PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Algoritmo de Roteamento Maze para Dispositivos Programáveis FPGA

por

PAULO CÉSAR FURLANETTO MARQUES

Dissertação de mestrado submetida como requisito parcial
à obtenção do grau de Mestre em Ciência da Computação.

Prof. Fernando Gehm Moraes
Orientador

PORTO ALEGRE

2002

RESUMO

Esta dissertação apresenta o desenvolvimento de um algoritmo de roteamento para dispositivos programáveis do tipo FPGAs. A principal motivação para o desenvolvimento deste algoritmo é encontrar uma solução eficiente para o problema de roteamento dos dispositivos programáveis FPGA. O trabalho apresenta a arquitetura de roteamento dos dispositivos programáveis da Xilinx, pois esta arquitetura é a utilizada pelo algoritmo. Os algoritmos acadêmicos de roteamento e suas características principais são apresentados em tabelas comparativas. Uma vez apresentados os algoritmos de roteamento encontrados na literatura, é apresentado o desenvolvimento do algoritmo de roteamento, comparando-o com o estado-da-arte. Por fim, apresenta-se uma proposta de implementação paralela do algoritmo, com o objetivo de reduzir o tempo de processamento para rotear grandes circuitos.

Palavras-chave: algoritmo Maze, algoritmos de roteamento, arquitetura de roteamento, CAD, FPGA.

ABSTRACT

This work presents the development of a routing algorithm for FPGA devices. The main motivation to develop such work is to find an efficient solution to deal with the growing complexity of present FPGAs. The Xilinx architecture, based on vertical and horizontal routing channels, is used in the present work. The main features of related academic algorithms are studied, and a performance comparison between them is given. Once the state-of-the-art is presented, the development of our algorithm is detailed, comparing it to related works. Finally, a parallel implementation of the developed routing algorithm is proposed, aiming to reduce the consumed CPU time required to route large benchmarks.

Key words: Maze algorithm, routing algorithms, routing architectures, CAD, FPGA.

AGRADECIMENTOS

Ao meu orientador, Prof. Fernando Gehm Moraes, pela amizade, pelo companheirismo e pelo incentivo ao longo dos anos até a conclusão deste trabalho. Por estar sempre disposto a ajudar, mesmo quando tudo parecia perdido. Pela perseverança e por ter acreditado que este trabalho fosse concluído.

À minha esposa, Renata Frediani Morsch, pelo carinho e pelo incentivo para que este trabalho fosse concluído. Pela compreensão nos momentos em que estive ausente.

Aos meus pais, Abrelino e Teresinha Marques, que sempre me apoiaram em todos os momentos até a conclusão deste trabalho.

Aos professores e funcionários do PPGCC por acreditarem que fosse possível concluir o trabalho.

À CAPES e à PERTO por financiarem parte deste trabalho.

À todos que, direta ou indiretamente contribuíram para o desenvolvimento e conclusão deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ANSI	American National Standards Institute
ASIC	Application-Specific Integrated Circuit
CAD	Computer Aided Design
CLB	Configurable Logic Block
CLK	Clock Signal
CGE	Coarse Graph Expansion
DLL	Delay Locked Loops
FPGA	Field-Programmable Gate Array
GBP	Greedy Bin Packing
GRM	General Routing Matrix
IOB	Input / Output Block
LEDA	Library of Efficient Data Types and Algorithms
LUT	Look-up Table
MCNC	Microelectronics Center of North Carolina
OGC	Orthogonal Greedy Coupling
PIA	Programmable Interconnection Array
PIP	Programmable Interconnection Point
PLD	Programmable Logic Device
PLL	Phase-Locked Loop
RAM	Random Access Memory
VLSI	Very Large Circuit Integration
VPR	Versatile Place and Route

1 Introdução

Com o avanço da tecnologia VLSI tornou-se possível desenvolver circuitos digitais com baixo custo, elevado número de transistores e alta freqüência de funcionamento. Um dos dispositivos mais beneficiado com esta tecnologia foram os componentes programáveis do tipo FPGA. Estes dispositivos têm auxiliado na prototipação de circuitos digitais, pois são facilmente re-programados através da utilização de ferramentas de CAD [SHE95] e estão sendo também empregados em escala industrial, quando o volume de produção não é tão elevado (tipicamente 30.000 peças/ano). Com a utilização destes dispositivos programáveis o tempo necessário para que um novo produto seja lançado no mercado reduziu drasticamente. Até então, o projeto de um circuito integrado dedicado (*ASIC*) levava vários meses para ser concluído.

Em um projeto eletrônico, três funções custo devem ser atendidas: minimizar o atraso nas redes, reduzir a área ocupada e reduzir a potência consumida pelo circuito. Cabe salientar que não é possível obter uma solução que satisfaça estes três compromissos, por serem dependentes uns dos outros. Por exemplo, ao tentarmos reduzir o atraso, a área ocupada tende a crescer, devido à necessidade de inserirmos um maior número de *buffers* ou pela utilização de transistores maiores. Para que estes problemas sejam resolvidos, os projetistas precisam desenvolver novas técnicas de mapeamento [CON94], posicionamento [EBE95] e roteamento eficientes, como pode ser observado em [HUR00] e em [CHA00], onde é proposto um algoritmo de roteamento paralelo.

Neste trabalho é estudado o problema de roteamento em dispositivos FPGA, com arquitetura de canais de roteamento horizontais e verticais [ROS90]. A escolha da arquitetura foi feita com base em duas observações: (*i*) é a tendência observada na indústria, como por exemplo, pelos fabricantes Xilinx e Altera ([XIL02] e [ALT02]); (*ii*) está sendo largamente utilizada no estudo de arquiteturas reconfiguráveis e projetos conjuntos de *hardware* e *software* (*co-design*).

Desenvolver um algoritmo de roteamento é uma tarefa árdua, considerando-se que os FPGAs possuem recursos de roteamento fixos e limitados [CHA00]. Os recursos de roteamento, nos dispositivos programáveis FPGA, de uma forma geral, estão dispostos em linhas e colunas, formando uma matriz.

Este trabalho tem como objetivo principal o desenvolvimento de um algoritmo para o roteamento de circuitos programáveis do tipo FPGA. Este roteador utiliza como base um algoritmo da classe *Maze Router*, adaptado para dispositivos FPGAs. O algoritmo proposto por Lee [LEE61] e Moore [MOO59] é utilizado para encontrar o caminho mínimo entre dois pontos em um grafo.

O Capítulo 2 apresenta uma revisão da arquitetura de roteamento de dispositivos programáveis FPGA: "*row-based*", "*sea-of-gates*", PLD hierárquico e Matriz Simétrica. As arquiteturas de roteamento do tipo "*row-based*", "*sea-of-gates*" e PLD hierárquico são apresentadas de forma sucinta. Este capítulo detém-se na arquitetura de roteamento do tipo Matriz Simétrica, pois para este tipo de dispositivo é desenvolvido o algoritmo. Para esta arquitetura são estudados os dispositivos Xilinx XC4000 e Xilinx Virtex. Os dispositivos Virtex possuem duas inovações importantes: o bloco lógico é duplicado em relação ao bloco lógico da família 4000 e os recursos de roteamento são utilizados na forma de "*cluster*", aumentando-se assim o roteamento local. Com o aumento do roteamento local há redução no atraso de propagação dos sinais elétricos [MAR00a].

O Capítulo 3 apresenta o algoritmo *Maze*, o qual é utilizado como base para a

implementação do presente trabalho, assim como importantes variações aplicadas a este algoritmo visando melhorar seu desempenho. Neste Capitulo é feita uma revisão do estado da arte dos algoritmos de roteamento propostos pela comunidade científica. Esses algoritmos são analisados levando-se em consideração o seu algoritmo de roteamento e as soluções encontradas a fim de facilitar o roteamento de circuitos digitais em FPGAs. É apresentada também comparação de desempenho entre os algoritmos de roteamento, tendo por métrica a quantidade de trilhas utilizadas para que os circuitos de teste [YAN91] sejam completamente roteados.

O Capítulo 4 apresenta a contribuição principal deste trabalho, que é o desenvolvimento do algoritmo de roteamento. Neste Capítulo são detalhadas as modelagens utilizadas para converter a estrutura do FPGA em uma estrutura de dados e o algoritmo de roteamento propriamente dito. A implementação é dividida em duas partes: propagação da frente de onda e a retropropagação. Na primeira parte, a partir de uma rede do circuito digital propaga-se uma frente de onda do pino de conexão inicial da rede até que um pino de conexão qualquer da mesma rede seja encontrado. A segunda parte é responsável pela fixação do nome da rede em todas as trilhas necessárias para que a conexão entre os pinos seja realizada. São também apresentados os resultados obtidos com a utilização do algoritmo sobre os circuitos de teste, comparando-os com os resultados obtidos com os roteadores acadêmicos.

O Capítulo 5 apresenta uma proposta de implementação paralela do algoritmo seqüencial de roteamento apresentado no Capitulo 4. Tal proposta de implementação paralela é baseada na observação da localidade das redes. Nos circuitos de teste foi possível notar que a maior parte das redes possui seus pinos de conexão muito próximos. Outro detalhe observado é que a grande maioria das redes possui *fanout* igual a 2, ou seja, a rede possui a origem e apenas um destino. Isso não significa que esta rede tem comprimento reduzido, porém o roteamento é simplificado pelo menor número de propagações necessárias. Propõe-se que a paralelização seja efetuada em uma rede de estações de trabalho, onde cada estação é responsável pelo roteamento de uma parte do circuito. Sendo assim, o algoritmo seqüencial pode ser distribuído em uma rede de estações de trabalho, reduzindo o tempo de processamento.

Finalmente, o Capítulo 6 apresenta as conclusões do presente trabalho.

2 Arquitetura de roteamento em FPGAs Xilinx

Neste Capítulo é realizada uma breve apresentação das arquiteturas de roteamento dos dispositivos programáveis FPGA Xilinx das famílias 4000 e Virtex.

Os dispositivos programáveis FPGA, de uma forma geral, são compostos de blocos lógicos, matrizes de chaves (*Switch Box*) e recursos de roteamento denominados de trilhas.

A Figura 1 apresenta as quatro principais arquiteturas de roteamento utilizadas em dispositivos programáveis FPGA [ROS93]: matriz simétrica, PLD hierárquico, mar de portas (*Sea-of-Gates*) e o *row-based*.

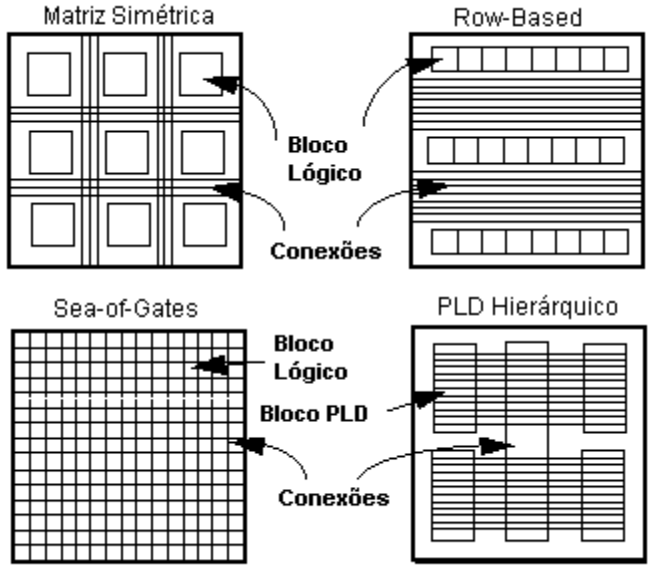


Figura 1 – Arquiteturas de dispositivos programáveis FPGA.

A arquitetura do tipo *sea-of-gates* é um dispositivo preenchido completamente por transistores ou blocos lógicas de baixa complexidade (tipicamente uma porta lógica *nand*). A vantagem desta arquitetura é a alta densidade de portas lógicas implementadas por unidade de área. Porém, como não há uma área dedicada para roteamento, é necessário que o mesmo seja feito sobre as células, muitas vezes inutilizando-as.

Os dispositivos com arquitetura do tipo *row-based* possuem linhas de blocos lógicos na horizontal, sendo a área de roteamento feita entre estas linhas. Observa-se que não há área de roteamento vertical dedicada, o que pode acarretar sacrifício de blocos lógicos para a passagem de conexão entre linhas não adjacentes.

As arquiteturas *row-based* e *sea-of-gates* são oriundas das metodologias de projeto de ASICs, *standard-cells* e *gate-array*. Ambas não lograram sucesso comercial em FPGAs devido às limitações impostas pelos escassos recursos de roteamento.

O dispositivo que tem a arquitetura do tipo PLD hierárquico é constituído por uma matriz de blocos lógicos, denominados *Logic Arrays Blocks*, sendo os mesmos interligados pelo recurso de roteamento conhecido como Matriz Programável de Interconexão (PIA). Esse tipo de dispositivo é dito hierárquico porque os blocos lógicos podem ser agrupados entre si, criando assim as Macro-

Células, sendo esse o nível 1. O nível 2 é formado pela conexão das Macro-Células entre si através da PIA. Os FPGAs com arquitetura do tipo PLD Hierárquico podem ser encontrados nos dispositivos fabricados pela Altera. O roteamento é bastante simplificado neste tipo de arquitetura, pois a cada nível de hierarquia utiliza-se basicamente uma rede do tipo *cross-bar*.

A arquitetura do tipo Matriz Simétrica é flexível no requisito roteamento [THA94], pois possui canais verticais e horizontais de roteamento. Deve-se observar que, para a área do circuito não ser completamente dominada pelo roteamento, a granularidade do bloco lógico deve ser alta [MAR00a]. Como veremos adiante, a família Xilinx 4000 possui 3 LUTs, sendo duas de 4 entradas e uma de 3 entradas, e a família VIRTEX possui 4 LUTs de 4 entradas.

O presente trabalho utiliza a arquitetura do tipo Matriz Simétrica por esta estar sendo largamente utilizada na indústria, e por apresentar desafios de implementação tanto de complexidade como de tempo de processamento.

A Seção 2.1 apresenta a arquitetura de blocos lógicos e roteamento da família Xilinx 4000 e a Seção 2.2 a família Virtex.

2.1 Família 4000

Os dispositivos FPGA Xilinx da família 4000 foram lançados em 1994 [XIL02].

A lógica combinacional que define a funcionalidade do circuito é armazenada em tabelas-verdade implementadas em hardware, denominadas LUT, de *Look-up Table*. Uma LUT de K entradas é uma combinação de uma memória de K bits e um multiplexador de K:1, que pode implementar qualquer função Booleana de K variáveis. As K entradas são usadas como endereço para uma memória de 2^K x 1-bit, que armazena a tabela verdade da função. Na Figura 2 apresentamos como a função $x = ab + \bar{b}c$ é implementada em uma LUT de 4 entradas. A LUT é constituída de 16 bits de memória ($2^{4\ \text{entradas}}$) e um multiplexador 16x1. Durante a fase de *configuração* do circuito, armazenamos a tabela-verdade nos 16 bits de memória. Durante a fase de *processamento*, cada uma das variáveis controla uma das entradas do multiplexador (observar que a entrada *d* fica com valor constante zero).

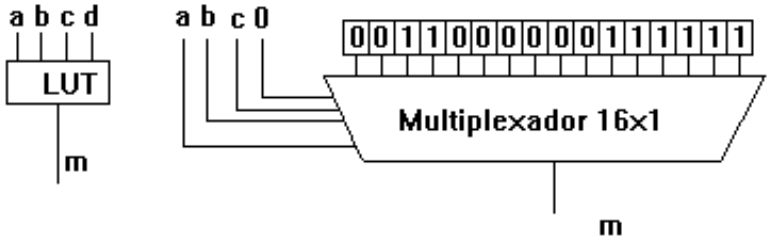
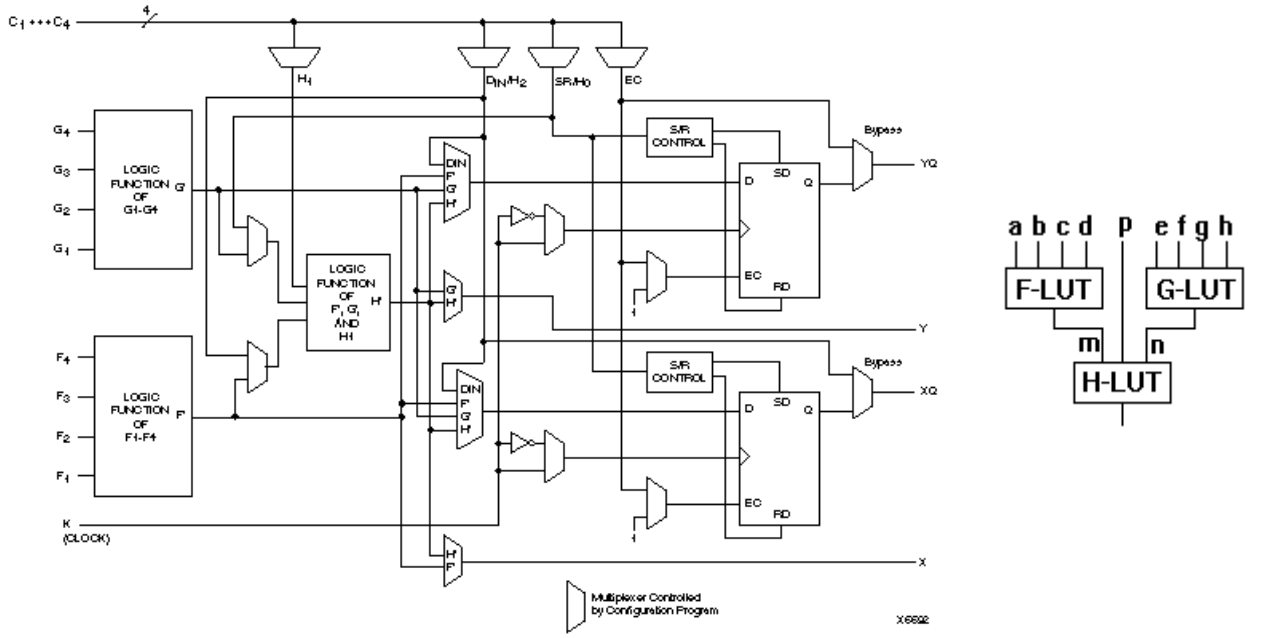


Figura 2 – Exemplo de LUT implementando a função $x = ab + \bar{b}c$ (x= Σ(2,3,10,11,12,13,14,15)).

O bloco lógico da família 4000 é constituído por duas LUTs de 4 entradas, denominadas de F-LUT e G-LUT respectivamente, e uma LUT de 3 entradas (H-LUT). A Figura 3a ilustra a constituição do bloco lógico da família 4000. As saídas das LUTs são denominadas F', H' e G', sendo que F' utiliza a saída X e G' usa a saída Y, através de multiplexadores. O sinal H' pode utilizar tanto as saídas X e Y. Todas as saídas podem ser diretas, complementadas ou bufferizadas, característica importante para a melhora do desempenho elétrico.

As entradas C_1, C_2, C_3 e C_4 são utilizadas para controle, denominadas no interior do bloco lógico como H_1, D_{in}, S/R e EC. A H-LUT recebe três entradas: F', G' e H1. O sinal D_{in}, faz com que as saídas passem por registradores (flip-flop tipo D) e estejam disponíveis nas saídas YQ e XQ. O pino S/R serve para setar ou resetar os registradores durante a inicialização ou reconfiguração ou ainda quando uma rede dedicada de *reset* é ativada. Os dois registradores possuem um sinal de clock comum (K) que pode ser habilitado ou não, dependendo da entrada EC (*Enable Clock*).



(a) representação completa do bloco lógico (b) representação simplificada

Figura 3 - Bloco lógico Xilinx XC4000.

A Figura 3b, apresenta de forma simplificada o bloco lógico da família 4000. Este bloco lógico pode implementar: (*i*) duas funções booleanas de quatro variáveis, utilizando as duas LUTs de 4 entradas; (*ii*) uma função de 5 variáveis, utilizando a F-LUT e a G-LUT para a implementação dos minitermos da função, sendo a H-LUT controlada pela quinta variável; (*iii*) algumas funções de 6 até 9 variáveis.

Cabe salientar que este bloco lógico, apesar de aceitar 9 entradas, não equivale a uma LUT com 9 entradas. Para isto seriam necessários 512 bits para armazenar todos os minitermos possíveis, tornando inviável a área de silício necessária e o tempo de reconfiguração. A implementação de funções com um maior número de variáveis reduz o número de blocos lógicos utilizados no circuito, em conseqüência disso a densidade aumenta e o atraso é reduzido. Apesar da aparente vantagem de aumentar a complexidade dos blocos lógicos, as ferramentas de síntese lógica não utilizam plenamente estes recursos, tornando o FPGA sub-utilizado.

Estudos acadêmicos demonstram que o melhor compromisso área *versus* desempenho é obtido com LUTs tendo 4 entradas [ROS93]. A aplicação deste estudo pode ser observado nos dispositivos comerciais, os quais contém LUTs de 4 entradas (Xilinx 4000 e Virtex, Altera Apex). O número de LUTs por bloco lógico também é estudado em trabalhos acadêmicos. [MAR00a] propõe que este número fique entre 8 e 10. A família 4000 possui 3 LUTs por bloco lógico (CLB), a família VIRTEX 4 LUTs por CLB e a família VIRTEX-II 8 LUTs por CLB.

2.1.1 Arquitetura de Roteamento

Os recursos de roteamento dos dispositivos da família 4000 possuem 4 tipos de segmentos de fio: linhas de propósito geral (*Single-length lines*), linhas de tamanho duplo (*Double- length lines*), linhas longas (*long lines*), linhas globais e matrizes de chaves. A família XC4000EX possui além desses recursos, as linhas quádruplas. Na Figura 4, são apresentados os recursos de roteamento, sendo as partes com o fundo cinza pertencentes apenas ao XC4000EX.

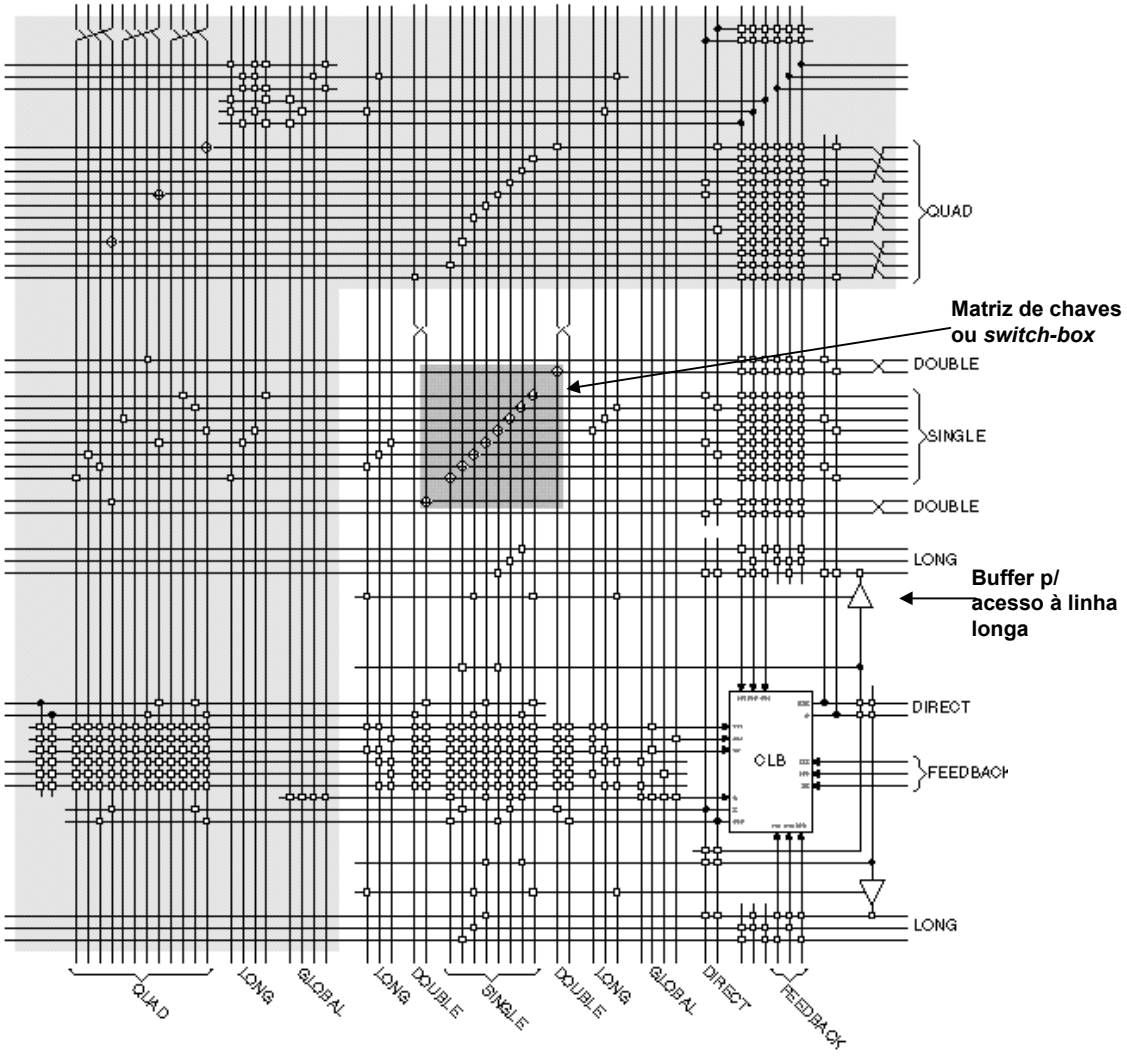


Figura 4 – Detalhe dos recursos de roteamento da Xilinx XC4000.

As *linhas longas* percorrem todo o comprimento ou largura do dispositivo, sendo utilizadas para transmitir sinais globais do circuito, diminuindo assim o *skew*[1] destes sinais [ZHU97]. Cada linha longa possui uma chave programável no centro, a qual separa a linha em duas partes independentes. As entradas e as saídas dos blocos lógicos podem conectar-se às linhas longas. As saídas do bloco que utilizarem este recurso podem ser realizadas através de *buffers tri-state* ou das linhas de propósito geral.

[1] Diferença no tempo de propagação do sinal entre sua origem e seus destinos. Em linhas longas o sinal pode chegar nos destinos em tempos diferentes, comprometendo o comportamento de circuitos síncronos. Este parâmetro pode ser traduzido como "escorregamento" do sinal.

Por outro lado, as *linhas globais* atravessam toda a extensão do dispositivo na vertical possuindo apenas chaves de conexão com as linhas de propósito geral (*single length lines*).

As *linhas de tamanho duplo* são duas linhas paralelas, tendo o dobro do tamanho das linhas de propósito geral. Essas linhas são conectadas a matrizes de chaves alternadas, ou seja, a cada dois blocos lógicos. Cada bloco lógico é associado a 4 linhas horizontais e 4 linhas verticais de tamanho duplo, podendo assim haver uma distribuição uniforme do sinal em distâncias intermediárias, com flexibilidade de roteamento.

As *linhas de propósito geral* estão distribuídas na vertical e na horizontal, utilizadas principalmente para conectar blocos lógicos adjacentes ou muito próximos. Quando a distância entre dois blocos lógicos é grande, esse tipo de recurso não é utilizado em razão do aumento do atraso causado pelo grande número de chaves utilizadas para realizar essa conexão.

A *matriz de chaves* (Figura 5) consiste de 10 chaves de interconexão, sendo que cada uma possui 6 pontos de interconexão programável (PIP). O PIP é um conjunto de transistores de passagem controlado por células de memória RAM, configuradas pelo CAD de síntese física. Essa matriz de chaves conecta as linhas de propósito geral e de tamanho duplo entre si. Observa-se que cada ponto de interconexão é muito caro em termos de área de silício, pois possui 6 chaves (transistores) e ao menos 6 transistores para armazenar o estado de cada uma das chaves. Com o objetivo de reduzir a área de silício, a matriz de chaves limita a possibilidade de conexões. Na Figura 5 observamos que cada conexão que entra na matriz de chaves só pode se conectar com outros três segmentos de fio, um em cada borda. O ideal seria uma conexão completa, mas o custo seria proibitivo.

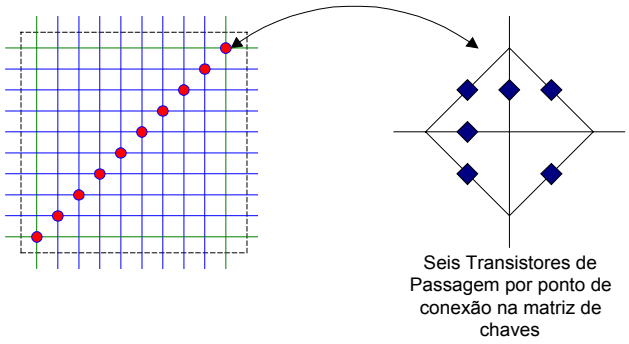


Figura 5 – Matriz de chaves da família XC4000.

