PMAZE: MODELAGEM E ROTEAMENTO PARA FPGAS

Paulo César Furlanetto Marques [1]
Fernando Gehm Moraes [2]
Ney Laert Vilar Calazans [3]

Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Instituto de Informática
Av. Ipiranga, 6681 - Prédio 30 / BLOCO 4
Telefone: +55 51 320-3611 - Fax: +55 51 320-3621
CEP 90619-900 - Porto Alegre - RS - BRASIL

Resumo

O presente trabalho tem por objetivo descrever uma contribuição para a etapa de síntese física de dispositivos programáveis do tipo FPGAs, especificamente a etapa de roteamento. Apresenta-se os principais parâmetros que devem ser modelados na arquitetura dos FPGAs para os algoritmos de roteamento. São apresentadas duas versões de roteamento: com e sem restrição nas matrizes de chaves. Para a versão sem restrições na matriz de chaves apresenta-se os algoritmos implementados (propagação e retro-propagação) e os resultados obtidos em termos de largura mínima de canal. A versão que modela restrições encontra-se em desenvolvimento, apresentando-se aqui resultados parciais. Ambos algoritmos, assim como os algoritmos acadêmicos e comerciais, tem por base o algoritmo de propagação de frente de onda, denominado Maze. Nossas principais contribuições são: (*i*) conexão de redes multi-ponto, utilizando um procedimento denominado de "propagação de aresta", o qual conecta os vários pontos de uma rede com o caminho mínimo; e (*ii*) método de propagação para matriz de chaves com restrições.

Abstract

This paper presents a routing algorithm for FPGAs programmable devices. We start by showing the architectural parameters internal to the FPGA which should be modeled for the routing algorithm. We present two versions of our routing algorithm: one without switch matrix modeling and another with. For the version without switch matrix modeling we show the implemented algorithms (propagation and back-propagation) and the results in terms of minimum width of routing channel. The version with restrictions is currently under development. We show partial results for this last version, using as benchmark a small circuit where rip-up and re-route is required. Both algorithms, as well as other academic and commercial tools, use the well-known Maze algorithm, which is based on wave-propagation from source to target(s) node(s). Our main contributions are: (*i*) multi-point net connection, using an algorithm called "edge propagation", which connects all nodes in a net using minimum routing resources; and (*ii*) a method to route under switch matrix restrictions.

---

[1] Mestrando em Ciência da Computação (PUCRS), Engenheiro Eletrônico (PUCRS, 1997)
E-mail: pmarques@andros.inf.pucrs.br

[2] Doutor em Informática, opção Microeletrônica (LIRMM, França, 1994), Engenheiro Eletrônico (UFRGS, 1987), Professor Adjunto do Instituto de Informática/PUCRS.
E-mail: moraes@andros.inf.pucrs.br

[3] Doutor em Ciências Aplicadas, opção Microeletrônica (UCL, Bélgica, 1993), Engenheiro Eletrônico (UFRGS, 1985), Professor Adjunto do Instituto de Informática/PUCRS.
E-mail: calazans@andros.inf.pucrs.br

1. Introdução

O elevado número de portas lógicas equivalentes em um dispositivo programável torna o processo de concepção manual impossível. Por exemplo, a família XC4000XV pode conter até 250000 portas lógicas. As ferramentas de CAD estão em crescente evolução para acompanhar o aumento da complexidade dos circuitos e as restrições cada vez mais estritas de projeto (baixo consumo e alta freqüência de funcionamento). O processo de concepção envolve 2 etapas: síntese lógica e síntese física. O presente trabalho tem por objetivo contribuir na etapa de síntese física, especificamente a etapa de roteamento, pois é nesta etapa onde deve-se controlar os elementos parasitas, afim de não comprometer o desempenho elétrico especificado em etapas mais abstratas de concepção.

O termo roteamento refere-se à implementação das rotas que os sinais irão percorrer, afim de realizar as conexões entre os diferentes equipotenciais no circuito. Em um circuito integrado pré-fabricado a definição das rotas é feita por programação de chaves ou de fusíveis, de maneira permanente ou temporária. A programação temporária é a que tem tido maior sucesso comercialmente pela sua facilidade de reprogramação, favorecendo o desenvolvimento de arquiteturas reconfiguráveis. Exemplos de FPGAs que utilizam essa tecnologia são XILINX 4000 [XIL98] e ALTERA FLEX 10k [ALT98]. Na família Xilinx, a arquitetura do FPGA é constituída por uma matriz simétrica de blocos lógicos programáveis pelo usuário, conectados por um conjunto de recursos de roteamento: linhas de diferentes tamanhos (linhas de propósito geral, linhas de tamanho duplo, linhas longas, linhas globais e linhas quádruplas) e matriz de chaves (*"switch boxs"*) [THA94]. A *matriz de chaves* consiste de chaves de interconexão, cada uma possuindo 6 pontos de interconexão programáveis. Essa matriz de chaves conecta as linhas de propósito geral e de tamanho duplo entre si.

