar os circuitos de *benchmark* usados neste trabalho foram seguidos os seguintes passos de CAD: 1) os circuitos de *benchmark*, que foram originalmente desenvolvidos visando a uma implementação com *standard cell*, foram mapeados tecnologicamente dentro de células lógicas de um FPGA usando o algoritmo Chortle [35]; 2) as células lógicas resultantes do mapeamento tecnológico foram posicionadas em localizações específicas em um FPGA usando uma implementação do algoritmo min-cut [17]; 3) finalmente, as células lógicas foram interconectadas durante o roteamento. Este último passo foi repetido diversas vezes para diferentes parâmetros dos roteadores global e detalhado. Os resultados após o roteamento são analisados de duas maneiras: 1) se a taxa de sucesso de roteamento atingiu 100%; 2) quando todas as conexões foram roteadas com sucesso, qual é o desempenho de velocidade do resultado final?.

O roteador global utilizado pelo SEGA foi uma adaptação do algoritmo de roteamento global LocusRoute para *standard cell* [83]. Este roteador global divide as redes multiponto do circuito em conexões de dois pontos e encontra caminhos de mínima distância através dos canais para cada conexão. O principal objetivo do algoritmo é distribuir as conexões entre os canais de forma que as densidades de

canal sejam balanceadas.

Um exemplo de saída de um roteador global [77], que é chamado de grafo sintético (*coarse graph*), G , para uma única conexão em um FPGA muito pequeno é ilustrado sobre o lado mais à esquerda na Figura 3.4. Os vértices e arcos em G são identificados pelas coordenadas mostradas na figura para o FPGA e definem a seqüência de canais que o roteador global escolheu para conectar o bloco lógico na localização (0,4) com um outro em (4,0).

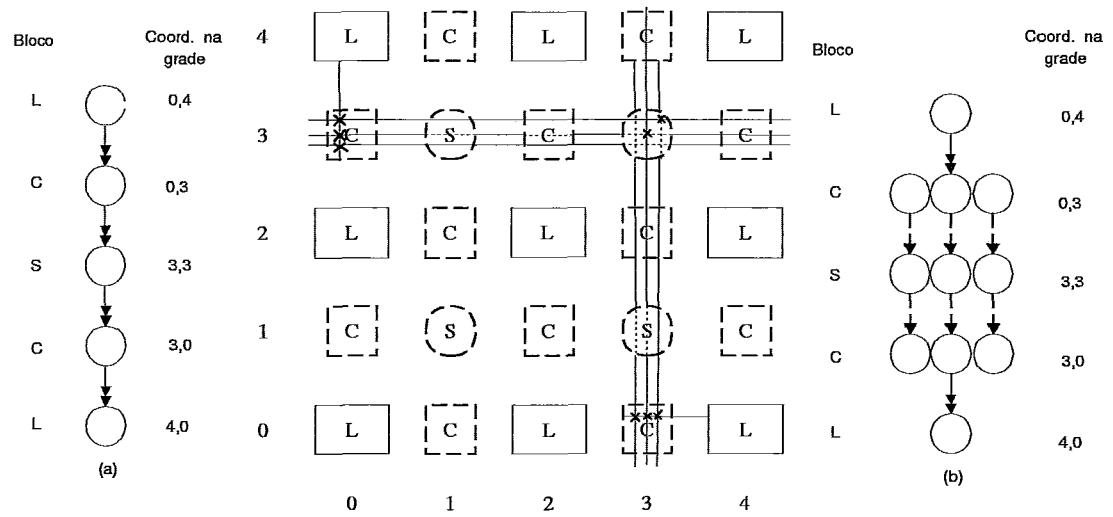

Figura 3.4: Grafo expandido D em (b) mostrando as Rotas Detalhadas para G em (a).

O roteamento detalhado é executado pelo SEGA (SEGment Allocator). Em termos gerais, SEGA é similar ao CGE, tendo como principal diferença a forma de tratar os segmentos de fios segundo os seus tamanhos. Assim, o SEGA consegue um resultado significativamente melhor do que o CGE (cerca de 25% melhor) em termos do desempenho de velocidade para os circuitos implementados.

Para rotear um circuito, SEGA inicialmente cria uma representação do FPGA para implementar cada rota global, a partir de um conjunto de parâmetros especificados pelo usuário, e então lê a saída do roteador global. Um grafo sintético é criado em uma estrutura de dados interna para cada conexão requerida. O roteamento detalhado então prossegue em duas fases principais: na fase 1, o roteador examina os segmentos de fios e *switches* de roteamento presentes no FPGA e enumera todas as alternativas para a rota detalhada de cada grafo sintético. Então, na fase 2, decisões

de roteamento específicas são feitas para cada conexão. Tais decisões são baseadas em funções de custo que refletem ou o atraso de roteamento associado com cada escolha, ou o efeito que cada alternativa irá ter sobre a roteabilidade do circuito completo.

Fase 1: Enumerando as Rotas Detalhadas

Como ilustrado na Figura 3.4, cada arco em D representa segmentos de fios específicos (um ou mais) que possam ser usados para implementar o arco correspondente em G. O grafo D tem os mesmos vértices que G, mas existe uma instância de cada vértice para cada caminho em um FPGA que parte do vértice raiz para o vértice folha. Os arcos de D são desenhados como linhas tracejadas para indicar que eles não são simples arcos. Cada arco, em D pode implicar o uso de múltiplos segmentos de fios. Nesse caso, múltiplas linhas tracejadas são mostradas. É importante frisar que o tamanho de um segmento de fio referenciado em D não é necessariamente o mesmo que o tamanho do arco correspondente em G, desde que um segmento de fio pode ser ou maior ou menor do que o próprio arco. Cada arco tem associado a ele um ou mais rótulos, um para cada segmento de fio que ele referencia. Os rótulos identificam os segmentos de fio correspondentes em um FPGA.

Fase 2: Seleção de Caminho

Nesta fase, SEGA coloca todos os grafos expandidos dentro de uma única lista de conexão. E então, baseado em funções de custo, o roteador seleciona um caminho para definir a rota detalhada para cada conexão na lista. Devido ao fato de que SEGA expande todos os grafos sintéticos antes de tomar qualquer decisão de roteamento, é possível considerar os efeitos colaterais que uma decisão feita para uma conexão venha a ter sobre as demais. O algoritmo desta fase é descrito a seguir, e mais adiante, serão enfocados os métodos para avaliar o custo de um caminho.

COLOQUE todas as conexões (grafos expandidos) em uma única lista de conexões.

ENQUANTO a lista de conexões não for vazia FAÇA

ORDENE a lista de conexões; SELECIONE o primeiro da lista.

ROTEIE a conexão selecionada usando o caminho de menor custo.

MARQUE a conexão como roteada e remova todos os caminhos desta conexão que estão na lista de conexões.

ENCONTRE todos os caminhos em que há conflito com o caminho selecionado e os remova como alternativas para as suas respectivas conexões. Se uma conexão perde a sua última alternativa de caminho, é então considerada não roteável.

ATUALIZE os custos de todos os caminhos afetados.

FIM-ENQUANTO

Se SEGA estiver sendo usado para otimizar área então a função de custo utilizada para sortear a lista de conexões é a mesma usada pelo CGE, *Demand(p)*, de modo que conexões que têm poucas rotas possíveis terão maior prioridade de escolha. Já quando busca-se otimizar o desempenho, SEGA ordena as conexões segundo os seus tamanhos e então faz uma seleção de caminho baseada no custo de uma função chamada *Delay(p)*. A função de custo *Demand(p)* para um caminho é a mesma que a função $c_f(.)$ usada pelo algoritmo CGE. A seguir a função *Delay(p)* é detalhada.

Função de Custo baseada em Desempenho de Velocidade

O propósito da função de custo *Delay(p)* é permitir ao SEGA selecionar quais caminhos representam a melhor escolha em termos de desempenho. Diferentes caminhos podem incluir maiores ou menores atrasos porque eles podem ter diferentes números de segmentos de fio ou seus segmentos de fio podem ser de tamanhos diferentes. Assim, avaliando *Delay(p)* com base no número e no tamanho dos segmentos de fio em um caminho, tem-se

$$Delay(p) = c_1 * NumSeg(p) + c_2 * SegLen(p)$$

onde $NumSeg(p)$ é similar à função de custo definida em [38] [88] e seu propósito é minimizar o número de segmentos de fios atribuídos para uma conexão. Os termos de custo são normalizados tal que eles variam de 0 a 1 e então $NumSeg(p)$ é definido como o quociente do "número real de segmentos de fio em p menos o mínimo possível" dividido pelo "número real de segmentos em p". $SegLen(p)$ é similar à

função usada em [88]. Seu propósito é minimizar a perda devido à atribuição de longos segmentos de fios para conexões curtas. Deste modo, $SegLen(p)$ é definido como o quociente do "comprimento total de segmentos de fio em p menos o comprimento total dos arcos no grafo sintético" dividido pelo "comprimento total dos segmentos de fio em p". Os fatores c_1 e c_2 são definidos como pesos binários que são utilizados para ligar ou desligar termos na função.

Roteando Redes Multipontos

Um aperfeiçoamento para SEGA considerado neste trabalho é a possibilidade do algoritmo considerar quais conexões são parte de redes multipontos, o que pode produzir melhores resultados do que considerar apenas o roteamento entre pares de pontos. Os resultados mostraram que esta possibilidade não é particularmente importante quando a meta de otimização é a roteabilidade, porque a função de custo *Demand(p)* tende a juntar conexões de dois pontos que são partes de uma mesma rede se elas se justapõem. Entretanto, quando a meta de otimização é desempenho de velocidade é vantajoso para SEGA *compartilhar* recursos de fiação entre conexões de uma mesma rede multiponto. Uma versão do SEGA modificado que pode ser usada para este propósito, é dada a seguir:

```
COLOQUE todas as conexões (grafos expandidos) em uma única lista de
conexões.
ENQUANTO a lista de conexões não for vazia FAÇA
        ORDENE a lista de conexões segundo o tamanho da rede.
        SELECIONE a rede mais longa.
        ORDENE as conexões da rede segundo os seus tamanhos.
        SELECIONE a conexão mais longa.
        ENQUANTO houver conexões não roteadas para a atual rede FAÇA
                SE esta é a primeira conexão roteada para a rede ENTÃO
                        ROTEIE a conexão selecionada usando o caminho
                mais rápido disponível.
                SENÃO
                        ROTEIE a conexão selecionada usando o caminho
                que tem o máximo número de segmentos compartilhados
                com a parte já roteada da rede.
                        SE tal caminho não está disponível ENTÃO
                                ROTEIE a conexão selecionada usando o ca-
                minho mais rápido disponível
                        FIM-SE
                FIM-SE
                MARQUE a conexão como roteada e remova todos os ca-
                minhos desta conexão que estão na lista de conexões.
                ENCONTRE todos os caminhos em que há conflito com
                o caminho selecionado e os remova como alternativas para
                as suas respectivas conexões. Se uma conexão perde a sua
                última alternativa de caminho, é então considerada não
                roteável.
                ATUALIZE os custos de todos os caminhos afetados.
        FIM-ENQUANTO
        MARQUE a rede como já roteada.
FIM-ENQUANTO
```

Pode ser observado, seguindo o pseudocódigo acima, que SEGA roteia todas as conexões de uma rede em particular antes de mover para uma outra rede. As redes são ordenadas por tamanho, tal que redes longas possam tentar obter vantagens através do uso de segmentos longos de fios. Este algoritmo tende a minimizar a resistência e capacitância das redes, resultando num incremento significativo no desempenho de velocidade.

Resultados

Em [64] temos uma extensa seção de resultados, dentre os quais podemos citar como sendo os mais importantes: um ganho médio de 25% em desempenho de velocidade dos circuitos avaliados neste trabalho, em relação aos melhores resultados publicados anteriormente, e a constatação de que, para qualquer que seja o esquema

de segmentação dos canais, há sempre uma melhoria no atraso médio das redes, embora quase sempre acompanhado do aumento do número mínimo de trilhas W do FPGA.

3.6.3 GBP (Greedy Bin-Packing)

Este trabalho [110] foi antecedido por [112], dos mesmos autores, onde após observarem que o problema clássico de roteamento de canal podia ser visto como um problema de *bin-packing* (empacotamento em "caixas") de intervalos 1-D, tiveram então a idéia de generalizar esta visão, de modo a enunciar o problema de roteamento FPGA 2-D como um problema de *bin-packing* de intervalos 2-D, e daí chegaram a interessantes conclusões. Primeiro, mostraram que este último problema podia ser reduzido em tempo polinomial ao problema de coloração de grafos, considerando-se a utilização de uma arquitetura disjunta. Segundo, o problema mesmo com esta redução, a qual associa um grafo G, transformando-o num problema de encontrar o número cromático deste grafo, denotado por $X(G)$, permanecia NP-completo. E finalmente, provaram que não há uma constante limitante sobre a taxa de mapeamento $\frac{W_d}{W_g}$, onde W_g e W_d correspondem, respectivamente, a mínima largura de canal requerido pelo roteamento global e detalhado, a menos que se imponha restrições sobre a topologia das rotas globais e como exemplo, os autores apresentam uma árvore H que atinge todos os blocos lógicos L, de modo que qualquer rota global representaria uma sub-árvore de H. Com isto a taxa $\frac{W_d}{W_g}$ ficaria limitada pela constante $\frac{3}{2}$. Infelizmente tais restrições topológicas não são aplicáveis na prática, pois a arquitetura do FPGA já vem definida pelo fabricante e não impõe restrições sobre a topologia das redes globais.

Será analisado, daqui por diante, um roteador de passo único, ou seja, um roteador que faz roteamento global e detalhado combinados em um só passo, o qual é baseado em heurísticas e tem como alvo a arquitetura detalhada no capítulo anterior. Tal roteador, na sua forma mais sofisticada, faz uso de duas heurísticas gulosas, que são acopladas de diversas maneiras. Como ambas as heurísticas têm uma mesma formulação básica, será detalhada somente esta.

Inicialmente será introduzido o modo como os autores formularam o problema, i.e., dado uma *netlist*, um posicionamento dos blocos lógicos e atribuição de pinos,

encontre um roteamento realizando todas as interconexões com o mínimo número de trilhas em uma arquitetura FPGA 2-D com blocos diagonais S e blocos C com flexibilidade total, i.e, $F_C = W$.

Em uma arquitetura FPGA 2-D disjunta, note-se que todas as trilhas com mesmo identificador de trilha formam domínios de trilhas disjuntos. As rotas detalhadas de quaisquer duas redes podem usar um mesmo domínio de trilha se, e somente se, elas não passam através se um mesmo bloco C. Além disso, uma dada rota global pode ser modelada num determinado grafo de coloração G_C, como mostrado na Figura 3.5, cujo número cromático é o mesmo que o número mínimo de trilhas necessárias para completar o roteamento detalhado [112]. Os vértices de G_C correspondem às redes e existe um arco entre dois vértices se, e somente se, suas respectivas conexões passam através de um mesmo bloco C.

Os autores apontam também neste trabalho uma anomalia inerente ao roteamento global, denominada GRA (*Global Routing Anomaly*) que faz com que não se possa afirmar que o roteamento global com menor densidade de canal sempre leva a uma economia de recursos de interconexão (i.e., número de segmentos de fio). Em alguns casos pode-se mesmo até chegar a um maior gasto de recursos de interconexão.

A Figura 3.5 ilustra tal anomalia, onde se tem dois exemplos de roteamentos globais válidos para o mesmo posicionamento de pinos (terminais). Pode ser observado, que para o segundo exemplo, embora a densidade de canal seja mais alta, ou seja, igual a 3, serão necessárias apenas três trilhas para completar o roteamento detalhado. Já para o primeiro exemplo, embora este tenha densidade de canal igual a 2, portanto mais baixa, serão necessárias quatro trilhas para completar o roteamento detalhado. O número de trilhas necessárias para completar o roteamento detalhado pode ser também visualizado pelo número mínimo de cores usadas para colorir os respectivos grafos G_C.

De modo a evitar tal anomalia, os autores adotaram uma técnica de roteamento em um único passo, consistindo em heurísticas eficientes, aplicadas ao problema formulado como um problema de *bin-packing*. Face à propriedade da arquitetura ser do tipo disjunta, podem ser visualizados os domínios de trilhas como *bins* a serem empacotados (preenchidos) por objetos(redes) de tamanhos (topologias) que se ajustem ao mesmo. Assim, então, o modelo conceitual apresentado é dado por:

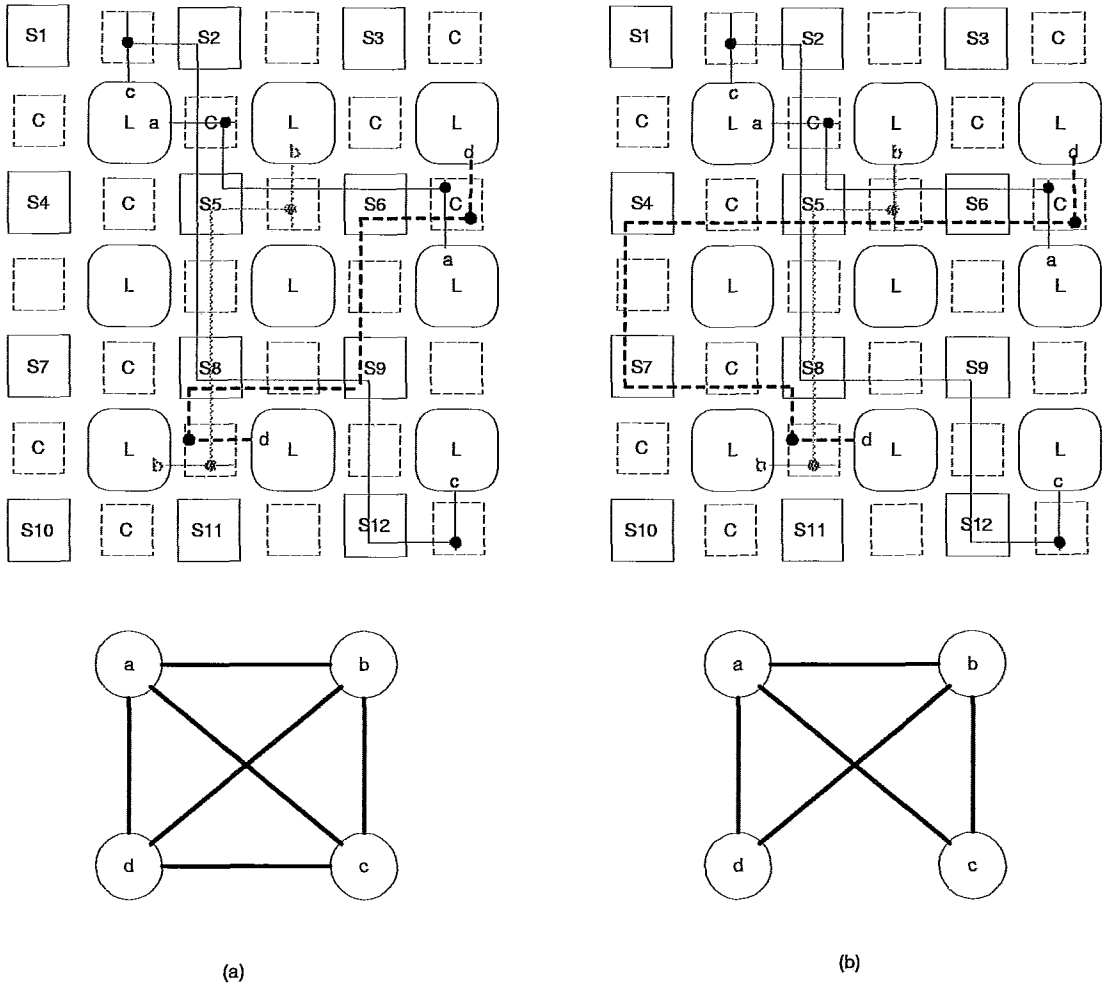

Figura 3.5: (a) A densidade de canal do roteamento global é somente 2, no entanto são necessárias quatro trilhas para completar o roteamento. (b) A densidade de canal do roteamento global é 3, mas agora somente três trilhas são necessárias para completar o roteamento.

- Um *bin* é um domínio de trilha.
- O objetivo é usar o mínimo número de *bins*.
- A geometria do *bin* é fixa e é a mesma para todos os *bins*. Para o problema de roteamento FPGA, a geometria de um *bin* é uma grade (malha), no estilo Manhattan.
- O tamanho dos objetos é variável, dependendo da topologia de cada rede.
- Alguns objetos não podem compartilhar um *bin*. Na terminologia de roteamento FPGA, tais objetos são redes, cujas rotas globais têm um bloco C em

comum.

O Roteador GBP (*Greedy Bin-Packing*) implementa a idéia de usar uma heurística *bin-packing* simples guiada pelo número de conexões pino a pino (redes de dois terminais) não roteadas de cada bloco C. Isto dá uma medida do congestionamento em cada bloco C, que é denominada de densidade de pinos de um bloco C. Em um roteador de um único passo, o roteamento global não é executado e as redes são diretamente roteadas de forma detalhada (roteamento detalhado). A cada vez que uma rede é roteada, a densidade de pinos de cada bloco C é atualizada e usada para selecionar a próxima rede a ser roteada.

Quando uma rede é processada, o algoritmo a atribui a um domínio de trilha como se estivesse empacotando-a em um *bin* ("caixa"). Este prossegue tentando rotear tantas redes quanto possível no domínio de trilha corrente até que o mesmo não possa acomodar mais qualquer uma das redes ainda não roteadas. Em seguida, passa a um novo domínio de trilha, elimina as redes roteadas da lista de redes e atualiza, se necessário, a densidade de pinos dos blocos C. A seguir, tem-se este algoritmo um pouco mais detalhado.

Algoritmo (GBP)

Passo 0 Decomponha todas as redes multipino em redes de dois pinos.

Passo 1 Para todo domínio de trilha

Processe toda a fila ordenada, consistindo em todas as redes de dois pinos, da maior para a menor. Faça pelo menos 5 (cinco) leituras desta fila, sendo que a cada leitura as regras de seleção são menos restritas;

primeira leitura: Selecione uma rede entre todas aquelas que têm pino localizado em um bloco C com a mais alta densidade de pino. Roteie (empacote) esta rede, e se possível, encontre um caminho com mínima distância Manhattan sem passar por blocos C com alta densidade de pino.

segunda leitura: Uma rede pode ser empacotada se é possível encontrar um caminho com mínima distância Manhattan sem passar por blocos C com alta densidade de pino.

terceira leitura: Selecione uma rede entre todas aquelas que têm pino localizado em um bloco C com a mais alta densidade de pino. A rede é roteada se existe um caminho para ela, tal que este seja no máximo um segmento maior do que a mínima distância Manhattan.

quarta leitura: Uma rede pode ser empacotada se é possível encontrar um caminho para ela, tal que este seja no máximo um segmento maior do que a mínima distância Manhattan.

quinta leitura: Se uma rede pode ser empacotada dentro do domínio correntemente empacotado, então ela é acomodada.

Esta heurística empacota tantas redes quanto possível de modo guloso dentro do domínio de trilha ainda não exaurido. Em cada domínio de trilha, a mais alta prioridade é dada para as redes mais longas que não provocam aumento na densidade de roteamento nos blocos C e que são roteáveis com mínima distância Manhattan. A idéia por trás disto é que heurísticas para *bin-packing* executam melhor quando os objetos maiores são colocados primeiro.

Os resultados apresentados para o GBP demonstram que o número total de trilhas usadas por este é 17% menor quando comparado aos melhores resultados existentes na literatura para roteadores de dois passos [18] [64]. A complexidade de tempo de CPU e de memória são $O(E * |L| * W)$ e $O(E + |L|)$, respectivamente, as quais são todas lineares no número $|E|$ de redes de dois terminais roteadas. Aqui, $|L|$ representa o número de LUTs e W é o número de trilhas de roteamento requerido. Outro dado interessante é que 90% das redes roteadas se mantiveram dentro da mínima distância Manhattan. Isto indica que o GBP tem um comportamento mais estável do que os roteadores de dois passos.

Ainda neste trabalho os autores apresentam duas versões aperfeiçoadas desta heurística básica, onde uma tenta reduzir o número de segmentos de roteamento, com um efeito implícito de redução do número de trilhas, e a outra tenta minimizar o número de domínios de trilhas, tendo como objetivo implícito minimizar o número de segmentos de fio. Estas duas são acopladas numa nova heurística, denominada acoplamento guloso ortogonal (*Orthogonal Greedy Coupling*) [111], assim chamada devido a ambas as heurísticas acopladas serem gulosas e ao mesmo tempo terem

objetivos ortogonais. Tal heurística consiste em usar a primeira heurística para preencher os primeiros bins e a partir de dado momento, alterna para a segunda heurística, ou ainda, usá-las de forma que os *bins* pares sejam preenchidos com base na primeira heurística e os ímpares com base na outra. Em ambas as situações, há um pequeno ganho de qualidade na solução, com especial atenção ao problema do baixo nível de ocupação dos últimos *bins*, cujo efeito é minimizado.

3.6.4 IKMB (Iterado Kou, Markowsky e Berman)

Em [1] os autores buscaram desenvolver um método que conseguisse combinar informação geométrica (por exemplo, para o objetivo de minimização do comprimento de fio) e topológica (por exemplo, para a seleção do caminho menos congestionado). É fato que uma técnica puramente topológica para o roteamento de FPGA pode ser incapaz de explorar certas informações geométricas disponíveis, tais como: proximidade física de pinos/blocos, distância da borda do FPGA, etc. Por outro lado, uma técnica estritamente geométrica pode falhar por não conseguir levar em consideração as restrições topológicas e combinatórias decorrentes da natureza discreta e limitada dos recursos de roteamento disponíveis num FPGA.

Deste modo, os autores desenvolveram um algoritmo híbrido, denominado IKMB (Iterated-KMB), que combina os algoritmos : (i) método de roteamento Iterado 1-Steiner [52], o qual sabidamente tanto tem um desempenho empírico excelente quanto é computacionalmente eficiente; (ii) método KMB (Kou, Markowsky e Berman) [59], que é um esquema de aproximação de Steiner em grafo, que pode ser implementado de forma eficiente. Para que se possa enunciar este algoritmo híbrido, é apresentado a seguir uma descrição geral dos dois algoritmos.

O algoritmo Iterado 1-Steiner aborda o seguinte problema: dado um conjunto P de n pontos sobre um plano Manhattan, encontre um conjunto S de pontos de Steiner tais que a árvore geradora mínima (MST) sobre $P \bigcup S$ tenha custo mínimo. A solução deste problema corresponde a árvore de Steiner retilínea mínima. O algoritmo se comporta da seguinte forma: dados os conjuntos P e S, definem-se as economias da MST de S com respeito a P como

$$\Delta\widehat{MST}(P,S) = \widehat{MST}(P) - \widehat{MST}(P \cup S)$$

onde a notação $\hat{T}$ representa a soma dos custos de todos os arcos da árvore T.

Denota-se por $H(P)$ o conjunto de Hanan dos pontos de Steiner candidatos (i.e., as interseções de todas as linha verticais e horizontais passando através de pontos de P). Para um dado conjunto de pontos P, um ponto 1-Steiner $x \in H(P)$ maximiza $\Delta\widehat{MST}(P, \{x\}) > 0$. O método I1S repetidamente procura pontos 1-Steiner e os inclui em S. O custo da MST sobre $P \cup S$ irá decrescer à medida que os pontos são adicionados e a construção da árvore termina quando não houver mais nenhum x com $\Delta\widehat{MST}(P, \{x\}) > 0$.

O segundo algoritmo, o método KMB, enfoca o seguinte problema: dado um grafo com pesos $G = (V, E)$, ou seja, cada arco $e_{ij} \in E$ tem um peso w_{ij}, e uma rede de terminais $N \subset V$ para conectar, encontre uma árvore $T' = (V', E')$ com $N \subseteq V' \subseteq V$ e $E' \subseteq E$ tal que $\sum_{e_{ij} \in E'} w_{ij}$ seja minimizado.