A Tabela 1 e a Figura 6 resumem os recursos de roteamento da família 4000. Observa-se, por exemplo, que cada bloco lógico da família XC4000E pode utilizar 24 linhas de roteamento vertical e 18 linhas de roteamento horizontal.

Tabela 1 – Recursos de roteamento por bloco lógico XC4000.

	XC4000E		XC4000EX	
	Vertical	**Horizontal**	**Vertical**	**Horizontal**
Singles	8	8	8	8
Doubles	4	4	4	4
Quads	0	0	12	12
Longlines	6	6	10	10
Direct Conects	0	0	2	2
Globals	4	0	8	0
Carry Logic	2	0	1	0
Total	24	18	45	32

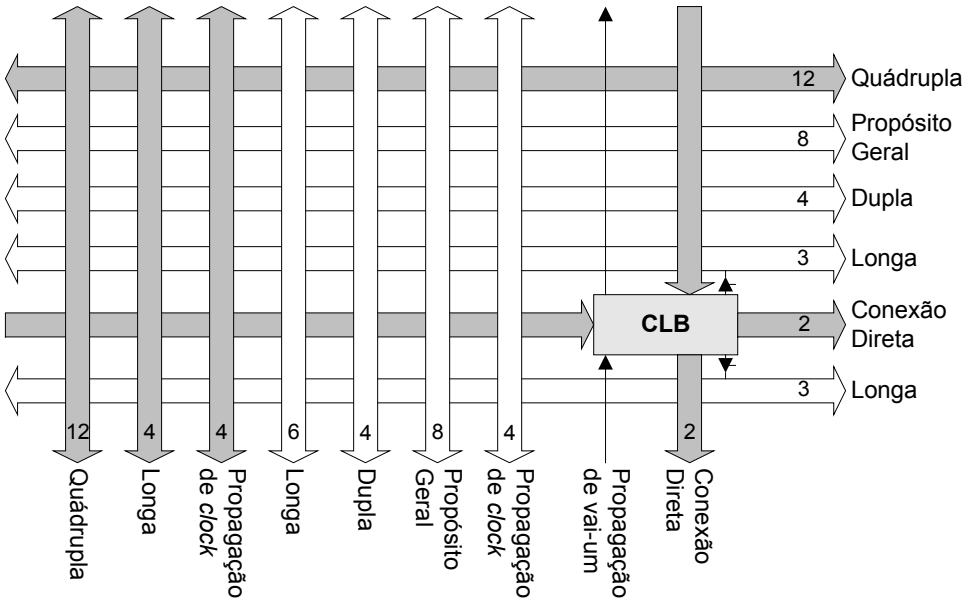


Figura 6 – Resumo dos recursos de roteamento.

Para definir a flexibilidade de roteamento de um FPGA são utilizados dois parâmetros: f_c e f_s [ROS90]. O parâmetro f_s define a flexibilidade da matriz de chaves, indicando o número total de conexões oferecidas a cada segmento de fio conectado à matriz. A Figura 7 ilustra duas situações distintas para o parâmetro f_s. A primeira situação, $f_s=\infty$, representa o máximo de flexibilidade para o roteamento, mas não uma situação real para os FPGAs atuais. A segunda situação, $f_s=3$, representa uma arquitetura real, impondo restrições aos algoritmos de roteamento. Para a família 4000 temos $f_s=3$.

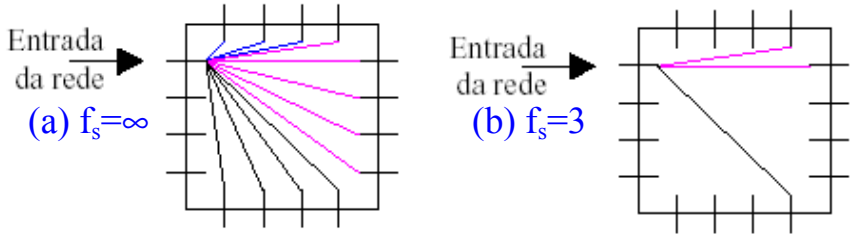


Figura 7 – Matriz de chaves com $f_s = \infty$ e $f_s = 3$.

A flexibilidade dos blocos de conexão é indicada por f_c, representado o número total de trilhas que os pinos de um bloco lógico podem acessar. Normalmente, este parâmetro é igual à largura do canal de roteamento. Para a família 4000 temos f_c igual à largura do canal de roteamento.

2.2 Arquitetura Virtex

Os dispositivos FPGA Xilinx da família Virtex foram lançados em 1998 [XIL02]. Esta família possui a particularidade de possuir uma arquitetura baseada em *clusters* [MAR00a], com o objetivo de minimizar o atraso de propagação dos sinais elétricos do circuito e de diminuir a área ocupada pelo circuito no dispositivo FPGA. O *cluster* é um conjunto de blocos lógicos interligados localmente por elementos de conexão de alta velocidade.

A Figura 8 apresenta uma visão geral da arquitetura da família Virtex, a qual é composta por Blocos Lógicos Programáveis (CLBs), Blocos de Entrada/Saída (IOBs) e Blocos de Memória

RAM (BRAMs). A ligação das CLBs e memórias aos blocos IOBs é feita através do *VersaRing*, que é composto por uma matriz de chaves de alta velocidade.

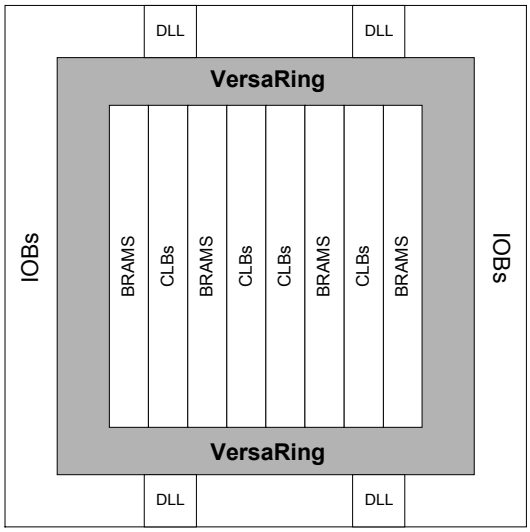


Figura 8 – Visão geral do dispositivo programável FPGA da família Virtex.

O bloco lógico da família Virtex, apresentado na Figura 9, é composto por dois *slices* idênticos, muito similares ao bloco lógico da família 4000. Observa-se que a estrutura interna do *slice* é simplificada em relação à CLB da família 4000, pois a LUT de 3 entradas foi suprimida. Cada *slice* contém 2 LUTs de quatro entradas, 2 flip-flops e recursos de propagação rápida de *carry*, para implementar operadores aritméticos.

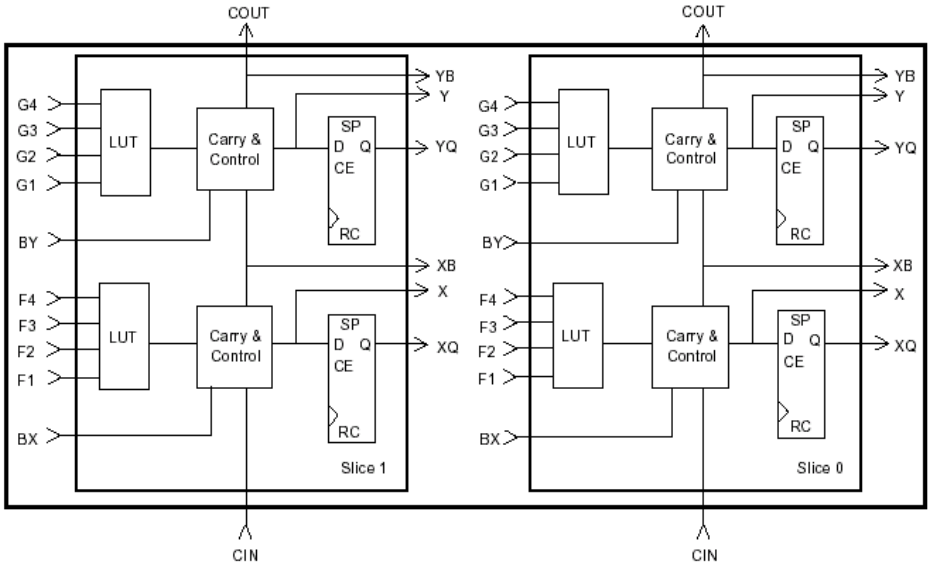


Figura 9 – Bloco lógico da família Virtex (2 *slices*).

2.2.1 Arquitetura de Roteamento

A arquitetura de roteamento da família Virtex pode ser dividida em 2 partes: matriz programável de roteamento, e linhas globais de distribuição de sinal de *clock*.

- Matriz Programável de Roteamento

A matriz programável de roteamento provê a conexão entre os CLBs, interligando o *roteamento local* com o *roteamento de propósito geral*. A Figura 10 mostra as conexões possíveis desse recurso de roteamento. Além do *roteamento local* e do *roteamento de propósito geral*, temos ainda o *roteamento de entrada/saída* e *roteamento dedicado*.

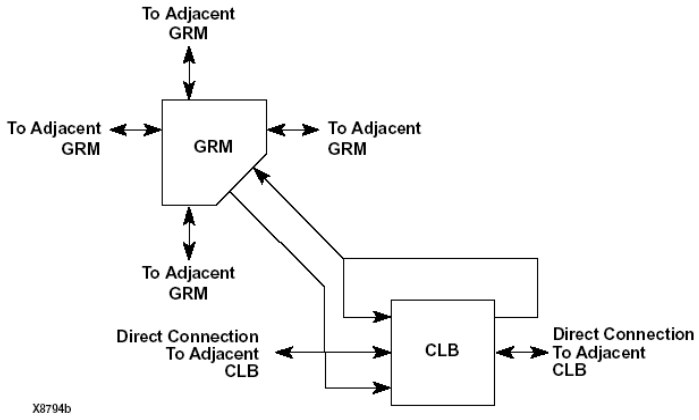


Figura 10 – Roteamento local da família Virtex.

O ***roteamento de propósito geral***, apresentado na Figura 11 é composto por 24 linhas de comprimento simples que interligam as GRMs (matriz geral de roteamento) adjacentes em quatro direções [LIM01]), 12 linhas de comprimento seis, denominadas *hex lines* (lembrar que na família 4000 havia linhas de comprimento 2 e 4, e que nesta família temos comprimento 6) e 12 linhas longas horizontais e verticais que percorrem todo o dispositivo. As *hex lines* conectam CLBs a intervalos de 3 em 3 CLBs. A Figura 12 [LIM01] ilustra 3 CLBs conectadas pela mesma *hex line*.

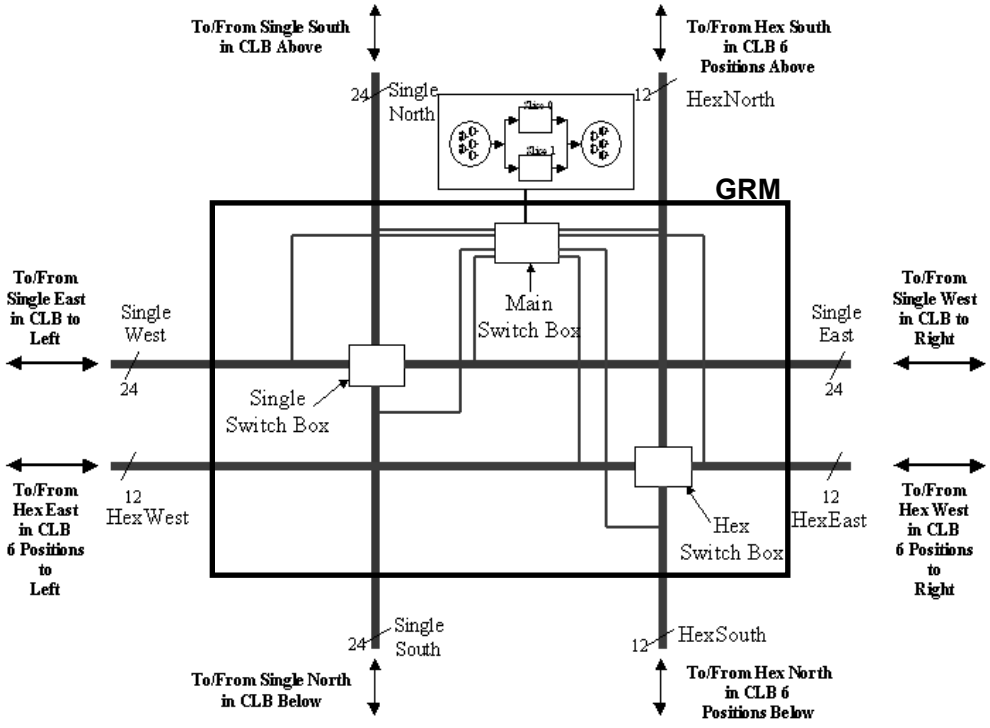


Figura 11 – Detalhamento das linhas de conexão associadas à matriz geral de roteamento (GRM).

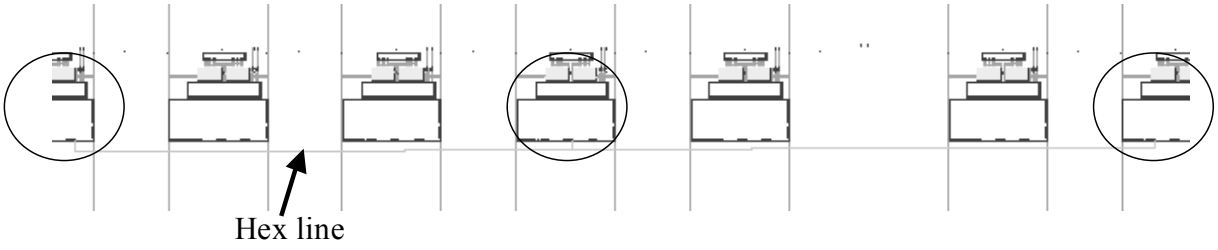


Figura 12 – Representação das linhas longas, as quais conectam de 3 em 3 CLBs.

O ***roteamento local***, também denominado como *VersaBlock*, realiza as conexões entre os blocos lógicos e a matriz geral de roteamento (GRM). O objetivo deste roteamento é realizar conexões de alta velocidade entre os *slices*, evitando que essas conexões passem pela matriz geral de roteamento eliminando o atraso produzido pela mesma, diminuindo assim o atraso de propagação do sinal. Há também um certo número de segmentos de fio que conectam diretamente CLBs vizinhas.

A Figura 13 ilustra parcialmente um roteamento feito em um dispositivo Virtex. O roteamento local, ou *VersaBlock*, é feito pelos multiplexadores de entrada e saída. A matriz geral de roteamento, GRM, é representada pelo retângulo *switch box*. A área cinza contém o roteamento de propósito geral.

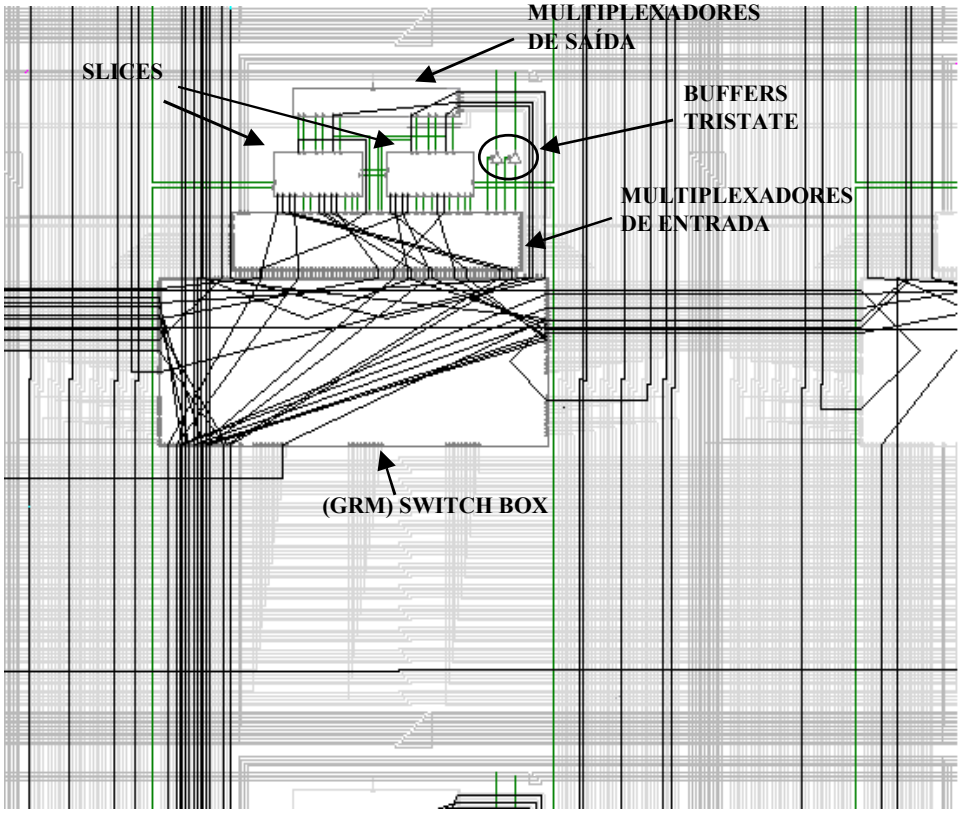


Figura 13 – Exemplo de circuito roteado da família Virtex.

Há 13 multiplexadores de entrada por CLB, os quais incluem as entradas F1-F4, G1-G4, CLK, SR, etc. Cada entrada possui um multiplexador associado, o que determina qual sinal proveniente da GRM irá alimentar os circuitos do *slice.* Há também 8 multiplexadores de saída por CLB.

O **roteamento de *entrada/saída***, como já comentado, é feito pela estrutura denominada *VersaRing*. O **roteamento dedicado** refere-se ao roteamento de *buffers tri-state*, propagação de *carry* e conexão dos DLLs.

- Distribuição Global de Clock

A Figura 14 representa como é realizada a distribuição global dos sinais de *clock*. Este tipo de distribuição se assemelha a uma espinha de peixe e também é utilizada em microprocessadores.

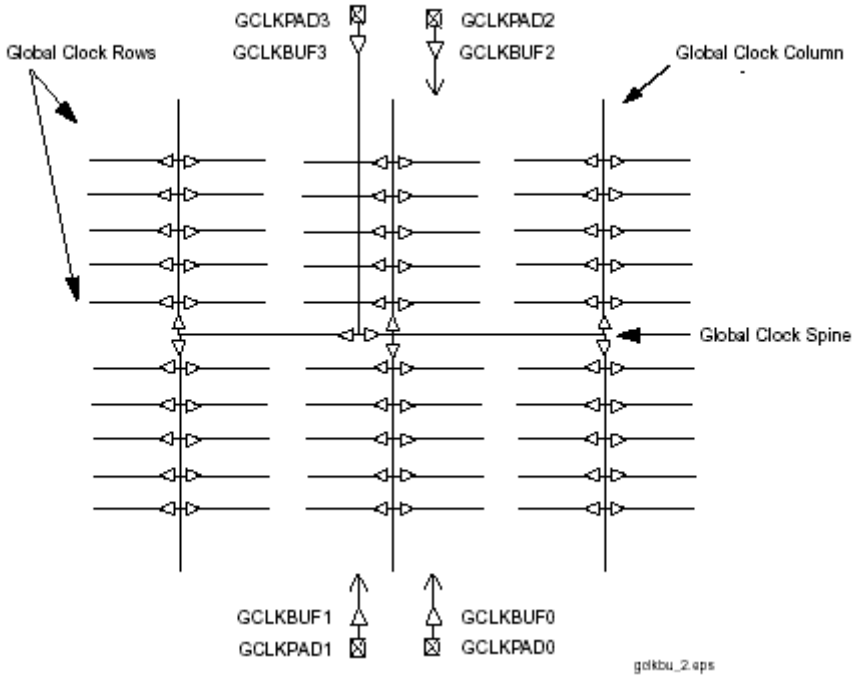


Figura 14 – Distribuição de *Clock na família VIRTEX.*

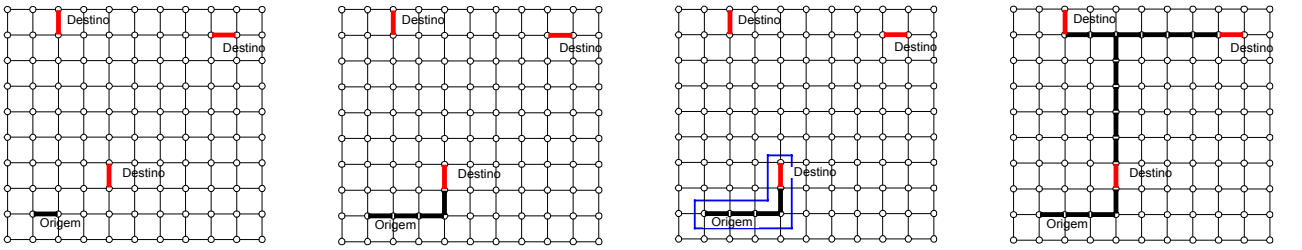
Há 4 entradas externas que podem ser utilizadas para sinal de *clock*, assim como 4 circuitos dedicados para sincronismo de *clock* (DLLs). Este sistema garante uma distribuição uniforme de sinais de sincronismo no interior do FPGA, evitando assim atrasos relativos (*skew*) em decorrência dos segmentos de fio longos que devem ser utilizados para que o sinal seja distribuído.

3 Roteamento para Arquiteturas Baseadas em Matriz Simétrica de Blocos Lógicos

O objetivo deste Capítulo é fornecer ao leitor uma visão geral dos algoritmos utilizados para a realização do roteamento detalhado para FPGAs. O algoritmo de base utilizado nos atuais sistemas de roteamento é o *Maze*, descrito na Seção 3.1. Após o detalhamento desse algoritmo de roteamento, serão apresentados alguns roteadores acadêmicos (Seção 3.2). Na Seção 3.3 é apresentada a comparação de desempenho entre os mesmos.

3.1 Algoritmo Maze

Lee [LEE61] e Moore [MOO59] propuseram um algoritmo para realizar o roteamento de redes com dois terminais, visando obter o menor caminho entre os mesmos. Desde então, o algoritmo básico vem sendo incrementado de forma a otimizar a execução (velocidade) e otimizar a quantidade de memória utilizada pelo mesmo. O algoritmo proposto por Lee e Moore e as versões derivadas desta implementação compõem a classe dos algoritmos Maze. O exemplo de uma rede sendo roteada através do algoritmo do Lee é ilustrada na Figura 15.



(a) grafo que representa o FPGA. Origem e destinos são arestas

(b) conexão entre a origem e o destino mais próximo

(c) propagação para o próximo destino através da conexão

(d) roteamento final para os 4 pontos da rede

Figura 15 – Rede sendo roteada pelo algoritmo do Lee.

A classe dos algoritmos Maze é utilizada para encontrar o menor caminho entre um par de pontos, chamados de origem (*source*) e destino(s) (*target(s)*), em uma matriz retangular ou quadrada. O processo de procura ocorre a partir da propagação de uma frente de onda da origem até que o destino seja encontrado. Esta frente de onda possui níveis distintos, que indicam a distância que a origem se encontra do destino. O nível inicial (origem) é 0, todos os pontos que fazem fronteira com a origem recebem o nível 1, e assim sucessivamente, até o destino. O objetivo principal de colocar níveis nas frentes de onda é garantir que o menor caminho seja o escolhido entre a origem e o destino.

Quando o destino é atingido pela frente de onda inicia-se então o processo de retropropagação, ou seja, o caminho encontrado na primeira fase é realizado de forma inversa, do destino para a origem, procurando sempre um nível inferior (frente de onda da propagação) até que o nível 0 seja encontrado. Assim é possível garantir que o menor caminho entre os dois pontos foi encontrado.

O algoritmo procura o destino simetricamente em todas as direções, usando a técnica de procura *breadth-first search*. Se o caminho entre a origem e o destino existir, é possível garantir que o caminho encontrado é o menor caminho entre os dois pontos. O pior caso que pode acontecer é de

a origem estar no centro da grade e o destino estar localizado na borda. Para que o destino seja atingido todos os pontos da grade terão de ser visitados.

Supondo que o algoritmo não conseguiu rotear todas as redes do circuito, é utilizada a técnica de "*rip-up & re-route*". Essa técnica consiste em re-incializar toda a estrutura de dados que representa o FPGA, re-ordenar as redes e executar uma nova tentativa de roteamento. A ordenação das redes segue a seguinte norma: as redes que não foram roteadas são colocadas no topo da lista de redes para que se obtenha sucesso no roteamento das mesmas em uma nova tentativa, e as redes que foram roteadas com sucesso na iteração anterior completam a lista.

O algoritmo do tipo Maze proposto por Lee, possui algumas variações. Hadlock [HAD77] propõe limitar o espaço de busca na grade pela definição da área que envolve todos os vértices da rede. A propagação só ocorrerá nos vértices que estão contidos nessa região. Consequentemente, quando a frente de onda encontrar uma borda qualquer dessa região não será mais propagada. No pior caso a complexidade em relação ao tempo de roteamento e o espaço utilizado é O(h . w), onde h e w são as dimensões do FPGA.

Soukup [SOU78] propõe diminuir o tempo de procura do destino, que no algoritmo original é muito alto. Este algoritmo consiste em propagar frente de onda em apenas uma direção até que o destino seja encontrado ou encontre um obstáculo, ou seja, um canal já roteado. Supondo que o destino tenha sido encontrado é feita a retropropagação para que a conexão seja realizada, se o destino não tiver sido encontrado. Então, a frente de onda continua a sua propagação em uma nova direção. Essa estratégia diminui o tempo de procura do destino, em compensação não consegue garantir que o menor caminho entre a origem e o destino seja encontrado. No pior caso a complexidade em relação ao tempo de roteamento e o espaço utilizado é O(h . w), onde h e w são as dimensões do FPGA.

Mais recentemente foi apresentada uma nova versão para o algoritmo Maze [HUR00]. O objetivo principal deste algoritmo é minimizar os componentes *RC* das redes, alcançando com isto um melhor desempenho elétrico. O autor relata que o desempenho do algoritmo é até 300 vezes superior ao algoritmo Maze clássico, por sua operação se basear em princípios de roteamento global.

3.2 Algoritmos e softwares de roteamento acadêmicos

3.2.1 CGE

O algoritmo básico do CGE (Coarse Graph Expansion) [BRO92] é dividido em duas fases: roteamento global e detalhado. O roteamento global é uma adaptação de um roteador global para projeto de ASICs, modelando as conexões como grafos. Nessa fase, o algoritmo expande o grafo inicial indicando o início e o fim da rede a ser roteada. A Figura 16 ilustra o FPGA modelado como um grafo, e à direita a seqüência de elementos do FPGA que são necessários para rotear a rede. Observa-se que a rede parte de um bloco lógico (L), passa por duas regiões de roteamento (C) e uma *switch box*, chegando ao seu destino, em outro bloco lógico (L).

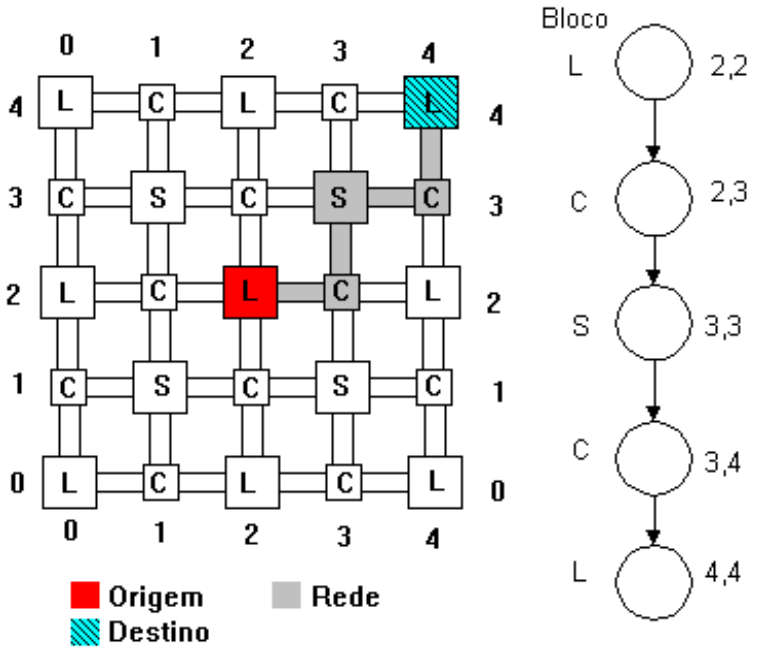
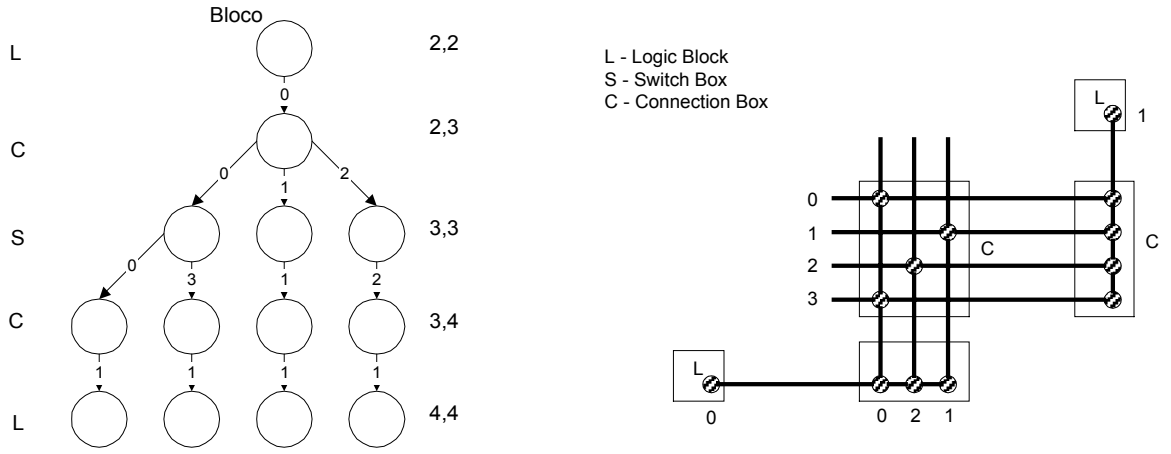


Figura 16 – Grafo criado no roteamento global (L: Bloco Lógico, C: Bloco de Conexão, S: Matriz de Chaves).

O roteamento detalhado é dividido em duas partes, expansão do grafo e realização da conexão. A primeira etapa expande o grafo gerado pelo roteamento global e armazena o número de caminhos possíveis para cada rede. A Figura 17a mostra o grafo expandido. Nesta figura observa-se que a região de roteamento conectada ao bloco lógico L(2,2) possui apenas três trilhas livres, enquanto que e a região de roteamento conectada ao bloco lógico L(4,4) possui quatro trilhas livres. A representação física deste grafo é apresentada na figura Figura 17b.



(a) Grafo expandido pela primeira fase do roteamento detalhado (b) Representação física do grafo

Figura 17 –Primeira fase do roteamento detalhado.

Na segunda fase, todas as possibilidades de conexão são colocadas em uma lista e baseado em uma função custo com o objetivo de reduzir o atraso, o algoritmo procura a melhor conexão. A Figura 18 apresenta um diagrama de blocos do *software* CGE.

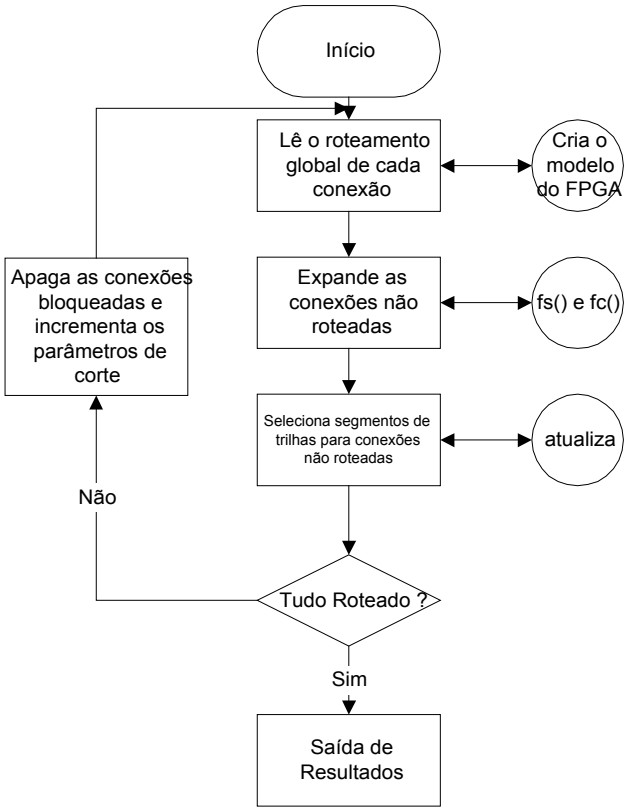


Figura 18 – Diagrama de blocos do software CGE.

3.2.2 Sega

O software SEGA [LEM97] é uma variação do *software* CGE. A diferença está na função custo, que leva em consideração os segmentos de fio longos para minimizar o atraso. Para a utilização deste software foi criado um método de roteamento global de dois estágios, seguido pelo roteamento detalhado.

Os parâmetros de entrada do SEGA compreendem uma lista de redes do circuito a ser roteado (*netlist*), e um arquivo que descreve a arquitetura do dispositivo. A partir destes dois arquivos é criado um grafo para cada rede.

O roteamento global utilizado é uma adaptação do algoritmo de roteamento global LocusRoute [ROS90]. Este roteador global separa o problema do roteamento global em dois sub-problemas distintos: (*i*) balancear a densidade dos canais de roteamento; (*ii*) atribuir um caminho específico para cada conexão. Para solucionar o primeiro sub-problema as redes multi-ponto são divididas em redes de conexão simples, ou seja, apenas entre dois pontos e depois se encontra a menor conexão possível, solução do segundo sub-problema. O objetivo principal deste algoritmo é distribuir o número de redes que passam por um determinado canal, a fim de obter uma distribuição uniforme das redes nos canais de roteamento. Os segmentos longos de fio são utilizados quando a distância entre blocos lógicos for grande.

O roteamento detalhado, este executado pelo SEGA, é composto de duas fases distintas. Na primeira fase são enumeradas todas as possibilidades para rotear cada rede do circuito. A segunda fase examina todas as alternativas de conexão e decide qual deve ser implementada, tendo como base uma função custo com quatro termos, que foi calculada enquanto a primeira fase era executada.

A Figura 19 apresenta uma tela depois do roteamento ter sido realizado. Este software permite também manipular os parâmetros F_c e F_s.

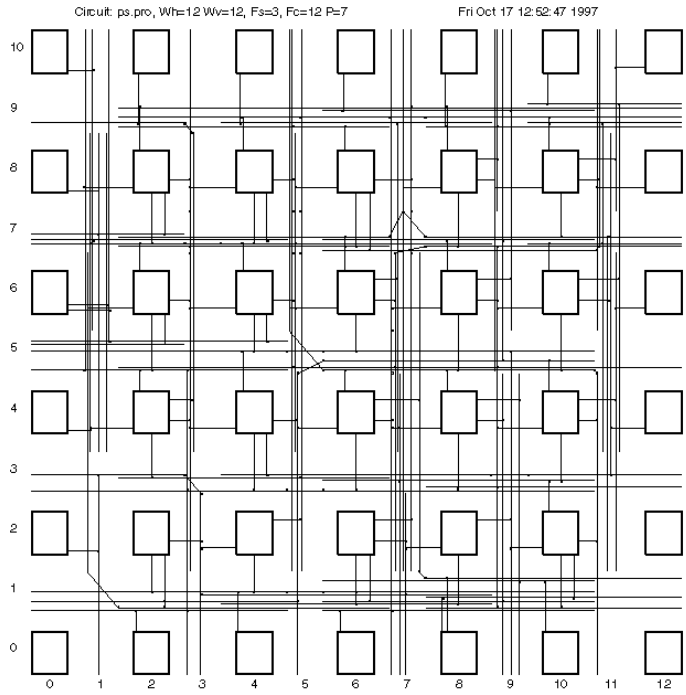


Figura 19 – Circuito roteado pelo software SEGA.

A Equação 1 mostra esta função-custo. Os dois primeiros termos referem-se à alocação de segmentos curtos ou longos, e os dois últimos termos referem-se à "roteabilidade" da conexão. O caminho que apresentar o menor custo é o escolhido. Os fatores w_α, w_β, w_c, w_f referem-se aos pesos (*weight*) aplicados a cada parcela da função-custo.

$$Cost = w_\alpha C_\alpha(p) + w_\beta C_\beta(p) + w_c C_c(p) + w_f C_f(p)$$

Equação 1– Função custo de quatro parâmetros, sendo que *p* designa um caminho de uma dada rede (*path*).

- $C_\alpha(p)$ – mede o desperdício dos segmentos longos de fio pelas conexões menores. O $C_\alpha(p)$ é definido pelo quociente do tamanho total desperdiçado dos segmentos longos em p pelo comprimento total dos segmentos em p. Quando um segmento longo é utilizado é comum que parte de seu comprimento não seja utilizado, daí o desperdício. Logo, em conexões curtas, segmentos longos devem ser evitados.
- $C_\beta(p)$ – mede o número de segmentos utilizados em uma conexão. O $C_\beta(p)$ é definido por $1-\frac{\min}{seg}$, sendo *min* o número mínimo de segmentos que p pode utilizar e *seg* o número segmentos em p. Quando mais próximo *seg* for do número mínimo de segmentos, menor será este fator.
- $C_c(p)$ - permite que o roteador selecione uma conexão mesmo que a mesma possua poucas alternativas de roteamento. Pode ser definida como a taxa do número de caminho restantes no grafo expandido.

- $C_f(p)$ – permite que o roteador selecione o caminho a ser conectado de forma a minimizar efeitos negativos nas outras conexões, ou seja, uniformiza a densidade de segmentos em cada canal. O parâmetro $C_f(p)$ é calculado armazenando todas as ocorrências dos segmentos em *p* que ocorrem em outras redes. Se *p* contém um total *i* de arestas, chamadas *e1*, *e2*, *e3* …*ei*, e cada aresta possui *j* outras ocorrências (*e1*, *e2*, *e3*, …, *ej*), então Cf(p) é dada por:

$$C_f(p) = \sum_i \sum_j \frac{1}{alt(e_j)}$$

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

3.2.3 VPR

O VPR [BET97a] (*Versatile Place and Route*) é uma ferramenta de posicionamento, roteamento global e roteamento detalhado para FPGAs. O algoritmo de posicionamento utilizado é o de têmpera simulada [KIR83] (*Simulated Annealing)*, onde a chave do problema está em uma função custo linear que leva em consideração a dificuldade de conexão em regiões com diferentes larguras de canal. O roteamento é realizado através de uma variação do algoritmo *PathFinder* [EBE95]. O algoritmo utiliza um algoritmo Maze para rotear cada rede. Se uma rede ficar bloqueada durante o roteamento, então é realizado *rip-up* e *re-route* até que a conexão bloqueada seja roteada.

Uma importante característica de *software* é flexibilidade para a descrição da arquitetura alvo, através de um arquivo de configuração. Neste arquivo os seguintes parâmetros podem ser capturados:

- no bloco lógico, o número de entradas e saídas;
- em que lados do bloco lógico podem ser acessadas as entradas e as saídas;
- a equivalência lógica entre os pinos de entrada e saída;
- número de *PADS* de E/S colocados em uma coluna ou uma linha do FPGA;
- a largura relativa dos canais de roteamento horizontais e verticais;
- a largura relativa dos canais em diferentes regiões de roteamento;
- a arquitetura da matriz de chaves, ou seja, quais conexões são permitidas;
- número de trilhas que cada pino de entrada do bloco lógico pode se conectar (F_c);
- valor de F_c para as saídas dos blocos lógicos;
- valor de F_c para os *PADS* de E/S.

A Figura 20 representa o fluxo de execução da ferramenta de CAD VPR. O arquivo de entrada deve ser uma lista de redes mapeada ou um circuito já posicionado por outra ferramenta de CAD.

O arquivo que contém as informações sobre o projeto pode ser escrito em todos os formatos aceitos pela ferramenta SIS [SEN92], o qual é utilizada para fazer a otimização lógica do circuito projetado. O passo seguinte é a realização do mapeamento tecnológico que é realizado pelo *software* FlowMap [CON94]. O arquivo gerado pela etapa de mapeamento tecnológico deve ser convertido para o formato *.net*, que é realizado pelo programa VPACK [BET97b] e possui as

seguintes palavras reservadas: .input, .output e .clb, que respectivamente indicam pino de entrada, pino de saída e início do bloco lógico. A sintaxe usada para definir o bloco lógico é a dada na Figura 21.

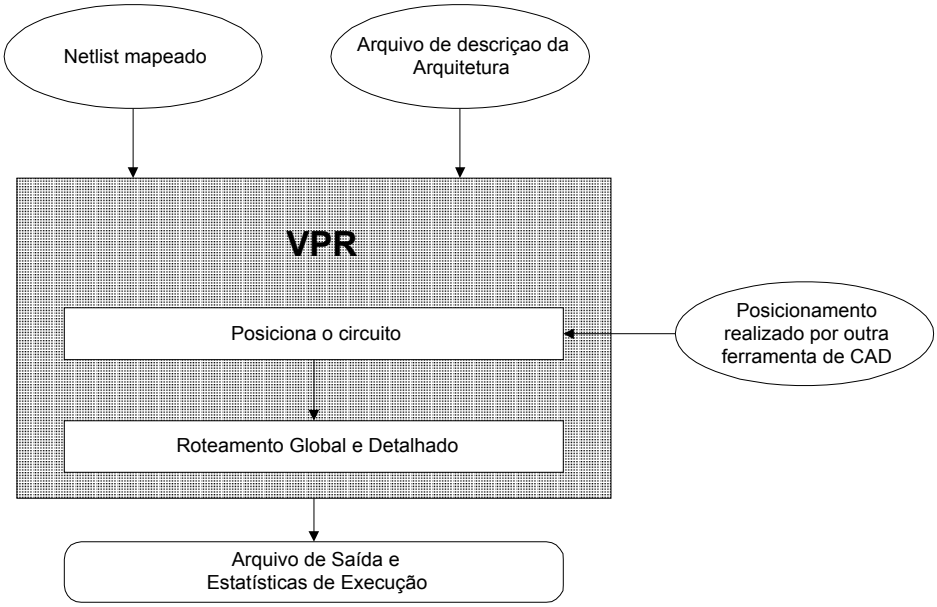


Figura 20 – Fluxo de execução do roteador VPR.

```
.clb <nome do bloco>
        pinlist: <rede A> <rede B> <rede C> …
        subblock: <nome do sub-bloco <número do pino > …
```

Figura 21 - Sintaxe do arquivo com o formato .net.

A palavra chave *pinlist* introduz uma lista contendo os nomes de todas as redes conectadas ao bloco lógico. A identificação é posicional em relação ao arquivo que descreve a arquitetura. Na versão atual do VPR [BET97b], o identificador *subblock* não é utilizado. A Figura 22 apresenta um exemplo de circuito convertido para o formato *.net*. Este circuito é constituído por um único bloco lógico, com três entradas e duas saídas. No exemplo apresentado, quando um pino não é conectado utiliza-se a palavra chave *open*, devido à conexão posicional entre a instância da CLB e sua declaração no arquivo de arquitetura.

```
.input a
  pinlist: a

.input bpad
  pinlist: b

.clb simple
  pinlist: a b open and2 open

.output out_and2
  pinlist: and2
```

Figura 22 - Exemplo de circuito em formato .net.

A Figura 23 apresenta, parcialmente, um arquivo com a lista de redes (*netlist*) que serve como ponto de partida para o roteador VPR. A Figura 24 apresenta o circuito depois da realização da etapa de posicionamento e a Figura 25 mostra como o circuito foi roteado pelo *software* VPR.

```
Addnet [a] R1C3.F1 R3C2.F3
Addnet [b] R2C1.F2 R1C3.F4
Addnet [c] R3C3.F1 R1C3.F2
Addnet [d] R3C3.F1 R1C2.F2
Addnet [e] R2C3.F4 R3C2.F4
```

Figura 23 - Exemplo de netlist.

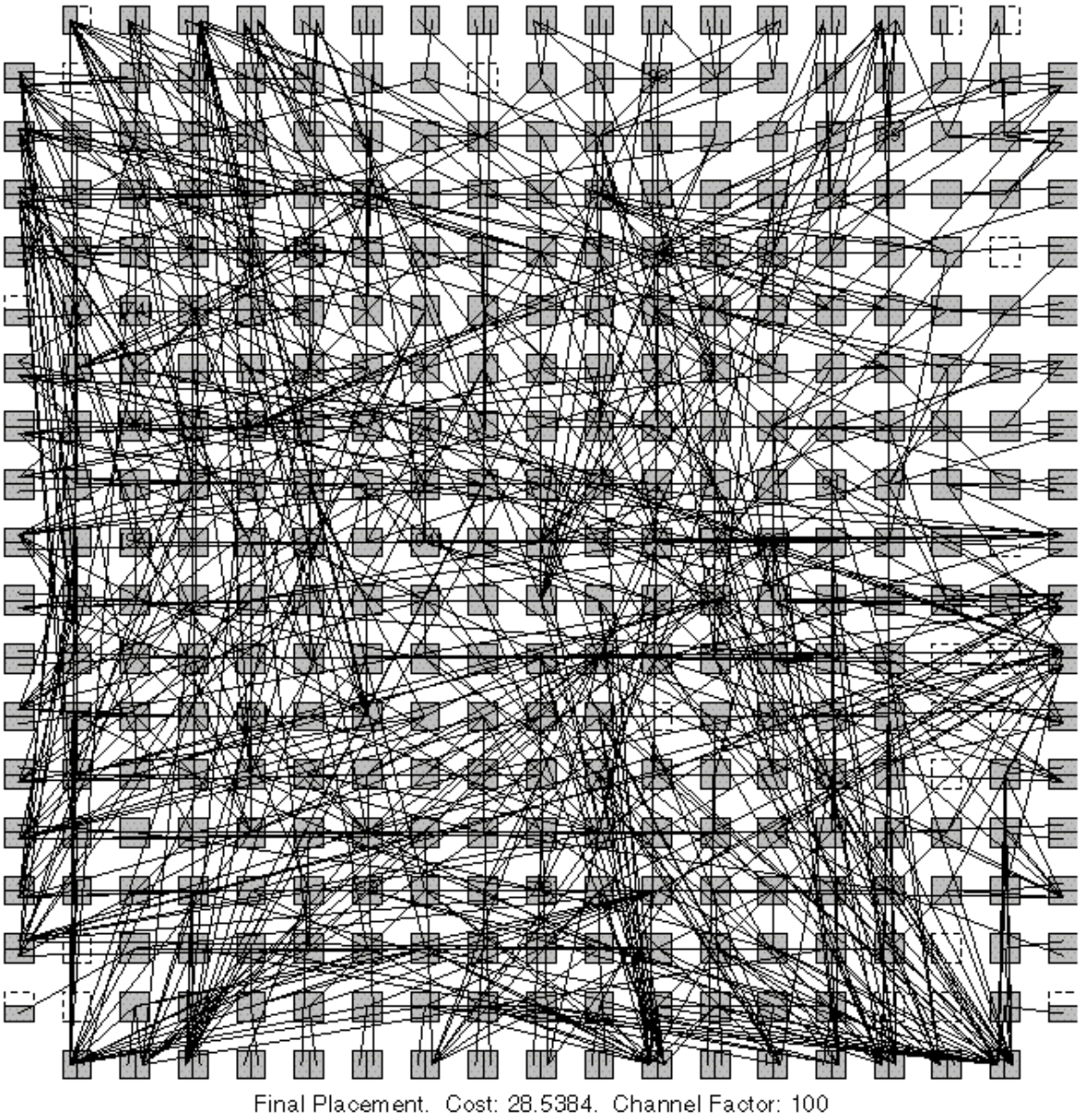


Figura 24 – Posicionamento realizado pelo software VPR.

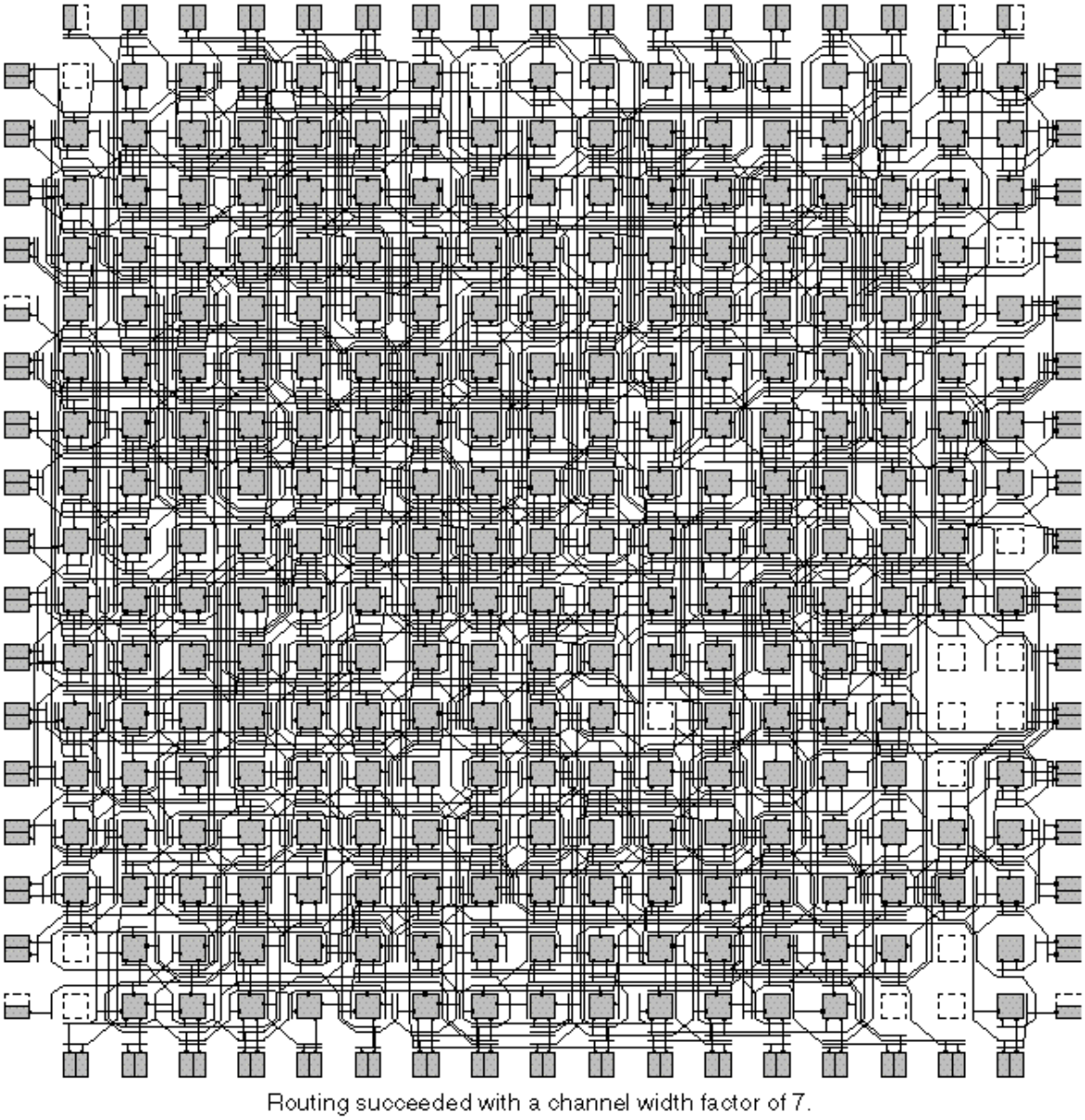


Figura 25 – Roteamento realizado pelo software VPR.

- VPR *Timing Driven*

Para FPGAs que possuem a arquitetura de seus blocos lógicos baseados em *clusters* o roteador VPR teve de ser modificado para que pudesse utilizar todos os recursos deste tipo de arquitetura de FPGA. Além de considerar *clusters*, esta versão do VPR considera em seu algoritmo de roteamento e posicionamento a otimização de desempenho [MAR00b]. A Figura 26 representa o fluxo de execução do roteador VPR Timing Driven.

A comparação do "*VPR timing driven*" com o VPR demonstrou que o novo algoritmo de posicionamento aumentou em 42% o desempenho de posicionamento e posteriormente de roteamento. Constatou-se também que houve a redução média de 5% na utilização dos segmentos de fio do dispositivo [MAR00b]. Os resultados foram obtidos através da utilização dos 20 circuitos de teste do conjunto de *benchmarks* MCNC [YAN91].

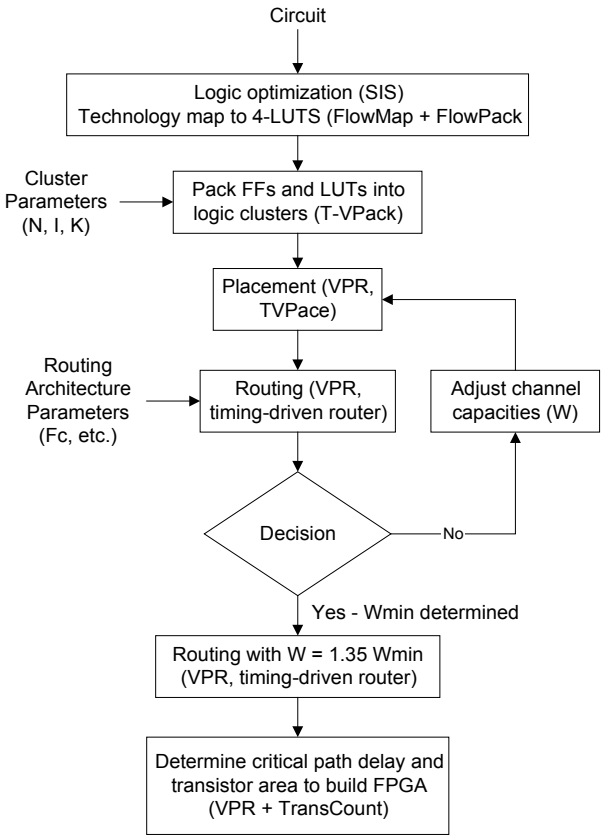


Figura 26 – Fluxo de execução do VPR timing Driven [MAR00b].

3.2.4 GBP e OGC

Este algoritmo combina o roteamento global com o roteamento detalhado em uma única etapa [WU94][WU95]. Assumindo que f_s=3 e f_c=largura do canal de roteamento, o FPGA é roteado considerando cada domínio de trilhas como uma caixa e com um algoritmo *greedy* procura-se rotear todas as caixas até que não existam mais caixas sem conexão. O método é utilizado pelo *software* GBP, é similar ao método heurístico *Best Fit Decreasing Bin-Packing* [COR89]. Foi observado que o GBP não consegue uma ótima densidade para o último domínio. No *software* OGC foi utilizado um outro mecanismo *greedy*, com uma modelagem diferente a fim de obter uma maior densidade nos domínios de trilhas, de forma a diminuir o roteamento utilizado. Nesses algoritmos, de uma única etapa, as redes multi-ponto (*Fanout > 1*) são roteadas uma de cada vez. Se o roteamento falhar em uma das redes, ela é movida para o começo da lista e o algoritmo é reiniciado, podendo assim ocorrer uma postergação indefinida.

3.2.5 TRACER

Os objetivos propostos por este algoritmo [LEE97] são: aumentar a "roteabilidade" e o desempenho (diminuir os atrasos) do circuito. O roteamento é realizado em duas etapas: roteamento inicial e *rip-up* e *re-route*. Na primeira etapa, as redes são roteadas seqüencialmente de acordo com as suas prioridades, ou seja, primeiro as redes onde o atraso se torna mais crítico. Durante a segunda etapa, é usado um *rip-up* e *re-route*, para solucionar problemas de violação dos recursos de roteamento (a rede ficou bloqueada) e os atrasos.

3.2.6 FPR

O FPR [ALE96][ALE95] é uma ferramenta para posicionamento e roteamento simultâneos para FPGAs das séries 3000 e 4000 da Xilinx. Este *software* combina uma estratégia geométrica recursiva para o posicionamento e roteamento global com um algoritmo baseado em coloração de grafos para o roteamento detalhado. O posicionamento e o roteamento global utilizam uma técnica denominada "*Thumbnail Partitioning*", que decompõe o circuito em um plano *m x n*. O roteamento detalhado é modelado como um grafo onde os vértices são os blocos lógicos e as arestas suas conexões. As duas etapas são resolvidas através de uma árvore de Steiner, utilizando um algoritmo heurístico [KOU81], que soluciona este problema em tempo polinomial. Este algoritmo tem como objetivo principal reduzir a área e reduzir a densidade de roteamento nos canais.

3.2.7 TCR

Este *software* é utilizado para rotear FPGAs simétricos, desenvolvido com base no XC3000 [RAM96]. O objetivo é rotear o circuito no FPGA com o menor atraso possível, impondo restrições. Esse *software* possui 4 passos: (1) calcular o atraso de cada rede e ordená-las em ordem decrescente de valor de atraso; (2) alocar os recursos de roteamento, identificando as redes (> atraso) que serão roteadas nos segmentos longos; (3) roteia o número máximo de redes nos segmentos longos, de forma a esgotar este recurso de roteamento e (4) roteia as redes que sobraram nos segmentos unitários[2]. No primeiro passo, é utilizada uma função custo para determinar o atraso da rede:

$$L\text{-}value = \alpha_1 * netslack + \alpha_2 * fanout + \alpha_3 * span + \alpha_4 * USP$$

Os parâmetros utilizados são os seguintes:

- *Netslack*: é a diferença entre o tempo requerido para o sinal ser propagado até o destino e o tempo real de propagação. Quanto mais negativo for o *netslack*, mais crítica é a rede.
- *Fanout*: é o número de pinos conectado à saída de um bloco lógico. Um *fanout* alto indica que a rede é crítica.
- *Span*: representa o comprimento da rede. Quanto maior for a distância entre os blocos lógicos de uma rede, mais crítica é a rede.
- USP: é um parâmetro opcional que é definido pelo usuário para indicar uma rede mais crítica.