A solução deste problema é conhecida como árvore de Steiner mínima em grafo (GSMT). Este problema é NP-completo, portanto suscita a necessidade do uso de heurísticas, dentre estas se notabiliza o método KMB, o qual inicialmente constrói um grafo completo G' sobre N, i.e. ,$G' = (N, N \times N)$, com o peso de cada arco e_{ij} igual a $dist_G(N_i, N_j)$, o custo do caminho mais curto em G entre N_i e N_j .Em seguida, calcule MST(G'), a árvore geradora mínima de G' e expanda cada arco e_{ij} de MST(G') dentro do correspondente caminho mais curto, denotado $path_G(N_i, N_j)$, atribuindo um subgrafo G'' que cobre N. Finalmente, calcule MST(G'') e elimine arcos desnecessários da MST(G'') até que todas os nós folhas sejam membros de N. A árvore resultante será denotada como KMB e o seu custo como $\widehat{KMB}$.

Agora, portanto, pode-se enunciar o algoritmo híbrido IKMB. Este corresponde a uma modificação do algoritmo Iterado 1-Steiner, de modo que ao gerar uma árvore que interligue um subconjunto N de pontos em um grafo, se utilize não mais uma MST e sim uma árvore de Steiner. Logo, ao invés de calcular a MST para que se possa determinar as "economias" de um nó (ponto) de Steiner candidato, é usado o algoritmo KMB para este propósito. Assim, dado um grafo $G = (V, E)$, a rede $N \subseteq V$ e um conjunto S de potenciais pontos de Steiner, a economia obtida é definida como sendo

$$\Delta\widehat{KMB}_G(N, S) = \widehat{KMB}_G(N) - \widehat{KMB}_G(N \cup S)$$

O algoritmo IKMB inicia computando a árvore KMB. Então, a cada iteração

o método IKMB repetidamente encontra nós candidatos de Steiner adicionais que reduzam o custo KMB total e os inclui no conjunto crescente de nós de Steiner S. O custo da árvore KMB sobre $N \cup S$ irá decrescer à medida que cada nó é adicionado e a construção da árvore termina quando não há mais nenhum $x \in V$ com $\Delta \widehat{KMB}_G(N \cup S, \{x\}) > 0$.

Um outro aspecto interessante apresentado em [1] é que esta técnica possibilita otimizar múltiplas funções objetivo de forma simultânea, através de uma generalização da heurística IKMB para grafos multi-pesos, onde cada critério de otimização tem seu próprio conjunto de arcos com pesos. Aqui a multi-objetividade é alcançada pela transformação desses arcos multi-peso em um único arco com peso correspondente à média ponderada destes. E então, a técnica pode ser aplicada como descrita anteriormente.

Os autores testaram o roteador IKMB para 10000 redes, geradas randomicamente, de cardinalidade (número de pinos) 3, 4, 5, 7 e 10, com coordenadas uniformemente distribuídas no intervalo de 0 a 10000. Então, cada rede foi roteada em um grafo do tipo grade induzido pela interseção das linhas verticais e horizontais passando através dos pinos. Os resultados obtidos foram comparados com os obtidos pelo método KMB, e demostram que, para todos os problemas testados, o IKMB gasta menos fio, ou seja, há uma diminuição do comprimento médio das redes, variando de 3,31% a 6,72%. O IKMB também tem seu desempenho comparado ao do CGE demostrando em alguns casos uma certa melhoria. A Tabela 3.1 resume esta última comparação.

Tabela 3.1: Comparação entre os roteador CGE e IKMB, em termos de máxima largura de canal requerida.

Circuito	CGE	IKMB	Tamanho do FPGA
BUSC	10	8	12 x 13
DMA	10	9	16 x 18
BNRE	12	11	21 x 22
DFSM	10	11	22 x 23
Z03	13	13	26 x 27

3.6.5 VPR (Versatile Placement and Routing)

Em [14] é apresentado algoritmo VPR que pode ser usado para executar posicionamento, roteamento global ou roteamento global/detalhado juntos. Dentre os trabalhos mais recentes, o VPR tem destacado sucesso, pois comparando seus resultados com os dos outros algoritmos, para os vinte maiores circuitos do benchmark MCNC [114], há uma expressiva redução do número mínimo de trilhas necessárias para concluir o roteamento do FPGA.

Desde que o VPR é capaz de executar a fase de *placement* das *netlists* do circuito e o roteamento global das redes, é possível combiná-lo com outros algoritmos que executem o roteamento detalhado. Além disto, o VPR oferece um vasta gama de parâmetros que podem ser usados para modificar o seu comportamento, como, por exemplo, o fator de redução de temperatura a cada iteração do seu algoritmo de *simulated annealing* para a fase de *placement.*

Uma outra característica bastante interessante é a sua flexibilidade quanto à arquitetura alvo do FPGA a ser roteado, permitindo que esta seja especificada em um arquivo, que será lido pelo VPR após a leitura das *netlists* do circuito. Também é possível modificar a flexibilidade dos blocos C e S.

Este trabalho é oriundo de um forte grupo de pesquisa em ferramentas de CAD para FPGA existente na Universidade de Toronto, e pode ser considerado como o sucessor dos algoritmos CGE [18] e SEGA [64], ambos desenvolvidos por este mesmo grupo. Aqui, no entanto, o roteamento detalhado, não mais é executado por expansão das rotas a partir de uma grafo sintético (*coarse graph*), mas sim por um algoritmo num estilo *maze* (labirinto), que corresponde a um aperfeiçoamento da implementação tradicional do algoritmo de Dijkstra (i.e. um roteador *maze* [61]). Para uma rede com k terminais, o roteador *maze* é invocado $(k-1)$ vezes para executar todas as conexões requeridas. Na primeira vez, o roteador *maze* inicia um onda a partir de um terminal e continua expandindo esta rede até que algum dos $(k-1)$ terminais da rede seja alcançado. E então o caminho do nó fonte para o nó destino corresponde a primeira parte do roteamento da rede. Em seguida, o roteador *maze* "esvazia" a onda em curso, e inicia um outra a partir da parte da rede já roteada até então. Assim, é fácil notar que após $(k-1)$ invocações do roteador *maze* todos os k terminais da rede estarão conectados.

Infelizmente, esta técnica consume muito tempo de CPU para o roteamento de redes que interligam terminais em um grande número de blocos lógicos. Tais redes, geralmente, permeiam uma grande parte do FPGA ou, possivelmente, todo ele. Além disso, ao analisar-se a última utilização do roteador *maze*, constata-se que a expansão, a partir da parte da rede já roteada até então, é computacionalmente cara, uma vez que certamente levará muito tempo para que a onda a ser expandida atinja o próximo terminal destino. Visando a sobrepujar esta dificuldade, o VPR implementa uma variação deste método. Quando um terminal destino de uma rede é alcançado, o VPR adiciona todos os recursos de roteamento (segmentos) necessários para conectar o terminal destino ao roteamento parcial corrente da rede (parte da rede já roteada) para a onda em expansão, com um custo zero. E diferentemente do anterior, este não esvazia a atual onda em expansão; pelo contrário, continua expandindo-a normalmente. Como o novo caminho adicionado ao roteamento parcial tem um custo zero, o roteador *maze* irá primeiro expandir em torno deste. E desde que este novo caminho é tipicamente pequeno, o custo computacional para adicioná-lo a esta nova onda em expansão é pequeno, e o próximo terminal destino será alcançado muito mais rapidamente do que se toda a onda fosse expandida em todo o seu entorno.

Foram testados os cicuitos presentes na tabela 3.2 que correspondem aos vinte maiores circuitos do *benchmark* MCNC [114], contendo FPGAs com um número de blocos lógicos variando entre 1047 e 8383. Em cada linha tem-se o nome do circuito, o seu tamanho em termos de blocos lógicos, o seu número de redes, o número de trilhas usadas para completar a tarefa (*placement* + roteamento global + roteamento detalhado) executado pelo VPR, e na última coluna, o número de trilhas usadas para completar o roteamento detalhado executado pela ferramenta SEGA, antecedida do VPR fazendo o *placement* e o roteamento global. Estes resultados demonstram o excelente desempenho do VPR em termos de máxima largura de canal requerida.

3.7 Análise Detalhada de Trabalhos Anteriores sobre Roteadores FPGA Paralelos

Com o rápido crescimento do nível de integração dos dispositivos FPGA, e por conseguinte, um forte aumento no tempo gasto pela fase de roteamento, tornou-se

Tabela 3.2: Descrição dos vinte maiores circuitos do *benchmark* MCNC.

Circuito	# Blocos Lógicos	# Nets	VPR (# trilhas)	SEGA (# trilhas)
alu4	1522	1536	10	16
apex2	1878	1916	11	20
apex4	1262	1271	12	19
bigkey	1707	1936	7	9
clma	8383	8445	12	25
des	1591	1847	7	11
diffeq	1497	1561	7	10
dsip	1370	1599	7	9
elliptic	3604	3735	10	16
ex1010	4598	4608	10	22
ex5p	1064	1072	13	16
frisc	3556	3756	11	18
misex3	1397	1411	10	17
pdc	4575	4591	16	33
s298	1931	1935	7	18
s38417	6406	6435	8	10
s38584.1	6447	6485	9	12
seq	1750	1791	11	18
spla	3690	3706	13	26
tseng	1407	1099	6	9
TOTAL	—	—	197	334

natural a idéia de se explorar paralelismo, principalmente com a popularização de redes de estações de trabalho.

Entretanto, o problema de roteamento possui características próprias que dificultam a implementação de uma solução paralela. Isto decorre do alto grau de interdependência entre o roteamento das redes de um circuito. Durante a pesquisa bibliográfica realizada para este trabalho foram encontrados apenas dois artigos, Chan e Schlag [25] e Chan, Schlag, Ebeling e McMurchie [26], que apresentam soluções para a paralelização da fase de roteamento especificamente em FPGAs. As idéias destes artigos são resumidas nesta seção.

3.7.1 Aceleração de um Roteador FPGA

Chan e Schlag [25] examinam métodos para acelerar um roteador FPGA em um ambiente de computaçao distribuída de redes de estações de trabalho, tendo co-processadores FPGA anexados ou múltiplas placas FPGAs. O roteador FPGA uti-

lizado para os experimentos foi o roteador seqüencial *Pathfinder* [70], que é aplicável a diversas arquiteturas FPGA. O *Pathfinder* é um roteador baseado no paradigma de negociação para FPGA, desenvolvido por McMurchie e Eberling. Este paradigma é baseado na adequação dos custos associados aos recursos de roteamento, a cada iteração, para resolver os conflitos de congestionamento.

Como a principal atividade de qualquer roteador é a busca por caminhos para as redes de sinal, foi desenvolvida uma estratégia de paralelização na qual a aceleração do roteamento está focada primordialmente na implementação em *hardware* desta fase de busca por caminhos. Assim, o paralelismo de granularidade fina é explorado no roteamento individual das redes.

O paralelismo de granularidade grossa é obtido pela separação de redes, de modo que estas possam ser roteadas em paralelo. Entretanto, há uma grande dificuldade em se fazer isto, pois o roteamento de redes separadas é fortemente acoplado. A dificuldade é que a rota tomada por cada rede depende do conhecimento das rotas das outras redes, uma vez que os recursos de roteamento não podem ser compartilhados. Tal dificuldade é superada pelo *Pathfinder* ao permitir que os recursos sejam compartilhados temporariamente, deixando estes conflitos para serem resolvidos nas próximas iterações através da adequação dos custos associados aos recursos de roteamento sendo compartilhados.

A estratégia adotada então é separar as redes por processador, cada um destes com uma cópia completa do grafo de roteamento detalhado e fazer a atualização local do congestionamento de recursos. É fácil imaginar um cenário no qual duas redes conflitantes podem oscilar entre duas rotas, desde que cada uma conheça a rota da outra rede na iteração anterior. Para tentar resolver isto, o grau de paralelismo, na atualização do congestionamento global dos recursos de roteamento, é limitado e uma alocação geográfica de redes a processadores é adotada.

Duas formas de comunicação entre os processadores foram avaliadas para a cooperação entre os processadores para a atualização dos custos de congestionamento. No modo síncrono, cada processador atualiza sua informação local sobre os custos de congestionamento dos recursos de roteamento, de acordo com as rotas das redes associadas a este processador, e ao final da iteração, todas as mudanças de custos são sincronizadas e cada processador começa a próxima iteração com custos globais

de congestionamento precisos. No modo assíncrono, cada processador comunica as mudanças nos custos de congestionamento resultantes do roteamento das redes associadas a ele, tão logo elas ocorram, para os outros processadores e atualiza sua própria informação local de acordo com as mensagens recebidas.

Como a informação sobre congestionamento local não é tão atual como na versão seqüencial, é possível que o *Pathfinder* paralelo nao convirja, ou que um grande número de iterações sejam necessárias, eliminando assim qualquer ganho de *speedup*. Entretanto, os autores relatam que este não foi o caso. Ressalte-se aqui que, quando restam poucas redes, os autores forçam o roteamento em um único processador, de modo a obter convergência.

Para a versão paralela do *Pathfinder* com comunicação síncrona entre os processadores também é relatada a convergência do algoritmo, embora tenha necessitado de um número maior de iterações. Entretanto, para uma execução usando 10 processadores para o circuito teste alu4 foi obtido apenas um *speedup* de 1,54.

A versão paralela do *Pathfinder* com comunicação assíncrona, sobre uma rede de estações de trabalho, utilizou um modelo mestre-escravo para a comunicação entre os processadores. O processador mestre ficava encarregado de receber e disseminar as atualizações de custos de congestionamento para todos os processadores. Os melhores *speedups* obtidos demonstraram um comportamento quase linear para dois pequenos circuitos testados, considerando cenários com 2, 3 e 4 processadores. Em particular, para o circuito s1A, quando 2 processadores foram usados, foi obtido um *speedup* de 2,37. Isto se deve ao fato de que as atualizações assíncronas da informação de congestionamento podem causar a obtenção de diferentes rotas; a versão paralela pode produzir um roteamento diferente do que o produzido pela versão seqüencial.

3.7.2 Um Roteador Paralelo FPGA baseado no Paradigma de Memória Distribuída

Chan, Schlag, Ebeling e McMurchie [26] mostram que o paradigama de roteamento baseado em negociação, aplicado com sucesso em diversos roteadores FPGA, pode ser paralelizado para alcançar melhor desempenho sem prejuízos significativos em termos de qualidade do roteamento. Acelerações deste paradigma incluem o uso de uma busca A* agressiva em [100], [104] e sua paralelização em [25]. Também é

apresentado um extenso estudo sobre a convergência do algoritmo de roteamento baseado em negociação.

Para que se determine uma estratégia de paralelização para um roteador FPGA deve ser considerado que tipo de paralelismo, se funcional ou de dados, poderá ser explorado pelo roteador. O núcleo de qualquer roteador emprega alguma forma de algoritmo de caminho mínimo. Uma abordagem de paralelismo funcional para paralelizar um roteador irá requerer a decomposição do núcleo do roteador em tarefas concorrentes. Uma abordagem de paralelismo de dados para paralelizar um roteador envolverá decompor o conjunto de redes do circuito a ser roteado em subconjuntos. Um típico grande projeto com FPGA tem muitos milhares de redes de sinal, de modo que o roteamento poderia ser trivialmente paralelizado se redes distintas pudessem ser roteadas em paralelo. Se isto for viável, o grau de paralelismo em princípio será limitado somente pelo número de redes de sinal. A dificuldade é que a rota tomada por cada rede depende do conhecimento das rotas das outras redes, visto que os recursos de roteamento não podem ser compartilhados.

Os autores fazem uma extensão do esquema da paralelização do algoritmo *Path-finder* apresentado em Chan e Schlag [25]. E uma nova meta de roteamento é adicionada que corresponde ao roteamento direcionado para minimizar o atraso (*delay-driven routing*). Como antes, este esquema de paralelização consiste em associar um subconjunto disjunto de redes para serem roteadas, a cada processador. Cada processador tem uma memória local privativa e cada processador está conectado a uma rede. Processadores trocam informações através de um mecanismo de troca de mensagens. Cada processador mantém sua versão local da estrutura de dados dos custos de congestionamento. A vantagem desta abordagem reside no fato de que nenhum mecanismo de exclusão mútua é necessário para manter a integridade da estrutura de dados. Entretanto, é possível que o valor do custo de congestionamento de um dado recurso possa não ser consistente entre diferentes processadores.

Em relação ao trabalho anterior de Chan [25], foi inserida uma variação no modo de comunicação assíncrona, onde um dado processador é designado para manter o controle das iterações correntes em cada processador. Com isto, se um processador está muito avançado, ou seja, já executou muitas ierações, então este entra num laço de espera pelos demais, onde nenhuma tarefa de roteamento é executada e somente

são feitas atualizações dos custos de congestionamento dos recursos de roteamento, de acordo com as mensagens recebidas. Isto se faz necessário porque a maneira segundo a qual os custos de congestionamento são atualizados depende do número de iterações k.

Tendo em mente que a distribuição das redes de sinal entre diferentes processadores afeta o balanceamento de carga e consequentemente o *speedup*, os autores ordenam as redes de sinal em ordem decrescente de *fanout* (espalhamento dos pinos de uma rede pela matriz do FPGA). Isto é feito para que as redes com maior *fanout* sejam roteadas primeiro e as redes de baixo *fanout* por último, uma vez que estas são menos críticas em termos de tempo e têm mais rotas alternativas. Além disso, como o tempo de roteamento de uma rede depende do seu *fanout*, a lista de redes é preprocessada para alcançar melhor balanceamento de carga entre os processadores. Em seguida, a lista é particionada entre os processadores de modo que cada processador fique com aproximadamente o mesmo número de nós destinos (*sinks*).

Resultados são apresentados para um conjunto de circuitos reais sobre uma arquitetura FPGA comercial, a família XC4000 FPGAs produzida pela Xilinx. O número de redes destes circuitos varia entre 118 e 1542 redes. Os resultados demonstram que esta implementação do *Pathfinder* paralelo tem bom desempenho, atingindo um *speedup* de 2,5 quando 3 processadores são usados. Mas este desempenho satura rapidamente, pois quando 4 ou mais processadores são utilizados o *speedup* obtido fica assintoticamente limitado a 3. Na verdade, para um único destes circuitos avaliados são obtidos melhores *speedups*, mas isto se deve ao fato deste circuito possuir muitos módulos pequenos.

Capítulo 4

Uma Heurística para o Problema da Árvore de Steiner Retilínea

O problema da árvore de Steiner retilínea consiste em tentar encontrar uma árvore de comprimento mínimo conectando um conjunto de terminais. Este é um dos problemas fundamentais na automação do roteamento de redes de circuitos eletrônicos, uma vez que uma interconexão de comprimento mínimo tem mínima capacitância total e também requer uma mínima quantidade de área. Assim sendo foi desenvolvida uma heurística bastante rápida, com baixa complexidade computacional, capaz de gerar boas soluções aproximadas, por vezes ótimas, para este problema, visando a seu uso no roteamento de redes em FPGAs, que tipicamente interconectam um pequeno conjunto de terminais, em média, menos de cinco terminais.

4.1 Introdução

Considere i e j dois pontos quaisquer pertencentes ao plano euclideano, com coordenadas (x_i, y_i) e (x_j, y_j), respectivamente . Defina o custo de um arco conectando i e j, como sendo igual a distância retilínea $|x_i - x_j| + |y_i - y_j|$. Então se T é um conjunto de k pontos (vértices) no plano, o problema de Steiner retilíneo é o problema de interligar os vértices em T, de tal forma que seja minimizado o custo total dos arcos usados. O conjunto de todos os vértices é denominado por V. É bem conhecido, que a solução para este problema será uma *árvore geradora mínima* (MST), sobre algum conjunto de vértices $T \cup S$, onde $S \subset V - T$ é chamado o conjunto de vértices de Steiner. Tal árvore é chamada a *árvore de Steiner retilínea mínima* (MRST).

A pesquisa sobre o problema de como encontrar a MRST tem sido guiada por

diversos resultados fundamentais. Primeiro, Hanan [42] mostrou que sempre existe uma MRST com pontos de Steiner escolhidos a partir dos pontos de intersecção de todas as linhas horizontais e verticais passando através de todos os pontos em T (veja Figura 4.2). E este resultado foi generalizado por Snyder [96] para todas as dimensões. Entretanto, um outro resultado devido a Garey e Johnson [37] mostrou que a despeito desta restrição sobre a espaço de soluções, o problema de encontrar a MRST permanece NP-completo, o que tem motivado um grande número de heurísticas para este problema. Um estudo amplo pode ser encontrado em Hwang, Richards e Winter [49] e em Maculan [67]. A Figura 4.1 mostra uma MST retilínea e uma MRST para um conjunto $|V|$ com 4 pontos, sobre um grafo suporte formado pela interseção das linhas verticais e horizontais passando pelos terminais.

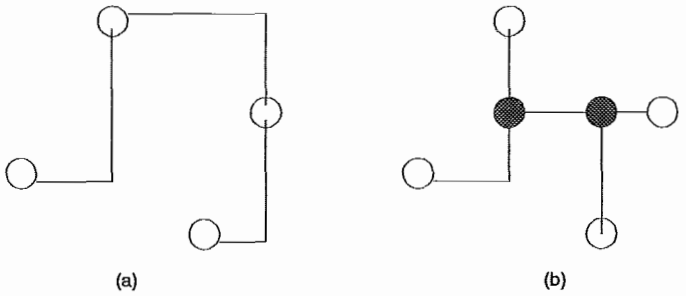

Figura 4.1: (a) Uma árvore geradora mínima retilínea (MST) e (b) uma MRST, onde os nós não hachuriados são os nós terminais do conjunto T e os nós hachuriados representam o conjunto S de pontos de Steiner.

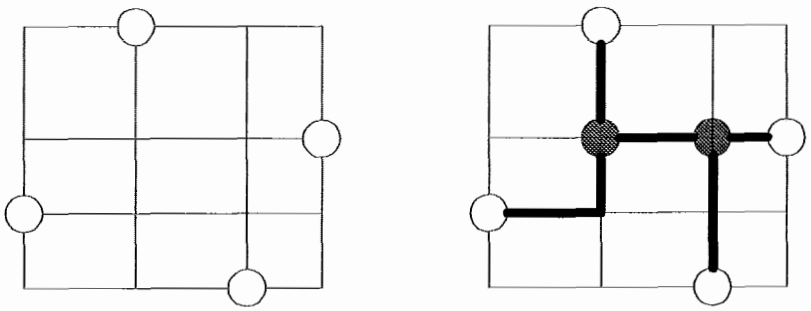

Figura 4.2: Teorema de Hanan. Sempre existe uma MRST com pontos de Steiner escolhidos a partir das intersecção de todas as linhas verticais e horizontais passando através de todos os pontos.

Yang and Wing [115] [116] [117] apresentam um algoritmo exato para o problema de Steiner, o qual relata resultados apenas para problemas muito pequenos, envolvendo no máximo 10 vértices. Outro algoritmo exato foi apresentado por Bahiense,

Maculan e Sagastizábal [9] que se baseia no uso do algoritmo do volume aplicado à relaxação lagrangeana de uma formulação inteira de fluxos não simultâneos de uma única mercadoria, onde para todos os problemas com $|V| \leq 3000$ e $|T| \approx 2\%V$ foi alcançada a otimalidade. Contudo, ainda que para FPGAs pequenos com apenas algumas centenas de redes, estes algoritmos exatos demandam um enorme esforço computacional, fazendo com que não se apliquem no contexto de roteamento em FPGAs.

Beasley [12] apresenta um algoritmo heurístico que estabelece a cada iteração uma melhoria da MST corrente através da inserção da MRST para cada subgrafo conexo com 4 vértices da MST. Lee, Bose e Hwang [62] apresentam um algoritmo similar ao algortimo de Prim [78] para MST. Hsu, Pan e Kubitz [46] apresentam algoritmos baseados no algoritmo de Prim [78] e de Kruskal [60] para a MST. Richards [81] apresenta uma implementação eficiente de um algoritmo devido a Hanan. Ho, Vijayan and Wong [43] apresentam um algoritmo também baseado em transformar a MST retilínea numa MRST.

Uma característica comum à maioria das heurísticas presentes na literatura tem sido a de tentar encontrar uma aproximação da MRST a partir de reduções sucessivas de custo para a MST retilínea inicial. No entanto, todas as heurísticas deste tipo possuem uma razão de desempenho de pior caso limitada a 3/2, uma vez que foi provado por Kahng e Robins [53] que $custo(MST)/custo(MRST) \leq 3/2$. No entanto, esta razão é a mesma para a simples MST retilínea, conforme demonstrado por Hwang [48]. Isto então tem motivado o desenvolvimento de esquemas alternativos para a aproximação da MRST, tendo dentre estes com melhor desempenho os algoritmos Iterado 1-Steiner [53] e Iterado-KMB [2].

Tanto em [53] como em [2], a característica inovadora é a de incorporar ao algoritmo alguma informação da geometria do problema, melhor dizendo, do modo como os terminais estão distribuídos. Infelizmente estes algoritmos possuem complexidade de tempo de $O(n^4 \log n)$. Griffith, Robins, Salome e Thang [39] apresentam uma melhoria da implementação do algoritmo Iterado 1-Steiner que possui complexidade de tempo $O(n^3)$.

Em Cabral, Aude e Maculan [22] é apresentada uma heurística que também se baseia na idéia de incorporar ao algoritmo alguma informação acerca da disposição

geométrica dos terminais. E isto é feito, de um modo inovador, ao ser definida uma noção de equidistância de um arco a todos os terminais. Este valor de equidistância representará o novo peso do respectivo arco, e com base nisto, é encontrada uma MST sobre a grade de Hanan. Em seguida, encontra-se o mínimo subgrafo conexo desta MST que interliga todos os terminais. O algortimo que implementa esta heurística apresenta duas outras características inovadoras: primeiro, ele consiste de um único passo; segundo, ele possui menor complexidade de tempo $O(kn)$, onde $k = |T|$, quando comparado com a melhor implementação do algortimo Iterado 1-Steiner.

4.2 Heurística Proposta

A idéia básica da heurística proposta é dirigir o algoritmo guloso de Kruskal [60] para MST, de modo que a MST encontrada contenha como um subgrafo seu a MRST (*Minimal Rectilinear Steiner Tree*). E desde que o algoritmo de Kruskal [60] para MST escolhe a cada passo o arco de custo mais baixo, que não gere ciclos, para ser incorporado a MST em construção, deve-se forçar que os arcos mais prováveis de virem a pertencer a MRST sejam escolhidos o mais cedo possível. Isto é feito através da substituição da métrica de distância retilínea para avaliação do custo do arco por uma nova medida que representa a equidistância do arco em relação aos terminais.

Assim, o peso de um arco será tanto menor quanto mais próximo, ou equidistante, estiver este arco de todos os nós a serem interligados.

Define-se então, o valor de equidistância de um arco $e = (i, j)$, como sendo

$$equidist(e) = \sum_{t \in T} \max \{\zeta(i,t), \zeta(j,t)\}$$

onde T é o conjunto de terminais e $\zeta(i,t)$ é o custo do caminho mínimo do nó i ao nó t.

Para ilustrarmos esta idéia consideremos o grafo $G = (V, E)$, $|V| = 6, |E| = 7$, com custos associados aos arcos, que refletem a distância retilínea entre seus nós, mostrado na Figura 4.3. Seja o conjunto de terminais $T = \{1, 3, 5\} \subset V$. É fácil observar que a MST(G) tem custo ótimo igual a 11, havendo três possibilidades para tal MST(G), como ilustrado na Figura 4.4.

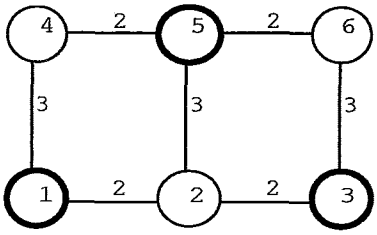

Figura 4.3: Grafo exemplo.

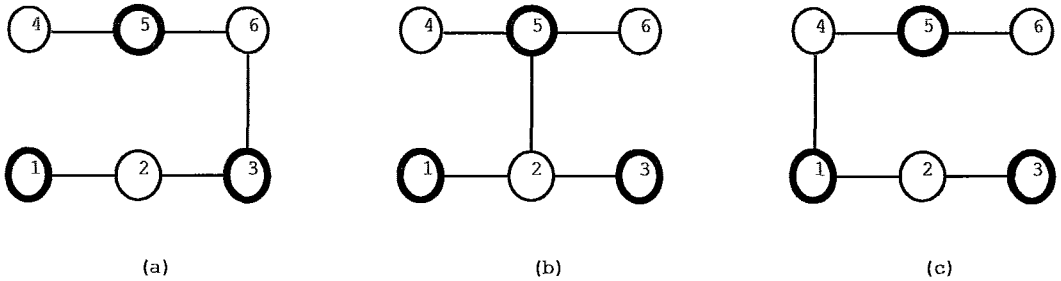

Figura 4.4: MST(G) ótimas.