Segundo o autor, para que este software apresente um bom desempenho, as constantes devem possuir os seguintes valores: $\alpha 1=-1$ e $\alpha 2=\alpha 3=\alpha 4=1$.

No segundo passo, as redes são indicadas para o roteamento em segmentos longos de fio em razão do atraso elevado. No passo seguinte é iniciado o processo de roteamento nos segmentos longos, de forma a esgotar os segmentos longos de fio. Para finalizar as redes restante são roteadas nas linhas de propósito geral.

[2] Nota-se que o XC3000 possui apenas dois recursos de roteamento: as linhas longas e as linhas de propósito geral.

3.3 Comparação entre os algoritmos de roteamento

Na Tabela 2 [LEM97] são apresentados os algoritmos de roteamento, seus algoritmos de base e as suas principais características.

Tabela 2 – Resumo das características dos algoritmos de roteamento [LEM97].

	LocusRoute	CGE	SEGA	GBP	OGC	IKMB	TRACER	SROUTE	FPR	VPR
CARACTERÍSTICA	[ROS90]	[BRO92]	[LEM93]	[WU94]	[WU95]	[ALE95]	[CHE95]	[WIL97]	[ALE96]	[LEM97]
Explora a equivalência dos pinos	s			n	n	n	n	s	n	s
Explora doglegs nos pinos de saída		s	s	s	s	s	s	s	s	
Explora doglegs nos pinos de entrada.		s/n[a]	s/n[a]	s	s	s	s	n	s	
Explora os segmentos longos de fio	s[b]	n	s	n	n	n	n	n	n	s[b]
Realiza rip-up e re-routing	s	s	n	n	n	s	s	n	n	s
Mecanismo de solução greedy	s	s	s	s	s	n	n	s	s[c]	n
Árvore de Steiner / Roteamento global baseado em arborescência	n			n	n	s	n	n	s	n
Roteamento global baseado no menor caminho	s			n	n	n	n	n	s	n
Roteamento global maze	n			n	n	n	s	s	n	s
Garantia de limites de performance	n	n	n	n	n	s[d]	n	n	s[d]	n
Resultados dependentes de ordenamento de redes	s	s	n	n	n	s	n	s	s	n
Posicionamento e roteamento simultâneo	n	n	n	n	n	n	n	n	s	s
Roteamento Global	s	n	n	n	n	n	n	n	n	n
Roteamento Detalhado	n	s	s	n	n	n	n	n	n	n
Combinação de roteamento global e detalhado	n	n	n	s	s	s	s	s	n	n

a - CGE/SEGA
b - O custo de quebra pode ser aplicado para auxiliar em uma melhor exploração dos segmentos longos de fio.
c - Na fase de particionamento do roteamento global é selecionado o método greedy.
d - Em cada rede é garantido o uso de um número menor ou igual a 2 vezes número mínimo de fios.

- **Equivalência de Pinos**. Os *softwares* LocusRoute, SROUTE e VPR exploram a equivalência entre os pinos dos blocos lógicos. Isso significa que se em uma função booleana as entradas estiverem no mesmo nível lógico, é possível colocar os sinais em qualquer uma das entradas F ou G. Com essa técnica é possível escolher o pino de entrada, de forma a obter redução na área ocupada pelo circuito após o roteamento. Caso seja usado um FPGA com pinos de entrada apenas na face superior e inferior é necessário 8% a mais de área de roteamento [BET96]. Na Figura 27, é possível observar que o bloco lógico da Xilinx família XC4000, possui uma entrada F e uma entrada G em cada uma de suas quatro faces.

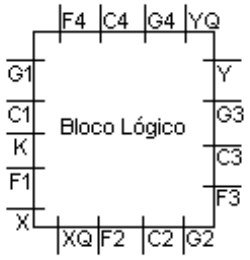


Figura 27 – Distribuição dos Pinos de E/S em um Bloco Lógico.

- **Exploração de *doglegs* nos pinos de entrada e saída**. Para explicar o que significa *doglegs* será utilizado um exemplo prático. Quando houver a necessidade de realizar duas conexões contidas no mesmo espaço de roteamento horizontal (Figura 28) [MOR90], de tal forma que a realização de uma conduz ao bloqueio da outra, haverá a necessidade de romper uma das conexões, através de uma ou mais chaves, criando assim uma rota alternativa para a conexão bloqueada.

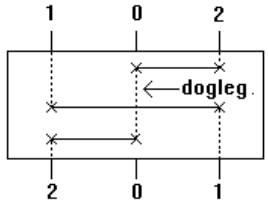


Figura 28 – Uso de *dogleg* para evitar bloqueio entre as conexões.

- **Exploração dos Segmentos de Fio Longo**. O objetivo desta técnica é reduzir o atraso provocado pelas resistências e capacitâncias parasitas provenientes das chaves de conexão, reduzindo o desempenho do circuito. Outra vantagem é que os sinais globais, por exemplo clocks ou resets, podem ser distribuídos uniformemente ao longo de todo o circuito. Os *softwares* que utilizam essa técnica são o LocusRoute, SEGA, VPR.
- ***Rip-up* e *Re-routing***. Essa técnica é utilizada principalmente nos *softwares* de roteamento de dois passos, roteamento global e roteamento detalhado e em poucos casos nos *softwares* que realizam o posicionamento e o roteamento simultâneos. O método consiste em uma primeira fase chamada de roteamento global, onde é atribuída a rota que cada rede deve seguir. Uma vez as rotas atribuídas, tem início a fase de roteamento detalhado, com o objetivo de realizar através dos recursos de roteamento disponíveis a conexão física dos blocos lógicos. Quando uma conexão não pode ser fisicamente realizada, o algoritmo executa a rotina de *rip-up* e *re-routing*. Essa rotina retira as conexões que bloqueiam determinada rede (*rip-up*), reordena a lista de redes, relançando o roteador sobre estas redes (removidas e não roteadas), processo denominado *re-route*.
- **Mecanismo de Solução Gulosa**. Os algoritmos gulosos utilizam o princípio da localidade para encontrar a solução. Consistem em dividir o problema em partes iguais e resolver apenas localmente, sendo que a solução encontrada é generalizada supondo que o comportamento seja o mesmo para todas as partes. A primeira vista, o algoritmo parece eficiente, pois se consegue solucionar o problema local, obrigatoriamente o problema global será solucionado a partir de uma generalização da solução local. No caso do roteamento de FPGA é possível encontrar uma solução ótima local, porém quando generalizada a solução poderá haver redes não roteadas,

sendo necessária a utilização de técnicas de *rip-up* e *re-route* para a completa solução do roteamento.

- **Árvore de Steiner**. Modela-se o FPGA como um grafo, onde os nodos são os blocos lógicos e as arestas a suas conexões. Como todas as redes possuem uma origem e um ou mais destinos, a origem é chamada de raiz e os nodos restantes são denominados nodos-folha. Em razão disto, a rede recebe o nome de árvore. A árvore de Steiner tem como objetivo principal encontrar o menor caminho entre a origem e os destinos. Quanto maior o número de nodos-folha, maior será a profundidade do grafo, contribuindo para o aumento da complexidade do problema. Encontrar a menor árvore valorada que esteja conectada a todos os vértices de G, é classificado como um problema NP-completo.

3.4 Desempenho dos algoritmos

A Tabela 3 apresenta a comparação entre os vários *softwares* de roteamento, onde os valores apresentados indicam a densidade mínima do canal. Esta tarefa é realizada através de circuitos de teste (*benchmarks*) e o parâmetro utilizado é o de número máximo de trilhas por canal de roteamento.

Tabela 3 – Tabela comparativa de desempenho entre roteadores acadêmicos [LEM97].

Os valores mostrados na tabela correspondem ao número mínimo de trilhas por canal para rotear todo o circuito.

Posicionamento	ALTOR [ROS85]								SPLACE	FPR [ALE95]	ALTOR	VPR
R. Global	LocusRoute [ROS90]			GBP [WU94]	OGC [WU95]	IKMB [ALE95]	TRACER [CHE95]	SROUTE [WIL97]			VPR [BET97a]	
R. Detalhado		CGE [BRO92]	SEGA [LEM93]								SEGA [LEM97]	
9symml	9	9	9	9	9	8	6	7	7	9	7	6
Alu2	10	12	10	11	9	9	9	9	8	10	8	7
Alu4	13	15	13	14	12	11	11	12	9	13	10	8
Apex7	13	13	13	11	10	10	8	9	6	9	10	5
Example2	17	18	17	13	12	11	10	11	7	13	10	5
K2	16	19	16	17	16	15	14	15	11	17	14	10
Term1	9	10	9	10	9	8	7	8	5	8	8	5
Too_large	11	13	11	12	11	10	9	11	8	11	10	7
Vda	14	14	14	13	11	12	11	12	10	13	12	9
TOTAL	112	123	112	110	99	94	**85**	94	**71**	103	**89**	**62**

Os valores apresentados pela Tabela 3, foram obtidos com a combinação de ferramentas de posicionamento, roteamento global e roteamento detalhado. A combinação dos roteadores acadêmicos VPR (posicionamento e roteamento global) e SEGA (roteamento detalhado) obteve a menor densidade de trilhas por canal. Com esta combinação foram necessárias 62 trilhas para que os 9 circuitos de teste fossem roteados com sucesso.

Esta comparação permite concluir que os algoritmos com roteamento global e detalhado em duas etapas (VPR e SEGA), podem ser competitivos com os algoritmos de roteamento de uma

etapa (TRACER e SROUTE), quando apenas a rotabilidade é importante. É possível notar que o melhor desempenho global entre os *softwares* que utilizam o posicionador ALTOR [ROS85] é alcançada pelo *software* TRACER [CHE95]. Para a obtenção do melhor desempenho é preciso combinar os *softwares* VPR e SEGA, onde o primeiro realiza o posicionamento e o roteamento global, indicando a rota que cada rede deve seguir, e o segundo, realiza o roteamento detalhado.

Existem outros circuitos de teste (*Benchmarks*) [YAN91], que foram utilizados na geração dos resultados da Tabela 4. Os valores para o VPR foram encontrados através da realização de testes no laboratório. Essa Tabela mostra a largura do canal necessária para que seja feito o posicionamento e o roteamento dos circuitos de teste. Na Tabela 4, observa-se que o VPR utilizou 197 trilhas enquanto o SEGA utilizou 334 trilhas.

Com o objetivo de incentivar os pesquisadores a pesquisar novas formas de se rotear circuitos grandes (número de redes maior que 1200), os autores do VPR propuseram um desafio [ROS02]. O desafio consiste em desenvolver um roteador capaz de rotear todos os circuitos da tabela acima, com o menor número de trilhas possível.

Tabela 4 - Largura mínima de canal necessária, para a realização do roteamento [BET97a].

Circuito	# BL	SEGA	VPR
Alu4	1522	16	10
Apex2	1878	20	11
Apex4	1262	19	12
Bigkey	1707	9	7
Clma	8383	25	12
Des	1591	11	7
diffeq	1497	10	7
Dsip	1370	9	7
Elliptic	3604	16	10
Ex1010	4598	22	10
Exp5	1064	16	13
Frisc	3556	18	11
Misex3	1397	17	10
Pdc	4575	33	16
S298	1931	18	7
S38417	6406	10	8
S38584.1	6447	12	9
Seq	1750	18	11
Spla	3690	26	13
Tseng	1047	9	6
Total	**---**	**334**	**197**

#BL: número de blocos lógicos.

O roteador SC-PathFinder apresentado em [CHA97] e [CHA00] conseguiu vencer o desafio reduzindo o número de trilhas necessárias para que os circuitos da tabela acima fossem roteados com sucesso. Esse roteador necessitou de apenas 188 trilhas para rotear os circuitos, nove

trilhas a menos do que necessitou o VPR. A Tabela 5 apresenta a comparação entre o VPR e o roteador SC-PathFinder, atual vencedor do desafio proposto, e algoritmo de melhor desempenho hoje encontrado na literatura.

Tabela 5 – Resultados do atual vencedor do desafio [ROS02].

Circuito	# BL	VPR	SC-Pathfinder
Alu4	1522	10	9
Apex2	1878	11	11
Apex4	1262	12	11
Bigkey	1707	7	6
Clma	8383	12	12
Des	1591	7	7
diffeq	1497	7	7
Dsip	1370	7	6
Elliptic	3604	10	9
Ex1010	4598	10	10
Exp5	1064	13	12
Frisc	3556	11	11
Misex3	1397	10	10
Pdc	4575	16	16
S298	1931	7	7
S38417	6406	8	7
S38584.1	6447	9	8
Seq	1750	11	11
Spla	3690	13	12
Tseng	1047	6	6
Total	**---**	**197**	**188**

4 Algoritmo Seqüencial de Roteamento

O objetivo deste Capítulo é apresentar a implementação do algoritmo seqüencial de roteamento. A Seção 4.1 descreve os arquivos de entrada de dados (lista de redes e arquivo de arquitetura) do algoritmo de roteamento e suas estruturas. Na Seção 4.2 é apresentada a modelagem de dispositivos programáveis FPGA na forma de um grafo parametrizado, e como foi modelado através de uma matriz de números inteiros e será apresentada a biblioteca de estruturas de dados e algoritmos LEDA. A Seção 4.3 descreve a implementação do algoritmo Maze e as estruturas utilizadas na sua implementação. A Figura 29 apresenta o fluxo de funcionamento do algoritmo de roteamento e suas entradas de dados.

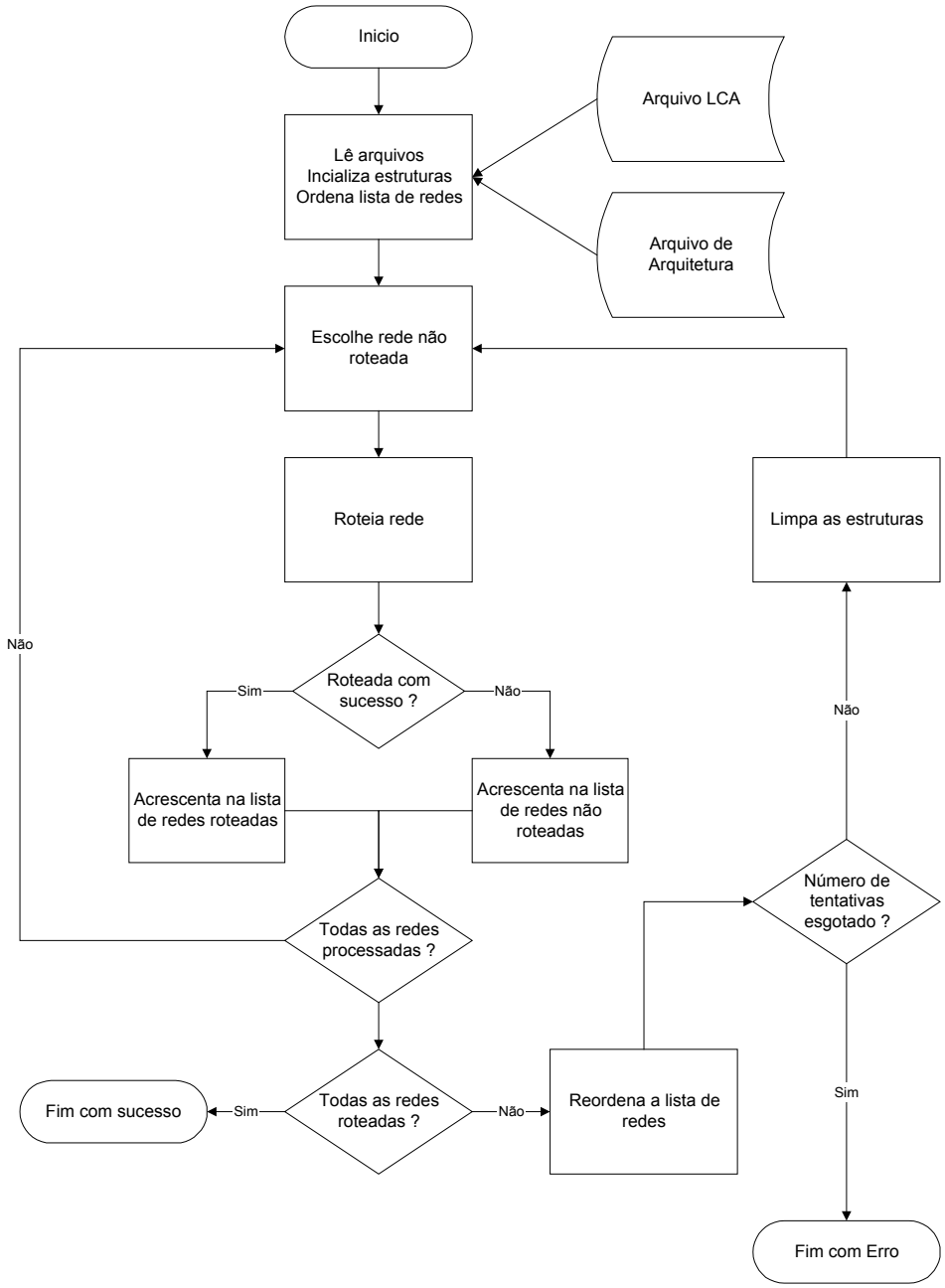


Figura 29 – Fluxo de execução do roteador implementado.

4.1 Entrada de Dados

A entrada de dados para a execução do algoritmo de roteamento é realizada através de dois arquivos, onde um contém a descrição da arquitetura do dispositivo FPGA e o outro a lista de redes do circuito a ser roteado. A Seção 4.1.1 apresenta o arquivo de arquitetura, as estruturas e como é realizada a leitura do mesmo e na Seção 4.1.2 é apresentado o arquivo que contém a lista de redes representando o circuito a ser roteado.

4.1.1 Arquivo de Arquitetura

No arquivo de arquitetura é definido o tipo de dispositivo que está sendo utilizado, o número de linhas e colunas de CLBs, o número de trilhas existentes em cada canal, a disposição dos pinos de conexão do Bloco Lógico e o número de vezes que o processo de roteamento deve ser realizado até o circuito ser totalmente roteado no dispositivo. Estas informações são armazenadas na estrutura apresentada pela Figura 30 e um exemplo do arquivo de arquitetura pode ser visto no Anexo I. Este formato foi definido no escopo deste trabalho, não sendo um formato padrão.

```
typedef struct {
    char fpgaTipo[TAM];
    int linhas;
    int colunas;
    int trilhas;
    int repeticoes;
    arc_pin *pinos;
}arc_arch;

typedef struct pin {
    char pNome[TAM];
    int pPosicao;
    struct pin *prox;
}arc_pin;
```

Figura 30 – Estrutura da Arquitetura do Dispositivo.

A estrutura "*arc_arch*" é composta por 5 variáveis e por uma lista da estrutura "*arc_pin*". A variável "*fpgaTipo*" informa qual é o tipo de dispositivo que será utilizado. As variáveis "*linhas*" e "*colunas*" indicam a quantidade de linhas e colunas de *CLBs* do dispositivo, ou seja, a quantidade de Blocos Lógicos existentes no dispositivo e a variável "*trilhas*" informa o número de trilhas existentes em cada canal de roteamento. A variável "*repeticoes*" indica o número de execuções de "*rip-up and re-route*" que devem ser executadas até o circuito ser roteado com sucesso.

Na estrutura "*arc_pin*" são armazenadas as informações relativas à posição dos pinos de conexão do Bloco Lógico. Ela é composta por duas variáveis e um ponteiro. A variável "*pNome*" indica o nome do pino, a variável "*pPosicao*" indica em qual direção está localizado o pino de conexão do Bloco Lógico. O ponteiro é utilizado para a criação de uma lista encadeada.

O pseudo código da rotina de leitura da arquitetura é apresentada na Figura 31.

```
Lê tipo do FPGA
Lê o número de linhas
Lê o número de colunas
Lê número de trilhas por canal
Lê o número de execuções do algoritmo
Enquanto não for fim de bloco {
   Lê nome do pino
   Lê a posição do pino
}
Lê indicação de final do arquivo de arquitetura
```

Figura 31 – Rotina de leitura do arquivo de arquitetura do dispositivo.

As posições dos pinos de conexão no arquivo de arquitetura são descritas por norte, sul, leste, oeste ou qualquer um dos lados do CLB ("*north*","*south*","*east*", "*west*" ou "*anyside*"). Após a leitura, essas denominações são convertidas em números inteiros (1, 2, 3, 4 ou 5 respectivamente) para que a execução do algoritmo seja simplificada.

4.1.2 Arquivo de lista de redes

O arquivo de lista de redes (".lca") contém todas redes que compõem o circuito a ser roteado. Cada linha do arquivo corresponde a uma única rede do circuito, todas elas possuem a palavra "*Addnet*" no início para indicar que a rede deve ser adicionada no circuito. A sintaxe de cada rede é ilustrada pela Figura 32 e no Anexo II é apresentado um exemplo de arquivo de lista de redes.

Addnet <nome da rede> <bloco lógico origem.pino> <bloco lógico destino.pino> <bloco lógico destino.pino> ...

Figura 32 – Sintaxe do Arquivo.

A estruturas que armazenam as redes estão ilustradas na Figura 33.

```
typedef struct {
    char Nome[TAM];
    int nNome;
      int xmin;
      int ymin;
      int xmax;
      int ymax;
      int nTerms;
    ap_ponto *rede;
}ap_rede;

typedef struct pto {
    int hv,x,y;
    struct pto *prox;
}ap_ponto;
```

Figura 33 – Estrutura da lista de redes.

A estrutura "*ap_rede*" é composta por 7 variáveis e uma lista da estrutura "*ap_ponto*" que indica todas as conexões da rede. A partir da estrutura "ap_rede" é criada uma lista contendo todas as redes que compõem o circuito. A variável "*Nome*" é do tipo char e armazena o nome da rede a ser roteada e a variável "*nNome*" é o nome da rede representado por um inteiro para facilitar a identificação da mesma em uma matriz tri-dimensional. As variáveis "*xmin*", "*ymin*", "*xmax*" e "*ymax*" são utilizadas para representar os limites de propagação para a rede que está sendo roteada. Essas variáveis são preenchidas com as coordenadas mínimas e máximas representando um retângulo de procura. Assim quando a propagação está ocorrendo e atingir a fronteira determinada por essas variáveis, o processo de expansão da frente de onda é interropido. E por fim, a variável "*nTerms*" indica a quantidade de pontos existentes na rede. Essa informação é muito importante para que a lista de redes seja ordenada de forma decrescente.

A estrutura "*ap_ponto*" possui três variáveis inteiras que identificam em qual canal este pino da CLB está conectado e um ponteiro que indica o próximo ponto da rede. A localização do canal é fornecida pela tripla (*hv*,*x*,*y*), onde *hv* indica se o canal está na vertical ou na horizontal, e as coordenadas *x* e *y* representam a linha e a coluna respectivamente no FPGA. A Figura 34 representa a lista de redes e as suas estruturas.

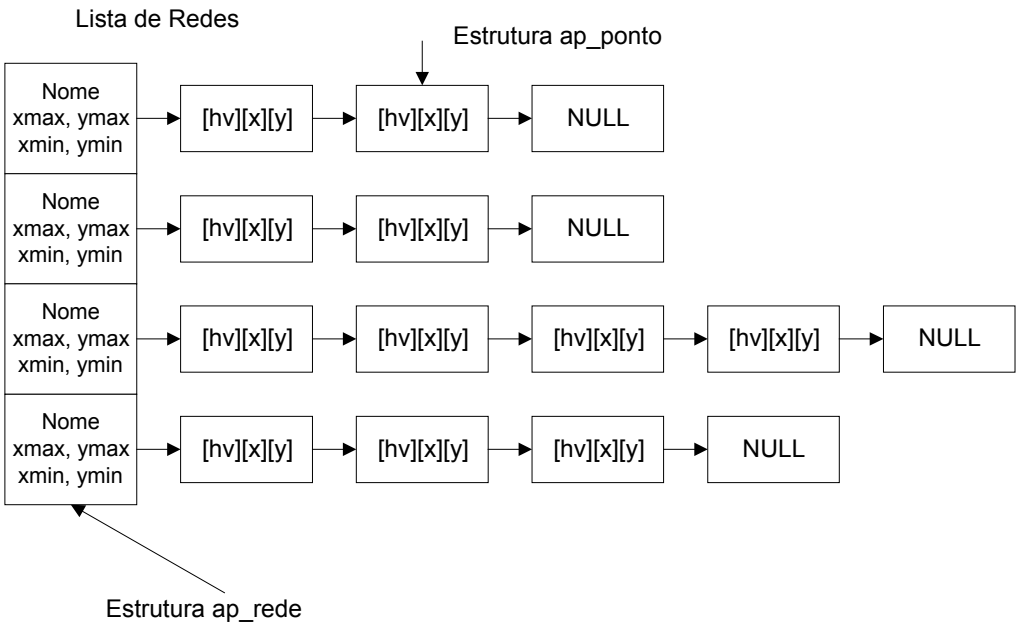


Figura 34 – Relacionamento das listas entre si.

A Figura 35 apresenta a rotina de leitura do arquivo de redes e a conversão das coordenadas dos Bloco Lógicos para lista de redes composta pela estrutura "*ap_rede*". Nessa rotina a primeira tarefa a ser realizada é a leitura de todo o arquivo para uma lista de *strings*, visando assim reduzir o tempo de processamento visto que não é realizado nenhum acesso a disco para a conversão da rede lida. Após a leitura do arquivo a rotina fica em um laço enquanto a lista de *strings* não for vazia, procura o nome da rede e associa à mesma um número inteiro seqüencial. Um novo laço é iniciado onde vão ser identificados e convertidos os pontos que compõem a rede até que a linha que está sendo analisada chegue ao fim. Nessa etapa é também definida a janela de propagação e no final do laço determina-se o número de pontos da rede.

```
Lê todas as redes para uma lista de strings
Enquanto lista de strings não está vazia {
   Zera variáveis
   Procura a palavra "Addnet"
   Coloca nome da rede na variável "Nome"
   Coloca na variável "nNome" número sequencial de leitura
   Ponteiro para a lista de pontos = NULL
   Enquanto não terminar a linha {
      Coordenadas (hv,x,y) = Converte Coordenadas
      Se xmin > coordenada x então xmin = coordenada x
      Se ymin > coordenada y então ymin = coordenada y
      Se xmax < coordenada x então xmax = coordenada x
      Se ymax < coordenada y então ymax = coordenada y
      Numero de pontos ++
   }
   Coloca na estrutura o numero máximo de pontos da rede
}
```

Figura 35 – Rotina de leitura e conversão do arquivo de redes.

A função "*Converte Coordenadas*" é utilizada para realizar a conversão das coordenadas lidas em coordenadas representadas por números inteiros (*hv*,*x*,*y*). A rotina de conversão das coordenadas é descrita pela Figura 36.

```
Recebe o ponto lido na forma de uma string (Exemplo: R1C1.F4)
Lê as coordenadas (x e y) e a posição do pino (NORTH, SOUTH, EAST, WEST)
Compara com o tamanho máximo do FPGA para verificar se existe a coordenada
Procura na estrutura da arquitetura a posição do pino
Se não encontrou a posição
  Retorna valores de coordenadas = -1
Se posição = NORTH {
  hv = horizontal
  x = x
  y = y
}
Se posição = SOUTH {
  hv = horizontal
  x = x+1
  y = y
}
Se posição = EAST {
  hv = vertical
  x = x
  y = y+1
}
Se posição = WEST {
  hv = vertical
  x = x
  y = y
}
Retorna coordenadas convertidas
```

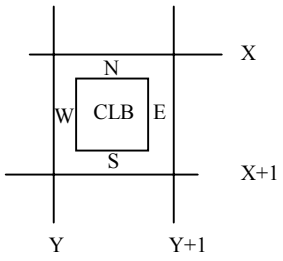


Figura 36 – Rotina que realiza a conversão das coordenadas.

A entrada dessa função é uma *string* com a seguinte formatação: "RxxCyy.PP", onde xx e yy representam a posição do Bloco Lógico no FPGA e "*PP*" é o nome do pino de conexão, nome esse que indica em que face do Bloco Lógico o pino está localizado (ver Anexo II). A definição da localização do pino encontra-se descrita no arquivo de arquitetura. Após as coordenadas e a posição do pino no Bloco Lógico terem sido encontradas é feita a verificação de se elas estão dentro dos limites do FPGA e se a posição do pino está definida no arquivo de arquitetura.

4.2 Modelagem do FPGA

Na Seção 4.2.1 é apresentada a modelagem do FPGA como um grafo parametrizado. Para a realização dessa modelagem era utilizada a biblioteca de estruturas de dados e algoritmos LEDA. Essa modelagem teve de ser abandonada, porque a biblioteca LEDA deixou de ser de domínio público. Na Seção 4.2.2, é apresentado o FPGA modelado como uma matriz de estruturas, com 3 dimensões.

4.2.1 Grafos

O dispositivo FPGA pode ser modelado como um grafo parametrizado onde os vértices são as matrizes de chaves ("*switch box*") e as arestas representam os canais de roteamento, como pode ser observado na Figura 37.

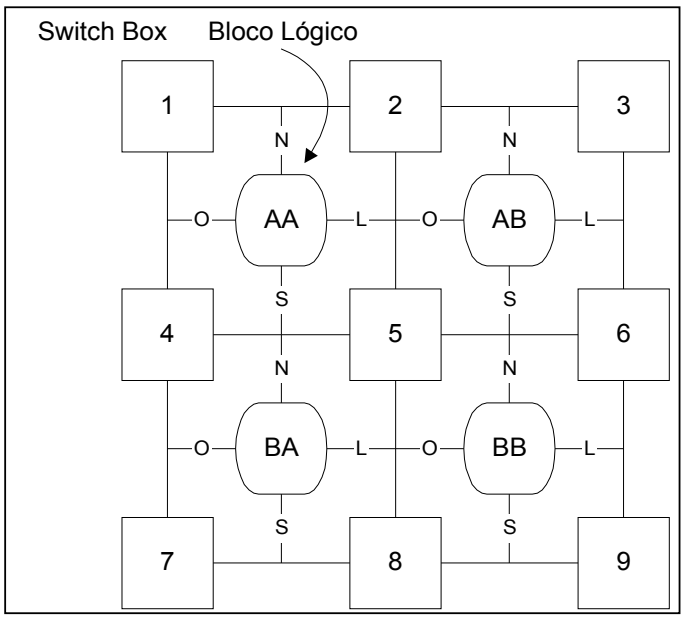


Figura 37 – FPGA modelado na forma de um grafo parametrizado.

Os Blocos Lógicos são identificados por uma dupla de letras (ou também par x, y) que informa a sua posição (linha, coluna) no dispositivo e os pinos de saída dos blocos são conectados a quatro canais de roteamento. Os canais de roteamento são conectados entre si pelas matrizes de chaves (*"Switch Box"*), e cada matriz de chaves deve ter no mínimo 2 canais de roteamento conectados.

Para a implementação dessa estrutura foi utilizada a biblioteca de estruturas de dados e algoritmos (LEDA [MEH89]), por possuir a implementação de estruturas complexas (listas, filas, grafos, etc) e funções de para o tratamento dessas estruturas já implementadas.

As principais características dessa biblioteca são:

- LEDA provê uma considerável coleção de tipos de dados e algoritmos de forma simplificada tornando acessíveis as operações com tipos complexos, por exemplo grafos, a qualquer programador.
- Possibilita uma especificação precisa e amigável para cada tipo de dado e algoritmo. As especificações são pequenas (apenas uma linha de programação), gerais (permite a utilização em diferentes implementações) e abstratas (mascaram todos os detalhes de implementação).
- Para uma estrutura de dados eficiente o acesso posicional é muito importante. Na LEDA, o conceito de item foi desenvolvido de uma forma abstrata e de uso simplificado.
- Contém implementações eficientes para cada tipo de dado, por exemplo, o algoritmo de procura Depth First Search, os algoritmos para descobrir o menor caminho em grafos (Dijkstra's e Minimum Spanning Tree) entre outros.

As operações de criação e modificação de grafos são bastante simplificadas com a utilização da biblioteca LEDA. LEDA é implementada como uma biblioteca de classes C++. Por exemplo, operação do tipo "para todos os nódos *v* do grafo G faça" é implementada como "forall_edges(v,G)" ou "para todas as arestas adjacentes *v* do grafo G" é implementada como "forall_in_out_edges(v,G)". Esses recursos permitem o desenvolvimento de programas para a solução de problemas com grafos de uma forma simplificada com funções de simples utilização. A Figura 38 apresenta as três estruturas que modelam o dispositivo programável FPGA.

```
typedef struct CHANNEL {
  int  counter;
  int visited;
  int used_tracks;
  int routing;
  list<TRACK> tracks;
  int block1;
  int block2;
  int rest;
  int *vagas;
  int *propag;
  };
typedef struct TRACK {
  int net;
  };
typedef struct SBOX {
  int  index;
  edge north;
  edge east;
  edge south;
  edge west;
  };
```

Figura 38 – Estruturas utilizadas para modelar o dispositivo programável FPGA.

As arestas do grafo possuem toda a informação de roteamento, armazenada na estrutura "CHANNEL". O contador "*counter*" é utilizado para indicar o nível de propagação da frente de onda através do grafo na fase de propagação, e vai sendo incrementado caso a aresta não tenha sido

visitada. Na fase de retropropagação este contador é analisado com o intuito de encontrar o menor caminho entre os vértices, que por sua vez vai sendo decrementado até que a origem da rede tenha sido encontrada. A variável "*visited*" indica se a aresta já foi localizada pela frente de onda ou não e a variável "*routing*" indica que o processo de roteamento está sendo realizado, sendo essa variável utilizada pela fase de retropropagação. O número de trilhas já preenchidas é indicado pela variável "*used_tracks*", que após o término do roteamento da rede todas as arestas pertencentes à mesma são incrementadas e o nome da rede é armazenado em uma lista. Para indicar à qual Bloco Lógico o canal está conectado existem duas variáveis para este fim: "*block1*" e "*block2*". O preenchimento destas variáveis é função da coordenada do Bloco Lógico, obtida através da Equação 2. A variável "*block1*" refere-se ao Bloco Lógico situado a esquerda do canal e a variável "*block2*" indica o Bloco Lógico que está a direita do canal. Se o canal está localizado na periferia do FPGA, as variáveis "*block1*" e "*block2*" podem assumir valores negativos. Os valores negativos indicam que não existem Bloco Lógico conectado ao mesmo.

$$RXXCYY = XX * 10000 + YY$$

Onde: XX indica a coordenada da linha

YY indica a coordenada da coluna

Equação 2 – Fórmula de conversão da coordenadas

Existem ainda duas estruturas auxiliares ligadas entre si, a primeira é uma lista onde estão relacionados os blocos lógicos e suas devidas conexões na rede e a outra contém as redes com os blocos lógicos que compõem a mesma, descritas anteriormente na Seção 4.1.

A estrutura TRACK é composta por um número inteiro que indica o nome da rede e é utilizada na estrutura CHANNEL para indicar se existe e qual é a rede que passa por aquele canal.

A estrutura SBOX é composta por um número inteiro seqüencial utilizado como índice e 4 apontadores do tipo "*edge*" (aresta) que indicam a direção do próximo vértice. O vértice seguinte pode estar localizado à direita, à esquerda, acima ou abaixo ("*east, west, north, south*"). Se o apontador possui o seu valor menor do que 0, indica que naquela direção a borda do dispositivo foi encontrada.

A primeira versão do algoritmo foi implementada com a utilização das estruturas acima citadas. Após a realização de testes preliminares constatou-se que essa implementação apresentava um baixo desempenho (tempo de processamento elevado). Os circuitos demoravam muito tempo para serem roteados. Outro fator que contribui para procurar uma nova forma de modelar o FPGA foi que a biblioteca LEDA deixou de ser de domínio público, dificultando desta forma o desenvolvimento de uma ferramenta acadêmica de livre distribuição. A partir destes dois fatores partiu-se para uma nova forma de modelar o FPGA.

4.2.2 Matriz

O dispositivo programável FPGA foi então modelado com uma matriz de estruturas com 3 dimensões. O FPGA foi dividido em canais que estão na posição horizontal e canais que estão na posição vertical (Figura 39). Esse tipo de modelagem facilita a implementação do algoritmo de

roteamento por lidar com índices representados por números inteiros e também contém todas as informações de roteamento apontando por estes índices. Os índices da matriz são organizados da seguinte forma: [h/v][L][C]. O primeiro índice informa se o canal a ser visitado está na horizontal (“0”) ou na vertical (“1”), e os outros dois indicam a linha (L) e a coluna (C) no interior do dispositivo programável FPGA. Então por exemplo, o canal [0][0][0] está situado na horizontal, na linha 0 e na coluna 0.

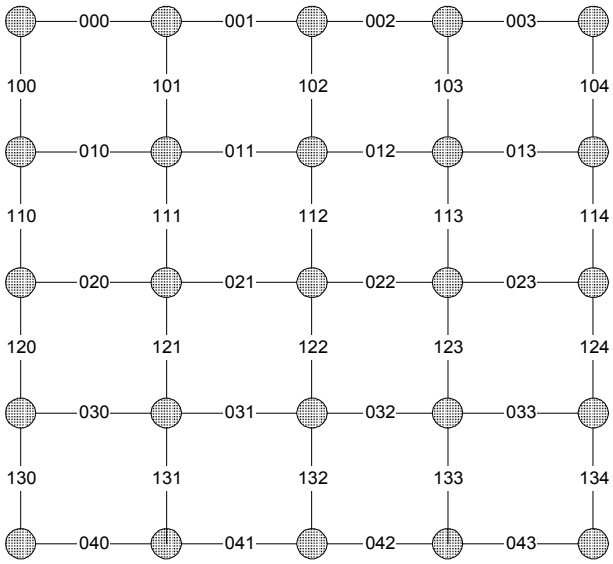


Figura 39 – FPGA modelado como uma matriz com 3 dimensões.

As estruturas utilizadas para esse tipo de modelagem estão representadas na Figura 40.

```
typedef struct {
    int Visited;
    int Counter;
    int nTrilhas;
    int Vagas;
    int *Trilhas;
}ap_canal;

typedef struct {
    char Nome[TAM];
    int nNome;
      int xmin;
      int ymin;
      int xmax;
      int ymax;
      int nTerms;
    ap_ponto *rede;
}ap_rede;
```

Figura 40 – Estruturas utilizadas para a implementação utilizando uma matriz.

A estrutura principal é chamada de “*ap_canal*” composta de 4 variáveis inteiras e um vetor de números inteiros. A variável “*Visited*” indica se o canal foi visitado ou não e a variável “*Counter*” é utilizada na propagação da frente de onda indicando qual o nível da onda que alcançou o canal. O número de trilhas vagas é indicado pela variável “*Vagas*”. A variável “*nTrilhas*” indica a

quantidade de trilhas existente no canal. Este valor é obtido após a leitura do arquivo de arquitetura do dispositivo. Essa estrutura tem um vetor de números inteiros chamado de "*Trilhas*", com o tamanho de "*nTrilhas*" alocado dinamicamente e é utilizada para armazenar o nome das redes que estão ocupando o canal, bem como as trilhas do canal que já estão ocupadas.

A estrutura "*ap_rede*" é inicializada na leitura do circuito, conforme a Figura 33.

No momento da execução do algoritmo de roteamento todas as estruturas são inicializadas de forma dinâmica com o objetivo de reduzir ao máximo a utilização de memória. A inicialização da matriz de estruturas está descrita na Figura 41. A entrada da função de inicialização é a estrutura que contém as informações da arquitetura do dispositivo. Após saber quais são as dimensões do FPGA, é feita a alocação dinâmica da primeira dimensão da matriz que indica se o canal está posicionado na horizontal ou na vertical (variável "*hv*"). Depois de alocar a primeira dimensão da matriz, as linhas da mesma são alocadas. E por fim, são alocadas as colunas da matriz.

```
Lê #linhas e #colunas
Aloca a primeira dimensão da matriz
Para hv=0 até hv menor que 2 {
  Enquanto #linhas + 1 != 0 {
    Aloca a segunda dimensão da matriz (linhas)
    Decrementa #linhas
    Enquanto #colunas + 1 != 0 {
      Aloca a terceira posição da matriz (colunas)
      Decrementa #colunas
    }
  }
}
```

Figura 41 – Rotina de inicialização da matriz com 3 dimensões

4.3 Algoritmo de Roteamento

A Seção 4.3.1 apresenta a implementação do algoritmo de roteamento do tipo Maze, a propagação de frente de onda sobre a matriz até que todos os pinos da rede sejam encontrados. A Seção 4.3.2 apresenta a retropropagação (ou "*Backtracking*") que é responsável por indicar e identificar todos os canais por onde a rede passa. Essa rotina de retropropagação retorna sempre a menor distância entre todos os pinos da rede. Na Seção 4.3.3, é apresentada a rotina de "*rip-up & re-route*". Se a lista de redes a ser roteada estiver vazia, mas pelo menos uma rede não for roteada, utiliza-se a rotina de "*rip-up & re-route*". Essa rotina re-inicializa a matriz que representa o FPGA, e monta uma nova lista de redes, para uma nova execução do algoritmo. A Figura 42 ilustra a rotina principal do algoritmo de roteamento.

Como pode-se observar na Figura 42, o algoritmo de roteamento pode ser dividido em 2 laços: principal e roteamento. O laço principal é controlado pelo de número de vezes que o roteamento deve ser aplicado sobre a lista de redes, a fim de conseguir rotear todas elas, definido no arquivo de arquitetura (Seção 4.1.1). Nesse laço é executada a rotina de "*rip-up & re-route*", somente quando pelo menos uma rede da lista de redes não foi roteada. A estrutura que representa o FPGA é re-inicializada e uma nova tentativa de roteamento é iniciada. No laço de roteamento são executadas a propagação da frente de onda e a rotina de retropropagação. As redes que são roteadas com sucesso são acrescentadas em uma *lista de redes roteadas* e as redes que não obtiveram

sucesso no roteamento são armazenadas na *lista de redes não roteadas*. O laço de roteamento só termina quando a lista de redes é vazia. Se todas as redes foram roteadas com sucesso, o algoritmo termina. Senão a *lista de redes roteadas* é acrescentada no final da *lista de redes não roteadas* e é feita uma nova tentativa de roteamento.

```
Enquanto Numero de iterações > 0 {
  Enquanto lista de redes não está vazia {
     Retira uma rede da lista de redes
     Enquanto lista de pinos da rede não está vazia {
        RedeAux = Roteia a rede
        Se RedeAux = NULO {
          Adiciona a rede na lista de redes não roteadas
        }
        Senão (
          Realiza a retropropagação ("backtracking")
          Adiciona a rede na lista de redes roteadas
        }
        Limpa rastros da frente de onda na matriz (FPGA)
     }
  }
  Se (Todas as redes foram roteadas ?) {
    Sai do laço principal
  }
  Senão {
    Re-ordena a lista de redes
    Limpa lista de redes roteadas e não roteadas
    Limpa toda a matriz
    Decrementa número de iterações
  }
}

Escreve arquivo de saída
Libera todas as estruturas alocadas
Fim
```

Figura 42 – Rotina principal do algoritmo de roteamento do tipo Maze

4.3.1 Maze

A primeira parte do algoritmo de roteamento do tipo *Maze* é a propagação da frente de onda a partir da *origem* da rede até que algum canal utilizado pela rede seja encontrado, ou seja, um *destino*. Antes de iniciar a propagação da frente de onda é necessário verificar se algum dos pinos de destino não está presente no próprio canal do pino de origem. Nessa etapa são criadas duas listas auxiliares: uma contendo os canais que estão sendo roteados (lista de roteamento) e uma que contém a lista de destinos (lista de destinos). Dessa forma é possível verificar se algum dos pinos de destino foi encontrado. A Figura 43 ilustra o funcionamento da propagação da frente de onda, o qual é detalhado nos parágrafos seguintes.

O primeiro passo do algoritmo é procurar os canais vizinhos à origem (nível 0 da frente de onda, variável "*counter*" da estrutura) e adicioná-los a lista de roteamento, com o nível de

propagação em 1. Se o canal estiver posicionado na horizontal, o mesmo pode ter 2 canais vizinhos na horizontal e 4 canais vizinhos na vertical. Se o canal estiver na posicionado na vertical, o mesmo pode ter 2 canais vizinhos na vertical e 4 canais vizinhos na horizontal. Cada canal de roteamento possui no mínimo 3 canais vizinhos, situação que ocorre na periferia do FPGA.

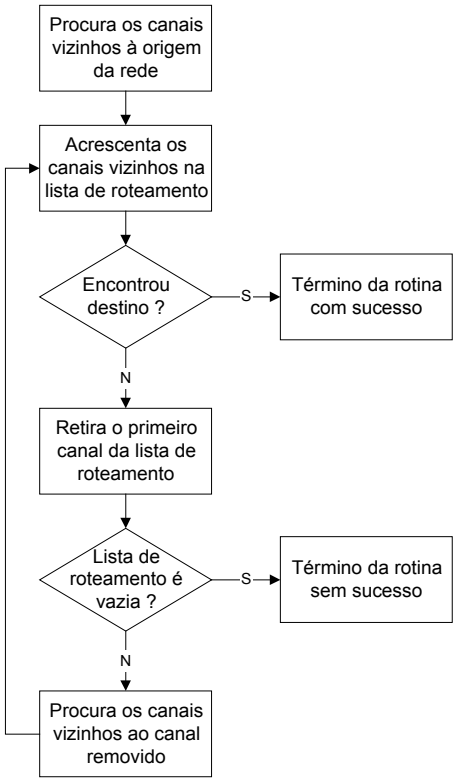
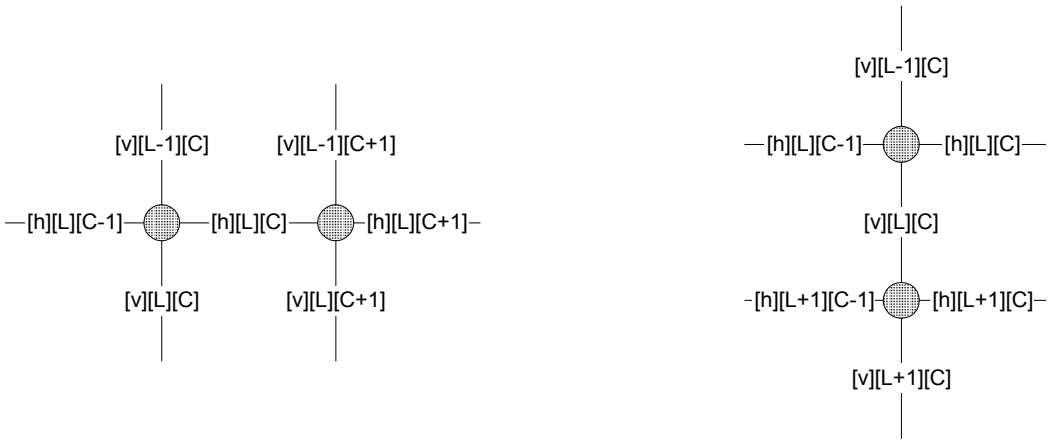


Figura 43 – Fluxograma da propagação da frente de onda.

A Figura 44a ilustra quais são as coordenadas dos canais vizinhos ao canal horizontal [h][L][C] e a Figura 44b as coordenadas dos canais vizinhos ao canal vertical [v][L][C].



(a) Vizinhos de um canal horizontal (b) Vizinhos de um canal vertical

Figura 44 – Método para encontrar as coordenadas dos canais vizinhos.

A Figura 45 apresenta a rotina de procura e validação dos canais vizinhos. Após encontrar os vizinhos deve-se verificar se as coordenadas vizinhas são válidas. As coordenadas inválidas referem-se a 5 condições: canal já visitado, canal sem trilha disponível, canal fora do quadrado envolvente da rede (*bounding box*), canal com coordenada negativa, ou canal com coordenada

maior que a dimensão do FPGA.

Esta rotina retorna uma lista com todos os vizinhos válidos de um dado canal. Esta lista é posteriormente acrescentada à *lista de roteamento*.

```
lista_vizinhos Encontra coordenadas vizinhas ( [h/v] [L] [C] ) {
lista_vizinhos = NULL;
Se canal é horizontal {
  append(lista_vizinhos, vizinho horizontal 1 [h][L][C-1] )
  append(lista_vizinhos, vizinho horizontal 2 [h][L][C+1] )
  append(lista_vizinhos, vizinho vertical 1 [v][L-1][C] )
  append(lista_vizinhos, vizinho vertical 2 [v][L-1][C+1] )
  append(lista_vizinhos, vizinho vertical 3 [v][L][C] )
  append(lista_vizinhos, vizinho vertical 4 [v][L][C+1] )
}
Se canal é vertical {
  append(lista_vizinhos, vizinho vertical 1 [v][L-1][C] )
  append(lista_vizinhos, vizinho vertical 2 [v][L+1][C] )
  append(lista_vizinhos, vizinho horizontal 1 [h][L][C-1] )
  append(lista_vizinhos, vizinho horizontal 2 [h][L][C] )
  append(lista_vizinhos, vizinho horizontal 3 [h][L+1][C] )
  append(lista_vizinhos, vizinho horizontal 4 [h][L+1][C-1] )
}

percorre lista_vizinhos
    se coordenadas inválida (ver parágrafo acima)
         remove vizinho da lista_vizinhos

retorna lista_vizinhos
}
```

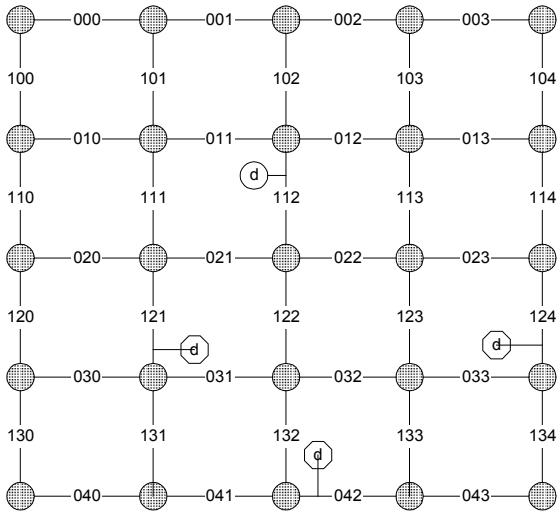
Figura 45 – Rotina de procura e validação dos canais vizinhos.

Para ilustrar este procedimento é utilizado como exemplo a rede "*d*" do circuito listado no Anexo II. Esta rede possui uma origem e três destinos, a Figura 46a apresenta a rede antes da conversão para coordenadas [h/v][L][C] ter sido realizada e a Figura 46b após a conversão. A Figura 47 apresenta a distribuição dos pinos da rede "*d*" no FPGA.

Addnet [d] R1C1.F3 R2C1.F1 R3C2.F4 R2C3.F3	**d** → 112 121 042 124
(a)	(b)

Figura 46 – Sintaxe da rede a ser roteada

O ponto de origem da rede "*d*" está localizado na coordenada "*112*", ou seja, o pino está localizado em um canal vertical, linha 1 e coluna 2 (tripla [h/v][L][C]). Os demais canais pertencentes a esta rede são armazenados na *lista de destinos*.



Obs.: Apesar do índice ser representado por 3 dígitos, na realidade são 3 índices independentes

Figura 47 – Posição da *origem* e dos *destinos* da rede "*d*"

Conforme a Figura 44, já realizamos os dois primeiros passos: construção da lista de canais vizinhos à origem e inserção desta lista na *lista de roteamento*. Esta lista de roteamento é apresentada na segunda coluna da Tabela 6, *primeira iteração.* Nesta lista não foi encontrado nenhum dos *destinos* da rede.

O primeiro canal é removido da *lista de roteamento*, inserindo-se nesta lista os canais vizinhos ao canal removido. A partir desta *lista de roteamento* modificada é feita uma nova busca por *destinos*. O procedimento é repetido até que seja encontrado um destino ou não seja mais possível propagar a frente de onda (*lista de roteamento* vazia).

A Tabela 6 mostra a lista de roteamento após as 2 iterações necessárias para que fosse encontrado o primeiro destino da rede "*d*".

Tabela 6 – Início e a primeira iteração do algoritmo.

Início	Primeira Iteração	Segunda Iteração
Canal [112] → 0 (Origem)	Canal [011] → 1	Canal [111] → 2
	Canal [012] → 1	Canal [013] → 2
		Canal [113] → 2
	Canal [122] → 1	Canal [132] → 2
		Canal [032] → 2
		Canal [031] → 2
	Canal [022] → 1	Canal [023] → 2
		Canal [123] → 2
	Canal [021] → 1	Canal [121] → 2 (Destino)

A Figura 48 representa no FPGA as duas iterações necessárias, no caso dessa rede, para que um de seus destinos seja encontrado. É possível observar que a primeira iteração encontra apenas 5 canais vizinhos para a origem, sendo que essa origem deveria ter 6 canais vizinhos (2

verticais e 4 horizontais). Isso acontece por causa da aplicação de limites (Seção 4.1.2) para a propagação da frente de onda, ou seja, a propagação só será realizada nos canais contidos na área tracejada. Esse limite ajuda a diminuir o tempo de processamento, pois não é necessário procurar os destinos fora do limite.

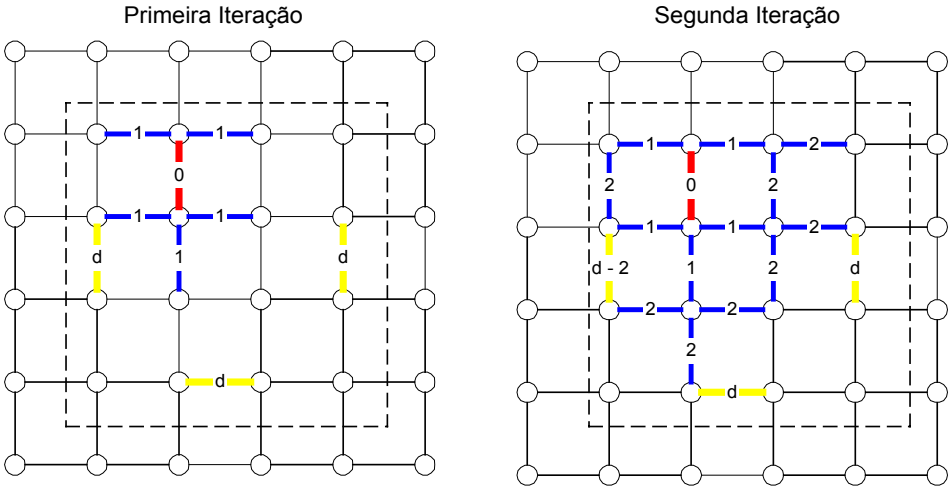


Figura 48 – Propagação da frente de onda.

Os vizinhos encontrados após a primeira iteração (nível 1 da frente de onda), são adicionados à lista de origens. Os canais vizinhos à origem passam a ser as origens para a propagação de mais um nível da frente de onda (nível 2). Antes de realizar uma nova propagação é necessário verificar se algum destes vizinhos é um canal de destino.

Quando um destino da rede é encontrado, tem início a fase de retropropagação ("*Backtracking*"), que será apresentada na Seção 4.3.2. Supondo que a retropropagação tenha sido realizada, o FPGA com a rede "*d*" parcialmente roteada é ilustrado na Figura 49.

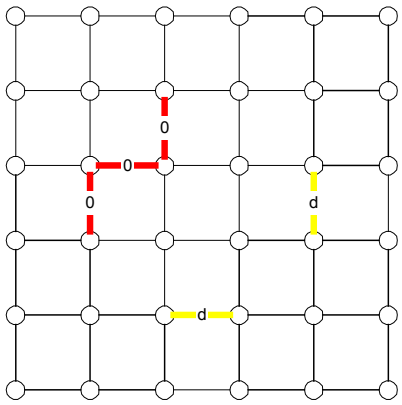


Figura 49 – FPGA com a rede "*d*" parcialmente roteada.

A partir da rede parcialmente roteada é iniciada uma nova propagação, onde as origens são os canais já roteados. Essa técnica é utilizada para que os destinos que restam sejam encontrados de uma forma mais eficiente. Ao invés da propagação partir de um único ponto (canal de origem da rede), propaga-se a partir de 3 canais. A Figura 50 mostra a propagação da frente de onda a partir da rede parcialmente roteada.