Pode-se perceber ainda que a árvore de Steiner retilínea mínima (MRST) para o grafo G, que interliga todos os vértices pertencentes $T \subset V$, possui custo ótimo igual a 7, ou seja, MRST(G)=7. Esta é obtida a partir de uma MST(G) através da poda recursiva de todos os nós folhas não terminais presentes na MST(G) e de seus arcos adjacentes. A Figura 4.5 ilustra a MRST(G).

A heurística proposta busca então induzir que qualquer MST(G) encontrada possua um subgrafo conexo que corresponda a uma boa aproximação da MRST(G), em termos de custo. Assim, tem-se que tentar fazer com que a heurística conduza para a MST da Figura 4.4(b), considerando o exemplo suprareferido.

Para tanto, calcula-se um valor de equidistância para cada arco. Por exemplo, para o arco (5,6), cujo nó adjacente mais afastado do terminal 1 dista 7 unidades, do terminal 3 dista 5 e do terminal 5 dista 2, resulta num valor de equidistância igual a 14. A figura 4.6 mostra o grafo G_e com os valores de equidistância para todos os seus arcos.

Agora se o algoritmo de Kruskal [60] for aplicado ao grafo G_e, ter-se-á a seguinte ordem de escolha de arcos na formação da MST(G_e), a saber: (1,2),(2,3),(2,5),(4,5) e (5,6). E então, após efetuar-se a poda dos nós não terminais e seus arcos adjacente na MST(G_e) é obtida a MRST(G) mostrada na Figura 4.5. Fazendo assim, é induzida uma ordem de escolha dos arcos que formarão uma MST(G), de modo que esta

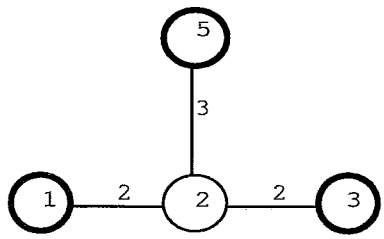

Figura 4.5: MRST(G) ótima.

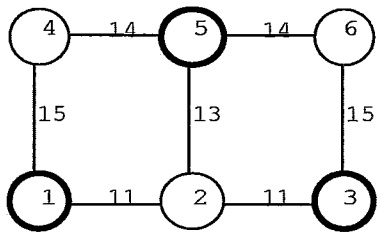

Figura 4.6: Grafo exemplo com valores de equidistância nos arcos.

venha a conter uma árvore de Steiner retilínea que seja uma boa aproximação da MRST(G).

4.3 Algoritmo

A seguir os passos do algoritmo que implementa a heurística proposta são enunciados:

Algoritmo (Aproximação da Árvore de Steiner Retilínea)

Passo 0 Para cada arco $e = (i, j) \in E$ calcule

$$equidist(e) = \sum_{t \in T} \max \{\zeta(i,t), \zeta(j,t)\}$$

onde $\zeta(i,t)$ é igual a soma dos custos de cada arco pertencente ao caminho mínimo que interliga o nó i ao terminal t.

Passo 1 Encontre a MST de G, usando como peso para cada arco o seu respectivo valor de equidistância. Esta pode ser facilmente obtida usando [48],[60] ou [78].

Passo 2 Elimine arcos na MST encontrada, se necessário, de modo que todos os nós folhas restantes sejam nós terminais. Tal árvore retilínea obtida desta forma corresponderá a uma aproximação da MRST para T em G.

A análise de complexidade deste algoritmo mostra que no passo 0, cada arco da grade é visitado duas vezes, durante a fase de expansão de cada nó terminal. De modo que, isto atribui a este passo uma complexidade de $O(k|E|)$, onde a cardinalidade do conjunto de arcos $|E| \approx 2n$ e $|V| = n$. Resultando assim numa complexidade de $O(kn)$ para o passo 0.

No passo 1, é possivel usar um algoritmo, como o proposto por Hwang [48], para construção da MST com complexidade igual a $O(n \log n)$. A complexidade do passo 2, que consiste na eliminação de nós e arcos da MST, é igual a $O(n)$. Portanto, pode-se perceber que se $k < \log n$, então o algoritmo tem como dominante a complexidade do passo 1, ou seja, $O(n \log n)$, caso contrário a complexidade do passo 0, $O(kn)$, torna-se dominante.

Cada passo deste algoritmo é executado uma única vez, o que o torna bastante interessante no tocante ao esforço computacional exigido. Além disso, em termos de complexidade de pior caso, ele supera os algoritmos com melhores resultados presentes na literatura, que são os algoritmos Iterado 1-Steiner [53] e Iterado-KMB [2], ambos com complexidade igual a $O(n^3)$.

O valor de equidistância de um arco guarda uma relação de visão global com todos os terminais, em contraste com a geração de um caminho mínimo entre um par de nós levando em consideração somente a distância retilínea como o custo de cada arco. Com esta informação global, é possível escolher o caminho contendo os arcos que estejam mais equidistantes de todos os outros terminais, quando existir mais de uma possibilidade para o caminho mínimo. Isto explica porque algumas heurísticas de sucesso para o problema de Steiner em grafos, como os algoritmos propostos por Kou, Markowsky e Berman (KMB) [59] e por Takahashi e Matsuyama [102], não tenham o mesmo sucesso, quando os pesos associados aos arcos correspondem à distância retilínea entre seus nós adjacentes. No entanto, se estas forem precedidas do passo 0 do algoritmo proposto aqui e trabalharem sobre o grafo agora com arcos rotulados com pesos de equidistância, então um comportamento similar ao da heurística proposta neste trabalho deve ser esperado.

O algoritmo proposto pode ser seguido por uma fase de refinamento, onde para cada terminal o grafo é dividido em dois subgrafos, quando possível. Um deles consistirá em todos os nós e arcos que estiverem a direita do terminal e o outro

consistindo em todos os nós e arcos remanescentes. Então o algortimo proposto é invocado para encontrar um MRST em cada subgrafo e finalmente as árvores encontradas são combinadas, de modo a formar a MRST final.

4.4 Resultados Computacionais

As tabelas mostradas aqui apresentam os resultados obtidos para os problemas presentes na OR-Library [11] . As tabelas 4.1 e 4.2 apresentam os resultados obtidos para o conjunto de problemas testes R, o qual é devido a Soukup e Chow [98], quando o algoritmo é executado sem e com fase de refinamento, respectivamente. Em ambas as tabelas, as colunas $|V|$, $|E|$, $|T|$ e Ótimo indicam o número de vértices, o número de arcos, o número de terminais e o comprimento da MRST, respectivamente, de cada instância do conjunto R. E as colunas MST, #MST(%), ST Rect. e #STR(%) mostram o valor da solução obtida pela heurística MST retilínea, o erro da solução decorrente desta solução $(MST/Otimo) * 100$, o valor da solução obtida pela heurística proposta neste trabalho e o erro desta solução $(STRect./Otimo) * 100$, respectivamente. A coluna Δ dá o percentual de redução alcançado pela heurística proposta $[(MST - STRect.)/Otimo] * 100$, onde um valor positivo indica um ganho em relação a heurística MST retilínea.

Para o conjunto R, composto de 46 instâncias com número de terminais variando de 3 a 62, pode-se constatar que a heurística proposta se comporta muito bem para problemas com até 4 terminais, tendo alcançado otimalidade para todos os problemas deste tipo. Porém, à medida que o número de terminais aumenta, o desempenho da heurística degrada bastante, pois a métrica de equidistância perde a sua representatividade. Isto acontece quando há muitos terminais na região interna do menor retângulo que engloba todo o conjunto de terminais.

De modo apenas a estabeler uma comparação com os outros algortimos que buscam máxima redução de custo em comparação com a MST retilínea, foi calculada a redução média alcançada para as 15 instâncias dos conjuntos ES10, ES20, ES30 e ES40, com 10, 20, 30 e 40 terminais, respectivamente, que são devidos a Beasley [11]. A taxa de redução média obtida foi de -0.88%, -11.01%, -25.98% e -29.73% quando a fase de refinamento não é executada. Estas taxas de redução melhoram significativamente quando a fase de refinamento é aplicada. Estas tornam-se 7.03%,

-0.56%, -8.42% e -12,68%, respectivamente. Isto mostra que a heurística proposta obtem piores resultados do que a heurística MST retílinea quando o número de terminais é muito grande (ES30 e ES40).

No tocante ao esforço computacional, o algoritmo que implementa esta heurística é bastante parcimonioso. Todas as instâncias do conjunto R executaram em menos de 1 segundo num PC Pentium II 300Mhz com 64 Mb de memória RAM.

Tabela 4.1: Resultados para o conj. de problemas R sem fase de refinamento.

Prob.	$\|V\|$	$\|E\|$	$\|T\|$	Ótimo	MST	#MST(%)	ST Rect.	#STR(%)	Δ
R01	15	22	5	187	197	5.35	187	0.00	5.08
R02	12	17	6	164	183	11.59	164	0.00	10.38
R03	28	45	7	236	259	9.75	260	10.17	-0.38
R04	64	112	8	254	266	4.72	284	11.81	-6.76
R05	12	17	6	226	249	10.18	226	0.00	9.24
R06	24	38	12	242	268	10.74	242	0.00	9.70
R07	30	49	12	248	274	10.48	248	0.00	9.49
R08	24	37	12	236	262	11.02	288	22.03	-9.92
R09	15	22	7	164	202	23.17	172	4.88	14.85
R10	36	60	6	177	205	15.82	177	0.00	13.66
R11	30	49	6	144	161	11.81	144	0.00	10.56
R12	27	42	9	180	190	5.56	210	16.67	-10.52
R13	42	71	9	150	160	6.67	185	23.33	-15.62
R14	36	60	12	260	260	0.00	260	0.00	0.00
R15	100	180	14	148	160	8.11	198	33.78	-23.75
R16	9	12	3	160	160	0.00	160	0.00	0.00
R17	48	82	10	200	202	1.00	249	24.50	-23.26
R18	182	337	62	404	454	12.38	529	30.94	-16.52
R19	168	310	14	188	213	13.30	227	20.74	-6.57
R20	6	7	3	112	142	26.79	112	0.00	21.12
R21	15	22	5	192	230	19.79	196	2.08	14.78
R22	16	24	4	63	66	4.76	63	0.00	4.54
R23	16	24	4	65	73	12.31	65	0.00	10.96
R24	16	24	4	30	35	16.67	30	0.00	14.28
R25	9	12	3	25	28	12.00	25	0.00	17.86
R26	9	12	3	15	17	13.33	15	0.00	11.76
R27	16	24	4	133	143	7.52	133	0.00	6.99
R28	12	17	4	24	280	16.67	24	0.00	14.28
R29	9	12	3	200	210	5.00	200	0.00	4.76
R30	28	45	12	110	125	13.64	110	0.00	12.00
R31	130	237	14	259	311	20.08	336	29.73	-8.03
R32	210	391	19	313	362	15.65	347	10.86	4.14
R33	132	241	18	268	301	12.31	311	16.04	-3.32
R34	272	511	19	241	275	14.11	290	20.33	-5.45
R35	240	449	18	151	171	13.25	187	23.84	-9.35
R36	6	7	4	90	90	0.00	90	0.00	0.00
R37	49	84	8	90	93	3.33	105	16.67	-12.90
R38	100	180	14	166	168	1.20	209	25.90	-24.40
R39	100	180	14	166	168	1.20	201	21.08	-19.64
R40	64	112	10	155	186	20.00	168	8.39	9.67
R41	144	263	20	224	245	9.38	273	21.88	-11.43
R42	81	144	15	153	164	7.19	192	25.49	-17.07
R43	195	362	16	255	296	16.08	301	18.04	-1.69
R44	196	364	17	252	266	5.56	312	23.81	-17.29
R45	270	507	19	220	247	12.27	237	7.73	4.05
R46	16	24	16	150	150	0.00	150	0.00	0.00

Tabela 4.2: Resultados para o conj. de problemas R com fase de refinamento.

Prob.	$\|V\|$	$\|E\|$	$\|T\|$	Ótimo	MST	#MST(%)	ST Rect.	#STR(%)	Δ
R01	15	22	5	187	197	5.35	187	0.00	5.08
R02	12	17	6	164	183	11.59	164	0.00	10.38
R03	28	45	7	236	259	9.75	242	2.54	6.56
R04	64	112	8	254	266	4.72	274	7.87	-3.00
R05	12	17	6	226	249	10.18	226	0.00	9.24
R06	24	38	12	242	268	10.74	242	0.00	9.70
R07	30	49	12	248	274	10.48	248	0.00	9.49
R08	24	37	12	236	262	11.02	236	0.00	9.92
R09	15	22	7	164	202	23.17	164	0.00	18.81
R10	36	60	6	177	205	15.82	177	0.00	13.66
R11	30	49	6	144	161	11.81	144	0.00	10.56
R12	27	42	9	180	190	5.56	210	5.56	0.00
R13	42	71	9	150	160	6.67	165	10.00	-3.12
R14	36	60	12	260	260	0.00	260	0.00	0.00
R15	100	180	14	148	160	8.11	174	17.56	-8.75
R16	9	12	3	160	160	0.00	160	0.00	0.00
R17	48	82	10	200	202	1.00	219	9.50	-8,41
R18	182	337	62	404	454	12.38	480	18.81	-5.72
R19	168	310	14	188	213	13.30	206	9.50	3.28
R20	6	7	3	112	142	26.79	112	0.00	21.12
R21	15	22	5	192	230	19.79	196	2.08	14.78
R22	16	24	4	63	66	4.76	63	0.00	4.54
R23	16	24	4	65	73	12.31	65	0.00	10.96
R24	16	24	4	30	35	16.67	30	0.00	14.28
R25	9	12	3	25	28	12.00	25	0.00	17.86
R26	9	12	3	15	17	13.33	15	0.00	11.76
R27	16	24	4	133	143	7.52	133	0.00	6.99
R28	12	17	4	24	280	16.67	24	0.00	14.28
R29	9	12	3	200	210	5.00	200	0.00	4.76
R30	28	45	12	110	125	13.64	110	0.00	12.00
R31	130	237	14	259	311	20.08	294	13.51	5.46
R32	210	391	19	313	362	15.65	344	9.90	4.97
R33	132	241	18	268	301	12.31	311	16.04	-3.32
R34	272	511	19	241	275	14.11	270	12.03	1.81
R35	240	449	18	151	171	13.25	176	23.84	-2.92
R36	6	7	4	90	90	0.00	90	0.00	0.00
R37	49	84	8	90	93	3.33	97	7.76	-4.3
R38	100	180	14	166	168	1.20	199	19.87	-18.45
R39	100	180	14	166	168	1.20	201	21.08	-19.64
R40	64	112	10	155	186	20.00	162	4.51	12.90
R41	144	263	20	224	245	9.38	259	15.62	-5.71
R42	81	144	15	153	164	7.19	178	16.33	-8.53
R43	195	362	16	255	296	16.08	286	12.15	3.37
R44	196	364	17	252	266	5.56	291	15.47	-9.39
R45	270	507	19	220	247	12.27	229	4.09	7.28
R46	16	24	16	150	150	0.00	150	0.00	0.00

Capítulo 5

Formulações de Programação Inteira

O problema de roteamento FPGA corresponde, em sua essência, ao problema de gerar árvores de Steiner, uma para cada rede de sinal (*net*), e empacotá-las dentro dos canais de roteamento do FPGA. O objetivo é minimizar a máxima largura de canal. Também é desejável que o comprimento total das árvores de Steiner (comprimento total das redes) seja minimizado, melhorando assim o desempenho do circuito.

5.1 Introdução

Consideraremos o problema de geração e empacotamento de árvores de Steiner num grafo de roteamento de FPGA. Este é o problema fundamental na fase de roteamento em projeto de circuitos baseados em FPGA. Tal grafo é ilustrado na figura 5.1, onde cada recurso de roteamento (segmento de fio, pino de E/S de bloco lógico e pino de entrada e saída (*pads* E/S) do circuito) corresponde a um vértice deste grafo. Se existe uma ligação unidirecional entre quaisquer dois recursos de roteamento s e t do grafo de roteamento, então haverá um arco $a = (s, t)$ orientado. Em Lengauer [65] tem-se uma excelente fundamentação teórica sobre os diversos problemas combinatórios que aparecem no projeto de circuitos integrados, provendo também uma notação bem definida, a qual será parcialmente adotada neste trabalho.

A partir do grafo suporte mostrado em 5.1(b), consideremos um grafo de roteamento G=(V,E), não orientado, comprimentos $w_e \in \Re_+$ e capacidades $c_e \in \mathcal{Z}_+$, para cada arco $e \in E$. $\Omega = N_i, i = 1, 2, ..., k$ é uma família de subconjuntos do conjunto

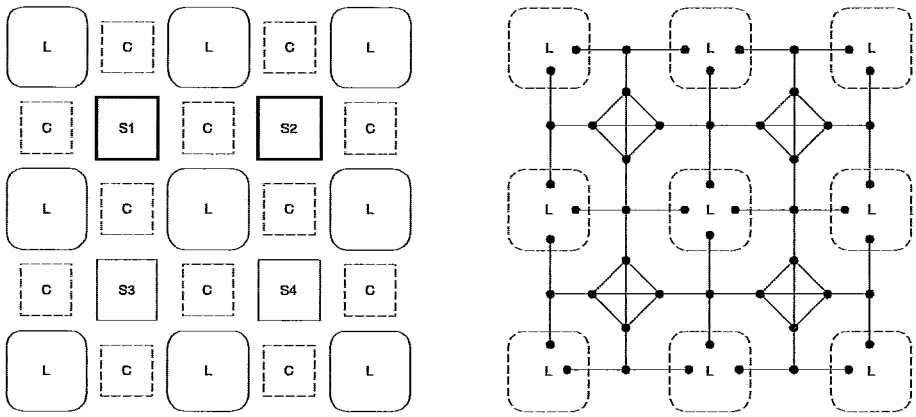

Figura 5.1: (a) Um exemplo de FPGA e seu (b) grafo suporte.

de nós V. Ω é chamado de *netlist* e cada subconjunto N_i define uma rede i. Os nós em N_i são os terminais da rede i. Denotemos $\Gamma = T_i, i = 1, 2, ..., k$ como sendo uma família de árvores tal que T_i conecta os nós em N_i. Tal árvore T_i é chamada uma árvore de Steiner para a rede i. Γ é dito ser um *roteamento*. O *congestionamento* $U(\Gamma, e)$ sobre um arco e para um roteamento Γ é o número de árvores T_i em Γ que contem o arco e.

$$U(\Gamma, e) = |\{i | T_i \in \Gamma, e \in T_i\}|. \tag{5.1}$$

O comprimento total de um roteamento Γ é dado por

$$w(\Gamma) = \sum_{e \in E} w_e U(\Gamma, e). \tag{5.2}$$

A sobrecarga sobre um arco $e \in E$ em um roteamento Γ é definida como

$$L(\Gamma, e) = \max\{0, U(\Gamma, e) - c_e\}. \tag{5.3}$$

Um arco é dito estar *supersaturado* se $L(\Gamma, e) > 0$. Um roteamento é considerado *válido* se nenhum arco está supersaturado. Os problemas de roteamento global podem ser restritos ou irrestritos.

Roteamento Global Restrito: Um roteamento *válido*, solução do problema de roteamento global restrito, é um roteamento Γ tal que nenhum arco $e \in E$ está supersaturado. O custo de um roteamento Γ é dado por (5.2). Então entre todos os roteamentos que roteiam todo o conjunto de redes Ω é escolhido aquele com menor comprimento total de fio.

Roteamento Global Irrestrito: Cada solução do problema de roteamento global irrestrito é um roteamento *válido*. O custo do roteamento é

$$\min_{e \in E} L(\Gamma, e) + w(\Gamma)$$

Assim, uma solução deste problema é um roteamento com mínima máxima largura de canal, e dentre estes aquele com com menor comprimento total de fio.

Ambas as versões do problemas de roteamento são conhecidas como sendo NP-difícil, pois incluem, como um caso especial, o problema de encontrar uma árvore de Steiner de comprimento mínimo, o qual é NP-difícil [37].

Métodos de programação inteira têm sido considerados por Lengauer [65], Lengauer e Lugering [66], Chopra [29], e Grotschel et al [41] [40].

O objetivo principal aqui é estimar mais precisamente limites inferiores para o problema do roteamento global irrestrito. E depois usar este limite inferior para restringir a máxima largura de canal na formulação do problema de roteamento detalhado, e que também poderá ser usado para definir a melhor política do número inicial de trilhas por processador na aplicação paralela desenvolvida neste trabalho. Se as restrições de capacidades sobre os arcos são ignoradas pode-se obter um simples limite inferior pela construção de um roteamento em que se utilize a árvore de Steiner mínima para cada rede.

5.2 Uma Formulação para o Problema de Roteamento Global

A seguir uma formulação de programação inteira para o problema de roteamento global irrestrito é enunciada. Em nosso trabalho, o objetivo é encontrar o máximo congestionamento sobre os arcos, ou seja, a mínima largura de canal W, definida como

$$W = \min_{e \in E} L(\Gamma, e). \tag{5.4}$$

Para a formulação do modelo, nos basearemos na idéia de modelar o problema da construção de cada árvore de Steiner T_i para sua respectiva rede i, como um problema de fluxo (ver Maculan [67]), onde um dos terminais da rede (*net*) é designado

como nó fonte ou origem (*source*) e os demais são designados como nós sumidouros ou destinos (*sinks*). De modo que, dado uma rede i haverá um um fluxo unitário partindo do seu nó fonte para cada um dos seus nós destinos.

Inicialmente o grafo $G = (V, E)$, não orientado, é transformado em um correspondente grafo $D = (V, A)$, orientado. Dado G=(V,E) com conjunto de arcos E, contrói-se o grafo $D = (V, A)$, orientado, com conjunto de arcos A, onde os arcos $a = (f, h)$ e $a' = (h, f)$ estão em A se e somente se o arco $e = [f, h]$ está em E. O comprimento w_a de um arco a é igual ao comprimento do correspondente arco não orientado e. Para cada rede i tem-se um dos vértices $s_i \in N_i$ como nó fonte. Para cada rede i e terminal $t \in N_i - \{s_i\}$ é definida uma mercadoria i, t. Dado um arco $a = (f, h)$ tem-se x_{fh}^{it} representando o fluxo da mercadoria i, t sobre o arco $a = (f, h)$. De modo a assegurar uma árvore de Steiner para cada rede $i \in \{1, 2, ..., k\}$, impõe-se um fluxo de uma unidade da mercadoria i, t oriunda de s_i para cada $t \in N_i - \{s_i\}$. Isto é feito usando-se as restrições de fluxo (5.5). Seja $I(f)$ o conjunto de todos os vértices $h \in V$ tal que $(h, f) \in A$ e $O(f)$ o conjunto de todos os vértices $h \in V$ tal que $(f, h) \in A$. Assim, um roteamento Γ sobre D consiste de um conjunto de árvores de Steiner, uma para cada rede (*net*).

$$\sum_{h \in O(f)} x_{fh}^{it} - \sum_{h \in I(f)} x_{hf}^{it} = \begin{cases} 1 & se \quad f = s_i \\ -1 & se \quad f = t \\ 0 & se \quad f \neq s_i, t, \end{cases} , \; f \in V, \; t \in N_i - \{s_i\} \qquad (5.5)$$

Se $(f, h) \notin A$ então a variável x_{fh}^{it} é simplesmente ignorada. As restrições de fluxo (5.5) são escritas para cada $(f, h) \in A$ e para toda rede $i \in \{1, 2, ..., k\}, t \in N_i - \{s_i\}$. Encontrando-se um fluxo satisfazendo (5.5) entre cada s_i e seus destinos $t \in N_i - \{s_i\}$, os arcos com fluxo positivo contêm uma árvore de Steiner gerando a rede i. Para capturar isto, são definidas as seguintes variáveis inteiras $y_{i,a}$ onde

$$y_{i,a} = \begin{cases} 1 & \text{se existe um fluxo positivo de qualquer das mercadorias } i, t \\ & \text{para } t \in N_i - \{s_i\} \text{ sobre o arco } a. \\ 0 & \text{caso contrário.} \end{cases}$$

O problema de roteamento global irrestrito pode então ser formulado da seguinte maneira:

$$\min \quad W$$

$$\text{s.a} \quad x \text{ satisfazendo (5.5)},$$

$$x_{fh}^{it} \leq y_{i,a}$$

$$\forall a = (f,h) \in A, i \in \{1,2,...,k\}, t \in N_i - \{s_i\}. \quad (5.6)$$

$$\sum_{i=1}^{k} y_{i,a} + \sum_{i=1}^{k} y_{i,a'} - W \leq 0$$

$$\forall a = (f,h), a' = (h,f), e = [f,h]. \quad (5.7)$$

$$W, y, x \geq 0, y \in \{0,1\}. \quad (5.8)$$

Esta formulação é uma generalização da formulação de fluxo multi-mercadorias para o problema da árvore de Steiner dada por Claus e Maculan [68], Beasley [10] e Wong [108].

Este modelo matemático representa bem a essência do problema de roteamento global, no entanto, ainda que para FPGAs pequenos, o número de variáveis é extremamente elevado. Consideremos, por exemplo, o circuito teste alu4, o qual necessita de um FPGA 40 x 40, com uma única LUT de quatro entradas em cada bloco lógico, cujo correspondente grafo de roteamento, com aproximadamente 22.400 arcos e 1536 redes, resultará numa formulação exigindo mais de 34 milhões de variáveis, pois o número de variáveis é da ordem de $O(k|A|)$. Este número, no entanto, pode ser reduzido se forem geradas apenas variáveis considerando, para cada rede, o subgrafo do grafo de roteamento, que engloba todos os seus terminais. De fato, tal restrição é usada na prática pelos algortimos de roteamento, como no VPR [14], o qual tem sua fase de expansão restrita aos segmentos de fio de blocos de conexão C que estejam até 3 canais de roteamento além do menor retângulo que engloba todos os terminais da rede.

5.3 Estendendo a Formulação para o Problema de Roteamento Detalhado

Consideremos o grafo $D_d = (V_d, A_d)$ construído a partir do grafo $D = (V, A)$ usado na representação do problema de roteamento global, tal que $V_d = V$, e para todo $a \in A$, tem-se $a_q \in A_d, q = 1, 2, ..., W_d$, onde $W_d \geq W$ é uma estimativa da mínima

largura de canal, definida com base na estimativa W fornecida pelo problema de roteamento global irrestrito. E para todo $a \in A$ tem-se $w_{a_q} = w_a, q = 1, 2, .., W_d$.

O problema de roteamento detalhado pode então ser formulado como

$$\min \quad \sum_{a \in A} \sum_{i=1}^{k} w_a y_{i,a}$$

s.a x satisfazendo (5.5),

$$x_{fh}^{it} \leq y_{i,a}$$

$$\forall a = (f,h) \in A_d, i \in \{1, 2, ..., k\}, t \in N_i - \{s_i\}. \quad (5.9)$$

$$\sum_{i=1}^{k} y_{i,a} + \sum_{i=1}^{k} y_{i,a'} \leq 1$$

$$\forall a = (f,h), a' = (h,f), a, a' \in A_d. \quad (5.10)$$

$$W, y, x \geq 0, y \in \{0,1\}. \quad (5.11)$$

Esta formulação possui um número de variáveis extremamente elevado, tendo dimensão W vezes maior do que a dimensão do problema de roteamento global irrestrito, respectivo. Outra característica inerente a esta formulação é que ela modela implicitamente um bloco de interconexão S (*switch box*) com flexibilidade total, ou seja, cada segmento de fio de um lado de um bloco S pode se conectar a qualquer outro segmento de fio que esteja num dos outros 3 lados de bloco S. Isto, na prática, é não realístico, pois como o número de *switches* necessário para implementar este bloco S é de $O(W^2)$, resultaria num circuito ocupando uma área enorme. Na prática, apenas algumas conexões estão disponíveis. Por exemplo, num bloco S-diagonal apenas $6W$ conexões são implementadas. Portanto, esta formulação é de interesse apenas teórico.