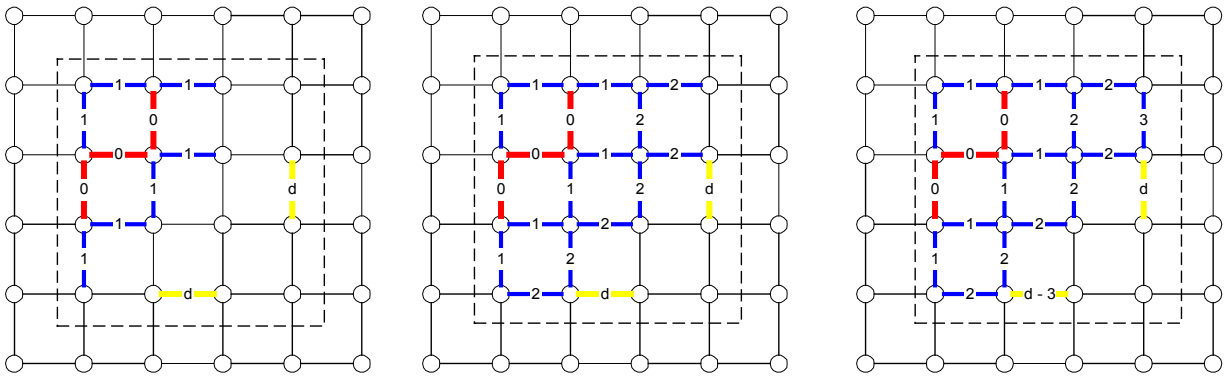


Figura 50 – Propagação da frente de onda a partir da rede parcialmente roteada

A Tabela 7 ilustra as 3 iterações necessárias para que o próximo *destino* da rede "*d*" seja encontrado. O canal [0][2][1] não possui nenhum vizinho para propagar a frente de onda, pois seus únicos vizinhos possíveis já tinham sido utilizados pelos outro canais da *lista de roteamento*.

Tabela 7 – Iterações para encontrar o próximo *destino* da rede "*d*".

Início	Primeira Iteração	Segunda Iteração	Terceira Iteração
Canal [112] → 0 (*Origem*)	Canal [011] → 1		
	Canal [012] → 1	Canal [013] → 2	Canal [114] → 3
		Canal [113] → 2	
	Canal [122] → 1	Canal [132] → 2	Canal [042] → 3 (*Destino*)
		Canal [032] → 2	
	Canal [022] → 1	Canal [023] → 2	
		Canal [123] → 2	
Canal [121] → 0 (*Origem*)	Canal [111] → 1		
	Canal [131] → 1	Canal [041] → 2	
	Canal [031] → 1		
Canal [021] → 0 (*Origem*)			

É possível também notar que a propagação ocorre de forma homogênea sobre o FPGA e obedecendo aos limites estabelecidos pelo tamanho da rede. Pode-se observar também que os dois destinos restantes encontram-se à mesma distância dos canais de origem. Sendo assim, apenas um deles será encontrado nessa propagação. No momento em que um dos destinos for encontrado a propagação da frente de onda será interrompida e o outro destino ficará para a próxima propagação. Isso ocorre porque para cada propagação com sucesso, temos apenas uma execução da rotina de retropropagação. Não há critério de escolha para qual desses destinos deve ser roteado primeiro.

A Figura 51 mostra como ficou a rede "*d*" roteada pelo algoritmo descrito.

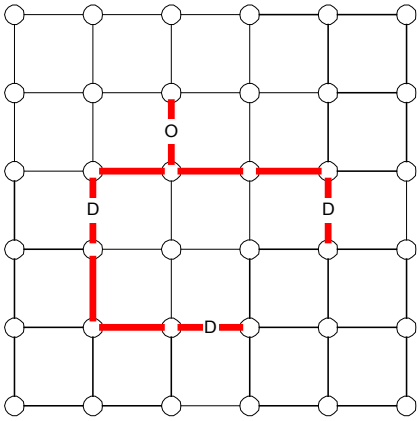


Figura 51 – Rede roteada pelo algoritmo Maze.

No Anexo II é descrito o circuito completo, de onde a rede "*d*" foi retirada apenas para ilustrar o roteamento pelo algoritmo proposto. As coordenadas do circuito convertidas e o seu roteamento completo podem ser observados nesse Anexo. O Anexo III apresenta a tela de execução do algoritmo proposto.

4.3.1.1 Propagação com e sem restrição de matrizes de chaves

A propagação sem restrição de matrizes de chaves, ou seja, $f_s = \infty$, não necessita verificar se as trilhas vagas na *origem* da rede permanecem vagas em todos os canais vizinhos. Quando não utilizada a restrição de matrizes de chaves, a área de propagação torna-se maior, pois qualquer segmento da rede pode utilizar qualquer trilha do canal. Para exemplificar será utilizado o circuito descrito no Anexo II. A Figura 52 representa a parte "*congestionada*" do circuito "*paper.lca*". Para a realização do mesmo não foi utilizada nenhuma restrição de matriz de chaves ($f_s = \infty$) e com a largura de canal igual a 2. Se não for aplicada a restrição o circuito pode ser roteado.

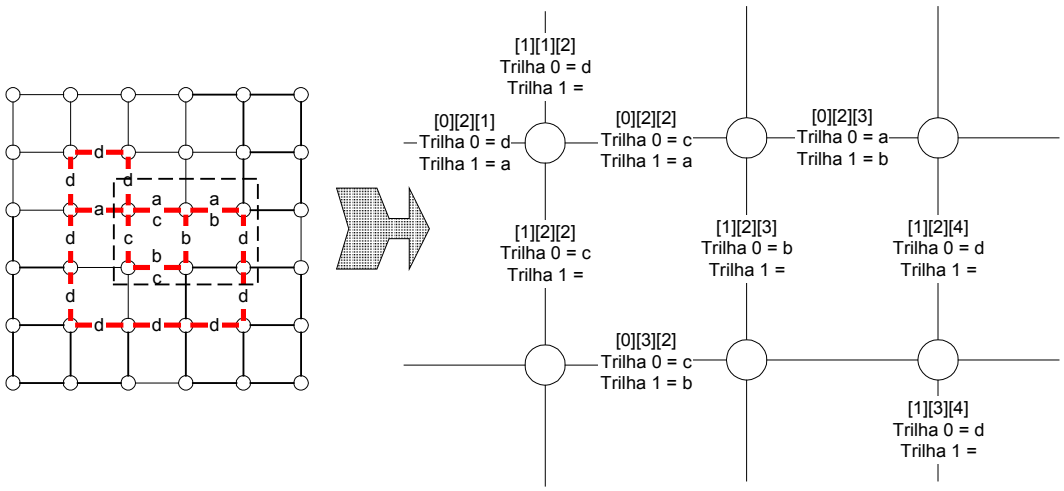


Figura 52 – Circuito exemplo roteado sem a restrição de matrizes de chaves.

Como é possível observar na Figura 52, a rede "*a*" ocupa a trilha 0 do canal [0][2][3] e ocupa a trilha 1 dos canais [0][2][1] e [0][2][2]. Se a restrição de matriz de chaves for aplicada o algoritmo esgota todas as tentativas de roteamento definidos no arquivo de arquitetura.

Quando a restrição de matrizes de chaves é aplicada ($f_s \neq \infty$) deve-se verificar se os canais vizinhos à *origem* e posteriormente todos os canais que vão compor a rede possuem a mesma trilha vaga. Pode acontecer de o canal possuir trilhas vagas, mas se não forem exatamente as mesmas trilhas vagas nos demais canais encontrados, o canal deve ser desconsiderado, porque não há caminho disponível para que possa encontrar um destino. A Figura 53 apresenta a parte "*congestionada*" do circuito descrito no Anexo II. A largura teve de ser aumentada em uma unidade (3), para que o circuito pudesse ser roteado aplicando as restrições de matriz de chaves.

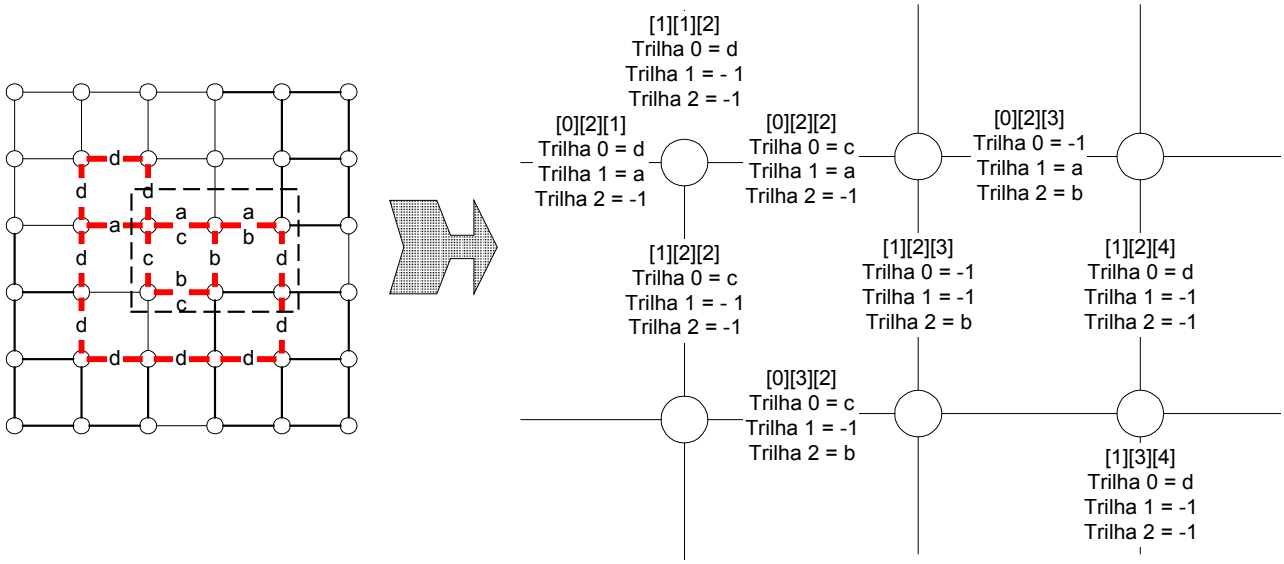


Figura 53 – Circuito exemplo roteado com restrição de matriz de chaves.

Na Figura 53 pode-se observar que cada rede ocupa sempre o mesmo canal em todos os seus componentes (canais).

4.3.2 Retropropagação (*Backtracking*)

A retropropagação possui esse nome por realizar a propagação da frente de onda na ordem inversa (destino até a origem da rede que está sendo analisada). Na propagação da frente de onda (origem até destino), descrita na Seção 4.3.1, a frente de onda se espalha pelos canais numerando-os em ordem crescente, onde a origem recebe o número 0. A retropropagação é dividida em duas partes: procura de vizinhos (canais) com nível de propagação menor do que o canal que está sendo analisado (rotina "*BackTracking*" da Figura 54) e a sinalização da rede nos canais (rotina "*Preenche Estrutura_FPGA*" da Figura 54), roteando assim a rede.

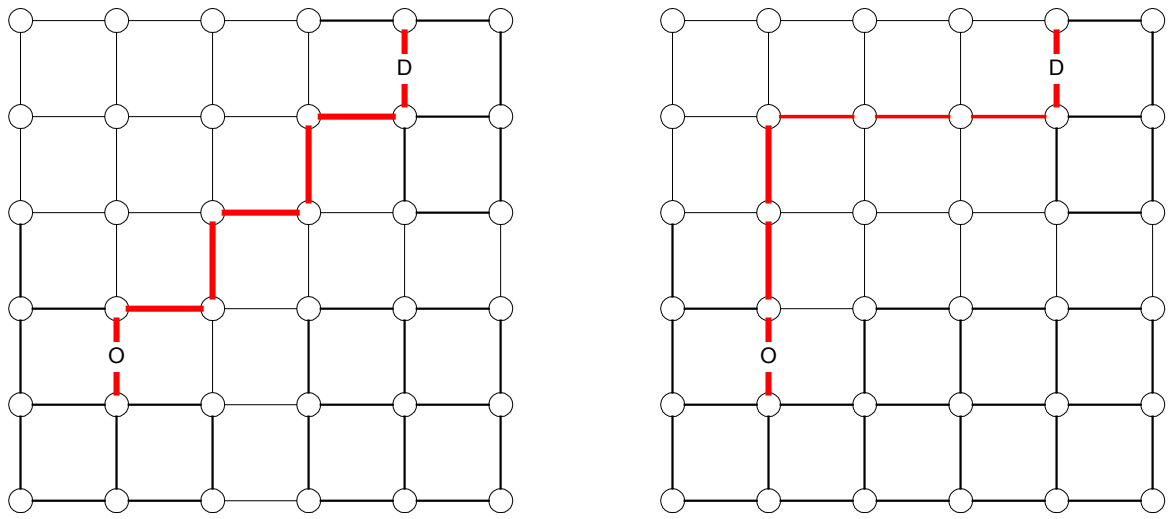
As restrições aplicadas a retropropagação são: o nível do canal atual deve ser menor que o nível do canal anterior, o canal deve ter sido visitado pela primeira frente de onda, o canal analisado deve possuir pelo menos uma trilha vaga e deve-se evitar a troca de direções nos canais para que sejam usados os segmentos longos de fio (não utilizados na presente implementação). Essa restrição melhora a desempenho do circuito, diminuindo o atraso da rede. Esse atraso é causado pela capacitância existente nas matrizes de chaves do FPGA. Quanto maior o número de matrizes de chaves possuir a rede em todo o seu percurso, maior será o atraso.

```
lista_roteada BackTracking(lista_propagação) {
    Percorre lista_propagação {
      Se (nível de propagação do vizinho for menor do que nível de propagação &&
        vizinho já foi visitado &&
        vizinho possui ao menos uma trilha vaga no canal) {
           Adiciona vizinho na lista_roteada
           nível de propagação = nível de propagação do vizinho
       }
       Se (nível de propagação == 0)
           sai do laço
    }
    Retorna lista_roteada
}

Preenche Estrutura_FPGA(lista_roteada) {
    Enquanto(lista_roteada não é vazia) {
       coloca nome da rede no canal
       diminui o número de trilhas vagas no canal
    }
    grava a rede roteada no arquivo de saída
}
```

Figura 54 – Pseudo-código da retropropagação.

A Figura 55a apresenta uma rede roteada sem a aplicação da restrição de troca de direções e a Figura 55b apresenta a mesma rede roteada utilizando a restrição de troca de direções.



(a) (b)

Figura 55 – Restrição de troca de direções dos canais.

A primeira parte é iniciada logo após a propagação da frente de onda, da origem até um dos destinos da rede. É nessa fase que são aplicadas duas das três restrições para a retropropagação. A primeira restrição aplicada é a do nível de propagação, que deve ser menor nos canais vizinhos ao que está sendo analisado. A segunda restrição é que os canais vizinhos devem ter sido "visitados" pela frente de onda (da origem até um dos destinos). Para demonstrar mais claramente como a

retropropagação funciona será utilizado o mesmo exemplo da Seção 4.3.1. A Figura 56 ilustra como é realizada a primeira etapa da retropropagação sobre o exemplo da rede “d”, descrita na Seção anterior.

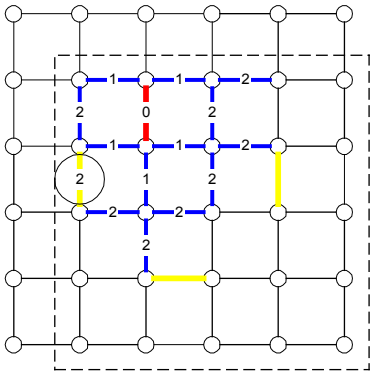


Figura 56 – Exemplo da retropropagação

O canal marcado com um círculo é o primeiro destino que foi encontrado na propagação da frente de onda. É possível observar que o mesmo possui o nível de propagação igual a 2. A procura dos vizinho é facilitada pela observação das restrições. O canal destino possui 3 canais vizinhos que podem indicar o caminho até a origem, pois os demais não foram visitados pela frente de onda, por estarem fora da área de procura da rede. Aplicando-se as duas restrições dessa etapa, conclui-se que o canal analisado possui dois canais vizinhos com o nível de propagação igual ao que está sendo analisado, e apenas um indicando o nível de propagação menor do que o nível de propagação inicial. A retropropagação é interrompida quando a origem da rede é encontrada.

O passo seguinte, após a origem ter sido encontrada, é a verificação da existência de trilhas vagas no canal. Os canais que formam a rede são armazenados em uma lista, sendo que essa lista é percorrida até que pelo menos uma trilha vaga seja encontrada. Tendo encontrado uma trilha vaga, o contador de trilhas vagas (“*Vagas*”, Seção 4.2.2) é diminuído de uma unidade.

Após saber que o canal possui ao menos uma trilha vaga, é necessário saber qual a trilha que pode ser utilizada pela rede. Depois de encontrada a trilha disponível, o nome da rede é colocado no vetor de trilhas (“Trilhas”, Seção 4.2.2). A rede “*d*” parcialmente roteada é apresentada na Figura 49 da Seção 4.3.1.

Nesta fase é que se aplica a restrição de canal (f_s). Se a restrição de canal for aplicada, todos os pontos da rede devem ser analisados a fim de que seja encontrada a mesma posição vaga no vetor de trilhas, para que o circuito seja completamente roteado. Se essa restrição não for aplicada, as trilhas são escolhidas em ordem seqüencial, não importando se todos os pontos do circuito utilizam a mesma trilha no canal. Sendo assim o espaço de procura para se rotear uma rede é reduzido.

Após a rede ser completamente roteada é necessário realizar a limpeza das estruturas. Para realizar a limpeza, as variáveis “*counter*” e “*routing*” recebem o valor 0. Tendo cuidado para não alterar a variável que informa a quantidade de trilhas vagas no canal (“*Vagas*”) e o vetor que indica os nomes das redes presentes no canal (“*Trilhas*”).

4.3.3 *Rip-up* & *re-route*

A técnica de "*rip-up & re-route*" é relativamente simples de ser implementada e é muito utilizada em algoritmos de roteamento. O "*rip-up & re-route*" é executado quando o circuito não pode ser totalmente roteado no FPGA. No "*rip-up*" é realizada uma "*limpeza geral*", sendo toda a estrutura que representa o FPGA reinicializada. Esse processo é chamado assim, por desfazer todas as conexões (redes) que obtiveram sucesso na fase de roteamento.

A primeira iteração do roteamento ocorre com a "*lista de redes*" ordenada conforme o "*fanout*" de cada rede, ou seja, número de trilhas utilizadas para rotear a mesma. Essa lista é ordenada de forma decrescente, pois redes com grande "*fanout*" podem ser bloqueadas pelas redes com "*fanout*" menor. Assim sendo, redes com alto "*fanout*" devem ser roteadas primeiro e assim sucessivamente.

Durante o processo de roteamento cada rede que vai sendo roteada com sucesso é inserida em uma lista denominada de "*lista de redes roteadas*". Por sua vez, as redes que não são roteadas são armazenadas em outra lista chamada de "*lista de redes não roteadas*". Ao final do roteamento, é verificado se todas as redes circuito foram roteadas, ou seja, se a "*lista de redes não roteadas*" é vazia. Se essa lista não estiver vazia, é necessario que as redes sejam reordenadas e inseridas na "*lista de roteamento*".

Essa reordenação é realizada da seguinte forma: todas as redes que não foram roteadas são inseridas na "*lista de roteamento*", para que sejam as primeiras redes a serem roteadas. E a "*lista de redes roteadas*" é inserida no final da "*lista de roteamento*". É imperativo para que o roteamento seja concluído com sucesso, que essa ordenação seja feita na forma descrita acima. Pode acontecer de uma rede muito longa, bloquear redes menores. Como conseqüência da nova ordenação, as redes não roteadas na primeira tentativa serão as primeiras a serem roteadas. Após estas terem sido roteadas, seque o roteamento das redes que foram roteadas com sucesso. O Anexo IV mostra a execução do algoritmo de roteamento no circuito de teste "*seq.lca*" e a execução do "*rip-up & re-route*".

O número de tentativas que o roteamento deve ser realizado, no caso do algoritmo apresentado (Seção 4.3), é definido no arquivo de arquitetura (Seção 4.1.1). Se o número de tentativas for excedido, então o circuito não pode ser roteado com sucesso. Nesse caso, sugere-se o uso de três alternativas: (i) aumentar o tamanho do FPGA (número de linhas e colunas) e não utilizar a restrição imposta pelo "*quadrado envolvente*", pois essa restrição diminui a área de procura do algoritmo; (ii) aumentar o número de trilhas disponíveis no FPGA; (iii) e por fim aumentar o número de vezes que o "*rip-up & re-route*" deve ser executado.

A Figura 57 apresenta a evolução do número de trilhas roteadas para o circuitos de teste "*seq.lca*", descrito em [YAN91]. Este circuito possui 1791 redes e a rede com maior "*fanout*" possui 234 pinos de conexão do Bloco Lógico com o canal de roteamento. Esta Figura mostra que foram necessárias 7 iterações para que o circuito fosse completamente roteado. Sendo assim é possível concluir que a quantidade de redes não roteadas tende a estabilizar ao longo das iterações que são efetuadas no circuito, até que o circuito seja totalmente roteado. Se número de redes roteadas não ficar estável, a reordenação do "*rip-up & re-route*" não está sendo executada corretamente.

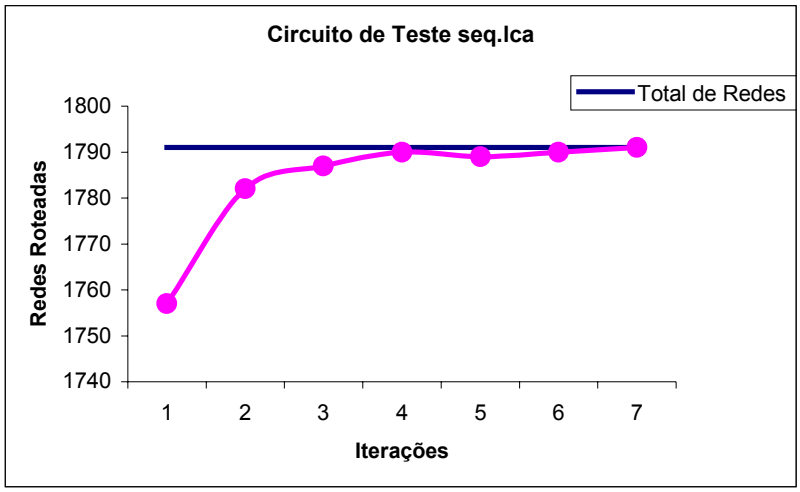


Figura 57 – "*Rip-up & re-route*" executado no circuito de teste "*seq.lca*".

Para ilustrar melhor este processo, será utilizado o exemplo descrito pela Figura 58, a tela de execução e as estatísticas de roteamento do mesmo são apresentadas no Anexo III.

Circuito *paper2.lca*	Coordenadas Convertidas
Addnet [a] R1C1.F4 R1C3.F4	[0] - [a] → 021 023
Addnet [b] R0C0.F1 R3C3.F4 R1C3.F3 R3C0.F2 R1C2.F3 R2C2.F1	[1] - [b] → 100 043 114 030 113 122
Addnet [c] R3C0.F1 R0C3.F2	[2] - [c] → 130 003

Figura 58 – Circuito exemplo paper.lca

A Figura 59 apresenta a localização das redes do circuito no FPGA. As origens das redes são os assinalados com um círculo e os destinos são representados pelos hexágonos. A largura de canal para esse circuito é igual a **1 (um)**.

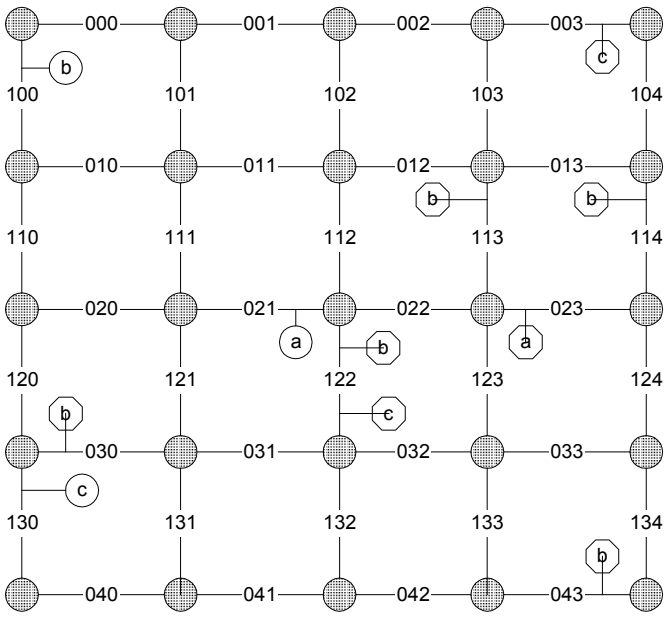


Figura 59 – Localização das *origens* e *destinos* do circuito exemplo.

Na Figura 60, os canais [0][2][1] e [0][2][3] são a *origem* e o *destino*, respectivamente da rede "a". A rede "a" não pode ser roteada porque teria de utilizar a trilha 0 do canal [0][2][2], mas a trilha 0 já está sendo ocupada pela rede "b". A rede "b" já tinha sido roteada conforme é apresentada na Figura 60. É possível observar também que na *origem* e no *destino* dessa rede existem trilhas vagas no canal.

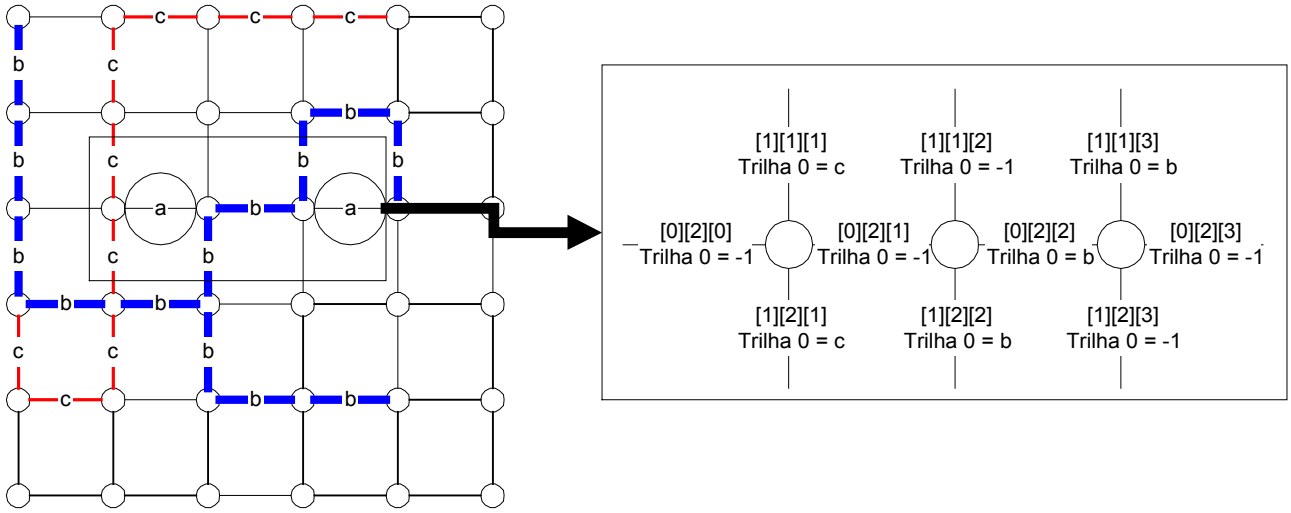


Figura 60 – Exemplo da técnica de "Rip-up & re-route".

A Tabela 8, apresenta a primeira tentativa de roteamento que não obteve sucesso e segunda tentativa, conseguindo assim que todo o circuito fosse roteado no FPGA. Após o "*rip-up & re-route*" toda as trilhas são roteadas com sucesso.

Tabela 8 – Execução do "*rip-up & re-route*".

1ª Tentativa	2ª Tentativa
[b] – [1] → 100 110 120 030 031 122 132 022 042 043 113 013 114	[a] – [0] → 021 022 023
[a] – [0] → Não pode ser roteada.	[b] – [1] → 100 110 120 030 031 122 032 123 133 013 043 114
[c] – [2] → 130 040 131 121 111 101 001 002 003	[c] – [2] → 130 040 131 121 111 101 001 002 003

A Figura 61 apresenta o circuito completamente roteado. Pode-se observar que a rede "b" precisou utilizar um canal a mais, do que o roteamento realizado pela primeira tentativa.

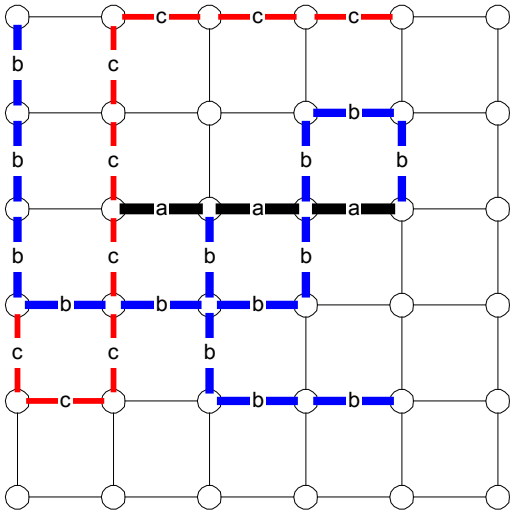


Figura 61 – Circuito completamente roteado após a execução do "*rip-up & re-route*".

4.4 Avaliação do algoritmo desenvolvido

Na Seção 4.4.1 são apresentadas as características dos circuitos de teste ("*benchmarks*") propostos por [YAN91]. A Seção 4.4.2 apresenta os resultados obtidos pelo algoritmo de roteamento descrito na Seção 4.3. Os resultados apresentados foram realizados com os dois tipos de modelagem descritos na Seção 4.2.