Uma outra formulação para este problema pode ser obtida considerando-se o grafo de roteamento detalhado, mostrado na figura 2.11, e cuja visão ampliada de uma parte deste é mostrada na figura 2.12. A figura 5.2, mostrada novamente por questão didática, ilustra o grafo de roteamento detalhado.

Seja o grafo de roteamento $\hat{G}_d = (\hat{V}_d, \hat{A}_d)$, orientado, com capacidades inteiras $c_r, c_r \geq 0$, onde o conjunto de vértices $\hat{V}_d$ corresponde ao conjunto de recursos de roteamento presentes no grafo de roteamento detalhado FPGA, ou seja, cada segmento de fio, nós fonte e destino de bloco lógico, pino de E/S de bloco lógico e

pino de *pad* E/S, corresponde a um vértice $r \in \hat{V}_d$. Dados quaisquer dois recursos de roteamento s e t em $\hat{V}_d$, se existe uma ligação (um *switch* programável) que permita o sinal ir de s para t então haverá um arco $a = (s, t)$, $a \in \hat{A}_d$, orientado.

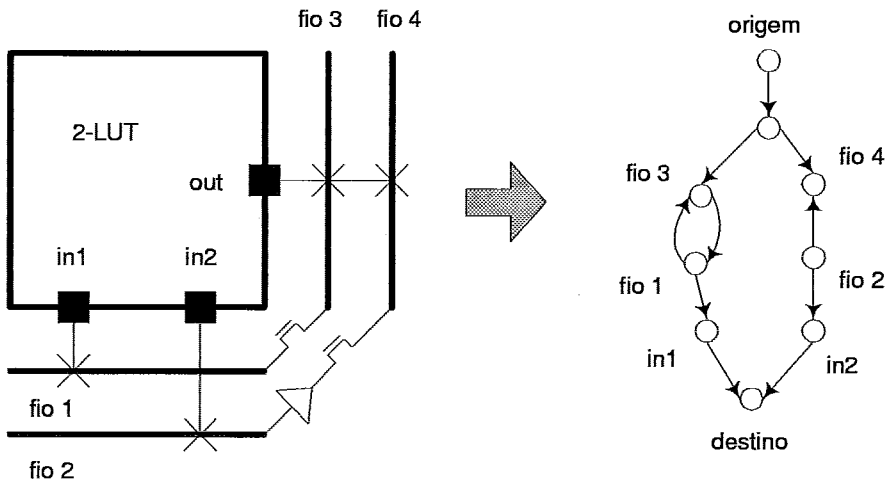

Figura 5.2: Modelando roteamento detalhado FPGA como um grafo orientado.

Do mesmo modo que para o problema de roteamento global, para cada rede i tem-se um dos vértices $s_i \in N_i$ como nó fonte. Para cada rede i e terminal $t \in N_i - \{s_i\}$ é definida uma mercadoria i, t. Dado um arco $a = (f, h)$ denota-se x_{fh}^{it} representando o fluxo da mercadoria i, t sobre o arco a. De modo a assegurar uma árvore de Steiner para cada rede $i \in \{1, 2, ..., k\}$, impõe-se um fluxo de uma unidade da mercadoria i, t oriunda de s_i para cada $t \in N_i - \{s_i\}$. Isto é feito usando-se as restrições de fluxo (5.12).

$$\sum_{h \in O(f)} x_{fh}^{it} - \sum_{h \in I(f)} x_{hf}^{it} = \begin{cases} 1 & se \quad f = s_i \\ -1 & se \quad f = t \\ 0 & se \quad f \neq s_i, t, \end{cases} \quad , f \in \hat{V}_d,\ t \in N_i - \{s_i\} \qquad (5.12)$$

Cada recurso de roteamento $r \in \hat{V}_d$ tem um peso associado w_r que representa um custo pela utilização do mesmo. Neste trabalho, w_r representa o comprimento do recurso. Também são definidas as seguintes variáveis inteiras $z_{r,i}$ onde

$$z_{r,i} = \begin{cases} 1 & \text{se o recurso } r \text{ é usado pela rede } i \text{ no transporte} \\ & \text{de suas mercadorias } (i,t),\ t \in N_i - \{s_i\} \\ 0 & \text{caso contrário.} \end{cases}$$

O problema de roteamento detalhado restrito consiste então em alocar recursos de roteamento $r \in \hat{V}_d$ a cada rede i, respeitando a disponibilidade de recursos de roteamento, e tendo como objetivo minimizar o comprimento total das redes. Este problema pode ser formulado da seguinte maneira:

$$\begin{aligned}
\min \quad & \sum_{r \in \hat{V}_d} \sum_{i=1}^{k} w_r z_{r,i} \\
\text{s.a} \quad & x \text{ satisfazendo (5.12)}, \\
& \sum_{h \in I(r)} \sum_{t \in N_i - \{s_i\}} x_{hr}^{it} \leq (|N_i| - 1) z_{r,i}, \forall i, \forall r \in \hat{V}_d. \qquad (5.13) \\
& \sum_{i=1}^{k} z_{r,i} \leq c_r, \forall r \in \hat{V}_d. \qquad (5.14) \\
& z, x \geq 0, z \in \{0,1\}. \qquad (5.15)
\end{aligned}$$

O conjunto de restrições (5.14) garante a exclusão entre redes sobre os recursos de roteamento (segmentos de fio, pinos de E/S) onde somente uma única rede (*net*) pode ser construída. Estes recursos de roteamento possuem capacidade $c_r = 1$, mas quando $r \in \hat{V}_d$, representa um nó destino (*sink*), a restrição (5.14) funciona como um limite sobre os números de redes que podem alcançar o bloco lógico, ou seja, para um bloco lógico contendo uma única LUT com quatro entradas (4-LUT) tem-se $c_r = 4$. Isto decorre do fato de que todos os pinos de entrada de uma LUT são logicamante equivalentes, e portanto, um sinal de uma rede (*net*) pode alcançar um bloco lógico destino por qualquer dos seus lados e de seus pinos, caso haja mais de um pino a cada lado. O conjunto de restrições (5.13) permite para uma dada rede i o compartilhamento do recurso r pelo fluxo de todas as suas mercadorias i, t e aloca o recurso r para a rota detalhada da rede i.

Como a formulação anterior, esta também é de interesse apenas teórico, ainda que uma redução do número de variáveis tenha sido alcançada pela modelagem explícita das conexões possíveis do bloco S que compõem a arquitetura FPGA.

5.4 Aplicando Relaxação Lagrangeana

Resolver diretamente o problema de roteamento global ou o problema de roteamento detalhado é praticamente impossível para FPGAs suficientemente grandes, devido à dimensão do conjunto de variáveis. Assim, uma técnica de decomposição deve

ser empregada. Uma boa escolha é a aplicação de relaxação lagrangeana devido à existência de subconjuntos de restrições fáceis de resolver, se considerados isoladamente. Em relação à formulação do problema de roteamento global irrestrito, há duas relaxações possíveis. A primeira possibilidade consiste em relaxar os conjuntos de restrições (5.6) e (5.7), e a segunda em relaxar o conjunto de restrições (5.5). Estas são discutidas a seguir.

Relaxando as restrições de acoplamento entre redes

Observando o conjunto de restrições (5.6) e (5.7), fica claro que estas são as únicas restrições que relacionam diferentes redes, uma com as outras. Isto motivou então o estudo de uma abordagem via Relaxação Lagrangeana, onde estes conjuntos de restrições são relaxados, resultando num problema lagrangeano que se decompõe em k subproblemas, um para cada rede, devido à relaxação do conjunto de restrições (5.7). Para cada rede i, por sua vez, o respectivo subproblema se decompõe em $|N_i - \{s_i\}|$ subproblemas de caminhos mínimos, um para cada terminal da rede i. Deste modo, obtem-se um alto nível de decomposição do problema, podendo todos os subproblemas gerados serem resolvidos em paralelo. Pode-se assim obter limites inferiores sobre a mínima largura de canal necessária para rotear um dado circuito. Estes limites podem então ser usados para definir o número inicial de domínios de trilhas em cada processador, considerando a aplicação paralela apresentada neste trabalho.

Seja $\lambda_e \geq 0$ e $\gamma_a^{it} \geq 0$ os conjuntos de multiplicadores de Lagrange associados com as restrições (5.7) e (5.6), respectivamente. Então podemos enunciar o seguinte problema lagrangeano

$$\begin{aligned}
\mathcal{L}(W, y, x, \lambda, \gamma) = \min \quad & W + \sum_{e \in E} \lambda_e \Big(\sum_{i=1}^{k} y_{i,a} + \sum_{i=1}^{k} y_{i,a'} - W\Big) + \\
& \sum_{a \in A} \sum_{i=1}^{k} \sum_{t=1}^{|N_i - \{s_i\}|} \gamma_a^{it} (x_{fh}^{it} - y_{i,a}) \\
\text{s.a} \quad & x \text{ satisfazendo (5.5)}, \\
& W, y, x \geq 0, y \in \{0, 1\}.
\end{aligned}$$

que pode ser reescrito como

$$\begin{aligned}\mathcal{L}(W,y,x,\lambda,\gamma) = \min \quad & (M - \sum_{e\in E}\lambda_e)W + \sum_{a\in A}\mu_a \sum_{i=1}^{k} y_{i,a} + \\ & \sum_{a\in A}\sum_{i=1}^{k}\sum_{t=1}^{|N_i-\{s_i\}|}\gamma_a^{it}(x_{fh}^{it} - y_{i,a}) \\ \text{s.a} \quad & x \text{ satisfazendo (5.5)}, \\ & W,y,x \geq 0, y \in \{0,1\}.\end{aligned}$$

onde $\mu_a = \mu_{a'} = \lambda_e$ para $e = [f,h], a = (f,h), a' = (h,f)$.

Investigando o dual da relaxação linear do problema de roteamento global, tem-se, em relação à variável W, a seguinte restrição $\sum_{e\in E}\lambda_e \leq 1$ com $\lambda_e \geq 0$. Caso a solução ótima de W^* seja estritamente positiva, por complementaridade das folgas tem-se $\sum_{e\in E}\lambda_e = 1$, e portanto, a variável W é eliminada da função do problema lagrangeano obtendo-se então

$$\begin{aligned}\mathcal{L}(W,y,x,\lambda,\gamma) = \min \quad & \sum_{a\in A}\mu_a\sum_{i=1}^{k} y_{i,a} + \sum_{a\in A}\sum_{i=1}^{k}\sum_{t=1}^{|N_i-\{s_i\}|}\gamma_a^{it}(x_{fh}^{it} - y_{i,a}) \\ \text{s.a} \quad & x \text{ satisfazendo (5.5)}, \\ & W,y,x \geq 0, y \in \{0,1\}.\end{aligned}$$

Deste modo, podemos perceber que a resolução do lagrangeano equivale a resolver k subproblemas independentes, um para cada rede i, que por sua vez tem o seu respectivo problema de encontrar uma árvore de Steiner retílinea (roteamento da rede), resolvido como $|N_i - \{s_i\}|$ problemas de caminho mínimo. Dado um $\lambda \geq 0$ e $\gamma \geq 0$, o valor do lagrangeano $\mathcal{L}(\lambda,\gamma)$ fornece um limite inferior sobre a mínima largura de canal W. Para maximizarmos o limite inferior obtido pela relaxação lagrangeana, resolveremos o seguinte dual lagrangeano:

$$\mathcal{D}(\lambda,\gamma) = \max_{\lambda\geq 0,\gamma\geq 0}\{\mathcal{L}(\lambda,\gamma)\}$$

Uma técnica usual para resolver o dual estabelecido por relaxacao lagrangeana, $\mathcal{D}(\lambda,\gamma)$, que tem sido largamente difundida na literatura é a otimização via subgradientes. Um outro método que poder ser aplicado é o método do volume [9].

Relaxando as restrições de conservação de fluxo

Uma outra possibilidade de relaxação para o problema consiste em relaxar o conjunto de restrições de conservação de fluxo (5.5).

Seja $\sigma_f^{it} \geq 0$ o conjunto de multiplicadores de Lagrange associados com as restrições (5.5), então podemos enunciar o seguinte problema lagrangeano:

$$\begin{aligned}\mathcal{L}(W,y,x,\sigma) = \min \quad & W + \sum_{i=1}^{k} \sum_{f \in V:\, f \neq s_i, t} \sum_{t=1}^{|N_i - \{s_i\}|} \sigma_f^{it} \left(\sum x_{fh}^{it} + \sum x_{hf}^{it}\right) \\ & + \sum_{i=1}^{k} \sum_{t=1}^{|N_i - \{s_i\}|} \sigma_t^{it} - \sum_{i=1}^{k} \sum_{t=1}^{|N_i - \{s_i\}|} \sigma_{s_i}^{it} \\ \text{s.a} \quad & x \text{ satisfazendo (5.6)e (5.7)}, \\ & W, y, x \geq 0, y \in \{0,1\}.\end{aligned}$$

Assim, obtem-se um problema lagrangeano que pode ser resolvido por inspeção, ou seja, se o coeficiente de x_{fh}^{it} na função lagrangeana for maior que 0, então este assume 0, caso contrário assume 1. Os valores de $y_{i,a}$ são determinados em conseqüência dos valores de x_{fh}^{it}, que por sua vez determinam W.

Neste trabalho, no entanto, não foi desenvolvida uma implementação computacional da abordagem descrita nesta seção, ficando portanto este estudo mais aprofundado como uma proposta de trabalho futuro.

Capítulo 6

Estratégias de Paralelização para Roteamento em FPGAs

Atualmente plataformas SMP (Multiprocessamento Simétricos) e *clusters* de estações de trabalho são muito comuns. Assim sendo, é uma evolução natural tentar usar o potencial de paralelismo desses ambientes para reduzir drasticamente o tempo gasto em alguns passos do ciclo de projeto físico, em particular, na fase de roteamento. Este trabalho apresenta um estudo de idéias de paralelização para a fase de roteamento em FPGAs, tendo sido implementadas a maioria destas idéias.

6.1 Introdução

O interesse por este problema surgiu naturalmente face às crescentes dificuldades em se rotear circuitos cada vez maiores. Hoje há no mercado pastilhas com mais de 10.000.000 portas lógicas utilizando tecnologia de 0.12μ e operando abaixo de $1,5$ Volts, resultado do crescente nível de integração, que naturalmente traz consigo um aumento considerável na complexidade do problema, tanto no tocante à explosão do tempo gasto na compilação da especificação do projeto, tipicamente feito em linguagens de descrição de hardware como VHDL, Verilog, etc; como também na exigência da adequação das ferramentas de CAD para que estas possam dar suporte às inovações tecnológicas.

Entretanto, na literatura da área de roteamento para FPGAs, quase nenhum estudo tem sido feito sobre o desenvolvimento de roteadores paralelos. Isto se deve primeiro à dificuldade em se tratar com o alto grau de interdependência das redes de sinal (*nets*) do circuito. Tendo em vista o objetivo de se construir circuitos

com a menor área possível, busca-se minimizar a largura de canal de roteamento do FPGA e, por conseguinte, uma rede ao ser roteada sofre influência das redes roteadas anteriormente. Segundo, os algoritmos de roteamento existentes consideram todo o grafo de roteamento, e portanto, necessitam, numa versão paralela, gerar cópias deste grafo em todos os processadores, o que implica trocar informações sobre o congestionamento em cada arco, ou seja, um processador necessita reunir informações de todos os outros processadores de modo a concluir se um dado arco está ou não disponível. Porém, o grafo de roteamento tem dimensões elevadas, impondo assim severos custos de comunicação entre os processadores.

As duas dificuldades suprareferidas têm, de certo modo, inibido o desenvolvimento de soluções que explorem o uso de paralelismo para o problema de roteamento em FPGA. Trabalhos pioneiros são apresentados em Chan e Schlag [25] e em Chan et al [26]. Nas próximas seções são descritas algumas das diversas idéias de paralelização que foram surgindo ao longo deste trabalho de pesquisa. Destas, algumas foram implementadas para avaliação de desempenho e outras apenas conceitualmente estabelecidas.

Nesta seção serão discutidas brevemente algumas idéias que podem ser aplicadas como estratégias para o roteamento paralelo. São elas:

- É fácil perceber que a tarefa mais árdua durante o trabalho de roteamento é a fase de busca por caminhos para rotear uma rede. Daí surge uma primeira idéia que consiste em usar paralelismo na busca por estes caminhos. Em Chan e Schlag [25] e Aude et al [8] esta fase, além de ser paralelizada, tem parte dela implementada diretamente em *hardware*. E em Chan et al [26] cada processador recebe um conjunto de redes e efetua o roteamento usando um algoritmo baseado no paradigma de negociação, como o *PathFinder* [70], sendo que a atualização dos custos de congestionamento é feita trocando informações com os outros processadores, sobre o congestionamento local em cada segmento de fio do FPGA, via um mecanismo de troca de mensagens. Esta estratégia possui alto custo de comunicação que cresce com o tamanho do grafo de roteamento.

- Ainda tendo em mente a busca por caminhos no grafo, por que não *clusterizar* o grafo, ou seja, dividir o FPGA em diversas regiões disjuntas de layout, de mo-

do que, a cada processador possa ser atribuída uma parte disjunta do grafo de roteamento? De fato, para o roteamento das redes (*nets*) circunscritas a qualquer uma dessas áreas de layout, seria necessário apenas inspecionar o grafo de roteamento da respectiva região de layout. No entanto, para as redes que possuam seus terminais espalhados por várias regiões de layout, os processadores haverão de se comunicar na tentativa de buscar os caminhos para tais redes. Isto acarreta um alto custo de comunicação, que, na presente análise, torna esta abordagem pouco eficiente para arquiteturas baseadas em memória distibuída e troca de mensagens. Entretanto, para arquiteturas baseadas em memória compartilhada, esta idéia de paralelização pode se adequar muito bem, devido à maior facilidade de implementar nestas plataformas aplicações paralelas com forte grau de acoplamento entre as tarefas. Esta idéia foi originalmente apresentada em Aude [7] para roteamento em duas camadas e estendida para FPGAs em Cabral, Aude e Maculan [23].

- Uma outra abordagem que surgiu da observação, em relação às características da arquitetura alvo do FPGA usando S-Box diagonal, foi a idéia de explorar a disjunção natural do grafo de roteamento face à possibilidade da separação dos segmentos de fios do FPGA em domínios de trilhas. A idéia proposta consiste em atribuir a cada processador uma partição do conjunto de domínios de trilhas e em seguida dividir o conjunto de redes a serem roteadas entre estes processadores. Cada processador roteia sua parcela de redes da lista de redes e ao não concluir o roteamento de toda a sua lista, encaminha a outros processadores as redes que este não conseguiu rotear. Isto é feito até que o roteamento termine ou que se conclua que não é possível rotear todo o conjunto de redes com a quantidade existente de trilhas, i.e., domínios de trilhas. Esta idéia foi originalmente proposta em Cabral, Aude e Maculan [23].

6.2 Explorando Regiões de Layout em Paralelo

Em [21] é proposta uma heurística para rotear grupos de *nets* em paralelo, onde o objetivo é decrescer o tempo de execução da fase de roteamento através do roteamento concorrente de *nets* alocadas a áreas disjuntas de layout. Duas *nets* podem

ser ditas disjuntas, em relação a área de layout, se os retângulos que englobam seus respectivos terminais não se interceptam. Este procedimento é, portanto, baseado na idéia de particionar o layout do FPGA em *clusters*, tal que uma partição do conjunto de *nets* seja associada a cada *cluster*. Cada *cluster* é associado a um único processador, que deverá rotear todas as *nets* pertencentes a este *cluster*. Os principais aspectos desta idéia são detalhadas a seguir:

- Para cada rede é determinado o menor retângulo que engloba todos os seus terminais;
- A área de layout do FPGA é dividida em um número fixo de *clusters*, considerando o número de processadores disponíveis e o tamanho do FPGA. Se o FPGA for muito grande, a sua divisão num número razóavel de *clusters* permitirá melhor balanceamento de carga. Os *clusters* são numerados e o *cluster* k é associado ao processador i. A cada nova iteração, o FPGA é dividido em *clusters* maiores, sendo necessário um número menor de processadores;
- Inicialmente todo *cluster* tem somente uma trilha disponível para rotear suas *nets*, mas durante o processo, se para algum *cluster* não for possível rotear quaisquer das *nets* remanescentes no seu cluster, este envia um pedido de incremento do número de trilhas, como uma mensagem para o processador mestre;
- A cada processador escravo são atribuídos um ou mais *clusters* e aos mesmos, cabe rotear todas as *nets* do respectivos *clusters*;
- Dentre os processadores existe um especial, chamado mestre. Os outros comportam-se como escravos. O processador mestre receberá pedidos enviados por processadores escravos e irá forçar uma atualização do FPGA, em termos do número de trilhas, em todos os *clusters* requerentes, quando necessária;
- A configuração da geometria dos *clusters* (tamanho, quantidade de *nets*) muda a cada iteração;
- A criação da *netlist* pertencente a cada *cluster* é feita considerando para cada *net*, se o retângulo que engloba todos os seus terminais está totalmente contido

em algum *cluster*. Em resumo, se uma dada *net* está dentro de algum *cluster* k, então ela é incluída na *netlist* pertencente ao *cluster* k, caso contrário esta *net* é colocada em uma lista de *nets* não roteadas. Tal lista é considerada apenas na próxima iteração, após o roteamento ter sido concluído em todos os *clusters*, e novos *clusters*, maiores do que os *clusters* da iteração anterior, terem sido definidos.

Esta idéia é ilustrada na figura 6.1, onde um FPGA 4 X 4 é mostrado. Supondo que quatro processadores estejam disponíveis, é possível dividir a área de layout do FPGA em quatro *clusters* (figura 6.1(a)), e a cada um deles associar um processador diferente. Deste modo, quatro *nets*, uma em cada *cluster* podem ser roteadas em paralelo. Após o término do roteamento nos quatro *clusters*, uma nova iteração é iniciada. Os tamanhos de cada *cluster* são atualizados e a lista de *nets* não roteadas é processada para que seja criada uma nova lista de *nets* para cada novo *cluster*. Somente após isto, o roteamento nos *clusters* é iniciado. Tal estratégia pode ser vista pelas figuras 6.1(a) a 6.1(e), como uma seqüência de iterações. A última iteraçao é mostrada na Figura 6.1(e), onde um único *cluster* é usado para representar todo o layout do FPGA. Outra possibilidade é executar o roteamento na ordem reversa, isto é, iniciar com um *cluster* muito grande, de modo a conter as *nets* que se espalham por todo o FPGA, e a cada iteração, reduzir o tamanho dos *clusters*.

O custo associado a esta estratégia, em termos de mensagens, é baixo, se os recursos de roteamento dos *clusters* não se interceptam como ilustrado na Figura 6.1. Isto decorre do fato de os recursos de roteamento não serem compartilhados pelos *clusters*, e assim poucas mensagens precisarem ser enviadas do processador mestre para os processadores escravos. As mensagens são as seguintes: o novo valor de W (número corrente de trilhas do FPGA), o tamanho do cluster e a *netlist* pertencente ao processador escravo. Também poucas mensagens são enviadas dos processadores escravos para o processador mestre. São elas: pedido de incremento unitário do número de trilhas W, nível de utilização de segmentos, e o status do roteamento executado e, no caso de fracasso, também a lista de nets não roteadas pelo escravo.

Para arquiteturas de memória compartilhada, é interessante notar que durante a fase de roteamento em cada *cluster*, cada processador atualiza partes disjuntas

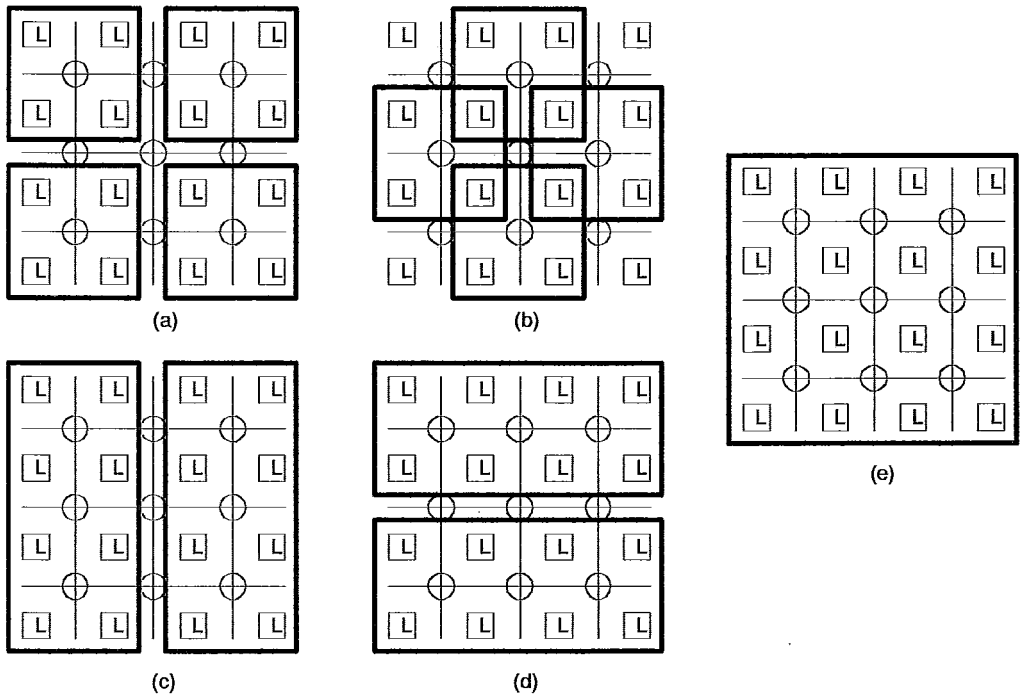

Figura 6.1: Roteamento por regiões de layout

da estrutura de dados que representa a área de layout. E após o término de uma iteração, o processador mestre tem acesso diretamente a todas as soluções. Entretanto, se *clusters* vizinhos têm regiões de roteamento em comum, haverá um aumento de comunicação proporcional ao tamanho da região compartilhada entre estes *clusters*.

Como a fase de roteamento é independente entre os *clusters*, isto gera também uma flexibilidade na escolha do melhor algoritmo de roteamento para cada *cluster* baseado em sua configuração, isto é, topologia das *nets*, recursos de roteamento disponíveis e tamanho do *cluster*.

Finalmente, o desempenho decorrente do uso desta estratégia é extremamente dependente do posicionamento dos circuitos. Quanto mais agrupados forem os circuitos fortemente interconectados, menor o custo de comunicação. No entanto, tal estratégia gera um paralelismo decrescente, com carga de trabalho em todos os processadores no início, e somente em um processador no final. Espera-se assim, que este procedimento possa ser capaz de reduzir o tempo gasto durante a fase de roteamento. E como cada rede é restrita a ser roteada dentro do seu *cluster*, provavelmente haverá um benefício adicional da redução do comprimento médio das

nets, significando um ganho em termos de desempenho para o circuito. Contudo, devido ao roteamento tardio das redes grandes, há grandes possibilidades de que esta restrição force um incremento exagerado do número de domínios de trilhas W.

6.3 Paralelizando por Domínios de Trilhas

Aqui será detalhada a proposta que correponde à maior contribuição deste trabalho. Esta consiste precisamente no particionamento do grafo de roteamento em domínios de trilhas. Isto somente é possível face às restrições arquiteturais impostas sobre o bloco de interconexão S do FPGA, de modo que segmentos de fios num dado bloco de conexão C podem apenas se conectar a segmentos de fios, com mesmo identificador de trilha, nos blocos de conexão C adjacentes. Isto é possível utilizando-se uma S-Box diagonal. A figura 6.2 ilustra a criação desses domínios de trilhas.