4.4.1 Circuitos de Teste ("*benchmarks*")

Os circuitos de teste utilizados para realizar a validação das estruturas de dados descritas anteriormente e do algoritmo proposto fazem parte dos exemplos utilizados em 1991 no "*MCNC International Workshop on Logic Synthesis*" [YAN91].

Na Tabela 9 são apresentadas as características dos circuitos de teste. Essas características são: número de pinos de entrada do circuito, número de pinos de saída do circuito e o número de redes que compõem os mesmos. O maior circuito, em número de redes, é o "*clma*" que possui 8445 redes. Dentre os circuitos de teste apresentados na Tabela 9, é possível destacar circuitos que são máquinas de estados finito (ex. s298), circuitos seqüenciais com lógica multi-nível (ex. clma), circuitos combinacionais multi-nível (ex. alu4) e circuitos combinacionais com lógica de 2 níveis (ex. apex2).

É importante analisar o *fanout* das redes dos circuitos, pois o roteador ordena as redes conforme esse parâmetro. O algoritmo implementado inicia o roteamento com as redes que possuem maior *fanout*. A Tabela 10 ilustra o percentual de "*fanout*" que cada um dos circuitos de teste.

Tabela 9 – Circuitos de Teste (MCNC).

Nome do Circuito	Nº de Entradas	Nº de Saídas	Nº de Redes
alu4	14	8	1536
apex2	39	3	1916
apex4	9	19	1271
bigkey	262	197	1936
clma	382	82	8445
des	256	245	1847
diffeq	N.D.*	N.D.*	1562
dsip	228	197	1599
elliptic	N.D.*	N.D.*	3735
ex1010	10	10	4608
ex5p	8	63	1072
frisc	N.D.*	N.D.*	3576
misex3	14	14	1411
pdc	16	40	4591
S298	3	6	1935
S38417	28	100	6435
S38584.1	38	304	6485
seq	41	35	1791
spla	16	46	3706
tseng	N.D.*	N.D.*	1099

*N.D. → Não Disponível

Tabela 10 – Percentual de fanout das redes dos circuitos de teste

Nome	% fanout									
	2	3	4	5	6	7	8	9	10	> 10
alu4	83 %	4 %	3 %	2 %	2 %	1 %	1 %	1 %	0 %	3 %
apex2	74 %	7 %	4 %	4 %	2 %	2 %	1 %	0 %	1 %	5 %
apex4	65 %	11 %	7 %	5 %	5 %	3 %	1 %	0 %	0 %	3 %
bigkey	70 %	17 %	0 %	0 %	0 %	0 %	2 %	10 %	0 %	0 %
Clma	73 %	9 %	5 %	3 %	1 %	2 %	1 %	1 %	1 %	4 %
des	72 %	9 %	3 %	2 %	3 %	1 %	1 %	1 %	1 %	7 %
diffeq	59 %	21 %	6 %	3 %	2 %	2 %	2 %	1 %	0 %	4 %
dsip	71 %	14 %	0 %	2 %	12 %	0 %	0 %	0 %	0 %	0 %
elliptic	58 %	32 %	5 %	0 %	0 %	0 %	0 %	0 %	0 %	4 %
ex1010	70 %	4 %	6 %	5 %	4 %	3 %	2 %	2 %	1 %	1 %
ex5p	55 %	13 %	11 %	6 %	5 %	3 %	2 %	1 %	1 %	3 %
frisc	66 %	18 %	4 %	2 %	2 %	0 %	0 %	0 %	0 %	7 %
misex3	76 %	6 %	4 %	3 %	1 %	1 %	1 %	1 %	1 %	6 %
pdc	77 %	9 %	3 %	2 %	1 %	1 %	0 %	0 %	0 %	6 %
s298	98 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	2 %
s38417	57 %	11 %	8 %	6 %	7 %	3 %	2 %	1 %	1 %	4 %
s38584.1	69 %	6 %	9 %	4 %	3 %	2 %	1 %	1 %	0 %	5 %
seq	75 %	8 %	4 %	2 %	1 %	1 %	1 %	1 %	0 %	6 %
spla	79 %	7 %	2 %	2 %	1 %	2 %	1 %	0 %	1 %	6 %
tseng	44 %	34 %	4 %	5 %	3 %	3 %	2 %	2 %	1 %	3 %

A Figura 62 apresenta a relação entre o número de pinos que compõem cada rede e o número de redes que formam o circuito. Pode-se notar que as redes com menor "*fanout*" são a maioria.

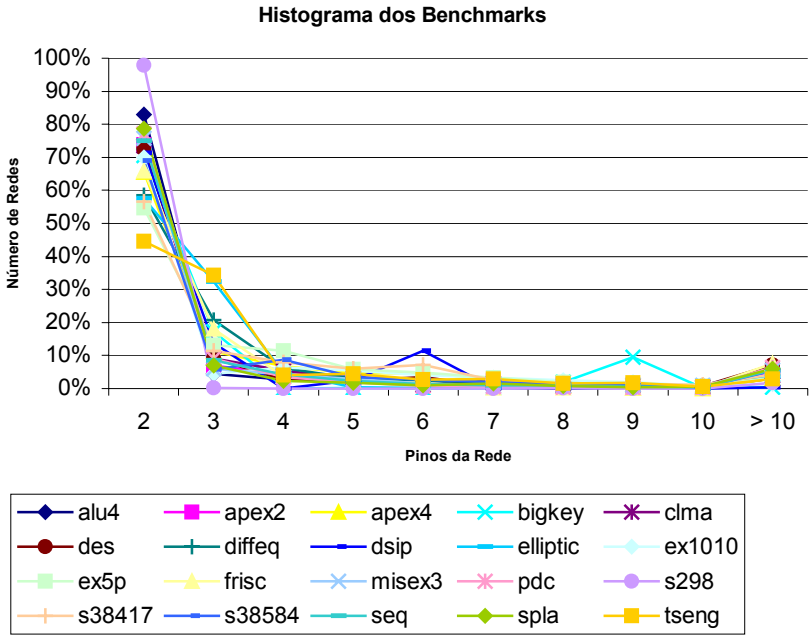


Figura 62 – Gráfico Fanout x Número de redes.

Analisando os dados contidos na Tabela 10 e a Figura 62, é possível observar que as redes multiponto são minoria em todos os circuitos. Estas redes são as mais difíceis de rotear, pois são as mais suscetíveis de serem bloqueadas. Por essa razão, deve-se antes de iniciar o roteamento, ordenar as redes em ordem decrescente de "*fanout*". Isso não quer dizer que, por a rede ter o maior "*fanout*" de todo o circuito, ela seja a mais difícil de ser roteada, pois os pinos podem estar muito próximos. Assim como, pode-se ter uma rede com "*fanout*" igual a 2, na qual a *origem* e o *destino* estarem em posições opostas no dispositivo. Nesse caso, a rede terá de cruzar praticamente todo o circuito para que a mesma seja roteada com sucesso.

Em todos os circuitos de teste, foi retirada a rede de *clock* por esta utilizar um canal de roteamento dedicado, sendo utilizado para distribuir sinal de *clock* em todo o FPGA. Isso é observado na família Virtex da Xilinx (ver. Figura 14, na Seção 2.2.1). Outras redes globais poderiam ser excluídas, como por exemplo, rede de *set/reset*, pois existem também recursos de roteamento dedicados para esse fim no FPGA.

4.4.2 Resultados obtidos

A Tabela 11 ilustra os resultados obtidos utilizando o algoritmo proposto com a modelagem por grafos.

A Tabela 12 e a apresentam os resultados obtidos utilizando a estrutura do FPGA sendo modelada através de uma matriz com 3 dimensões. Sendo o roteador executado em sistemas operacionais diferentes (*Windows 2000* e *Unix*).

Tabela 11 – Tabela de resultados do algoritmo com o FPGA modelado como um grafo (versão LEDA).

Nome do Arquivo	Número de Trilhas	Ltotal*	CPU (hh:mm:ss) Sun Sparc 20	R & R**	Dimensão	%RR ut.***	Número de Redes	VPR
alu4	**9**	18204	0:29:30	8	40x40	61,67%	1536	10
apex2	**10**	26068	0:54:54	9	44x44	65,83%	1916	11
apex4	12	18285	0:6:31	2	36x36	57,20%	1271	12
Bigkey	**6**	16059	5:0:54	19	54x54	45,06%	1936	7
clma	12	111129	19:57:43	7	92x92	54,12%	8445	12
des	7	20803	2:45:53	23	63x63	36,85%	1847	7
diffeq	8	13783	0:13:23	3	39x39	55,22%	1562	7
dsip	7	13587	2:11:23	5	54x54	32,68%	1599	7
elliptic	10	39721	4:15:63	4	61x61	52,51%	3735	10
ex1010	10	60978	4:41:24	7	68x68	64,98%	4608	10
ex5p	**12**	16112	0:11:10	4	33x33	59,83%	1072	13
frisc	11	47618	4:18:42	9	60x60	59,14%	3576	11
misex3	10	18369	0:16:35	4	38x38	61,97%	1411	10
pdc	16	81562	1:58:9	3	68x68	54,32%	4591	16
s298	7	18401	2:48:10	9	44x44	66,38%	1935	7
s38417	8	56802	5:33:26	5	81x81	53,45%	6435	8
s38584.1	9	54471	16:23:21	3	81x81	45,56%	6485	9
Seq	11	23970	0:16:4	3	42x42	60,33%	1791	11
Spla	13	55045	1:1:36	3	61x61	55,98%	3706	13
tseng	6	9158	0:19:4	7	33x33	68,02%	1099	6
Total	**194**		**73:43:51**					**197**

*Número Total de trilhas utilizadas / **Número de repetições / ***Porcentagem dos recursos de roteamento utilizados

Tabela 12 – Tabela de resultados do algoritmo com o FPGA modelado como uma matriz (versão atual).

Nome do Arquivo	Número de Trilhas	Ltotal*	CPU (hh:mm:ss) Pentium III 800	R & R**	Dimensão	%RR ut.***	Número de Redes	VPR
alu4	9	18167	0:05:40	4	42 x 42	53,34%	1536	10
apex2	10	26073	0:13:22	6	46 x 46	57,79%	1916	11
apex4	11	18329	0:01:46	3	38 x 38	53,41%	1271	12
bigkey	6	17074	0:41:58	15	56 x 56	43,04%	1936	7
clma	12	111261	1:12:38	2	94 x 94	50,83%	8445	12
des	7	20902	0:02:09	7	65 x 65	33,76%	1847	7
diffeq	8	14595	0:01:27	3	41 x 41	50,51%	1562	7
dsip	7	14465	0:07:28	3	56 x 56	31,25%	1599	7
elliptic	10	41760	0:34:51	4	63 x 63	50,19%	3735	10
ex1010	10	60962	0:24:02	3	70 x 70	59,63%	4608	10
ex5p	12	16131	0:00:59	4	35 x 35	50,46%	1072	13
frisc	11	49451	0:29:46	7	62 x 62	55,75%	3576	11
misex3	10	18361	0:01:36	3	40 x 40	53,31%	1411	10
pdc	16	81441	0:13:57	2	70 x 70	49,79%	4591	16
s298	7	18555	0:11:01	4	46 x 46	58,75%	1935	7
s38417	8	59422	0:44:14	3	83 x 83	52,02%	6435	8
s38584.1	8	57473	2:12:29	2	83 x 83	50,31%	6485	9
seq	10	24054	0:01:58	3	44 x 44	58,10%	1791	11
spla	13	54968	0:06:12	2	63 x 63	50,82%	3706	13
tseng	6	10014	0:07:42	26	35 x 35	62,65%	1099	6
Total	**191**		**7:35:15**					**197**

*Número Total de trilhas utilizadas / **Número de repetições / ***Porcentagem dos recursos de roteamento utilizados

A etapa de posicionamento foi realizada pela ferramenta acadêmica de posicionamento e roteamento VPR. Após o posicionamento ter sido realizado, houve a necessidade de converter o arquivo posicionado em uma lista de redes, para isso foi utilizado o conversor "*net2lca*" desenvolvido durante este trabalho.

Para a obtenção dos dados contidos nas tabelas acima não foi aplicada a restrição de matriz de chaves, ou seja, $f_s = \infty$. Os testes utilizando a modelagem por grafos apresentaram uma redução muito pequena no número de trilhas. O VPR utiliza 197 trilhas e a implementação descrita na Seção 4.2.1 utilizou 194 trilhas. Esta melhoria pode ser vista na Tabela 11, nos circuitos *Alu4*, *apex2*, *bigkey* e *ex5p*, que precisaram de uma trilha a menos do que o VPR, para que fosse completamente roteado. Quando os circuitos foram testados com a modelagem por matriz (Seção 4.2.2), observou-se que foi possível reduzir o número de trilhas para 191, conforme os dados apresentados na Tabela 12. Os circuitos onde foi obtida essa melhora foram: *alu4*, *apex2*, *apex4*, *bigkey*, *ex5p*, *s38584.1* e *seq*.

Tendo como base os dados da Tabela 11 e da Tabela 12, pode-se observar que o número de repetições ("*Rip-up and Re-route*") foi reduzido em praticamente todos os circuitos. Com a utilização da representação do FPGA na forma de uma matriz de estruturas reduziu-se o tempo de execução em 90%.

O circuito *Apex4* foi roteado aplicando-se ao roteamento a restrição de trilhas ($f_s = 3$) e retirando-se a restrição imposta pelo quadrado envolvente da rede. Para que o roteamento obtivesse sucesso, foi necessário que o número de trilhas disponíveis por canal fosse aumentado em 17 trilhas, ou seja, passou de 12 trilhas por canal para 29 trilhas por canal. Nesse circuito 3% das redes possuem o "*fanout*" maior do que 10, ou seja, 39 redes possuem mais de 10 pinos de conexão. Sendo que dessas 39 redes, 32 possuem mais do que 50 pinos de conexão.

Durante a implementação do algoritmo de roteamento com modelagem através de matriz de estruturas foi implementado um visualizador de redes. Este visualizador permite ver como a rede ficou distribuída no FPGA. Dessa forma, pode-se verificar que o roteador implementado está realizando o roteamento de circuitos digitais corretamente. O Anexo V apresenta a tela de execução do aplicativo de visualização de redes roteadas.

A Tabela 13 apresenta a comparação entre as ferramentas acadêmicas de roteamento VPR, VPR Timing Driven e Maze, sendo a última implementada no decorrer deste trabalho. Pode-se notar que o número de trilhas por canal aumentou quando foi utilizada a nova versão do VPR que visa otimizar o desempenho do circuito, ou seja, diminuindo o atraso relativo nas redes que compõem os circuitos de teste.

Tabela 13 – Comparação dos tempos de execução dos roteadores VPR, VPR *Timing Driven* e Maze (roteador implementado) .

Circuito	VPR	D_{max}^{*}	VPR - Time	D_{max}^{*}	Maze	D_{max}^{*}
alu4	0:44:53	10	4:13:33	11	0:05:40	10
apex2	1:19:19	10	5:26:09	12	0:13:22	11
apex4	1:07:01	12	1:24:24	15	0:01:46	12
Bigkey	0:36:32	6	8:38:37	6	0:41:58	7
clma	14:49:35	11	22:14:46	14	1:12:38	12
des	0:53:01	7	1:17:01	8	0:02:09	7
diffeq	0:28:22	8	0:47:27	8	0:01:27	7
dsip	0:23:33	7	1:15:48	7	0:07:28	7
elliptic	3:04:31	9	11:30:43	12	0:34:51	10
ex1010	3:35:37	10	13:11:04	11	0:24:02	10
exp5	1:05:43	12	1:18:24	15	0:00:59	13
frisc	4:33:27	11	6:29:38	15	0:29:46	11
misex	0:45:23	10	2:35:52	11	0:01:36	10
pdc	16:10:02	16	16:59:56	20	0:13:57	16
s298	0:30:20	7	2:50:28	8	0:11:01	7
s38417	2:56:46	7	8:40:19	8	0:44:14	8
s38584.1	7:39:34	8	11:20:20	8	2:12:29	9
seq	2:09:08	10	3:27:30	12	0:01:58	11
spla	9:35:03	12	5:14:28	16	0:06:12	13
tseng	0:16:09	6	0:23:17	7	0:07:42	6
Totais	**72:43:59**	**189**	**129:19:44**	**224**	**7:35:15**	**191**

D_{max} → *Densidade máxima por canal*

5 Proposta de Implementação Paralela

Este Capítulo tem como objetivo principal apresentar uma proposta de implementação paralela do algoritmo seqüencial descrito no Capítulo anterior. A execução é distribuída em uma rede de estações, onde cada estação é responsável pelo roteamento de uma parte do circuito. Na Seção 5.1 é apresentado um exemplo de um roteador paralelo encontrado na literatura. A Seção 5.2 apresenta a modelagem do FPGA para que a execução paralela do algoritmo seja possível.

5.1 Exemplo de roteador com execução paralela

O "*SC-PathFinder*" [CHA97] é um roteador com execução paralela e distribuída. Esse roteador é o atual vencedor do desafio proposto em [ROS02], utilizando apenas 188 trilhas para rotear os 20 circuitos teste propostos em [YAN91].

Esse algoritmo é uma modificação realizada no algoritmo de roteamento "PathFinder" [MCM95]. Nessa nova versão do algoritmo foram feitas 2 modificações. A primeira modificação é explorar o processamento paralelo e distribuído de uma rede de estações ("*Cluster*"). No *cluster*, cada estação é responsável pelo roteamento de uma parte do circuito. A segunda modificação foi implementar a troca de informações entre as estações, ou seja, compartilhando a informação de congestionamento dos recursos de roteamento. Esse processamento paralelo não necessita de semáforos, pois cada estação possui todas as informações sobre o dispositivo e sobre o circuito que esta sendo roteado.

O algoritmo é denominado de "negociador de caminho" A primeira iteração deste algoritmo distribui um conjunto de redes do circuito para cada estação do *cluster*. Estes conjuntos de redes são roteados independentemente. Uma vez todas as estações do *cluster* tendo concluído o processamento, as redes pertencentes aos canais congestionados são denominadas de redes compartilhadas. Estas redes compartilhadas têm seu custo aumentado, e uma "negociação" de qual tem mais prioridade determina a ordem do "*rip-up and re-route*" (a função custo depende do histórico de compartilhamento da rede e do atraso estimado desta). O processo de roteamento termina quando não há mais redes compartilhadas.

O ganho de desempenho obtido deve-se ao fato de que o tempo de comunicação entre estações de uma mesma rede local, na ordem de milisegundos, é muito inferior ao tempo de processamento de um conjunto de redes. O roteamento realizado por este algoritmo é denominado de grão grande, pois todo o circuito é roteado e posteriormente é feita a análise de congestão. Rotear um conjunto muito pequeno de redes em cada estação não traria desempenho, pois o custo de comunicação ultrapassaria o de roteamento.

5.2 Proposta de Roteamento Paralelo

O princípio que guia a presente proposta de implementação paralela de roteamento é a localidade das redes no interior do circuito. O princípio é particionar o circuito em diferentes áreas, denominadas *quadrantes*, executando paralelamente o algoritmo em cada um destes quadrantes. Observar que o paralelismo é de grão-grande, como na proposta anterior, porém com uma abordagem totalmente diferente. A proposta é rotear redes em áreas independentes, sem haver a

possibilidade de congestionar canais e posteriormente negociar quais redes têm prioridade para ficar no canal e quais devem sair.

5.2.1 Partição

O primeiro passo do procedimento do roteamento paralelo é particionar o circuito em quadrantes. A Figura 63 ilustra um FPGA com dimensão 64 x 64 blocos lógicos. A partição inicial, nível 0, corresponde ao tamanho total do FPGA (64 x 64). Nesta figura observa-se que o FPGA foi particionado até o nível 6, resultando em um quadrante com dimensão de 8 x 8 blocos lógicos.

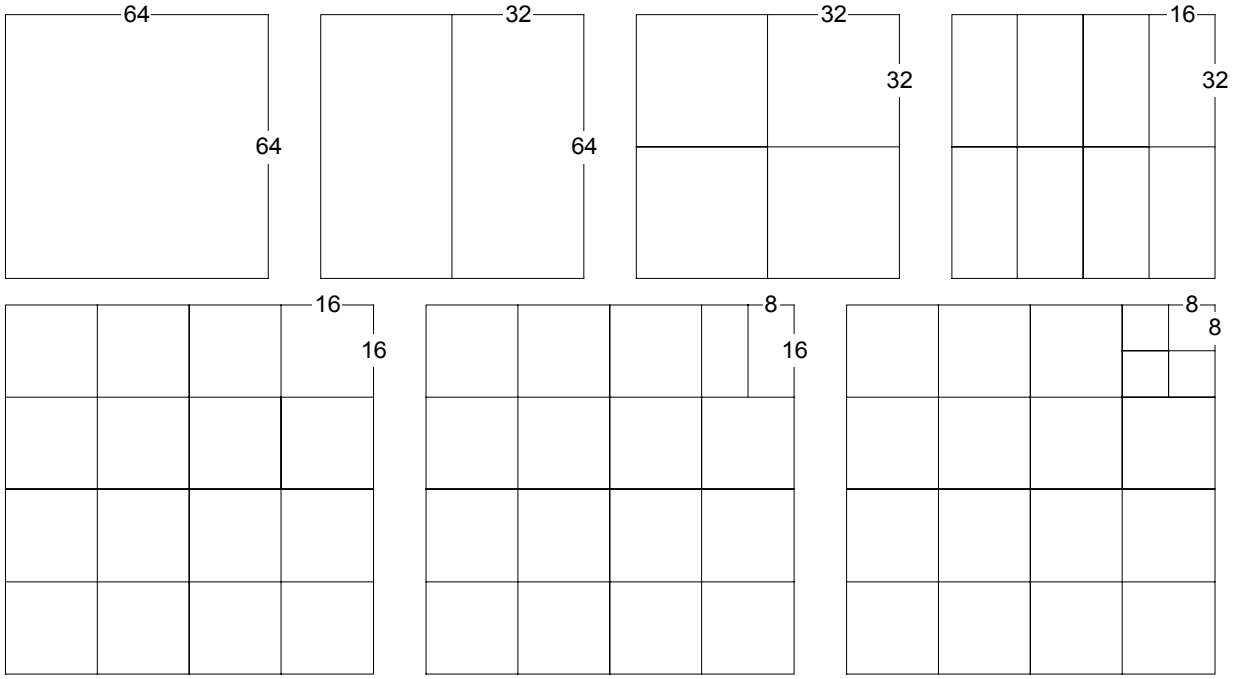


Figura 63 – Particionamento do FPGA.

A Tabela 14 apresenta a análise da ocupação de redes por quadrante para os *benchmarks* de roteamento.

Com o intuito de evitar partições pequenas, optou-se por dividir o FPGA até quadrantes de dimensão 8 x 8 blocos lógicos. Caso a partição continue, o número de redes por quadrante fica muito reduzido, e o tempo de comunicação prevalece sobre o de processamento.

Por exemplo, uma rede que esteja na partição 5 implica que todos os seus pinos estão contidos neste quadrante. Observando-se os dados fornecidos pela tabela pode-se concluir que em todos os circuitos a maioria das redes pode ser roteada em partições de tamanho 5 ou 6, corroborando a hipótese de localidade das redes.

Tabela 14 – Quantidade de redes que podem ser roteadas em uma partição.

Nome do Circuito	Nº Redes	Tam. Do FPGA	Nível de partição													
			6		5		4		3		2		1		0	
			RP	%	RP	%	RP	%	RP	%	RP	%	RP	%	RP	%
tseng.lca	1099	34	0	0%	520	47%	85	8%	90	8%	111	10%	222	20%	71	6%
apex4.lca	1271	37	0	0%	415	33%	89	7%	117	9%	204	16%	184	14%	262	21%
misex3.lca	1411	39	0	0%	619	44%	100	7%	108	8%	217	15%	169	12%	198	14%
Alu4.lca	1536	41	0	0%	808	53%	103	7%	200	13%	96	6%	135	9%	194	13%
diffeq.lca	1561	40	0	0%	765	49%	127	8%	253	16%	144	9%	138	9%	134	9%
Dsip.lca	1599	55	0	0%	838	52%	221	14%	159	10%	183	11%	43	3%	155	10%
Seq.lca	1791	43	0	0%	732	41%	120	7%	252	14%	175	10%	271	15%	241	13%
Des.lca	1847	64	850	46%	166	9%	146	8%	175	9%	201	11%	205	11%	104	6%
apex2.lca	1916	45	0	0%	757	40%	162	8%	305	16%	273	14%	188	10%	231	12%
s298.lca	1935	45	0	0%	1487	77%	107	6%	172	9%	87	4%	37	2%	45	2%
bigkey.lca	1936	55	0	0%	1119	58%	235	12%	194	10%	176	9%	46	2%	166	9%
frisc.lca	3576	61	0	0%	1713	48%	379	11%	428	12%	347	10%	368	10%	341	10%
Spla.lca	3706	62	0	0%	1673	45%	437	12%	514	14%	432	12%	317	9%	333	9%
elliptic.lca	3735	62	0	0%	1894	51%	466	12%	575	15%	381	10%	206	6%	213	6%
pdc.lca	4591	69	1922	42%	289	6%	369	8%	361	8%	620	14%	457	10%	573	12%
ex1010.lca	4608	69	1434	31%	305	7%	480	10%	560	12%	1067	23%	477	10%	285	6%
S38417.lca	6435	82	2674	42%	693	11%	1182	18%	543	8%	668	10%	480	7%	195	3%
s38584.1.lca	6485	82	3446	53%	491	8%	833	13%	442	7%	526	8%	579	9%	168	3%
clma.lca	8445	93	4017	48%	593	7%	1079	13%	1032	12%	786	9%	576	7%	362	4%

RP → Número de redes por Partição

5.2.2 Procedimento de roteamento

A etapa de particionamento divide o circuito até alcançar quadrantes de dimensão 8 *x* 8 blocos lógicos. Uma vez particionado o circuito, cria-se uma lista de redes para cada quadrante. Cada quadrante pode ser visto como um FPGA independente.

Tendo definido o número de partições, a próxima etapa é realizar a distribuição das partições em uma rede de estações. Uma das estações é denominada de *host*, sendo responsável por realizar a partição, criar a lista de redes por partição e distribuir as listas para as diversas estações.

Cada estação realiza o roteamento de suas redes independentemente. O algoritmo seqüencial utilizado difere do algoritmo apresentado no Capítulo anterior pelo fato de não restringirmos a propagação da frente de onda ao quadrado envolvente da rede e sim pelo tamanho do quadrante da partição. Esta relaxação fornece um espaço de procura maior para a rede, sem contudo pesquisar em todo o FPGA, estando limitado ao quadrante. É importante ressaltar que no nível de quadrante também há "*rip-up and re-route*".

À medida que as estações forem concluindo o roteamento parcial os dados são enviados ao *host*. O *host* recebe a *lista de rede roteadas* e a *lista de redes não roteadas* das estações. As *redes roteadas* são marcadas como obstáculos e as *redes não roteadas* são promovidas para o quadrante de dimensão imediatamente superior.

A Figura 64 ilustra a proposta do algoritmo paralelo, no nível de ações a serem executadas.

```
Enquanto todas as redes não forem roteadas ou o número de iterações for menor
que o máximo permitido:
 {
   Particionar o circuito até quadrante 8x8

   Enquanto o tamanho da partição for menor que o tamanho do FPGA:
   { - Criar lista de redes para cada quadrante
     - Se há redes roteadas no circuito criar estrutura de dados representando
       obstáculos
     - Enviar lista de redes e obstáculos as para estações
      (processamento paralelo)
     - Enquanto houver estação processando( )
       { - Inserir a "lista das redes roteadas" recebidas das estações na
           estrutura do FPGA (obstáculos)
         - Marcar as redes não roteadas para serem roteadas no quadrante
           subseqüente
       }
     - Passar para a partição subseqüente (8x16, 16x16, 32x16, 32x32, …)
   }

   Para as redes não roteadas:
   { - Desfazer o roteamento e inicializar a estrutura de dados
     - Rotear as redes não roteadas, considerando todo o FPGA com área
       disponível (rip-up and re-route)
     - Marcar estas redes como obstáculos
   }
  }
```

Figura 64 – Proposta do algoritmo paralelo de roteamento, representado a parte executada no host.

A segunda iteração do algoritmo contém as redes do segundo nível de partição (por exemplo 8 *x* 16) mais as redes não roteadas da partição anterior. As redes não roteadas na(s) partição(ões) anterior(es) têm maior prioridade para o roteamento. O *host* além de enviar para cada estação a lista de redes, envia também os obstáculos impostos pelas redes já roteadas. Este processo continua até o nível 0, ou seja, até haver apenas uma estação roteando o circuito.

Alcançando-se o nível 0, havendo redes não roteadas, estas são marcadas como redes para "*rip-up and re-route*". Toda a estrutura de roteamento é desfeita, as redes marcadas são roteadas tendo toda a área do FPGA disponível e o processo de partição reinicia.