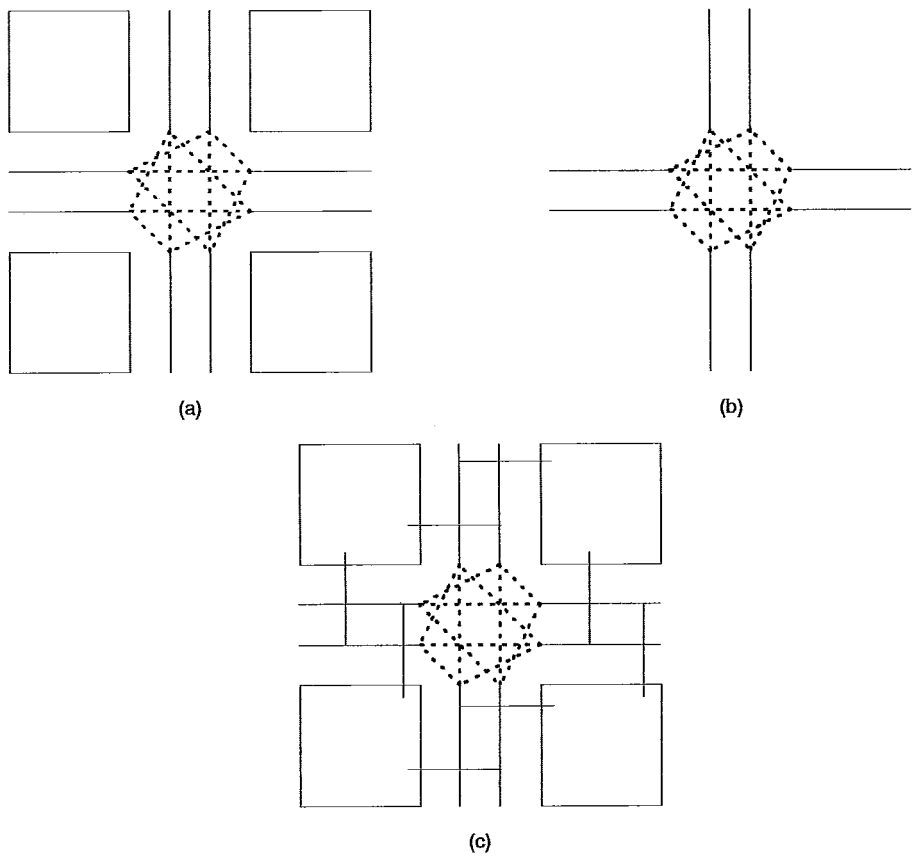

Figura 6.2: (a) Um FPGA 2x2 (b) dois domínios de trilhas (c) grafo de roteamento

A partir dessa disjunção, em termos de segmentos de fio, do grafo de roteamento detalhado do FPGA, é proposta uma estratégia de paralelizaçao que consiste em alo-

car a cada processador um ou mais domínios de trilhas, sendo que cada domínio de trilha será alocado a apenas um processador. Feito isto, pode-se então, dividir o conjunto de redes (*nets*) a serem roteadas, particionado-o no número de processadores disponíveis e/ou número de partições. Cada processador receberá uma destas partições. Este trabalho aborda a disjunção de redes entre processadores, ou seja, cada partição do conjunto de redes só pode ser atribuída a um dado processador. De outro modo, teria que se decidir, em algum momento, em qual dos processadores uma dada rede teria seu roteamento mantido em definitivo.

O paralelismo desejado é então obtido com os processadores roteando seus conjuntos de redes nos seus respectivos domínios de trilhas. O roteamento é concluído, se não houver nenhuma rede não roteada em nenhum dos processadores. Caso contrário, uma lista de todas as redes não roteadas será criada e novamente particionada e redistribuída entre os processadores, evitando-se que uma mesma rede seja novamente associada ao mesmo processador.

Se não for possível completar o roteamento após um certo número destas redistribuições de redes não roteadas, presume-se que será necessário aumentar o número de domínios de trilhas. Tal incremento tem que ser parcimonioso, além de se considerar a qual processador, no caso de incremento unitário, ou a quais processadores, no caso de incremento não unitário, serão atribuídas as novas trilhas. Tal escolha considera o quão saturados estão os respectivos domínios de trilhas, isto é, o percentual do número de segmentos de fios usados em relação ao número total de segmentos de fios existentes nos domínios de trilhas.

A seguir serão discutidos dois modelos topológicos de comunicação, o modelo Mestre-Escravo e o modelo de Vizinhança Restrita. Destes, o modelo mestre-escravo foi o adotado neste trabalho.

Modelo Mestre-Escravo

Nesta estratégia de paralelização, há que se destacar dois aspectos cruciais: 1) a elaboração da lista de redes não roteadas; 2) a decisão de incremento do número de domínios de trilhas num dado processador.

O modelo Mestre-Escravo foi inicialmente adotado por simplificar as ações supra-

mencionadas, que passam a ser coordenadas e geridas por um processador denominado Mestre. A figura 6.3 ilustra este modelo baseado num cenário com p processadores e um FPGA com W trilhas, onde $W = W_0 + W_1 + W_2 + ... + W_{p-1}$.

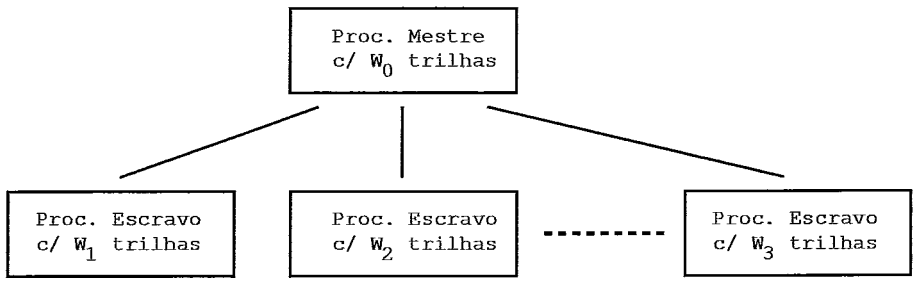

Figura 6.3: Modelo Mestre-Escravo

Neste tipo de modelagem (topologia), toda a comunicação é feita somente entre o processador mestre e os processadores escravos e vice-versa., de modo que não é possível dois processadores escravos se comunicarem. Esta característica se apresenta como vantagem, no tocante ao aspecto da implementação computacional, mas possui a desvantagem de diminuir o nível de cooperação entre os processadores e de tornar serial parte do processamento, podendo vir a se tornar um gargalo do modelo.

A elaboração da lista de redes não roteadas é feita pelo processador mestre através da união das listas de redes não roteadas de cada processador, que são recebidas como mensagens oriundas de processadores escravos, quando do insucesso no roteamento do conjunto de redes atribuído ao processador escravo.

O incremento do número total de domínios de trilhas, também gerido pelo processador mestre, baseia-se nas mensagens do nível de saturação oriundas de cada processador escravo (quantidade de segmentos de fio usados). Assim, é possível fazer um diagnóstico do nível total de utilização de segmentos de fio em todos os domínios de trilhas do FPGA, e então relacionar esta informação com o número e tamanho das redes (*nets*) ainda não roteadas. Com base nisto, é então definida qual política de incremento do número de trilhas/processador será adotada. Cabe assim ao processador mestre autorizar a um processador escravo a criação de mais um domínio de trilha localmente.

O término do algoritmo, segundo este modelo, também é detectado pelo processador mestre quando a lista de redes não roteadas está vazia, i.e., quando após a

última redistribuição da lista de redes não roteadas, todos os processadores enviarem mensagem de sucessso no roteamento de suas respectivas listas de redes não roteadas.

Modelo de Vizinhança Restrita

Como mencionado anteriormente, o modelo Mestre-Escravo centraliza o processo de tomada de decisão, eliminando qualquer cooperação neste processo. Aqui se propõe um outro modelo que visa a obter cooperação entre os processadores no processo de tomada de decisão, sem, no entanto, aumentar demasiadamente o custo de comunicação. Assim, um modelo de Vizinhança Restrita também é proposto, onde cada processador coopera apenas com uma pequena parcela dos processadores, denominamos seus vizinhos.

A figura 6.4 ilustra este modelo considerando dois cenários: o primeiro para 3 processadores tendo vizinhaça 1 (número de vizinhos para quem se manda mensagens) e outro para 5 processadores e vizinhança 2.

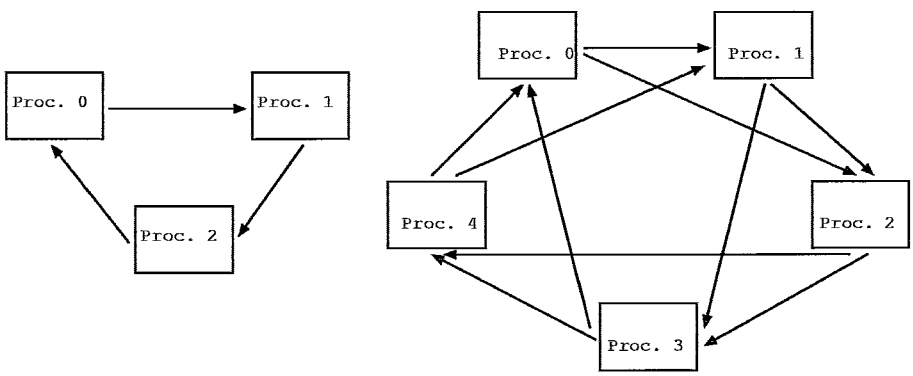

Figura 6.4: Modelo de vizinhaça restrita

Nesta modelagem, quando um processador não concluir o roteamento de toda a sua lista de redes, ele pode particionar sua lista de redes não roteadas no número de vizinhos para quem pode enviar mensagens e então enviar a cada um destes uma partição da lista de redes não roteadas. Fazendo isto, pode-se garantir que a qualquer momento cada rede ainda não roteada constará da lista de redes a rotear de um único processador.

Poder-se-ia também enviar toda a lista de redes não roteadas num processador para todos os seus vizinhos, o que exigiria, em algum momento, que os proces-

sadores cooperassem num esforço de restabelecer a consistência, pois poderia haver alguma(s) rede(s) que já tivesse(m) sido roteada(s) em mais de um processador.

A despeito da vantagem obtida pelo maior nível de cooperação entre os processadores, há uma perda em qualidade em relação ao número de trilhas, devido ao fato de que nenhum processador tem uma visão global do processo, como o mestre tem no modelo anterior. Além disso, torna-se complexo aqui o processo de incremento do número total de trilhas e deteçcão de término do algoritmo. Para solucionar estas dificuldades, pode-se propor que um dos processadores seja eleito como líder. Este se comunicará diretamente com todos os outros, sendo assim responsável pelo diagnóstico da evolução do roteamento e pela sinalização de término de execução em todo o sistema. Neste modelo, a decisão de criação de novos domínios de trilhas em cada processador seria um decisão local, baseda no seu nível de ocupação e no nível de ocupação dos processadores vizinhos, de quem este processador recebe mensagens.

6.3.1 Acoplamento dos Pinos de E/S dos Blocos Lógicos

Como pode ser observado na figura 6.2, o roteamento de redes em diferentes domínios de trilhas não exclui a possibilidade de conflito, pois a despeito da garantia de exclusão sobre os segmentos de fios, pode ocorrer compartilhamento de pinos de E/S de blocos lógicos. Para superar esta dificuldade foram propostas algumas alternativas:

- Permitir que diferentes redes compartilhem pinos de E/S de blocos lógicos durante determinado período de execução de algoritmo de roteamento, utilizando-se, em seguida, um procedimento de remoção e reroteamento (*rip up e reroute*) dessas redes, com exceção daquelas roteadas pelo processador selecionado para manter o roteamento de suas redes. Este procedimento é chamado a cada vez que um compartilhamento deste tipo é detectado. Como a cada vez pelo menos uma rede dentre as redes conflitantes, isto é, que compartilham algum pino, tem seu roteamento mantido, tornando indisponível os pinos ocupados por ela, garante-se a convergência do algoritmo paralelo que use esta alternativa.

- Gerar um grafo de interseção $G_I = (V, E)$, não orientado, onde cada rede corresponda a um nó deste grafo, havendo um arco entre quaisquer duas redes, se

suas *netlist* possuem interseção, ou seja, se existe algum bloco lógico conectado a ambas as redes, significando que ambas concorrem por pinos de entrada neste bloco lógico. A idéia é particionar G_I num determinado número de partições $P_i, i = 1, 2, .., m$, tal que $P_i \subset V$, e para toda rede $n \in P_i, n \notin V - \{P_i\}$. Em seguida, todas as redes presentes no corte $\delta(m) = \{n_1 \in \bigcup_{i=1}^{m} P_i | \exists e = [n_1, n_2] \in E, n_1 \in P_i, n_2 \notin P_i\}$ formam um novo grafo de interseção $G'_I = (V', E')$, onde $V' = \delta(m)$ e o conjunto de arcos $E' = \{e = [n_1, n_2] \in E | n_1 \in \delta(m), n_2 \in \delta(m)\}$. Então o novo grafo de interseção G'_I é particionado em $m/2, m \geq 4$, partições. Este procedimento é repetido até que o grafo não possa mais ser reparticionado. Fazendo isto, obtem-se em cada nível de particionamento conjuntos de redes não conflitantes, que podem ser roteados em paralelo , isto é, dada uma partição $P_i \subset V$, tem-se o conjunto $\hat{P}_i$ das redes internas à partição P_i, onde $\hat{P}_i = \{n_1 \in P_i | \forall e = [n_1, n_2] \in E, n_2 \in P_i\}$.

De fato, uma vez que todas as redes pertencentes a um corte $\delta(m)$ tenham sido roteadas e sejam conhecidas em todos os processadores a nova ocupação de pinos, pode-se rotear em paralelo todos os conjuntos de redes $\hat{P}_i, i = 1, 2, ..., m$. Obviamente todas as redes de um mesmo conjunto $\hat{P}_i$ devem ser roteadas pelo mesmo processador.

Esta proposta obtém um alto nível de paralelismo quando estão sendo roteadas as redes internas do nível mais baixo (nível 1) de particionamento. Contudo, há um gargalo decorrente do roteamento seqüencial das redes presentes no último corte gerado, quando nenhum novo reparticionamento é possível.

- Dividir o conjunto de pinos de E/S de blocos lógicos entre os processadores. Inicialmente o conjunto de redes é particionado no número de processadores disponíveis, segundo algum critério que leve a um balanceamento de carga satisfatório. Em seguida, cada partição de redes é atribuída a um único processador e cada processador recebe uma única partição de redes. E para cada processador percorre-se sequencialmente seu conjunto de redes, de modo que para cada rede é reservado um ou mais pinos de entrada em cada bloco lógico que conste da sua lista de nós destinos. Após esta divisão de pinos, que deve ser executada por um único processador, é construída uma lista de pinos por

processador que é enviada a cada um dos outros processadores. A partir deste momento, cada processador pode rotear seu conjunto de redes nos seus domínios de trilhas privativos usando apenas os pinos reservados às suas redes. Nenhuma possibilidade de conflito existe, gerando assim um alto grau de paralelismo, pois não há necessidade de comunicação entre os processadores, exceto a sinalização de um processador para os demais quando do término da sua execução. Uma desvantagem dessa estratégia reside no fato de que esta associação antecipada de pinos à rede pode, em princípio, piorar a qualidade do roteamento.

6.4 O Roteador TDR (*Track Domain Router*)

Este trabalho enfoca o desenvolvimento de um roteador paralelo que explore a possibilidade de desacoplamento, em termos de segmentos de fios, do grafo de roteamento detalhado em FPGAs, decorrente da criação de domínios de trilhas. Assim ao nosso roteador paralelo foi dado o nome de TDR (*Track Domain Router*).

Inicialmente, cada processador cria seu próprio grafo de roteamento detalhado, contendo apenas os domínios de trilhas pertencentes a ele. Para determinar o número inicial de domínios de trilhas por processador é utilizada a estimativa da mínima largura do canal de roteamento (mínimo número de domínios de trilhas) fornecidos pelo conhecimento prévio de resultados obtidos por outros roteadores para o conjunto de circuitos testes avaliados neste trabalho. Tal estimativa seria muito mais precisa se limites inferiores sobre o problema de roteamento global fossem obtidos via relaxação lagrangeana aplicada a um modelo de programação inteira para este problema, como formulado e proposto neste trabalho.

Entretanto, ao criar-se um grafo de roteamento em cada processador, tem-se replicação de pinos de E/S, tanto dos blocos lógicos como dos *pads* de E/S. Quanto aos *pads* de E/S, não há risco de conflito de redes, já para os pinos de E/S de blocos lógicos, há uma clique de conflito entre todas as redes que têm um bloco lógico comum as suas listas de nós destinos (*sinks*). Para tratar com o acoplamento entre redes decorrentes deste tipo de conflito foram propostas três alternativas, já citadas na seção anterior. Cada uma delas resultou numa implementação paralela que será descrita em detalhes nas próximas seções. Estas três alternativas guardam entre

si uma ordem cronológica de desenvolvimento dentro do contexto deste trabalho, fato que motivou a apresentação das duas primeiras alternativas, apesar de apresentarem baixos ganhos em tempo de execução de roteamento, quando comparadas ao roteamento seqüencial.

Para efetuar o roteamento das redes, em cada processador, foi adaptado o algoritmo de roteamento VPR [14]. Tal escolha foi efetuada por dois motivos: primeiro, este algoritmo detém os melhores resultados em termos de mínima largura de canal, presentes na literatura; segundo, o seu código com cerca de 10.000 linhas, escritas em C, está disponível na internet no endereço http://www.eecg.toronto.edu/~vaughn/vpr para *download*. Um terceiro motivo que reforça a certeza de ter sido esta uma escolha acertada é a publicação recente de um trabalho [26] que apresenta a paralelização de um algoritmo de roteamento para FPGAs, baseado no paradigma de negociação, o mesmo no qual se baseia o algoritmo VPR.

6.4.1 Evitando compartilhamento dos Pinos de E/S dos Blocos Lógicos via Ciclos de Negociação

Esta alternativa foi implementada segundo o modelo mestre-escravo para a topologia de comunicação e divisão de tarefas. O processador mestre não efetua roteamento de redes, apenas coordena a divisão de carga de trabalho entre os processadores escravos.

Inicialmente, o processador mestre define o número inicial de domínios de trilhas de cada processador e o envia aos respectivos processadores escravos. Cada processador escravo então cria seu grafo de roteamento detalhado com o número de domínios de trilhas autorizado pelo processador mestre. Em seguida, o processador mestre seleciona um conjunto de redes e o envia para para um processador escravo selecionado. O processador mestre repete este ciclo até que a lista de redes não roteadas esteja vazia ou não haja mais processador escravo ocioso para receber trabalho. O processador escravo recebe um conjunto de redes que devem ser roteadas nos seus domínios de trilhas. Caso consiga rotear todo o conjunto de redes, este informa ao processador mestre um status de sucesso e também informa qual o percentual utilizado da quantidade total dos seus segmentos de fio. Caso contrário, relata um status de insucesso e devolve ao processador mestre as redes não roteadas.

O processador mestre ao receber uma mensagem de sucesso atualiza seu banco de informações sobre o nível de ocupação de segmentos no processador escravo, depois seleciona um novo conjunto de redes adequado à disponibilidade total de segmentos de fios no processador escravo e então o envia. Caso não haja mais redes não roteadas, o processador escravo é mantido como ocioso. Se o processador mestre receber uma mensagem de insucesso, então este primeiro atualiza a lista de redes não roteadas incorporando as redes devolvidas pelo processador escravo e em seguida procede da mesma forma que para o recebimento de uma mensagem de sucesso.

Se por várias vezes, consecutivamente, um processador escravo passa a não conseguir rotear todo o conjunto de redes recebido, então o processador mestre autoriza ao escravo construir mais um domínio de trilhas.

Quando todas as redes já tiverem sido roteadas e todos os processadores escravos estiverem ociosos, o processador mestre comanda o início de um ciclo de negociação. Cada iteração do algoritmo é formada por passo de roteamento das redes, seguido por um passo de verificação de existência de compartilhamento de pinos de E/S de blocos lógicos. Num ciclo de negociação, cada processador escravo forma uma lista dos pinos de E/S de blocos lógicos correntemente ocupados pelas redes roteadas em seus domínios de trilhas e envia esta lista para o processador mestre, que reúne todas estas listas numa única lista global de ocupação de pinos. Para cada pino é indicado o número de redes que o compartilham. Então, o processador mestre envia esta lista global de ocupação de pinos para cada processador escravo. Assim é possível a cada processador escravo desfazer o roteamento de suas redes que estejam compartilhando algum pino de E/S de bloco lógico. Este, por sua vez, forma uma lista de redes, cujo roteamento foi desfeito e a envia ao processador mestre. O processador mestre recebe as listas de redes desfeitas, monta uma nova lista de redes não roteadas, informa aos processadores escravos quais pinos de E/S de blocos lógicos ainda estão livres e reinicia o roteamento em paralelo como descrito antes. A execução termina quando não houver mais nenhuma rede compartilhando algum pino de E/S de bloco lógico.

Se num ciclo de negociação todas as redes conflitantes forem desfeitas (removidas), haverá um número grande de ciclos de negociação, podendo inclusive não haver convergência do algoritmo. Assim sendo, um processador escravo que possua redes

conflitantes, ou seja, que compartilham pinos, é escolhido para manter o roteamento de suas redes. Além disso, redes roteadas em iterações anteriores não podem ser desfeitas. Fazendo isto, consegue-se garantia de convergência para o algoritmo.

Considerando-se o uso da topologia mestre-escravo e do paradigma de troca de mensagens num ambiente de memória distribuída, a figura 6.5 apresenta o pseudocódigo deste algoritmo para o processador mestre e a figura 6.6 para o processador escravo.

Alguns aspectos são cruciais nesta idéia, implicando diretamente no tempo computacional e na mínima largura de canal requeridos. São eles: seleção do conjunto de redes, seleção do processador escravo, política de incremento de domínios de trilhas e definição da configuração inicial do número de domínios de trilhas por processador.

A seleção do conjunto de redes a ser enviado a um processador escravo deve ter sua cardinalidade e características das redes contidas no conjunto ajustadas à configuração corrente dos domínios de trilhas do processador escravo, ou seja, devem ser levados em consideração o número de domínios de trilhas existentes no processador escravo e o respectivo nível de ocupação dos segmentos de fios. Neste trabalho, foi adotada uma política de impor limites sobre a máxima cardinalidade do conjunto de redes e sobre o máximo número de segmentos estimados como necessários para rotear as redes do conjunto. Quanto mais justos estes limites, mais comunicação se faz necessária, pois o processador mestre fará mais iterações do laço de distribuições de redes, onde são enviadas as redes a serem roteadas nos processadores escravos.

A seleção do processador escravo é feita com base no nível de utilização corrente. Sempre é escolhido aquele com maior percentual de segmentos de fio disponíveis, o que intuitivamente reflete a maior possibilidade de todo o conjunto de redes ser roteado nos domínios de trilhas deste escravo.

Se for adotada uma política de incremento de trilha que privilegie o tempo computacional gasto, então esta deve ser tal que, tão logo um processador escravo não consiga rotear o conjunto de redes recebido, este receba autorização para criar um novo domínio de trilha. No entanto, tal política aumenta demasiadamente a área de roteamento do FPGA, levando à criação de domínios de trilhas com baixíssima ocupação de segmentos de fios. Por outro lado, se tal política for por demais parcimoniosa, ter-se-á um FPGA ocupando uma área menor ao custo de um aumento

substancial no tempo computacional requerido. A política adotada neste trabalho foi a de incrementar o número de domínios de trilhas num processador escravo tão logo este tenha cinco insucessos consecutivos no roteamento das redes a ele enviadas, e o nível de utilização corrente em todos os processadores seja superior a 25%.

Quanto à configuração dos domínios de trilhas por processador, foi adotada uma mesma configuração para todos os processadores escravos, ou seja, cada um deles começa com a mesma quantidade de domínios de trilhas. Algumas tentativas foram feitas usando configurações não uniformes, que, por vezes, resultaram em melhores resultados, mas, em outros, em piores. De fato, este é um ponto que carece de uma avaliação mais profunda.

6.4.2 Provendo exclusão dos Pinos de E/S dos Blocos Lógicos via Particionamento Hierárquico

Esta alternativa também foi implementada segundo o modelo mestre-escravo para a topologia de comunição e divisão de tarefas. Novamente o processador mestre não efetua o roteamento de redes. Contudo além de coordenar a divisão de carga de trabalho, também faz o particionamento hierárquico do conjunto de redes. A idéia é gerar um grafo de interseção $G_I = (V, E)$, onde cada rede corresponderá a um nó deste grafo e existirá um arco entre quaisquer duas redes se existir algum bloco lógico conectado a ambas as redes, significando que ambas as redes concorrem por pinos de entrada do bloco lógico. Para ilustrar esta idéia considere um grafo de interseção G_I particionado em 2 partições P_1 e P_2 tal que $P_1 \cup P_2 = V$ e $P_1 \cap P_2 = \emptyset$. A partir de P_1 e P_2 três outros conjuntos são derivados, são eles: o conjunto de redes internas da partição P_1, denotado por $\hat{P_1} = \{n_1 \in P_1 | \forall e = [n_1, n_2] \in E, n_2 \in P_1\}$; o conjunto de redes internas da partição P_2, denotado por $\hat{P_2} = \{n_1 \in P_2 | \forall e = [n_1, n_2] \in E, n_2 \in P_2\}$; e o conjunto de redes presentes no corte, denotado por $\delta(2) = \{n_1 \in (P_1 \cup P_2)\ \exists e = [n_1, n_2], n_1 \in P_1, n_2 \in P_2\}$. Deste modo, roteando-se primeiro as redes do corte $\delta(2)$, os conjuntos de redes $\hat{P_1}$ e $\hat{P_2}$ podem ser roteados em paralelo, em seguida.

Para se alcançar um maior grau de paralelismo é necessário que seja gerado um número maior de partições internas, então o grafo de interseção G_I é inicialmente particionado em um número razoável de partições e, depois, a partir das redes

presentes no corte, um novo grafo de intersecção G'_I é criado envolvendo somente estas redes. Então, este novo grafo de interseção G'_I é particionado em $m/2$ partições, onde $m \geq 4$. Neste trabalho foi adotado m igual a 32 para o primeiro nível de particionamento e o último nível de particionamento é alcançado quando tem-se m=2.

A figura 6.7 ilustra esta idéia, mostrando para um grafo de intersecção entre 8 redes, dois níveis de particionamento associados e um esquema de paralelização do roteamento dessas redes considerando um cenário com 2 processadores.

Na primeira iteração as redes 1 e 2 são roteadas seqüencialmente pelo processador 1. O processador 2 permanece inativo aguardando o término do roteamento no processador 1, de modo a receber informação sobre quais pinos de E/S de blocos lógicos foram ocupados pelas redes 1 e 2. Na iteração 2, as redes internas às partições do nível 2 podem ser roteadas em paralelo pelos dois procesadores. Ao término desta iteração os processadores recebem um do outro a lista de ocupação dos pinos de E/S. A iteração seguinte pode, então, ser iniciada, com uma ou mais partições internas de redes sendo alocadas a cada processador.

Após as redes pertencentes ao corte $\delta(m)$ do último nível terem sido roteadas seqüencialmente e as respectivas ocupações de pinos de blocos lógicos serem conhecidas em todos os processadores escravos, o processador mestre deve selecionar uma ou mais partições internas de redes para enviar ao processador escravo selecionado.

Ressalte-se que todas as redes de uma dada partição interna têm que ser roteadas no mesmo processador, caso contrário não há garantia da exclusão de pinos. Com isto, se um processador escravo tem um insucesso no roteamento de qualquer rede de partição interna, então ele devolve ao mestre toda a partição recebida, de modo que esta possa ser avaliada em outro processador escravo.

Como cada processador escravo recebe partições internas inteiras, e dado que estas podem representar bastante trabalho, foi adotada uma política de incremento de trilhas mais branda, isto é, a cada dois insucessos consecutivos o processador escravo é autorizado a criar um novo domínio de trilhas. Quanto à configuração inicial nos processadores escravos, novamente é adotado o mesmo número de domínios de trilhas em cada processador.

Como observado na figura 6.7, há um paralelismo crescente à medida que as

iterações avançam. De fato, na última iteração, onde as redes do primeiro nível de particionamento são roteadas, alcança-se um alto grau de paralelismo. Em cada iteração, se o número de partições internas é maior do que o número de processadores, é possível promover uma melhor distribuição de carga, atribuindo as partições de redes internas $\hat{P}_i$ ainda não roteadas aos processadores escravos, tão logo estes se tornem ociosos. Contudo, o roteamento das redes presentes no último corte tem que ser feito seqüencialmente em um único processador enquanto os outros aguardam. Se esta fase for rápida, *speedups* razoáveis são alcançados. Caso contrário, o ganho com a paralelização pode ser muito pequeno. Isto depende crucialmente da quantidade e das características das redes presentes no último corte, pois se todas as redes com alto *fanout* (com muitos terminais espalhados por todo o FPGA) pertencerem a este último corte, então a maior parte do algoritmo seqüencial estará sendo repetida aqui. Infelizmente, neste estudo foi percebido que as redes de alto *fanout* se concentram no último corte, minimizando assim o ganho de tempo esperado por esta estratégia. Neste trabalho, foi utilizado o algoritmo de particionamento METIS [55], para particionar o grafo de interseção $G_I = (V, E)$ em cada nível de particionamento.

O pseudocódigo relativo às ações dos processadores mestre e escravo, para esta estratégia, é ilustrado nas figuras 6.8 e 6.9, respectivamente.

6.4.3 Particionando o Conjunto de Pinos de E/S dos Blocos Lógicos entre Processadores

Nesta terceira estratégia é alcançado um desacoplamento total entre os domínios de trilhas, permitindo finalmente um alto grau de paralelismo. A idéia básica é particionar o conjunto de pinos de E/S de blocos lógicos entre os processadores com base nas redes atribuídas a cada um destes. Inicialmente é definido um particionamento do conjunto de redes entre os processadores e, em seguida, cada partição de redes é processada rede a rede. Para cada rede da partição é reservado um ou mais pinos de entrada em cada bloco lógico que deva receber um sinal desta rede. Então os pinos reservados são atribuídos, exclusivamente, ao processador associado a esta partição de redes. Após todas as partições terem sido processadas uma lista global de reservas de pinos E/S de blocos lógicos por processador é montada, e enviada para

cada processador escravo. Cada processador escravo recebe também sua respectiva partição de redes.

Assim, para fazer com que cada processador escravo roteie suas redes nos seus domínios de trilhas sem que haja risco de compartilhamento de pinos, basta que durante a fase de expansão do algoritmo *maze router*, este seja proibido de expandir para pinos de E/S de blocos lógicos não reservados para o processador escravo que está roteando.

Alguns aspectos merecem especial atenção, pois têm impacto direto no número mínimo de domínios de trilhas que serão requeridos para rotear todas as redes do circuito. São eles: o particionamento do conjunto de redes e o procedimento de reserva de pinos.

O particionamento do conjunto de redes define o balanceamento de carga entre os processadores. Assim, se houver um processador escravo com uma carga de trabalho superior aos demais, então o tempo de execução deste processador definirá o tempo total de execução. Contudo, mensurar a carga de trabalho de uma partição de redes não é muito fácil, pois ela depende de quantas redes estão contidas na partição e de quão espalhada pela matriz do FPGA é cada rede da partição. O algoritmo de roteamento VPR usado em cada processador escravo no estilo *maze router* (roteador de labirinto) executa, para cada rede, uma onda de expansão a partir do nó origem da rede. Se a rede possui poucos terminais e estes estão próximos do nó origem, a propagação desta onda de expansão será rápida. Entretanto, se a rede possui muitos terminais espalhados por todo FPGA, o tempo de propagação será muito maior. De fato, o tempo cresce quadraticamente com o aumento da área do menor retângulo que engloba todos os terminais da rede. Neste trabalho foram avaliados dois esquemas de particionamento do conjunto de redes. O primeiro corresponde a construção de uma lista ordenada dos nós do grafo de interseção, com base numa estimativa w_i para o comprimento da rede i, em termos de segmentos de fio. Esta lista é então particionada de forma alternada entre os processadores, de modo que se k processadores são utilizados, então cada processador i receberá as redes cujas posições na lista ordenada sejam $i, i+k, i+2k, i+3k, ...$ e assim por diante. O segundo corresponde a utilização do algoritmo de particionamento METIS [55], para particionar o grafo de interseção $G_I = (V, E)$ no número de processadores

disponíveis, de forma a minimizar o corte e gerar partições balanceadas. Para tanto, foi gerada uma estimativa do comprimento w_i em termos de segmento de fios para cada rede, e esta estimativa é usada como peso para o vértice $n_i \in V$.

O procedimento de reserva de pinos tem influência no grau de flexibilidade do roteamento das redes. Se, para um dado bloco lógico, todas as redes que têm sinal chegando neste bloco lógico estiverem em partições disjuntas, então a ordem em que as partições são lidas definirá exatamente a atribuição de um pino de E/S fixo para cada rede. Neste aspecto, o particionamento tem impacto significativo. Por outro lado, se para um dado bloco lógico apenas uma rede alcança o bloco lógico, então todos os pinos de entrada deste bloco lógico devem ser reservados para esta rede, permitindo assim um alto grau de flexibilidade no roteamento desta rede, quanto à escolha por qual pino e qual lado o sinal desta rede alcança este bloco lógico.

Da mesma forma que nas duas alternativas anteriores, é definido inicialmente o mesmo número de domínios de trilhas para cada processador. Já a política de incremento de domínio de trilhas passa a ser descentralizada, de modo que cada processador escravo cria mais um novo domínio quando ainda houver redes não roteadas na sua partição de redes e seus domínios de trilhas estiverem saturados, ou seja, com alto nível de ocupação.

A figura 6.10 ilustra as ações dos processadores mestre e escravo nesta estratégia.

```
TDR-CN(netlist, arquitetura do FPGA);
1  Para cada processador escravo p faça
2      Determine W_p ;          {Número inicial de dominios}
3      envie W_p para o processador p;
4  fim-para
5  enquanto existir rede não roteada ou compartilhamento de pinos faça
6      enquanto existir rede não roteada ou processador trabalhando faça
7          enquanto existir rede não roteada e processador ocioso faça
8              selecione um conjunto de redes, Ω, ainda não roteadas;
9              selecione um processador escravo ocioso p;
10             se os dominios de trilha de p estão saturados então
11                 envie autorização para criação de um novo dominio de trilha em p
12             fim-se;
13             envie o conjunto Ω para o processador p;
14         fim-enquanto;
15         se existe processador escravo p trabalhando então
16             receba msg de status de roteamento em p;
17             se msg indicar insucesso então
18                 receba a lista de redes não roteadas por p;
19                 atualize lista de redes não roteadas do mestre;
20             fim-se;
21             receba estatisticas do roteamento em p;
22         fim-se;
23     fim-enquanto;
24     Para cada processador p faça
25         envie solicitação de lista de pinos para p;
26         receba a lista de ocupação de pinos em p;
27         atualize lista global de ocupação de pinos;
28     fim-enquanto;
26     se houver compartilhamento de pinos então
27         Para cada processador p faça
28             envie a lista de ocupação global de pinos para p;
29             receba a lista de redes desfeitas em p;
30             atualize lista de redes não roteadas do mestre;
31         fim-enquanto;
32     fim-se;
33 fim-enquanto;
34 Para cada processador escravo p faça
35     envie msg de término de execução;
36 fim-para
```

Figura 6.5: Pseudocódigo do mestre para o alg. TDR com ciclos de negociação

```
TDR-CN(netlist, arquitetura do FPGA);
1  termino ← falso;
2  receba número inicial de domínios de trilhas W_p
3  crie grafo de roteamento com W_p dominios de trilhas
4  enquanto termino == falso faça
5      receba msg do processador mestre
6      se msg == conjunto de redes Ω então
7          RotearRedes(Ω, sucesso)
8          se sucesso então
9              envia msg sucesso para o processador mestre;
10         senão
11             envia msg insucesso para o processador mestre;
12             envia subconjunto de redes não roteadas de Ω para o mestre;
13         fim-se;
14         envia estatísticas do roteamento no escravo
12     fim-se;
13     se msg == término-roteamento Ω então
14         termino ← verdade;
15     fim-se;
16     se msg == autorização de criação de domínio então
17         crie novo domínio de trilha;
18     fim-se;
19     se msg == solicitação de lista de pinos então
20         prepare lista de ocupação de pinos;
21         envie lista de ocupação de pinos para o mestre ;
22     fim-se;
19     se msg == lista de ocupação global de pinos então
20         desfaça roteamento das redes compartilhando pinos;
21         envie lista de redes desfeitas para o mestre ;
22     fim-se;
23 fim-enquanto;
```

Figura 6.6: Pseudocódigo do escravo para o alg. TDR com ciclos de negociação

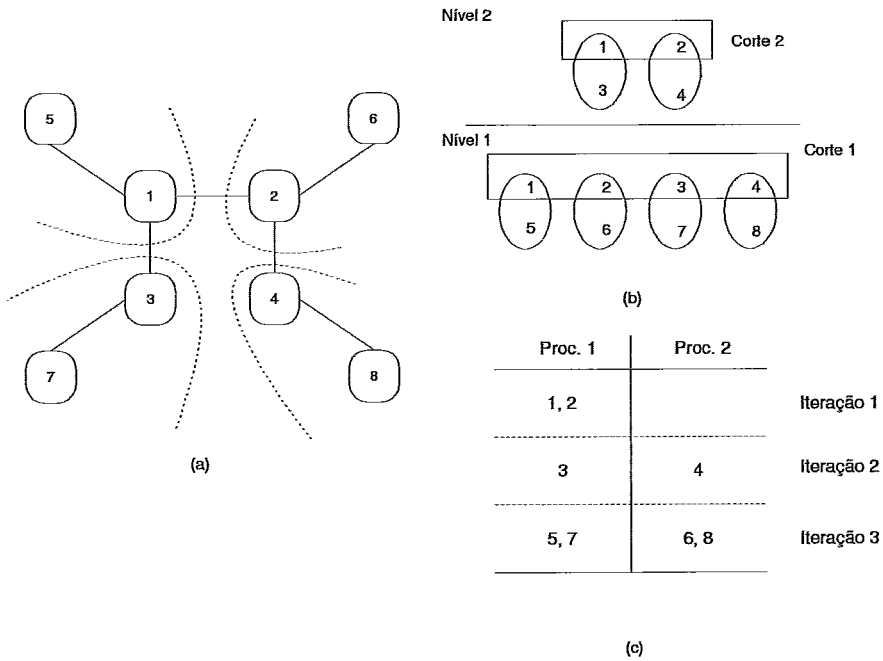

Figura 6.7: (a) Um grafo exemplo (b) níveis de particionamento (c) esquema de execução

```
TDR-PH(netlist, arquitetura do FPGA);
1  Para cada processador escravo p faça
2      Determine W_p ;          {Número inicial de dominios}
3      envie W_p para o processador p;
4  fim-para
5  contrua grafo de interseção G_I = (V, E);
6  efetue particionamento hierarquico de G_I;
7  enquanto existir rede não roteada faça
8      enquanto existir neste nivel partição de redes não roteada ou proc. trabalhando faça
9          enquanto existir neste nivel partição de redes não roteada e proc. ocioso faça
10             selecione um processador escravo ocioso p;
11             selecione uma partição de redes internas P̂ não roteada;
12             se os dominios de trilha de p estão saturados então
13                 envie autorização para criação de um novo dominio de trilha em p
14             fim-se;
15             envie a partição P̂ para o processador p;
16         fim-enquanto;
17         se existe processador escravo p trabalhando então
18             receba msg de status de roteamento em p;
19             se msg indicar insucesso então
20                 receba a partição de redes não roteadas por p;
21                 atualize lista de partições internas não roteadas do mestre;
22             senão;
23                 receba estatisticas do roteamento em p;
24             fim-se;
25         fim-se;
26     fim-enquanto;
27     Para cada processador escravo p faça
28         envie solicitação de lista de ocupação de pinos para p;
29         receba lista de pinos ocupados pelas redes de p;
30         atualize lista global de ocupação de pinos;
31     fim-para
32     Para cada processador escravo p faça
33         envie lista global de ocupação de pinos para p;
34     fim-para
35     avance para o próximo nivel;
36 fim-enquanto;
37 Para cada processador escravo p faça
38     Envie msg de término de execução;
39 fim-para
```

Figura 6.8: Pseudocódigo do mestre para o alg. TDR com particionamento hierárquico

TDR-PH(*netlist, arquitetura do FPGA*);

```
1  termino ← falso;
2  receba número inicial de domínios de trilhas W_p
3  crie grafo de roteamento com W_p dominios de trilhas
4  enquanto termino == falso faça
5      receba msg do processador mestre
6      se msg == partição de redes internas P̂ então
7          RotearRedes(P̂, sucesso)
8          se sucesso então
9              envia msg sucesso para o processador mestre;
10             envia estatísticas do roteamento no escravo
11         senão
12             envia msg insucesso para o processador mestre;
13             desfaz roteamento parcial da partição de redes internas P̂;
14             envia partição de redes internas P̂ para o mestre;
15         fim-se;
16     fim-se;
17     se msg == término-roteamento Ω então
18         termino ← verdade;
19     fim-se;
20     se msg == autorização de criação de domínio então
21         crie novo domínio de trilha;
22     fim-se;
23     se msg == lista global de ocupação de pinos então
24         atualize lista de pinos livres;
25     fim-se;
26     se msg == solicitação de lista de ocupação de pinos então
27         preparar lista de ocupação de pinos;
28         envia lista de ocupação de pinos para o mestre;
29     fim-se;
30 fim-enquanto;
```

Figura 6.9: Pseudocódigo do escravo para o alg. TDR com particionamento hierárquico

TDR-PP(*netlist, arquitetura do FPGA*);

```
1  se processador == MESTRE então
2      Para cada processador escravo p faça
3          Determine W_p ;              {Número inicial de dominios}
4          Envie W_p para o processador p;
5      fim-para
6      contrua grafo de interseção G_I = (V, E);
7      crie grafo de roteamento com W_p dominios de trilhas
8      particione G_I no número de processadores disponiveis;
9      contrua a lista global de reservas de pinos por processador
       com base na partição de redes associada a cada um deles;
10     Para cada processador escravo p faça
11         envie lista global de reservas de pinos para o processador p;
12         envie partição de redes p para o processador p;
13     fim-para
14 senão;
15     receba número inicial de domínios de trilhas W_p
16     crie grafo de roteamento com W_p dominios de trilhas
17     receba lista global de reservas de pinos ;
18     receba partição de redes;
19 fim-se;
20 enquanto existir rede não roteada na partição deste processador faça
21     selecione uma rede n ainda não roteada na partição;
22     se os dominios de trilha do processador estiverem saturados então
23         crie de um novo dominio de trilhas
24     fim-se;
25     RotearRedes(n, sucesso)
26     se sucesso então
27         marque a rede n como roteada
28     fim-se;
29 fim-enquanto;
```

Figura 6.10: Pseudocódigo para o alg. TDR com particionamento de pinos

Capítulo 7

Resultados Computacionais

Considerando as estratégias de paralelização propostas, são apresentados neste capítulo resultados comparativos de desempenho, em termos de *speedup* e também em termos do número total de trilhas, para um conjunto de circuitos de teste. Para a versão do algoritmo TDR, que usa a estratégia de divisão de pinos de E/S de blocos lógicos, foram testados os vinte maiores circuitos do benchmark MCNC [114]. Para as outras duas versões anteriores, o desempenho para apenas alguns circuitos testes foi avaliado, de modo a ilustrar alguns aspectos relevantes.

7.1 Descrição do Ambiente Computacional

Todos os experimentos foram realizados sobre uma rede de estações de trabalho IBM RS/6000 43P 200MHz, cada uma com 64 Mb de RAM. Tanto nos experimentos com o algoritmo seqüencial VPR, como nas versões do algoritmo TDR, o *cluster* de estações de trabalho foi usado de forma exclusiva, não havendo outras aplicações compartilhando CPU e/ou gerando qualquer outro tráfego na rede.

O algoritmo TDR, em todas as suas três versões, foi completamente desenvolvido em linguagem C e para implementar o mecanismo de troca de mensagens foi usada uma biblioteca MPI (*Message Passing Interface*) padrão, disponibilizada pela IBM para o sistema AIX. Para a criação do grafo de roteamento detalhado e visualização gráfica do roteamento obtido foram utilizadas as rotinas desenvolvidas por V. Betz, para o VPR [14], disponíveis juntamente com o código do VPR em http://www.eecg.toronto.edu/~vaughn/vpr.

Todos os tempos de execução, citados neste capítulo, para as três versões do algoritmo TDR, são dados sempre em segundos.

7.2 Roteador Seqüencial VPR

Para o roteamento do conjunto de redes, em cada processador escravo, foi utilizada uma adaptação do algoritmo VPR [14]. A adaptação consiste, basicamente, em restringir a fase de expansão durante o roteamento de uma rede, quando o algoritmo busca por caminhos no grafo de roteamento detalhado. Além disso, esquemas adicionais de remoção e de reroteamento de redes (reinicialização parcial dos custos de recursos de roteamento) foram incorporados ao algoritmo VPR.

Também foi retirado do algoritmo VPR o procedimento de busca binária usado para encontrar o número mínimo de trilhas necessário para rotear as redes de um dado circuito. Em cada iteração do VPR é tentado o roteamento do conjunto de redes usando um número fixo de trilhas W_{iter}. Se este consegue concluir o roteamento ou verifica que o roteamento é inviável, então um novo valor para W_{iter}, menor ou maior que o valor anterior é tentado na próxima iteração, respectivamente.

A tabela 7.1 mostra para cada circuito, os dados referentes à melhor iteração do algortimo VPR, que consegue todas as suas redes. As colunas W_{iter}, C_{med}, C_{max} e T_{iter} fornecem o número de trilhas requeridas, o comprimento médio das redes, o comprimento máximo de rede e o tempo gasto na iteração, respectivamente. A coluna T_{tot} mostra o tempo total de execução do VPR. A coluna #BLs dá o número de blocos lógicos de cada circuito. Ressalte-se aqui, que para alguns destes circuitos foi observado que o número mínimo de trilhas requeridas pelo roteamento W_{iter} não correspondeu aos valores relatados em Betz [14]. Tais circuitos são: diffeq, dsip, ex1010, frisc, misex3, pdc, s38584.1 e seq, que diferiram em +1, -1, +1, +1, +1, +2, -1 e +1 trilhas, respectivamente. Isto se deve, certamente, aos parâmetros utilizados, tais como: número máximo de iterações, limite sobre a dilatação do menor retângulo que engloba todos os terminais de uma rede e fatores de penalização sobre o compartilhamento de recursos de roteamento. Os mesmos parâmetros utilizados aqui foram mantidos para o roteamento das redes, em cada processador escravo, para as três versões do algoritmo TDR proposto.

Atualmente, os melhores resultados, em termos de área, foram alcançados pelo roteador SC-*Pathfinder* [26], que para alguns dos circuitos de teste avaliados neste trabalho conseguiu rotear usando menos uma trilha. Infelizmente não foi possível

Tabela 7.1: Resultados obtidos pelo VPR para os vinte maiores circuitos MCNC

Circuito	#BLs	Melhor iteracao				T_{tot}
		W_{iter}	C_{med}	C_{max}	T_{iter}	
alu4	1522	10	13,47	938	254	1018
apex2	1878	11	14,54	610	362	1890
apex4	1262	12	16,13	525	299	1426
bigkey	1707	7	7,99	1026	105	593
clma	8383	12	14,81	2770	3486	15150
des	1591	7	11,47	588	404	1546
diffeq	1497	8	8,79	714	99	628
dsip	1370	6	7,84	1215	122	463
elliptic	3604	10	11,56	2116	773	2763
ex1010	4598	11	13,83	1388	996	6211
ex5p	1064	13	17,01	713	434	2944
frisc	3556	12	14,81	2541	1005	5296
misex3	1397	11	14,82	744	250	1281
pdc	4575	18	22,19	1380	4427	25993
s298	1931	7	8,97	935	104	497
s38417	6406	8	9,28	2340	645	3715
s38584.1	6447	8	8,17	3507	1052	4564
seq	1750	12	15,43	1121	382	1840
spla	3690	13	17,06	751	1594	11033
tseng	1407	6	7,52	566	55	208

efetuar a comparação dos resultados deste trabalho com o SC-*Pathfinder*, em termos de desempenho, pois o código fonte deste roteador não está disponível. Assim, as comparações de resultados são feitas apenas em relação ao VPR, uma vez que este é o algoritmo usado em cada processador escravo para rotear as redes.

7.3 Algoritmo TDR usando ciclos de negociação

As figuras 7.1 e 7.2 ilustram para os circuitos alu4 e apex2, respectivamente, o comportamento do algoritmo TDR-CN (Ciclos de Negociação) quando esta estratégia é empregada. Em cada ponto da figura é indicado o número de redes conflitantes no respectivo ciclo de negociação, por exemplo, para o circuito alu4, no ciclo 1 há 506 redes conflitantes. Para o exemplo com o circuito alu4 foram usados quatro processadores escravos, cada um começando com seis domínios de trilhas. Para o exemplo com o circuito apex2 foram usados três processadores escravos, cada um começando com seis domínios de trilhas. Os circuitos alu4 e apex2 possuem 1536

e 1916 redes, respectivamente. Pode-se observar que um percentual alto, 67,05% e 66,96%, respectivamente, das redes destes circuitos é roteado, em definitivo, ao final da primeira iteração, enquanto o restante das redes tem seu roteamento desfeito. Para os circuitos alu4 e apex2 são removidos, ao final da primeira iteracão, 506 e 633 redes conflitantes, respectivamente. Infelizmente o número de redes conflitantes não diminui rapidamente de uma iteração para outra. Isto leva a um grande número de ciclos de negociação onde, em cada um destes, é gerado um forte desbalanceamento de carga de trabalho entre os processadores, o que resulta num desempenho ruim para esta estratégia.

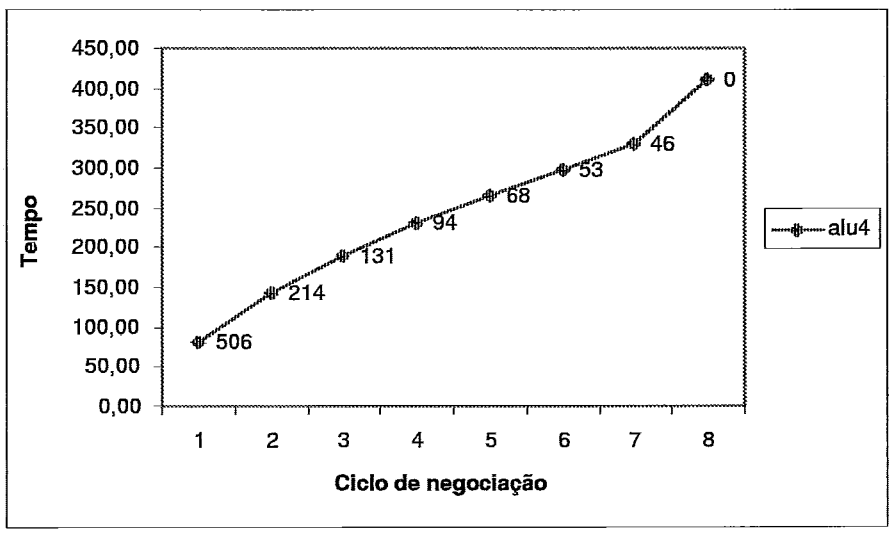

Figura 7.1: Ciclos de negociação do algoritmo TDR-CN para o circuito alu4

A tabela 7.2 demonstra que esta estratégia não oferece nenhum ganho de desempenho, muito pelo contrário. Contudo, observando apenas os dados relativos ao término da primeira iteração: número de trilhas W e o tempo gasto até o início do ciclo, percebe-se que todas as redes são roteadas rapidamente sem que o número de trilhas em cada processador escravo tenha sido incrementado. Os circuitos alu4 e apex2 requerem 254 e 362 segundos, respectivamente, para serem roteados, na melhor iteração do algoritmo VPR, enquanto que o algoritmo TDR-CN roteia todas as redes, atingindo o primeiro ciclo de negociação em apenas 81,69 e 235,58 segundos, respectivamente. Desta observação resultou novo esforço para resolver o problema de compartilhamento de pinos entre redes usando informação a priori sobre que redes poderiam conflitar em função dos pinos de E/S dos blocos lógicos.

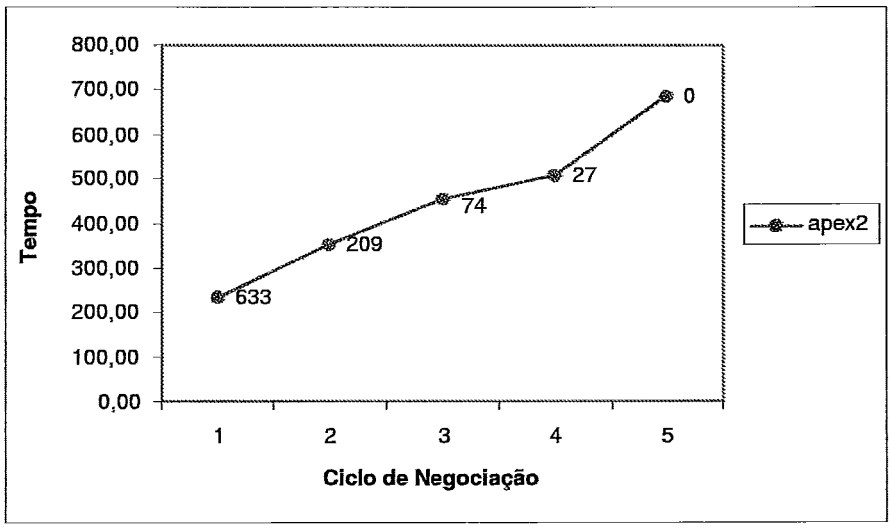

Figura 7.2: Ciclos de negociação do algoritmo TDR-CN para o circuito apex2

Tabela 7.2: Desempenho do TDR c/ ciclos de negociacao versus VPR

			Iteracao inicial		Iteracao final	
Circuito	W_{VPR}	T_{VPR}	W	T	W	T
alu4	10	254	24	81,69	24	410,19
apex2	11	362	18	235,58	20	685,88

7.4 Algortimo TDR usando particionamento hierárquico de redes

As tabelas 7.3 e 7.4 ilustram a distribuição de redes por níveis de particionamento hierárquico, para os circuitos alu4 e apex2, respectivamente. No primeiro nível de particionamento o conjunto de redes é dividido em 32 partições, no segundo nível o conjunto de redes sobre o corte do nível anterior é dividido em 16 partições. Isto se repete até seja alcançado o nível de particionamento com 2 partições. O roteamento inicia com as redes sobre o corte do último nível de particionamento sendo roteadas. E somente após isto é que se inicia o roteamento das partições internas, começando no último nível com duas partições internas e terminando no primeiro nível com 32 partições internas.

Nesta estratégia as redes sobre o corte no nível de particionamento com apenas 2 partições são roteadas via ciclos de negociação envolvendo todos os processadores escravos. Esta fase corresponde à maior parte do tempo de execução, pois, embora, o percentual de redes sendo roteado seja pequeno, 12,82% e 11,89% para os

circuitos alu4 e apex2, respectivamente, o esforço computacional exigido é muito grande, em função do fato de praticamente todas as redes grandes, ou seja, aquelas que se espalham por quase toda a matriz de blocos lógicos do FPGA, pertencerem a este conjunto de redes. Como grande parte das redes ficam distribuídas pelas 32 partições internas do primeiro nível de particionamento e, portanto, podem ser roteadas com um alto grau de paralelismo, era de se esperar que isto refletisse, de forma acentuada, num ganho de tempo de execução. Entretanto, estas redes, embora numerosas, demandam pequeno esforço computacional para serem roteadas, pois possuem poucos terminais, que, em geral, se encontram em blocos lógicos próximos uns dos outros, devido à ação do algoritmo de *placement.*

Tabela 7.3: Distribuiçao de redes por níveis para o circuito alu4

Número de partições em cada nível de particionamento	% redes internas	% redes sobre o corte
32	67,05	32,95
16	6,7	26,26
8	3,84	22,39
4	4,36	18,03
2	5,20	12,82

Tabela 7.4: Distribuiçao de redes por níveis para o circuito apex2

Número de partições em cada nível de particionamento	% redes internas	% redes sobre o corte
32	67,85	32,15
16	7,15	25,00
8	4,22	20,77
4	5,06	15,70
2	3,82	11,89

A tabela 7.5 mostra a distribuição do tempo de execução por nível de particionamento para uma execução do circuito alu4, usando 5 processadores escravos, sendo que o número inicial de trilhas em cada processador é igual 5. O tempo total desta execução foi de 107,88 segundos. Pode-se observar que a fase de paralelismo associada ao particionamento hierárquico da redes demora a se iniciar e demora menos de 25% do tempo total de execução para rotear quase 90% das redes do circuito, o que ilustra a discussão supramencionada.