O algoritmo é encerrado quando todas as redes forem roteadas ou quando a dimensão do quadrante for igual à dimensão do FPGA e o número de iterações para "*rip-up and re-route*" tiver sido excedido.

Espera-se com este algoritmo ter uma redução no tempo de processamento e uma redução no número de trilhas necessárias para rotear os circuitos. A redução no tempo de processamento é esperada devido ao roteamento ser feito em paralelo. Em uma partição 8 *x* 8, em um circuito com uma dimensão até 64 *x* 64 blocos lógicos, há 64 quadrantes. Em um *cluster* com 32 estações, cada estação realizaria o roteamento de apenas 2 quadrantes. Uma crítica à proposta é a eventual existência de nodos ociosos no *cluster* quando o tamanho da partição se aproxima do tamanho do FPGA. A redução no número de trilhas é esperada pelo fato do roteamento e o "*rip-up and re-route*" ser hierárquico.

6 Conclusão

O presente trabalho apresentou: (*i*) o estado da arte dos algoritmos de roteamento para FPGAs; (*ii*) o desenvolvimento de um roteador seqüencial, que tem o algoritmo Maze como base de implementação e (*iii*) a proposta de paralelização do algoritmo apresentado.

A comunidade acadêmica continua pesquisando novas técnicas de se melhorar o desempenho dos algoritmos existentes, tanto em área ocupada, desempenho elétrico e tempo de processamento. Neste trabalho foram citados alguns roteadores acadêmicos. O roteador que teve a maior ênfase foi o VPR, por conseguir rotear os circuitos de teste com um reduzido número de trilhas. Para que os circuitos de teste fossem roteados pelo VPR foram utilizadas 197 trilhas. Três anos mais tarde foi desenvolvido um novo roteador ("*SC-pathfinder*") que conseguiu rotear os mesmos circuitos de teste com um número inferior ao VPR. Esse roteador precisa de 188 trilhas para rotear com sucesso todos os circuitos de teste. O "*SC-pathfinder*" apresenta uma nova metodologia para o roteamento de FPGAs, utilizando um *cluster* de estações e técnicas de programação paralela.

O algoritmo de roteamento seqüencial foi desenvolvido utilizando-se a linguagem C (*ANSI* C), sendo o mesmo portável para diferentes sistemas operacionais. Essa implementação apresentou a modelagem do FPGA na forma de um grafo e também apresentou o FPGA modelado como uma matriz de estruturas com 3 dimensões. A modelagem escolhida para a implementação foi a matriz de estruturas, por esta ser mais versátil e pelo fato da biblioteca LEDA não estar mais disponível para domínio público sendo assim o roteador implementado pode ser distribuído gratuitamente. Para que a implementação e a modelagem do FPGA fossem validadas foram realizados testes com os mesmos *benchmarks* utilizados pelo VPR e o *SC-pathfinder*. Para a obtenção dos resultados foram necessários 74 horas (Sun Sparc 20), na versão com o FPGA modelado com um grafo e 7,35 horas na versão com a modelagem de matriz de estruturas (Pentium III 800MHz). Outro ponto que deve ser salientado é o número de trilhas que cada canal deve conter. Quanto menor for o número de trilhas por canal, melhor será a utilização dos recursos disponíveis no dispositivo, ou seja, melhor será a distribuição do circuito no mesmo. O algoritmo apresentado neste trabalho necessita de 191 trilhas para rotear todos os circuitos de teste.

O roteador *SC-pathfinde*", o qual realiza roteamento paralelo, tenta rotear todas as redes do circuito de uma só vez, sem levar em consideração os bloqueios existentes. Nossa proposta difere desta, utilizando como base o princípio da localidade das redes. A idéia principal da proposta é dividir o FPGA em "mini-FPGAs", e distribuí-los em uma rede estações. Assim cada estação será responsável por rotear uma parte do circuito, que posteriormente será agrupado pela estação "*host*". O "*rip-up & re-route*" é hierárquico. Realizado nos "mini-FPGAs" e posteriormente realizado também pela estação "*host*", onde as redes não roteadas serão roteadas antes que uma nova partição seja feita.

A principal contribuição científica deste trabalho é disponibilizar ao meio acadêmico um algoritmo de roteamento para FPGAs, e uma proposta de roteamento paralelo para redução do tempo de processamento.

Para trabalhos futuros sugere-se a implementação deste algoritmo paralelo, assim como introduzir no algoritmo tratamento de linhas de tamanho diferenciado.

7 Bibliografia

[ALE95] M. J. Alexander, J. P. Cohoon, J. L. Ganley, G. Robins. "Performance-Oriented Placement and Routing for Field-Programmable Gate Arrays". In: *EURO-DAC '95*, 1995, pp. 80–85.

[ALE96] M. J. Alexander, G. Robins. "New Performance-Driven FPGA Routing Algorithms". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 1505– 1517, Dec. 1996.

[ALT02] ALTERA Co. "Apex II Devices: The Next-Generation APEX SOPC". Capturado em: http://www.altera.com/, Janeiro 2002.

[BET96] V. Betz, J. Rose, "Directional Bias and Non-Uniformity in FPGA Global Routing Architectures". In: *IEEE/ACM International Conference on Computer Aided Design*, 1996, pp. 652–659.

[BET97a] V. Betz, J. Rose. "VPR: A New Packing, Placement and Routing Tool for FPGA Research". In: *7th International Workshop on Field-Programmable Logic*, 1997, pp. 213-222.

[BET97b] V. Betz. "VPR and VPACK User's Manual (Version 3.99)". Capturado em: http://www.eecg.toronto.edu/~vaughn/vpr/terms.html. Agosto 1997.

[BRO92] S. D. Brown, R. J. Francis, J. Rose, Z. G. Vranesic. "Field-Programmable Gate Arrays". Kluwer Academic Publishers, 1992, 206p.

[CHA97] P. K. Chan, M. D. F. Schlag. "Acceleration of an FPGA router". In: *Proc. 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines*, 1997, pp. 175 – 181.

[CHA00] P. K. Chan, M. D. F. Schlag. "Acceleration of an FPGA Router". In: *FPGA'2000: International Symposium on Field-Programmable Gate Arrays*. Capturado em: http://www.cse.ucsc.edu/~pak/router.ps, Agosto 2000, 7p.

[CHE95] C-D. Chen, Y-S Lee, A.C.-H. Wu, Y-L Lin. "TRACER-FPGA: a router for RAM-based FPGA's". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 371–374, Mar. 1995.

[CON94] J. Cong, Y. Ding. "FlowMap: an Optimal Technology Mapping Algorithm for Delay Optimization in Look-Up Table Based FPGA Designs". *IEEE Transactions on Computer Aided Design*, vol. 13(**1**), pp. 1–12, Jan. 1994.

[COR89] T. H. Cormen, C. E. Leiserson, R. L. Rivest. "Introduction to Algorithms". McGraw-Hill Book Company, 1989, 1028p.

[EBE95] C. Ebeling, L. McMurchie, S. A. Hauck, S. Burns. "Placement and Routing Tools for Triptych FPGA". *IEEE Transactions on Very Large System Integrated*, vol. 3(**4**), pp. 473–482, Dec. 1995.

[HAD77] F. O. Hadlock. "The Shortest Path Algorithm for Grid Graphs". *Networks*, vol. 7, pp. 323–334, 1977.

[HUR00] S-W. Hur, A. Jagannathan, J. Lillis. "Timing-Driven Maze Routing". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19(**2**), pp. 234-241, Fev. 2000.

[KIR83] S. Kirkpatrick, C. D. Gellat, Jr., M. P. Vecchi. "Optimization by Simulated Annealing". *Science*, nº 4598, 1983, pp. 498–516.

[KOU81] L. Kou, G. Markowski, L. Berman. “A Fast Algorithm for Steiner Trees”. *Acta Informatica*, vol. 15, pp. 141-145, 1981.

[LEE61] C. Lee. “An Algorithm for Path Connections and its Applications”. *IRE Transactions on Electronic Computers*, VEC-10, pp. 346-365, Set. 1961.

[LEE97] Y-S. Lee, A. C.-H. Wu. “A Performance and Routability Driven Router for FPGAs Considering Path Delays”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 179–185, Fev. 1997.

[LEM93] G. G. F. Lemieux, S. D. Brown. “A Detailed Routing Algorithm for Allocating Wire Segments in Field-Programmable Gate Arrays”. In: *4th ACM/SIGDA Physical Design Workshop*, 1993, pp 215-226.

[LEM97] G. G. F. Lemieux, S. D. Brown, D. Vranesic. “On Two-Step Routing for FPGAs”. In: *International Symposium on Physical Design*, 1997, pp. 60 – 66.

[LIM01] F. Lima. “Designing Single Event Upset Mitigation Techniques for Large SRAM-Based PFGA Devices”. Proposta de Tese, Instituto de Informática, UFRGS, 2001, 72p.

[MEH89] K. Mehlhorn, S. Näher. “LEDA, a Library of Efficient Data Types and Algorithms”. In: *Proceedings of 14th Symposium on Mathematical Foundations of Computer Science*, 1989, vol. 379, pp. 88-106.

[MAR00a] A. Marquardt, V. Betz, J. Rose. “Speed and Area Tradeoffs in Cluster-Based FPGA Architectures”. *IEEE Transactions on Very Large Scale Integration Systems*, vol. 8(**1**), pp. 84–93, Fev. 2000.

[MAR00b] A. Marquardt, V. Betz, J. Rose. “Timing-Driven Placement for FPGAs”. In: *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2000, pp. 203–213.

[MCM95] L. McMurchie, C. Ebeling. “Pathfinder: a Negotiation-Based Performance-Driven Router for FPGAs”. In: *Proceedings of 3rd International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, 1995, pp. 111–117.

[MOO59] E. Moore. “Shortest Path Through a Maze”. In: *Proceedings of the International Symposium on Switching Circuits*, 1959, pp. 285–292.

[MOR90] F. G. Moraes. “Trago – Síntese Automática de Leiaute para Circuitos em Lógica Aleatória”. Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul, 1990, 205p.

[RAM96] S. Raman, C. L. Liu, L. G. Jones. “A Timing-Constrained Incremental Routing Algorithm for Symmetrical FPGAs”. In: *Proceedings of European Design and Test Conference ‘96*, 1996, pp. 170–174.

[ROS85] J. Rose, W. M. Snelgrove, Z.G. Vranesic. “ALTOR: an Automatic Standard-Cell Layout Program”. In: *1985 Canadian Conference on Very Large Scale Integration*, 1985, pp. 169–173.

[ROS90] J. Rose. “Parallel Global Routing for Standard Cells”. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 9(**10**), pp. 1085–1095, Out. 1990.

[ROS93] J. Rose, A. Gamal, A. Sangiovanni-Vicentelli. “Architecture of Field-Programmable Gate Arrays”. *Proceedings of the IEEE*, vol. 81(**7**), pp. 1013–1028, Jul. 1993.

[ROS02] J. Rose. “The FPGA Place-and-Route Challenge”. Capturado em: http://www.eecg.toronto.edu/~vaughn/challenge/challenge.html, Janeiro 2002.

[SEN92] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, A. Sangiovanni-Vincentelli. “SIS: A System for Sequential Circuit Synthesis”. Technical Report UCB/ERL M92/41, U.C. Berkeley, May 1992.

[SOU78] J. Soukup. “Fast Maze Router”. In: *Proceedings of 15º Design Automation Conference*, 1978, pp. 100-102.

[SHE95] N. Sherwani. “Algorithms for VLSI Physical Design Automation”. Kluwer Academic Publishers, 1995, 538p.

[THA94] S. Thakur, D. F. Wong, S. Muthukrishnan. “Algorithms for a Switch Module Routing Problem”. In: *Proceedings of European Design Automation Conference*, 1994.

[WIL97] S. J. E. Wilton. “Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memories,” Tese de Doutorado, Universidade de Toronto, 1997. Capturado em: http://www.ece.ubc.ca/home/staff/faculty/stevew/etc/www/, Julho 1997.

[WU94] Y.-L. Wu, M. Marek-Sadowska. “An Efficient Router for 2-D Field Programmable Gate Arrays”. In: *Proceedings of European Design Automation Conference*, 1994, p. 412–416.

[WU95] Y.-L. Wu, M. Marek-Sadowska. “Orthogonal Greedy Coupling – A New Optimization Approach to 2-D FPGA Routing”. In: *Proceedings of Design Automation Conference*, 1995, p. 568–573.

[XIL02] XILINX Inc. “The Programmable Gate Array Data Book”. Capturado em: http://www.xilinx.com, Janeiro 2002.

[YAN91] S. Yang. “Logic Synthesis and Optimization Benchmarks, Version 3.0”. Microeletronics Centre of North Carolina, North Carolina, CA, Tech. Report, 1991.

[ZHU97] K. Zhu, D. F. Wong. “Clock Skew Minimization During FPGA Placement”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 376–385, Abr. 1997.

Anexo I

Arquivo de Arquitetura paper.arc.

```
.arch X4000E
.lines 4
.columns 4
.tracks 3
.repeat 200
.clbpins
.pin F1 west
.pin F2 north
.pin F3 east
.pin F4 south
.pin G1 west
.pin G2 south
.pin G3 east
.pin G4 north
.pin C1 west
.pin C2 south
.pin C3 east
.pin C4 north
.pin K  west
.pin X  west
.pin Y  east
.pin XQ south
.pin YQ north
.endpins
.end
```

Anexo II

Lista de redes paper.lca.

```
Addnet [a] R1C1.F4 R1C3.F4
Addnet [c] R1C2.F4 R2C1.F3 R3C2.F2
Addnet [b] R2C2.F4 R2C3.F2
Addnet [d] R1C1.F3 R2C1.F1 R3C2.F4 R2C3.F3
```

Coordenadas Convertidas

```
[0] - [d] → 112 121 042 124
[1] - [c] → 022 122 032
[2] - [A] → 021 023
[3] - [b] → 032 023
```

Na Figura 65 é possível observar o circuito acima convertido na estrutura do FPGA. O pino de origem é representado pelo círculo e os demais pontos da rede são representados pelo hexágono.

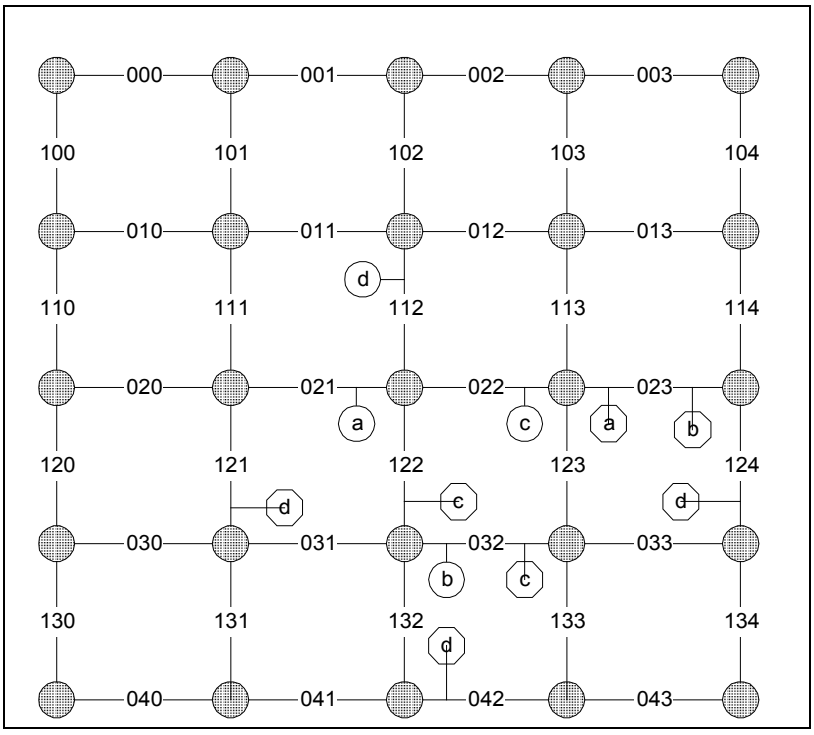


Figura 65 – Representação gráfica do circuito mapeado no FPGA

Para o roteamento do circuito acima, não foi necessário que o algoritmo realizasse o "*rip-up and Re-route*". A Figura 66 apresenta as coordenadas do circuito completamente roteado. Na primeira tentativa houve sucesso, conseqüentemente não foi preciso uma nova iteração. Esse circuito não pode ser roteado com uma largura de canal (número de trilhas menor do que 2), por se excluírem mutuamente do canal.

```
1ª Tentativa
Rede        Limite       Canais Utilizados
[d] - 3 - (5,5)(0,0)     112 121 021 042 041 131 124 134 043
[c] - 2 - (4,3)(1,1)     022 122 032
[a] - 0 - (3,4)(1,0)     021 023 022
[b] - 1 - (4,4)(1,1)     032 023 123
```

Figura 66 – Primeira e segunda tentativas de roteamento

A Figura 67 ilustra o circuito listado no arquivo *paper.lca*, roteado pelo algoritmo de roteamento desenvolvido. O congestionamento pode ser observado na Figura 67 pela área marcada pelo quadro tracejado.

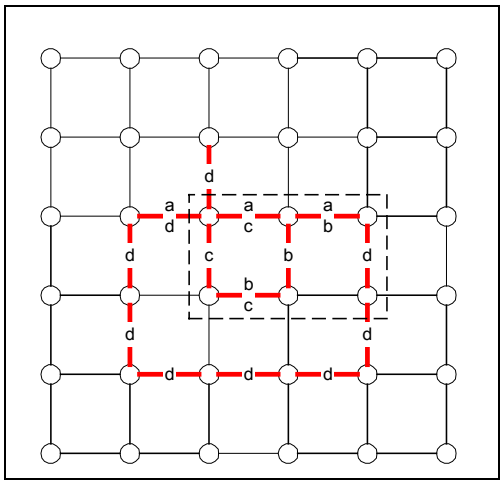


Figura 67 – Circuito roteado pelo algoritmo de roteamento do tipo Maze

Anexo III

A Figura 68 apresenta a execução do algoritmo de roteamento no circuito *paper.lca*

```
Lendo as opcoes do programa ...
Lendo o arquivo de arquitetura ...
Inicializando as estruturas ...
Criando a estrutura do FPGA ...
Criando o vetor de trilhas e preenchendo com -1 ...

Numero total de trilhas: 150
Lendo o arquivo de redes ...
Leitura das redes em processo ...
Convertendo as redes lidas ...
Numero de Redes Convertidas: 4
Ordenando a lista de redes - #Termos: 4 ...
Redes antes do Roteamento
Rede: [0] - [a] -> 021 023
Rede: [1] - [b] -> 032 023
Rede: [2] - [c] -> 022 122 032
Rede: [3] - [d] -> 112 121 042 124
3 - 4 ### ...  Ok ! --> Canal: 0
2 - 3 ### ..  Ok ! --> Canal: 0
0 - 2 ### .  Ok ! --> Canal: 1
1 - 2 ### .  Ok ! --> Canal: 2

######################################################################
Tempo de execucao: 0 s
Rip-up & Re-route: 0

Total de Trilhas: 180
Total Utilizado: 18
Percentual utilizado: 10.00

Circuito Roteado com Sucesso !

######################################################################

fpga[0][2][1] - [3] [0] [-1]
fpga[0][2][2] - [2] [0] [-1]
fpga[0][2][3] - [-1] [0] [1]
fpga[0][3][2] - [2] [-1] [1]
fpga[0][4][1] - [3] [-1] [-1]
fpga[0][4][2] - [3] [-1] [-1]
fpga[0][4][3] - [3] [-1] [-1]
fpga[1][1][2] - [3] [-1] [-1]
fpga[1][2][1] - [3] [-1] [-1]
fpga[1][2][2] - [2] [-1] [-1]
fpga[1][2][3] - [-1] [-1] [1]
fpga[1][2][4] - [3] [-1] [-1]
fpga[1][3][1] - [3] [-1] [-1]
fpga[1][3][4] - [3] [-1] [-1]
```

Figura 68 – Tela de execução do algoritmo de roteamento

A Figura 69 apresenta as estatísticas após a execução do algoritmo.

```
######################################################################
Tempo de execucao: 0 s
Rip-up & Re-route: 0
```

```
Total de Trilhas: 180
Total Utilizado: 18
Percentual utilizado: 10.00

Circuito Roteado com Sucesso !

######################################################################
```

Figura 69 – Estatísticas de execução do algoritmo de roteamento

A Figura 70 apresenta a execução do algoritmo de roteamento no circuito *paper2.lca*

```
Lendo as opcoes do programa ...
Lendo o arquivo de arquitetura ...
Inicializando as estruturas ...
Criando a estrutura do FPGA ...
Criando o vetor de trilhas e preenchendo com -1 ...

Numero total de trilhas: 50
Lendo o arquivo de redes ...
Leitura das redes em processo ...
Convertendo as redes lidas ...
Numero de Redes Convertidas: 3
Ordenando a lista de redes - #Termos: 6 ...
Redes antes do Roteamento
Rede: [0] - [a] -> 021 023
Rede: [1] - [b] -> 100 043 114 030 113 122
Rede: [2] - [c] -> 130 003
1 - 6 ### .....  Ok ! --> Canal: 1
0 - 2 ###  - [[a]] -  Not Ok !
2 - 2 ### .  Ok ! --> Canal: 1

Total de Redes  Roteadas: 2


Tentativa de Rip-up & Re-Route n#1
0 - 2 ### .  Ok ! --> Canal: 1
1 - 6 ### .....  Ok ! --> Canal: 1
2 - 2 ### .  Ok ! --> Canal: 1

######################################################################
Tempo de execucao: 0 s
Rip-up & Re-route: 1

Total de Trilhas: 60
Total Utilizado: 25
Percentual utilizado: 41.67

Circuito Roteado com Sucesso !

######################################################################

fpga[0][0][1] - [2]
fpga[0][0][2] - [2]
fpga[0][0][3] - [2]
fpga[0][1][3] - [1]
fpga[0][2][1] - [0]
fpga[0][2][2] - [0]
fpga[0][2][3] - [0]
fpga[0][3][0] - [1]
fpga[0][3][1] - [1]
fpga[0][3][2] - [1]
```

```
fpga[0][4][0]  -  [2]
fpga[0][4][3]  -  [1]
fpga[1][0][0]  -  [1]
fpga[1][0][1]  -  [2]
fpga[1][1][0]  -  [1]
fpga[1][1][1]  -  [2]
fpga[1][1][3]  -  [1]
fpga[1][1][4]  -  [1]
fpga[1][2][0]  -  [1]
fpga[1][2][1]  -  [2]
fpga[1][2][2]  -  [1]
fpga[1][2][3]  -  [1]
fpga[1][3][0]  -  [2]
fpga[1][3][1]  -  [2]
fpga[1][3][3]  -  [1]
```

Figura 70 – Tela de execução do algoritmo de roteamento sobre paper2.lca

A Figura 71 apresenta as estatísticas após a execução do algoritmo sobre o circuito paper2.lca.

```
Iteracao 1:
      Redes Roteadas: 2
      Redes Nao Roteadas: 1


######################################################################
Tempo de execucao: 0 s
Rip-up & Re-route: 1

Total de Trilhas: 60
Total Utilizado: 25
Percentual utilizado: 41.67

Circuito Roteado com Sucesso !

######################################################################
```

Figura 71 – Estatísticas do roteamento do circuito paper2.lca

Anexo IV

A Figura 72 apresenta a execução do algoritmo de roteamento, sobre o circuito de teste seq.lca.

```
Lendo as opcoes do programa ...
Lendo o arquivo de arquitetura ...
Inicializando as estruturas ...
Criando a estrutura do FPGA ...
Criando o vetor de trilhas e preenchendo com -1 ...

Numero total de trilhas: 40500
Lendo o arquivo de redes ...
Leitura das redes em processo ...
Convertendo as redes lidas ...
Numero de Redes Convertidas: 1791
Ordenando a lista de redes - #Termos: 234 ...

Total de Redes     Roteadas: 1757
Total de Redes Nao Roteadas: 34
Re-ordenando as redes

 Rip-up & re-route: 1

Total de Redes     Roteadas: 1782
Total de Redes Nao Roteadas: 9
Re-ordenando as redes

 Rip-up & re-route: 2

Total de Redes     Roteadas: 1787
Total de Redes Nao Roteadas: 4
Re-ordenando as redes

 Rip-up & re-route: 3

Total de Redes     Roteadas: 1790
Total de Redes Nao Roteadas: 1
Re-ordenando as redes

 Rip-up & re-route: 4

Total de Redes     Roteadas: 1790
Total de Redes Nao Roteadas: 1
Re-ordenando as redes

 Rip-up & re-route: 5

Total de Redes     Roteadas: 1789
Total de Redes Nao Roteadas: 2
Re-ordenando as redes

 Rip-up & re-route: 6

Total de Redes     Roteadas: 1790
Total de Redes Nao Roteadas: 1
Re-ordenando as redes

 Rip-up & re-route: 7
```

```
######################################################################
Tempo de execucao: 1749 s
Rip-up & Re-route: 7

Total de Trilhas: 41400
Total Utilizado: 24041
Percentual utilizado: 58.07

Circuito Roteado com Sucesso !

######################################################################
```

Figura 72 – Execução do algoritmo de roteamento sobre um circuito de teste (seq.lca)

Pode-se notar que o número de redes não roteadas vai diminuindo a cada ordenação e novo roteamento. Se o número de rede não roteadas não diminuir rapidamente, esse fato indica que a ordenação não foi realizada corretamente.

Anexo V

A Figura 73 apresenta a execução do aplicativo de visualização de redes. O símbolo "#" indica os pinos de conexão utilizados pela rede roteada.

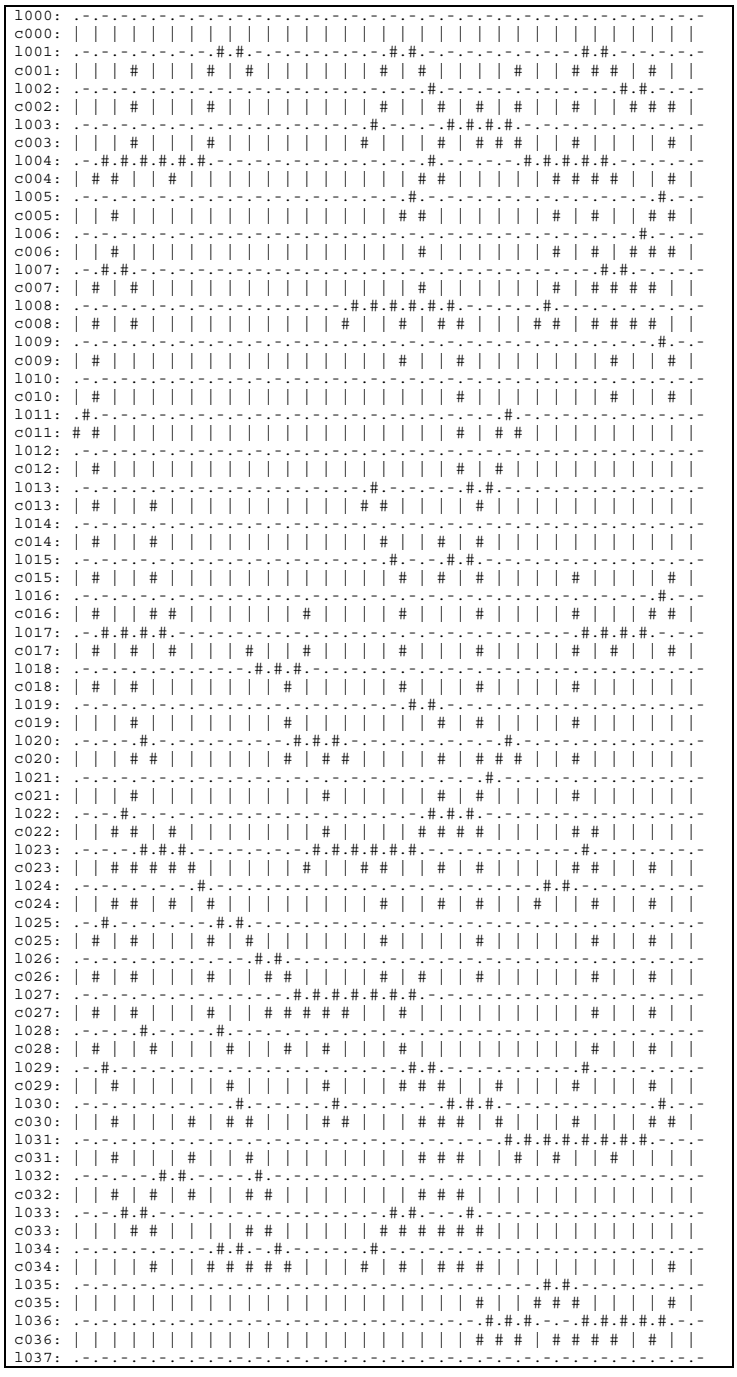


Figura 73 – Execução do Visualizador de Redes