A presença desta fase de roteamento das grandes redes, sobre o corte do último nível de particionamento, resolvida via ciclos de negociação ou mesmo via roteamento seqüencial, em apenas um dos processadores escravos, implica perda de desempenho minimizando assim o *speedup* obtido. Face a isto, buscou-se uma maneira de tornar possível separar-se também as grandes redes, de modo a serem roteadas sem risco de conflitos posteriores. A solução encontrada para este problema é apresentada na seção seguinte.

Tabela 7.5: Distribuiçao do tempo de execução por níveis de particionamento

Número de partições em cada nível de particionamento	Iniciada depois de (segs)
32	90,47
16	88,97
8	87,19
4	84,93
2	77,36

A figura 7.3 mostra o *speedup* alcançado pelo TDR-PH (Particionamento Hierárquico), para os seguintes circuitos testes, alu4 e apex2, que requerem praticamente o mesmo número mínimo de trilhas para que todo o seu conjunto de redes seja roteado, entretanto referem-se a FPGAs com diferentes números de blocos lógicos. O *speedup* apresentado na figura 7.3 corresponde à melhor escolha, em termos de desempenho, do número inicial de domínios de trilhas nos processadores escravos. E demonstra que o desempenho alcançado é baixo, mesmo quando cinco ou mais processadores escravos são usados.

A tabela 7.6 resume as melhores parametrizações para o algoritmo TDR-PH, considerando uma relação ponderada de desempenho versus área, e então estabelece uma comparação com os resultados obtidos pelo algoritmo VPR [14]. Pode-se notar que, apesar do *speedup* obtido, esta estratégia aumenta bastante o tamanho do FPGA. Isto decorre do fato de que o roteamento das redes de uma partição interna é penalizada pelo roteamento das redes já roteadas das partições internas anteriormente recebidas pelo processador escravo.

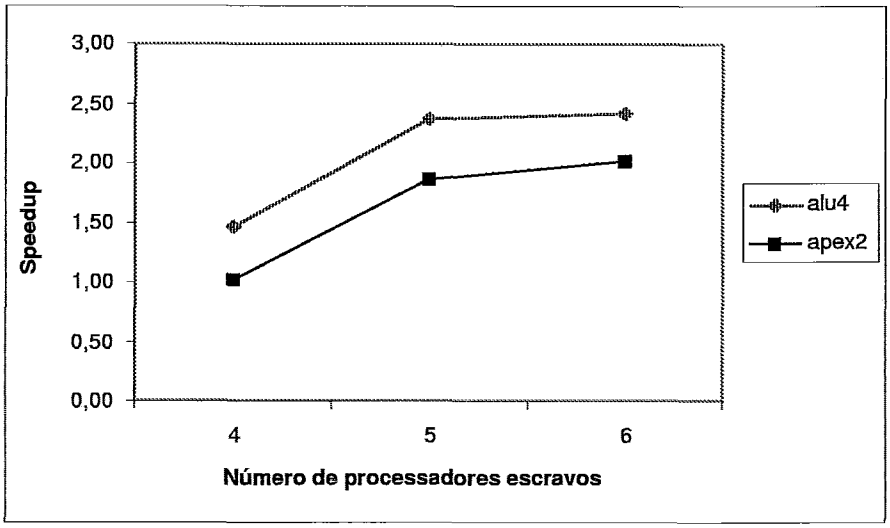

Figura 7.3: Speedup do algoritmo TDR-PH

Tabela 7.6: Resultados obtidos pelo TDR-PH para alguns circutos testes

	Melhor iteração do VPR				Melhor parametrização para TDR-PH					
Circuito	W_{iter}	C_{med}	C_{max}	T_{iter}	W	C_{med}	C_{max}	T	W_{inic}	#procs
alu4	10	13,47	938	254	18	11,49	720	189,65	4	4
apex2	11	14,54	610	362	26	12,47	414	239,70	5	5

7.5 Algortimo TDR usando divisão de pinos de E/S de blocos lógicos

Nesta estratégia, o conjunto de redes é particionado entre os processadores, e com base nesta atribuição de redes a processadores, é feita a respectiva divisão de pinos E/S de blocos lógicos entre os mesmos. Em seguida, cada processador roteia seu conjunto de redes utilizando apenas os pinos de E/S reservados a este processador.

O particionamento do conjunto de redes tem influência direta tanto no *speedup* alcançado como no número total de trilhas requerido. De fato, se as partições não forem uniformes quanto ao esforço computacional necessário para rotear o conjunto de redes associado, então haverá desbalanceamento de carga entre os processadores, prejudicando o *speedup* obtido pelo roteador paralelo TDR-PP (Particionamento de Pinos). Além disso, como o mesmo número inicial de trilhas é atribuído a cada processador, pode ocorrer que num processador este número seja insuficiente, enquanto que em outros processadore este número esteja superdimensionado. Isto implica maior número total de trilhas requerido.

A tabela 7.7 ilustra para os circuitos s38417 e s38584.1, a distribuição do tempo de execução do algoritmo de roteamento nos processadores para cada configuração de número de processadores e número inicial de trilhas. O coeficiente de variação relativo trata-se de uma medida relativa de dispersão, útil para a comparação em termos relativos do grau de concentração em torno da média de série distintas. Este corresponde ao desvio padrão dividido pela média da distribuição expresso percentualmente. Para esta distribuição, esta medida indica que, a despeito do esperado, não há um padrão claro que caracterize a influência do aumento do número de processadores sobre o balanceamento de carga.

Tabela 7.7: Distribuição do tempo de execução entre processadores

Circuitos	#Procs	$W_{inicial}$	Coeficiente de Variação Relativo
s38417	3	3	11,98
	4	3	6,69
	5	3	16,03
	6	3	6,01
	3	4	5,86
	4	4	9,70
	5	4	11,11
	6	4	8,53
s38584.1	3	4	6,62
	4	4	7,60
	5	4	6,46
	6	4	11,57
	3	5	5,59
	4	5	8,49
	5	5	5,25
	6	5	10,26

Os circuitos de teste, com exceção do circuito pdc, foram divididos em quatro grupos, conforme tabela 7.8, de modo a facilitar a apresentação do *speedup* alcançado pelo roteador TDR-PP para estes problemas. Os grupos foram definidos com base no tamanho da matriz de blocos do FPGA necessária para posicionar o circuito e no tempo requerido pelo roteamento. O circuito pdc é apresentado sozinho num único grafo por duas razões: primeiro o fato de que o *speedup* alcançado neste caso ter sido absolutamente superlinear, e segundo, a sua escolha para ilustrar o efeito do parâmetro número máximo de domínios de trilhas por processador sobre o desempenho obtido.

Tabela 7.8: Grupos de circuitos testes

Grupo	Circuitos
01	alu4, apex2, apex4, des, misex3, seq
02	bigkey, dsip, diffeq, s298, tseng
03	elliptic, frisc, s38417, s38584.1, ex5p
04	clma, spla, ex1010

Para um número fixo de processsadores, o número inicial de trilhas também tem influência no *speedup* que pode ser alcançado. A figura 7.4 ilustra este efeito para os circuitos alu4 e apex4, num cenário em que quatro processadores são utilizados. A figura 7.5 ilustra esta influência, para o circuito frisc, considerando cenários com 4, 5 e 6 processadores.

Pode-se observar que inicialmente ao se disponibilizar mais domínios de trilhas em cada processador, tem-se uma melhoria no tempo necessário para efetuar o roteamento do circuito, especialmente acentuada para o cicuito apex4. Isto deve-se ao fato de que nenhum processador necessita aumentar seu número de domínios de trilhas e refazer o roteamento das redes que não conseguiram ser roteadas nos domínios de trilhas iniciais. No entanto, a partir de um dado valor, o tamanho do grafo de roteamento gerado com este número de trilhas torna-se muito grande e faz com que a fase de expansão, durante o roteamento de uma rede, seja mais demorada, aumentando assim o tempo de execução.

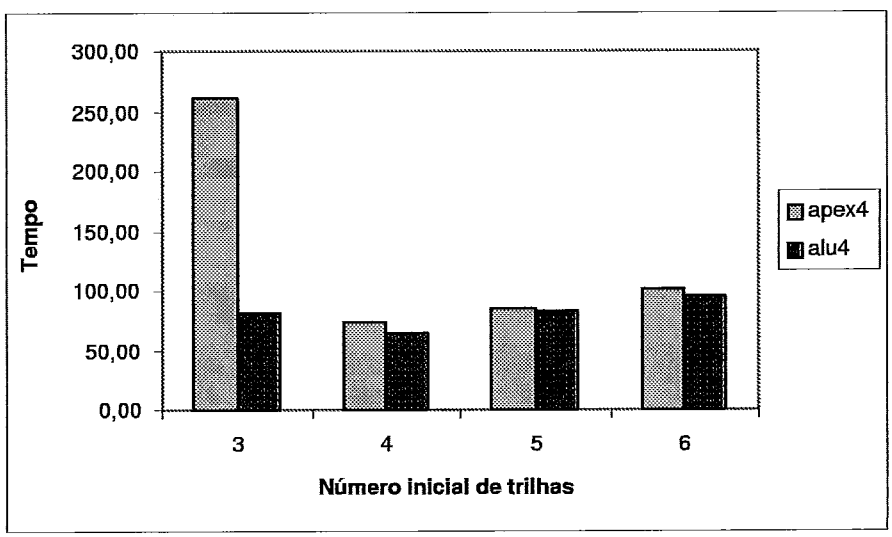

Figura 7.4: Influência do número inicial de domínios de trilhas sobre o *speedup* alcançado

Para o circuito alu4, quando três domínios de trilhas são inicialmente alocados a cada processador, o algoritmo TDR-PP usa duas trilhas a mais do que o mínimo número de trilhas requerido pelo algotimo VPR, porém um *speedup* de 3,32 é alcançado. Para o circuito apex4, quando 3 ou 4 domínios de trilhas são inicialmente alocados a cada processador, o algortimo TDR-PP usa quatro domínios de trilhas a mais do que requerido pelo algoritmo VPR, alcançando um *speedup* de 1,15 e 4,1, respectivamente. Isto mostra que esta estratégia leva à conclusão do roteamento do circuito em muito menos tempo do que é requerido pelo algoritmo VPR, enquanto ainda produz boas soluções em termos de área.

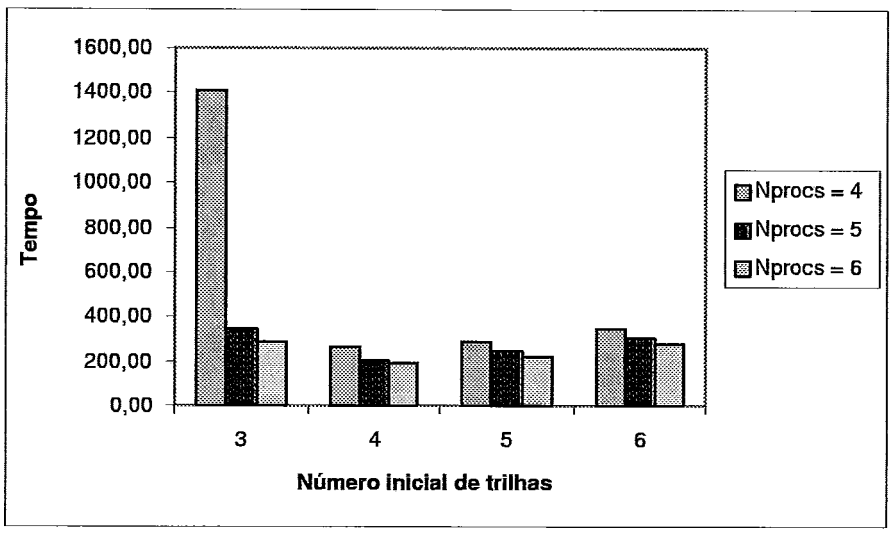

Figura 7.5: Influência do número inicial de domínios de trilhas sobre o *speedup* alcançado, para o circuito frisc

As figuras 7.6, 7.7, 7.8 e 7.9 mostram os ganhos, em termos de tempo, obtidos com a paralelização do roteamento dos circuitos dos grupos 1, 2, 3 e 4, respectivamente . Os dados apresentados consideram o melhor *speedup* alcançado, dentre as diversas parametrizações testadas de número de processadores e número inicial de trilhas. A figura 7.6 demonstra que o algoritmo TDR-PP alcança um comportamento quase linear para os circuitos testes do grupo 1. Porém, para os circuitos do grupo 2, que exigem baixo esforço computacional, ou seja, são roteados muito rapidamente, o algoritmo TDR-PP apresenta um comportamento sublinear, com ganho pouco acentuado, conforme é ilustrado pela figura 7.7. Isto se explica pelo fato de que estes circuitos ocupam um FPGA pequeno, dado que requerem um número baixo

de trilhas.

As figuras 7.8 e 7.9 demonstram que para os circuitos dos grupos 3 e 4 que correspondem aos circuitos mais pesados computacionalmente, obtém-se um ótimo desempenho para o algoritmo TDR-PP. Em alguns casos, como para o circuito s38584.1 por vezes um comportamento superlinear é alcançado, por exemplo, quando são usados quatro procesadores escravos. Isto deve-se primeiro a um particionamento de redes que levou a um bom balanceamento de carga entre os processadores. Segundo, o acesso restrito a apenas alguns domínios de trilhas do grafo de roteamento detalhado, cuja representação na memória é não contígua deve ter provocado melhor otimização do uso da memória cache, uma vez que um número menor de operações de troca de conteúdo da memória cache deva ter sido necessário.

Em outros casos, como para o circuito clma, o maior circuito do *benchmark* avaliado, obteve-se um comportamento sublinear, que se explica pelo fato de que para os diferentes números de processadores não foi alcançado um bom balanceamento de carga.

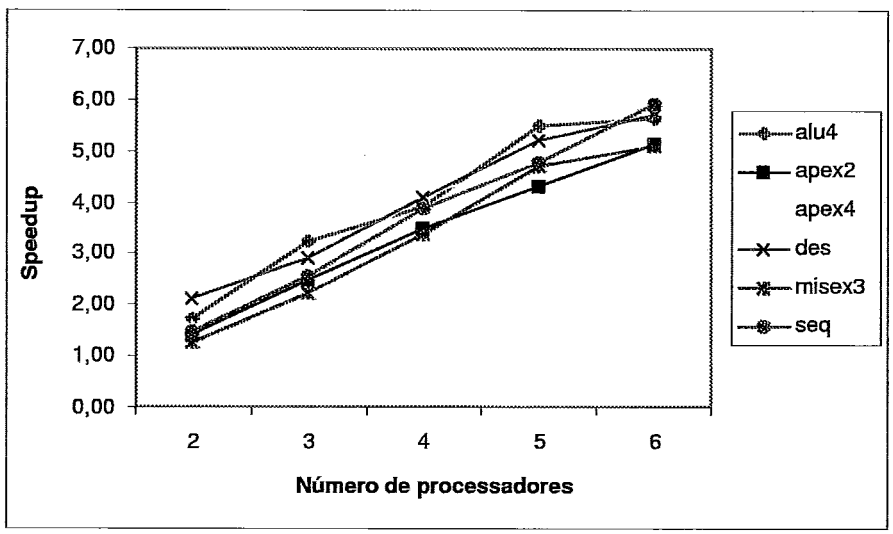

Figura 7.6: *Speedup* do algoritmo TDR-PP para os circuitos do grupo 1

A tabela 7.9 resume as melhores parametrizações para o algoritmo TDR usando esta estratégia, considerando uma relação de desempenho X área, e então estabelece uma comparação com os resultados obtido pelo algoritmo VPR [14].

Para os circuitos ex5p, frisc, misex3, seg, pdc e spla foram obtidas soluções com ótimos *speedups* e com um aumento bem pequeno da área do FPGA. O circuito ex5p,

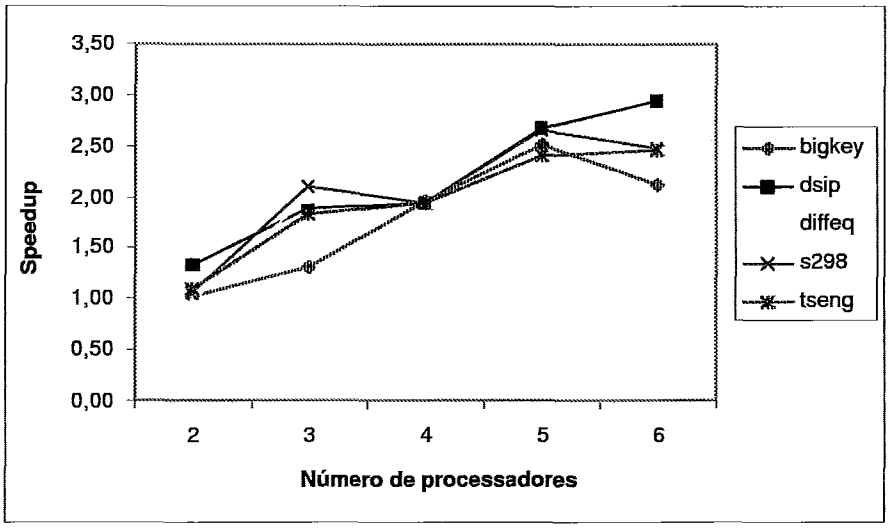

Figura 7.7: *Speedup* do algoritmo TDR-PP para os circuitos do grupo 2

por exemplo, foi roteado cinco vezes mais rápido enquanto que apenas 3 trilhas a mais foram necessárias. Entretanto, para os circuitos clma e dsip, embora tenha havido bons *speedups*, o número de trilhas utilizado foi quase o dobro do mínimo obtido pelo VPR.

Outrossim, esta estratégia apresenta, em geral, bons *speedups* com um pequeno custo adicional em termos de área. Mas uma vez que se adquira qualquer pastilha de FPGA comercial, cujo número máximo de trilhas é definido pelo fabricante, se o circuito couber facilmente no FPGA, o problema passa a ser manter a característica da prototipação da ferramente CAD para FPGAs. Assim, ainda que o circuito projetado use mais trilhas para ser roteado do que o número mínimo de trilhas, o mais importante é que se consiga rotear rapidamente o circuito, após cada modificação efetuadas no projeto do circuito para que este possa assim ser testado.

Em geral, devido ao roteamento obtido usar um número maior de trilhas, consegue-se rotear as redes com menor comprimento médio e também menor comprimento máximo de rede. Porém, para uns poucos circuitos, este comportamento não foi observado, por exemplo, para o circuito dsip e clma, diffeq, ex1010, frisc e s298 houve uma piora nessas medidas, embora tenham sido roteados usando mais trilhas. Isto decorre, certamente, pelo fato de que o procedimento de reserva de pinos torna indisponíveis alguns pinos de entrada em cada bloco lógico para cada processador, provocando uma perda de flexibilidade do algoritmo de roteamento quanto à escolha

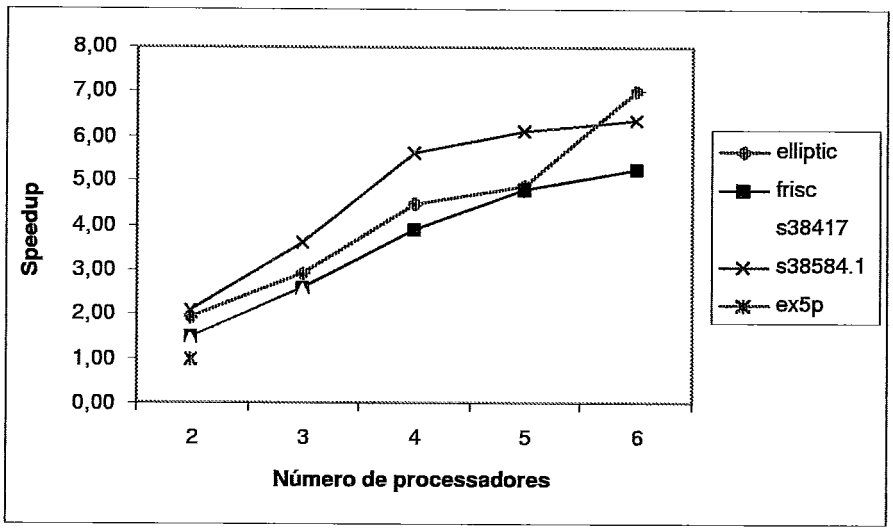

Figura 7.8: *Speedup* do algoritmo TDR-PP para os circuitos do grupo 3

do melhor pino de entrada em um bloco lógico para completar um conexão, o que força a construção de redes mais longas. Um provável congestionamento excessivo em algum processador pode também ter contribuído. Já para o circuito s38417 foi obtida uma redução expressiva do comprimento máximo de rede.

Em todos os resultados apresentados até aqui, para cada processador era construído um grafo de roteamento correspondente a um FPGA com um número de trilhas igual ao número mínimo de trilhas obtido pela melhor iteração do VPR. Isto deve-se ao fato de que não é possível prever quantas trilhas serão necessárias em cada processador, para que este possa rotear todo o seu conjunto de redes. Assim sendo, quando um processador incrementa o seu número de domínios de trilhas, ele apenas libera o acesso de parte da estrutura de dados já construída que corresponde a um domínio de trilhas ainda não utilizado, e com isso torna estes nós do grafo de roteamento disponíveis para roteamento. A figura 7.10 ilustra, para o circuito pdc, o modo como o número máximo de domínios que podem ser criados num processador afeta o *speedup* obtido, ou seja, mostra que foram consideradas duas execuções do algoritmo TDR-PP para o circuito pdc, com o número máximo de domínios por processador igual a 8 e 16, respectivamente.

Se o conjunto de redes for dividido em um número de partições maior do que o número de processadores, pode ser obtido um melhor balanceamento, desde que seja provido um esquema dinâmico de divisão de pinos de E/S de blocos lógicos. No

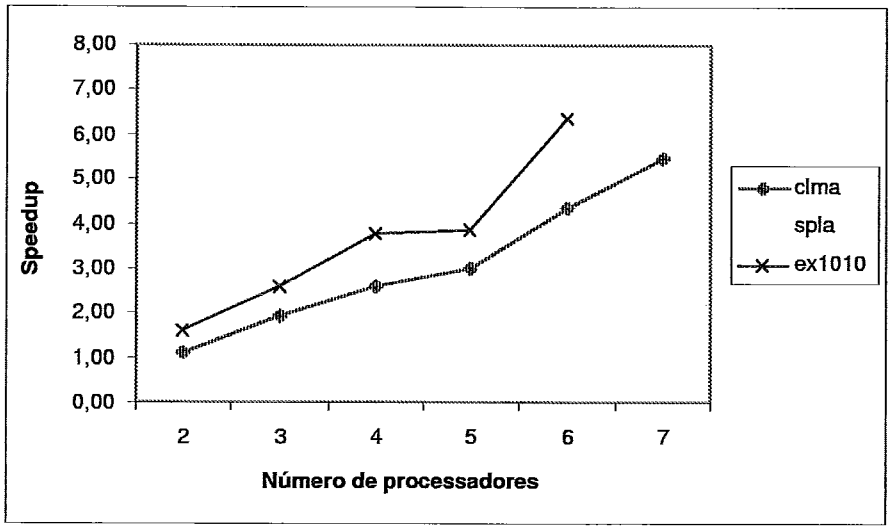

Figura 7.9: *Speedup* do algoritmo TDR-PP para os circuitos do grupo 4

entanto, fazendo isto, perde-se em termos de área do FPGA.

Se uma partição grande de redes é enviada a um processador, o algoritmo VPR que fez o roteamento local negocia entre as redes a formação de suas árvores de roteamento, conseguindo assim que nenhuma rede tenha seu comprimento penalizado. Por outro lado, se partições pequenas são enviadas, então as redes de uma dada partição não poderão negociar a ocupação de um dado segmento de fio com as redes de uma partição já roteada, ou seja, elas não poderão tentar usar este segmento de fio, uma vez que a rede que o utiliza não poderá mais ser desfeita. Isto faz com que as redes das últimas partições a serem roteadas tenham seu comprimento penalizado.

Um outro aspecto importante diz respeito à forma pela qual é executado o particionamento. Pode-se aplicar um algoritmo de particionamento específico ou uma rotina de particionamento aleatório. Neste trabalho, foi usado um esquema de particionamento bastantes simples, no qual cada rede é alocada a um dado processador, se o resto da divisão do número seqüencial desta rede na lista de redes, pelo número de processadores for igual ao número seqüencial deste processador na lista de processadores. Também foi avaliada a utilização de um algoritmo de particionamento de grafos, o algoritmo METIS [55].

Os experimentos efetuados mostraram que a simples heurística de particionamento da lista de redes consegue distribuir, mais equitativamente, o grupo das redes grandes, enquanto que o algortimo METIS ao particionar o grafo de interseção, us-

Tabela 7.9: Comparação dos algoritmos VPR e TDR-PP, em termos de área.

Circuito	Melhor iteração do VPR				Melhor parametrização para TDR-PP					
	W_{iter}	C_{med}	C_{max}	T_{iter}	W	C_{med}	C_{max}	T	W_{inic}	#procs
alu4	10	13.47	938	254	12	12,50	836	80,29	3	4
apex2	11	14.54	610	362	15	14,41	494	84,21	3	5
apex4	12	16.13	525	299	16	15,93	425	70,35	4	4
bigkey	7	7.99	1026	105	12	7,85	928	80,58	4	3
clma	12	14,81	2770	3486	24	16,59	4070	1799,80	8	3
des	7	11,47	588	404	10	11,13	559	192,11	5	2
diffeq	8	8,79	714	99	9	8,94	779	82,07	3	3
dsip	6	7,84	1215	122	12	9,16	1795	64,43	4	3
elliptic	10	11,56	2116	773	12	11,21	2216	264,75	4	3
ex1010	11	13,83	1388	996	12	14,15	1364	622,50	6	2
ex5p	13	17,01	713	434	16	16,50	684	83,97	4	4
frisc	12	14,81	2541	1005	15	14,93	2344	385,83	5	3
misex3	11	14,82	744	250	12	14,61	679	145,70	4	3
pdc	18	22,19	1380	4427	20	21	1235	603,86	5	4
s298	7	8,97	935	104	9	9,22	957	49,39	3	3
s38417	8	9,28	2340	645	10	9,13	1803	448,69	5	2
s38584.1	8	8,17	3507	1052	12	8,14	3747	186,95	3	4
seq	12	15,43	1121	382	13	14,70	752	202,66	4	3
spla	13	17,06	751	1594	16	16,35	739	349,98	4	4
tseng	6	7,52	566	55	9	7,86	611	29,92	3	3

ando apenas pesos nos vértices representando uma estimativa do comprimento das redes permitiu que grandes redes com muitos vizinhos em comum fossem alocadas a uma mesma partição. Assim o algoritmo METIS, apesar de aparentemente dividir os nós em partições balanceadas, em termos de estimativa de comprimento total de fios e de número de redes, na verdade estas partições não estão balanceadas em termos de esforço computacional requerido, devido ao particionamento não equitativo das grandes redes. A figura 7.11 ilustra esta influência do esquema de particionamento no balanceamento de carga obtido, que se reflete no *speedup* alcançado, para o circuito teste elliptic, comparando essas duas estrátegias de particionamento.

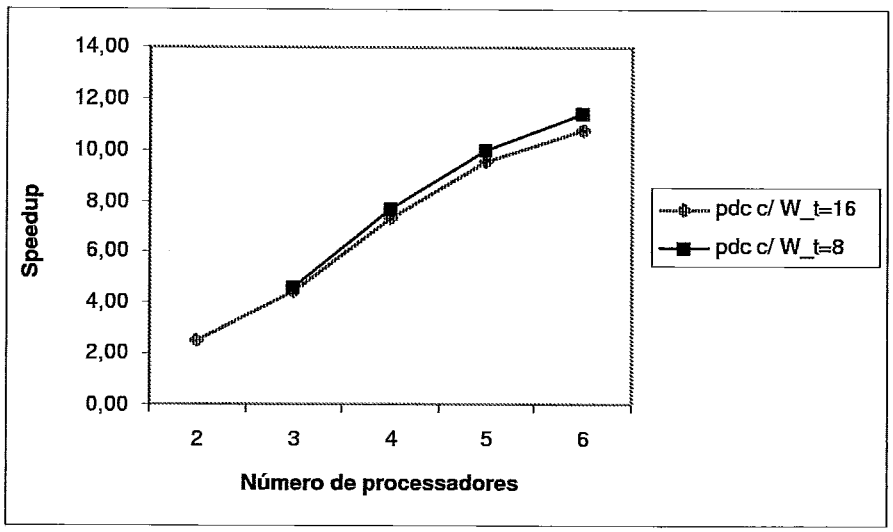

Figura 7.10: Influência do número máximo do trilhas sobre *speedup* do algoritmo TDR-PP para o circuito pdc

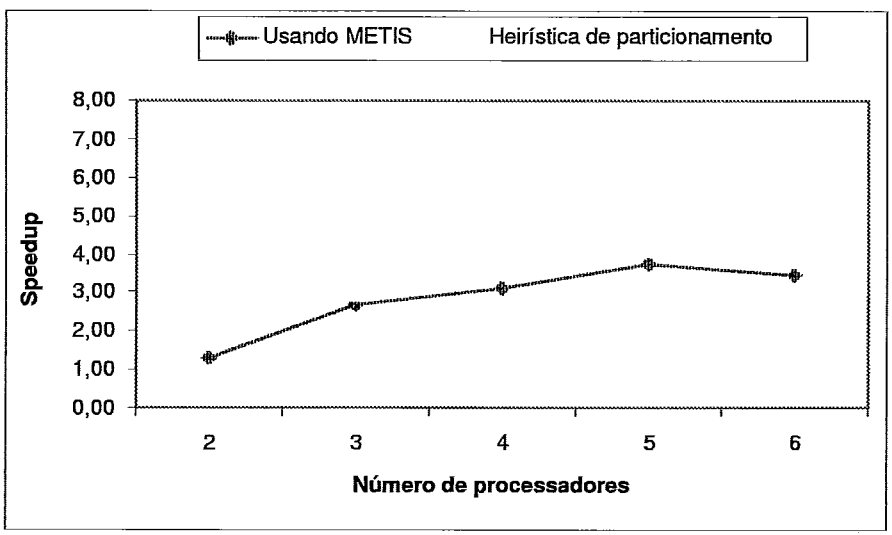

Figura 7.11: Influência da heurística de particionamento sobre o *speedup* alcançado, para o circuito elliptic

Capítulo 8

Conclusões e Trabalhos Futuros

Este trabalho incorpora o caráter inovador do paralelismo, que começa a ser explorado pelas ferramentas de CAD para o projeto de circuitos baseados em FPGAs. O foco de aplicação da estratégia de paralelização proposta neste trabalho é o cenário do ciclo de desenvolvimento de projeto onde um mesmo circuito é roteado a cada vez que o seu projeto é modificado. Uma forma inovadora de paralelização da fase de roteamento é apresentada neste trabalho, onde, ao se explorar características especias da arquitetura *island-style* (baseada em arrays) de FPGA, conseguiu-se desacoplar parcialmente o grafo de roteamento detalhado. E a partir do particionamento do conjunto de redes entre os processadores foi desenvolvido um esquema de atribuição de pinos de entrada de blocos lógicos a processadores, de modo a se obter um desacoplamento total do grafo de roteamento detalhado, em termos de domínios de trilhas. Tal estratégia de desacoplamento total, apesar de tornar o roteamento menos flexível em relação às opções de entrada num bloco lógico para uma rede, permitiu que bons *speedups* fossem obtidos, em geral, com pequena piora em termos de aumento de área e do comprimento máximo de rede. Em alguns poucos casos houve um aumento excessivo do número de trilhas, talvez decorrente de um particionamento patológico.

O desempenho alcançado pelo algoritmo TDR-PP demonstra que esta estratégia de paralelização consegue rotear os circuitos muito rapidamente atingindo *speedup* linear para a maior parte dos circuitos experimentados. Por exemplo, para o circuito teste pdc, que possui 4631 blocos lógicos e 4591 redes, o tempo de roteamento caiu de 4427 segundos para 603 segundos, quando quatro processadores com cinco trilhas inicialmente foram usados, sendo utilizadas 20 trilhas, apenas duas a mais do

que o número de trilhas requerido pelo algoritmo sequencial VPR. Também houve uma redução do comprimento médio e do comprimento máximo de rede. E devido ao seu custo de comunicação extremamente baixo, o algoritmo TDR-PP consegue escalar o *speedup* com o aumento do número de processadores. O bom desempenho obtido demonstra a importância de um algoritmo desta natureza, que resgata uma das principais vantagens do FPGA que é a da prototipação, uma vez que os ciclos de modificação, compilação e teste do circuito sendo projetados serão muito mais rápidos.

Além disso, o crescimento do nível de integração nos dispositivos FPGA moderno, e portanto, do grafo de roteamento associado, a tarefa de roteamento torna-se-á ainda mais crítica, passando a consumir um número grande de horas. Este crescimento certamente ressaltará as vantagens do algoritmo TDR-PP, pois a diminuição do custo computacional referente à fase de expansão durante o roteamento de uma rede, se tornará ainda mais acentuada.

Além dos bons resultados de desempenho alcançados com a exploração de paralelismo durante a fase de roteamento de circuitos FPGA, neste trabalho os resultados computacionais obtidos para a heurística proposta para o problema da árvore de Steiner retilínea demonstram sua eficiência para resolver problemas com poucos terminais, que se aplica muita bem ao roteamento em FPGAs onde em média as redes têm menos de cinco terminais.

Muitos são os aspectos relacionados à paralelização da fase de roteamento de circuitos baseados em FPGAs que demandam novas pesquisas, são eles:

- Novas estratégias de paralelização que consigam, de fato, dividir o trabalho de roteamento das redes explorando outras características arquiteturais do FPGA, ou usando algum esquema menos restritivo, em relação a quais pinos de entrada, em um dado bloco lógico, uma determinada rede pode usar, do que o procedimento de reservas de pinos de entrada de blocos lógicos a processadores escravos, adotado neste trabalho, que por vezes impõe uma única opção de entrada num bloco lógico para uma rede. Também deve ser explorado o paralelismo em outros ambientes como: plataformas de memória compartilhada e híbridas.

- Desenvolvimento de novas heurísticas de particionamento ou melhor orientação das heurísticas de particionamento de grafo existentes, para que gerem partições mais balanceadas do grafo de conflito de redes, em termos de esforço computacional para rotear as redes destas partições. Para que isto ocorra, é necessário também que o grafo de conflito de redes represente mais precisamente as possibilidades de conflitos entre as diversas redes do circuito, face à competição por pinos de entrada de blocos lógicos durante o roteamento destas redes.

- Investigação das formulações de programação inteira para o problema de roteamento global e de suas possíveis relaxações lagrangeanas.

Em relação ao algoritmo TDR-PP apresentado neste trabalho pretende-se efetuar as seguintes extensões:

- Investigar novas formas de particionamento de redes que promova melhor balanceamento de carga entre os processadores, de modo que também possam ser usados diferentes números inicias de trilhas nos processadores escravos. Espera-se assim, obter não somente melhor desempenho para o algoritmo TDR-PP, mas principalmente torná-lo mais competitivo em termos de número total de trilhas.

- Substituir o algoritmo VPR, que atualmente faz o roteamento das redes em cada processador escravo, e cuja fase de construção das redes é feita por $k-1$ invocações do algortimo de caminhos mínimos, onde k é o numero de terminais da rede, por um outro algoritmo de roteamento também baseado no mesmo princípio de negociação, mas que utilizará para o problema da árvore de Steiner retilínea mínima a heurística apresentada neste trabalho. Tal heurística permitirá construir uma aproximação da árvore de Steiner retilínea mínima, bem mais rapidamente, uma vez que sua complexidade de tempo é $O(kn)$, onde $k = |T|$ e T denota o conjunto de terminais da rede.

Referências Bibliográficas

[1] Alexander, M. J., Cohoon, J. P., Ganley, J. L., Robins, G. "An Architecture-Independent Approach to FPGA Routing Based on Multi-Weighted Graphs". In *Proc. European Design Automation Conf.*, pp. 259–264, 1994.

[2] Alexander, M. J., Robins, G. "A New Approach to FPGA Routing Based on Multi-Weighted Graphs". In *Proc. ACM/SIGDA Intl. Workshop on Field-Programmable Gate Arrays*, 1994.

[3] Alexander, M.J., Cohoon, J.P., Ganley, J.L., Robins, G. "Perfomance-Oriented Placement and Routing for Field-Programmable Gate Arrays". In *European Design Automation Conference*, 1995.

[4] Alexander, M.J., Robins, G. "New Perfomance Driven FPGA Routing Algorithms". In *Design Automation Conference*, 1995.

[5] Alpert, C., Chan, T., et al, D. Huang. "Quadratic Placement Revisited". In *Design Automation Conference*, pp. 752–757, 1997.

[6] Anderson, J., Seth, S., Roy, K. "A Coarse-Grained FPGA Architecture for High-Performance FIR Filtering". In *Sixth International Symposium on Field Programmable Gate Arrays*, 1998.

[7] Aude, J. "Uma Proposta de Implementação do Algoritmo de Lee no Multiprocessador Multiplus". In *Anais do VI Congresso da Sociedade Brasileira de Microeletrônica*, pp. 555–568, 1991.

[8] Aude, J., Lopes, E., Martins, M., Pinto, S. "ARCO - A Cost-Effective and Flexible Hardware Maze Router". *Microprocessing and Microprogramming*, v. 38, pp. 149–159, 1993.

[9] Bahiense, L., Maculan, N., Sagastizábal, S. *On the Convergence of The Volume Algorithm.* Technical report, Programa de Engenharia de Sistemas e Computação,UFRJ, 1999.

[10] Beasley, J. "An Algorithm for The Steiner Problem in Graphs ". *Networks*, v. 14, pp. 147–159, 1984.

[11] Beasley, J. "OR-Library: distributing teste problems by eletronic mail ". *Journal of the Operational Research Society*, v. 41, pp. 1069–1072, 1990.

[12] Beasley, J. "A heuristic for Euclidean and rectilinear Steiner problems ". *European Journal of Operational Research Society*, v. 58, pp. 284–292, 1992.

[13] Betz, V., Rose, J. "Directional Bias and Non-Uniformity in FPGA Global Routing Architectures". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, v. 11, n. 5, pp. 620–628, 1992.

[14] Betz, V., Rose, J. "VPR: A New Packing, Placement and Routing Tool for FPGA Research". In *International Workshop on Field Programmable Logic and Applications*, 1997.

[15] Betz, V., Rose, J. "How Much Logic Should Go in a FPGA Logic Block?". *IEEE Design and Test Magazine*, pp. 10–15, 1998.

[16] Betz, V., Rose, J., Marquardt, A. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, London, 1999.

[17] Breuer, M. "Min-Cut Placement". *Journal of Design Automation and Fault Tolerant Computing*, pp. 343–362, 1977.

[18] Brown, S., Rose, J., Vranesic, Z. "A Detailed Router for Field-Programmable Gate Arrays". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, v. 11, n. 5, pp. 620–628, Maio 1992.

[19] Brown, S. D., Khellah, M., Lemieux, G. "Segmented Routing for Speed-Performance and Routability in Field-Programmable Gate Arrays". *Journal of VLSI Design*, v. 4, n. 4, pp. 275–291, 1996.

[20] Cabral, L. *Roteamento FPGA: um enfoque heurístico e de programação inteira* . Exame de qualificação ao doutorado, Progr. de Eng. de Sistemas e Computação, COPPE/UFRJ, 1998.

[21] Cabral, L., Aude, J., Maculan, N. "FPGA Routing: A Brief Survey and a Parallel Strategy". In *Anais do Simpósio Brasileiro de Pesquisa Operacional*, pp. 1624–1633, 1999.

[22] Cabral, L., Aude, J., Maculan, N. "Uma Heuristica para o Problema da Àrvore de Steiner Retilínea". In *Anais do X CLAIO - Conferência Latino-Americana de Pesquisa Operacional e Sistemas*, Cidade de México, 2000.

[23] Cabral, L., Aude, J., Maculan, N. "TDR: A Distributed-Memory Parallel Routing for FPGAs". In *submetido ao Custom Integrated Circuits Conference*, 2001.

[24] Carte, W., Duong, K., et al, R. H. Freeman. "A User Programmable Reconfigurable Gate Array ". In *Proc. 1986 Custom Integrated Circuits Conference*, pp. 233–235, 1986.

[25] Chan, P., Schlag, M. "Acceleration of An FPGA Router". In *IEEE FCCM*, pp. 175–181, 1997.

[26] Chan, P., Schlag, M., Ebeling, C., McMurchie, L. "Distributed-Memory Parallel Routing for Field-Programmable Gate Arrays ". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, v. 19, n. 8, pp. 850–862, 2000.

[27] Chang, Y. W., Thakur, S., Zhu, K., Wong, D. F. "A New Global Routing Algorithm for FPGAs". In *Proc. IEEE Intl. Conf. Computer-Aided Design*, 1994.

[28] Chen, C. D., Lee, Y. S., Wu, A. C. H., Lin, Y. L. "A Perfomance and Routability Driven Router for FPGAs Considering Path Delays". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, v. 14, n. 3, pp. 371–374, 1995.

[29] Chopra, S. "Comparison of formulations and a heuristic for packing Steiner trees in a graph". *Annals of Operations Research*, v. 50, pp. 143–171, 1994.

[30] Chung, K., Singh, S., Rose, J., Chow, P. "Using Hierarchical Logic Blocks to Improve the Speed of Field Programmable Gate Arrays". In *ACM Symp. on FPGAs*, 1999.

[31] Cong, J., Leung, K. S., Zhou, D. "Performance-Driven Interconnect Design Based on Distributed RC Delay Model". In *Proc. ACM/IEEE Design Automation Conf.*, pp. 606–611, 1993.

[32] Corporation, Vantis. *VF1 Field Programmable Gate Arrays.* Preliminary Data Sheet, 1998.

[33] Dijkstra, E. W. "A Note on Two Problems in Connection With Graphs". *Numerische Mathematik*, v. 1, pp. 269–271, 1959.

[34] Ebeling, C., McMurchie, L., Hauck, S. A., Burns, S. "Placement and Routing Tools for the Triptych FPGA". *IEEE Trans. on VLSI*, pp. 473–482, 1995.

[35] Francis, R., Rose, J., Vranesic, Z. "Chortle-crf: Fast Technology Mapping for Lookup Table-based FPGAs ". In *Proc. 28th Design Automation Conference*, pp. 227–233, 1991.

[36] Gajski, D., Kuhn, R. "Guest Editors' Introduction : New VLSI Tools". *IEEE Computer*, v. 6, n. 12, pp. 11–14, 1983.

[37] Garey, M., Johnson, D. "The rectilinear Steiner tree problem is NP-complete". *SIAM Journal on Applied Mathemathics*, v. 32, n. 4, pp. 826–834, 1977.

[38] Greene, J., Roychowdhury, V., Kaptanoglu, S., Gamal, A. El. "Segmented Channel Routing". In *Proc. 27th Design Automation Conference*, pp. 567–572, 1990.

[39] Griffith, J., Robins, G., Salowe, J. S., Zhang, T. "Closing the Gap: Near-Optimal Steiner Trees in Polynomial Time". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, v. 13, n. 11, pp. 1351–1365, 1994.

[40] Grotschel, M., Martin, A., Weismantel, R. *Packing Steiner trees: a cutting plane algorithm and computational results.* Technical report, Konrad-Zuse-Zentrum fur Informtaionstechnik Berlin, 1992.

[41] Grotschel, M., Martin, A., Weismantel, R. *Packing Steiner trees: Polyhedral investigations.* Technical report, Konrad-Zuse-Zentrum fur Informtaionstechnik Berlin, 1992.

[42] Hanan, M. "On Steiner's problem with rectilinear distance ". *SIAM Journal on Applied Mathemathics*, v. 14, pp. 225–265, 1966.

[43] Ho, J., Vijayan, G., Wong, C. "A new approach to the rectilinear Steiner tree problem". In *Digest of the IEEE International Symposium on Eletrical Networks*, pp. 14–15, 1971.

[44] Hsieh, H., Carter, W., et al, J. Ja. "Third-Generation Architecture Boosts Speed and density of Field-Programmable Gate Arrays ". In *Proc. 1990 Custom Integrated Circuits Confererence*, pp. 31.2.1–31.2.7, 1990.

[45] Hsieh, H., Duong, K., et al, J. Ja. "A Second Generation User-Programmable Gate-Array ". In *Proc. 1987 Custom Integrated Circuits Conference*, pp. 515–521, 1987.

[46] Hsu, Y. C., Pan, Y., Kubitz, W. J. "A path selection global router ". In *24th ACM/IEEE Design Automation Conference*, pp. 641–644, 1987.

[47] Huang, D., Kahng, A. "Partitioning-Based Standard-Cell Global Placement with an Exact Objective". In *ACM Symp. on Physical Design*, pp. 18–25, 1997.

[48] Hwang, F. K. "An O(n log n) algorithm for rectilinear minimal spanning trees ". *Journal of the ACM*, v. 26, pp. 177–182, 1979.

[49] Hwang, F. K., Richards, D. S., Winter, P. *The Steiner Tree Problem.* Annals of Discrete Mathematics, Vol 53, North-Holland, 1992.

[50] Inc., Actel. *FPGA Data Book and Design Guide*, 1994.

[51] Inc., ALTERA. *Data Book*, 1998.

[52] Kahng, A. B., Robins, G. "A New Class of Iterative Steiner Tree Heuristics With Good Performance". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, v. 11, pp. 893–902, 1992.

[53] Kahng, A. B., Robins, G. "On perfomance bounds for a class of rectilinear Steiner tree heuristics in arbitrary dimension". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, v. 11, pp. 1462–1465, 1992.

[54] Kahng, A. B., Robins, G. *On Optimal Interconnections for VLSI.* Kluwer Academic Publishers, 1995.

[55] Karypis, G., Kumar, V. *METIS 3.0: Unstructured graph partitioning and sparse matrix ordering system.* Technical report, Department of Computer Science, University of Minnesota, 1997. Disponível na internet em http://www.cs.umn.edu/~metis.

[56] Khellah, M., Brown, S., Vranesic, Z. "Modelling Routing Delays in SRAM-based FPGAs". In *Proc. 1993 CCVLSI*, pp. 6B.13–6B.18, 1993.

[57] Khellah, M., Brown, S, Vranesic, Z. "Minimizing Interconnection Delays in Array-based FPGAs". In *Proc. 1994 Custom Integrated Circuits Confererence*, 1994.

[58] Knopman, J. *Algoritmos genéticos e de simulated annealing: aplicação ao problema do placement e técnicas de paralelização.* PhD thesis, COPPE/UFRJ, 1996.

[59] Kou, L., Markowsky, G., Berman, L. "A Fast algorithm for Steiner Trees". *Acta Informatica*, v. 15, pp. 141–145, 1981.

[60] Kruskal, J. B. "On the shortest spanning subtree of a graph and the travelling salesman problem ". In *Prodeedings of the American Mathematical Society*, v. 7, pp. 48–50, 1956.

[61] Lee, C. Y. "An Algorithm for Path Connections and its Applications". *IRE Trans. Electron. Comput.*, v. 10, pp. 346–365, 1961.

[62] Lee, J. L., Bose, N. K., Hwang, F. K. "Use of Steiner's problem in subotimal routing in rectilinear metric ". *IEEE Transactions on Circuits and Systems*, v. 23, pp. 470–476, 1976.

[63] Lee, Y. S., Wu, A. C. H. "A Perfomance and Routability Driven Router for FPGAs Considering Path Delays". In *Design Automation Conference*, 1995.

[64] Lemieux, G. G., Brown, S. D. "A Detailed Routing Algorithm for Allocating Wire Segments in Field-Programmable Gate Arrays". In *Proc. ACM/SIGDA Physical Design Workshop*, 1993.

[65] Lengauer, T. *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley, 1990.

[66] Lengauer, T., Lugering, M. *Integer programming formulations of global routing and placements problems.* Technical report, University of Paderborn, Alemanha.

[67] Maculan, N. "The Steiner problem in graphs ". *Annals of Discrete Mathematics*, v. 31, pp. 185–212, 1987.

[68] Maculan, N., Claus, A. *Une Nouvelle Formulation du Probléme de Steiner sur un Graphe.* Technical report, Centre de Recerche sur les Transports, Université de Montréal, 1983.

[69] Marquardt, A., Betz, V., Rose, J. "Using Cluster-Based Logic Blocks to Improve FPGA Speed and Density". In *ACM Symp. on FPGAs*, 1999.

[70] McMurchie, L.E., Ebeling, C. "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs". In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 111–117, 1995.

[71] Microelectronics, AT&T. *Optimized Reconfigurable Cell Array (ORCA) Series Field Programmable Gate Arrays.* Advanced Data Sheet, 1993.

[72] Mulvey, J., Crowder, H. "Cluster Analysis: An Application of Lagrangian Relaxation". *Management Science*, v. 25, n. 4, pp. 329–340, 1979.

[73] Nag, S., Rutenbar, R. "Perfomance-Driven Simultaneous Place and Route for Row-Based FPGAs". In *Design Automation Conference*, pp. 301–307, 1994.

[74] Nag, S., Rutenbar, R. "Perfomance-Driven Simultaneous Place and Route for Island-Style FPGAs". In *ICCAD*, pp. 332–338, 1995.

[75] Pacheco, P. S. *Parallel Programming with MPI.* Morgan Kaufmann Publishers, 1997.

[76] Palczewski, Mikael. "Plane Parallel A* Labirinto Router". In *29th ACM/IEEE DAC*, pp. 691–697, 1992.

[77] Preas, B., Lorenzetti, M. *Physical Design Automation of VLSI Systems.* Benjamim/Cummings Publishing Company, 1988.

[78] Prim, R. "Shortest connection networks and some generalisations". *Bell System Technical Journal*, v. 36, pp. 1389–1401, 1957.

[79] Quicklogic, . *An Introduction to Quicklogic's pASIC Devices and SpDE Development Enviroment.* Data Sheet from Quicklogic, 1991.

[80] Rao, S. K., Sadayappan, P., Hwang, F. K., Shor, P. W. "The Rectilinear Steiner Arborescence Problem". *Algorithmica*, v. 15, n. 1, pp. 277–288, 1992.

[81] Richards, D. "Fast heuristic algoritms for rectilinear Steiner trees ". *Algorithmica*, v. 4, pp. 191–207, 1989.

[82] Ries, B., Ettelt, G. "Speed: Fast and Efficient Timing Driven Placement". In *IEEE Int. Symp. on Circuits and Systems*, pp. 377–380, 1995.

[83] Rose, J. "Parallel Global Routing for Standard Cells". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, v. 9, n. 10, pp. 342–362, Outubro 1990.

[84] Rose, J., Brown, S. "Flexibility of Interconnection Structures in Field-Programmable Gate Arrays". *IEEE Journal of Solid State Circuits*, v. 26, n. 3, pp. 277–282, Março 1991.

[85] Rose, J., Francis, R., Lewis, D., Chow, P. "Architecture of Field Programmable Gate Arrays: The Effect of Logic Block Funcionality on Area Efficiency". *JSSC*, pp. 1217–1227, 1990.

[86] Rose, J., Vranesic, Z. G., Snelgrove, W. M. " ALTOR: An Automatic Standard Cell Layout Program". In *Proc. Can.Conf. VLSI*, pp. 168–173, 1985.

[87] Roy, K. "A Bounded Search Algorithm for Segmented Channel Routing for FPGAs and Associated Channel Architecture Issues". *IEEE Transactions on Computer-Aided-Design of Integrated Circuits*, pp. 1695–1705, 1993.

[88] Roy, K., Mehendale, M. "Optimization of Channel Segmentation for Channelled Architecture FPGAs". In *IEEE Custom Integrated Circuits Conference*, pp. 4.4.1–4.4.4, 1992.

[89] Roy, K., Nag, S. "Automatic Synthesis of FPGA Channel Architecture for Performance and Routability". *IEEE Transactions on VLSI Systems*, pp. 508–511, 1994.

[90] Roy, K., Nag, S. "On Channel Architecture and Routability for FPGA's under Faulty Conditions". *Lecture Notes in Computer Science*, 1994.

[91] Roy, K., Nag, S. "On Routability for FPGA's under Faulty Conditions". *IEEE Transactions on Computers*, 1995.

[92] Roy, K., Prasad, S. "Power Dissipation Driven FPGA Place and Route under Delay Constraints". *Lecture Notes in Computer Science*, 1994.

[93] S. D. Brown, J. Rose e Z. G. Vranesic, R. J. Francis. *Field-Programmable Gate Arrays.* Kluwer Academic Publishers, 1992.

[94] Sait, S. M., Youssef, H. *VLSI Physical Design Automation - Theory and Practice.* IEEE Press, New York, 1995.

[95] Snir, M., Otto, S. W., et al, S. Huss-Lederman. *MPI The Complete Reference* . Scientific and Engineering Computation Series, MIT Press, Cambridge, Massachusetts, 1996.

[96] Snyder, T. L. "On minimal rectilinear Steiner Trees in all dimensions ". In *Prodeedings of the Sixth Annual ACM Symposium on Computacional Geometry*, pp. 311–320, 1990.

[97] Soukup, J. "Circuit Layout". In *Proc. of the IEEE*, v. 60, pp. 1281–1304, 1981.

[98] Soukup, J., Chow, W. "Set of test problems for the minimum length connection networks". *ACM/SIGMAP Newsletter*, v. 15, pp. 48–51, 1973.

[99] Sun, Y., Wang, T. C., Wong, C. K., Liu, C. L. "Routing for Symmetric FPGAs and FPICs". In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pp. 486–490, 1993.

[100] Swartz, J. S., Betz, V., Rose, J. "A Fast Routability-driven Router for FPGAs". In *Int. ACM/SIGDA Symp. Field-Programmable Gate Arrays*, 1998.

[101] Swartz, W., Sechen, C. "Timing Driven Placement for Large Standard Cell Circuits". In *Design Automation Conference*, pp. 211–215, 1995.

[102] Takahashi, H., Matsuyama, A. "An Approximate Solution for the Steiner Problem in Graphs". *Mathematica Japonica*, v. 24, pp. 573–577, 1980.

[103] Technologies, Lucent. *FPGA Data Book*, 1998.

[104] Tessier, R. "Negotiated A* routing for FPGAs". In *Fifth Canadian Workshop on Field-Programmable Devices*, 1998.

[105] Trimberger, S. M. *Field-Programmable Gate Array Technology.* Kluwer Academic Publishers, 1994.

[106] Tseng, B., Rose, J., Brown, S. "Using Architectural and CAD Interactions to improve FPGA Routing Architectures". In *First International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, pp. 3–8, 1992.

[107] Wilton, S.J.E. *Architectures and Algorithms for Field Programmable Gate Arrays with Embedded Memories.* PhD thesis, Universidade of Toronto, 1997.

[108] Wong, R. "A Dual Ascent Approach to Steiner Tree Problem on a Directed Graph ". *Mathematical Programming*, v. 28, pp. 271–287, 1984.

[109] Wu, Y.-L., Chang, D. "On the NP-Completeness of Regular 2-D FPGA Routing Architectures and a Novel Solution". In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pp. 362–366, 1994.

[110] Wu, Y.-L., Marek-Sadowska, M. "An Efficient Router for 2-D Field Programmable Gate Arrays". In *European Design and Test Conf.*, pp. 412–416, 1994.

[111] Wu, Y.-L., Marek-Sadowska, M. "Orthogonal Greedy Coupling - A new Optimization Approach to 2-D FPGA Routing". In *Design Automation Conference*, 1995.

[112] Wu, Y.-L., Tsukiyama, S., Marek-Sadowska, M. "Graph Based Analysis of FPGA Routing". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, v. 15, n. 1, pp. 33–44, 1996.

[113] Xilinx, . *The Programmable Gate Array Data Book.* Xilinx, Inc., 1994.

[114] Yang, S. *Logic Synthesis and Optimization Benchmarks, Version 3.0.* Technical report, Microelectronics Centre of North Carolina, 1991.

[115] Yang, Y., Wing, O. "An algorithm for the wiring problem". In *Digest of the IEEE International Symposium on Eletrical Networks*, pp. 14–15, 1971.

[116] Yang, Y., Wing, O. "Optimal and suboptimal solution algorithms for the wiring problem". *IEEE Transactions on Circuit Theory*, v. 19, pp. 508–510, 1972.

[117] Yang, Y., Wing, O. "On a multinet wiring problem". *IEEE Transactions on Circuit Theory*, v. 20, pp. 250–252, 1973.

[118] Zelikovsky, A. Z. "An 11/6 Approximation Algorithm for the Network Steiner Problem". *Algorithmica*, v. 9, pp. 463–470, 1993.