Diferentemente das algoritmos de roteamento para circuitos tipo *standard-cells* ou *gate-arrays* (ASICs), o roteamento para FPGAs não permite a implementação dos algoritmos de roteamento baseados em canal de roteamento. A diferença básica está nas conexões verticais: em ASICs as conexões são realizadas por células de passagem (*feedthroughs*) em qualquer posição horizontal, nos FPGAs há uma distribuição uniforme de chaves de interconexão, em posições fixas.

Os roteadores, tanto acadêmicos quanto comerciais, têm utilizado como base o algoritmo Maze. Este algoritmo é utilizado para encontrar o menor caminho entre a origem e o(s) destino(s) de uma rede, em uma matriz. O processo de consiste em, à partir da origem propagar frentes de onda até que o destino seja encontrado. As frentes de onda são identificadas por níveis. A origem é o nível zero, a próxima frente de onda tem origem na anterior e recebe o nível anterior mais um, e assim sucessivamente até o destino. Quando o destino é atingido pela frente de onda inicia-se o processo de retro-propagação. O caminho encontrado na primeira fase é realizado de forma inversa, do destino para a origem, para que seja traçado o caminho entre os dois pontos. Este tipo de procedimento é seqüencial, roteando rede após rede. Quando uma determinada conexão não pode ser realizada, devido a bloqueios causados por outras redes, esta rede é colocada no início da lista de redes e o processo de roteamento é re-inicializado.

A seção 3 deste artigo apresenta a comparação entre roteadores acadêmicos, objetivando mostrar as características principais destas ferramentas. A seção 4 apresenta nosso roteador. São apresentadas duas implementações: sem restrições de chaves de interconexão entre os canais e outra versão com restrições. Nesta seção é mostrada a modelização do FPGA e os algoritmos desenvolvidos. A seção 5 apresenta nossos resultados preliminares, e finalmente nossas conclusões e trabalhos futuros.

2. Roteadores Acadêmicos

A Tabela 1 apresenta a comparação entre vários softwares de roteamento. Em alguns casos há uma ferramenta para roteamento global e outra para o roteamento detalhado. Esta comparação é realizada através de circuitos de teste (*"Benchmarks"*) e o parâmetro utilizado é o número máximo de trilhas por canal de roteamento. Após a tabela serão ressaltadas algumas características importantes destas ferramentas.

Global	LocusRoute **[ROS90]**		GBP	OGC	IKMB	VPR	TRACER	VPR
Detalhado	CGE	SEGA				SEGA		
Referência	**[BRO92]**	**[LEM93]**	**[WU94]**	**[WU95]**	**[ALE95]**	**[LEM97]**	**[LEE95]**	**[BET97a]**
9symml	9	9	9	9	8	7	7	6
Alu2	12	10	11	9	9	8	9	8
Alu4	15	13	14	12	11	10	11	9
Apex7	13	13	11	10	10	10	8	8
Example2	18	17	13	12	11	10	10	9
K2	19	16	17	16	15	14	14	12
Term1	10	9	10	9	8	8	7	7
Too_large	13	11	12	11	10	10	9	8
Vda	14	14	13	11	12	12	11	10
Total	123	112	110	99	94	89	85	77

Tabela 1 - Tabela comparativa de desempenho [BET97a]

Os softwares LocusRoute e VPR exploram a ***equivalência de pinos*** dos blocos lógicos. Isso significa que se em uma função booleana as entradas estiverem no mesmo nível lógico, é possível escolher em qual lado do bloco lógico o sinal será conectado. Segundo [BET96], caso não seja implementada esta característica é necessário 8% a mais de área de roteamento.

Os softwares LocusRoute, SEGA e VPR ***utilizam segmentos de tamanho variável***, com o objetivo de reduzir o número de chaves nas conexões, e assim melhorar o desempenho elétrico do circuito.

Os softwares LocusRoute, CGE, IKMB, Tracer, e VPR implementam ***Rip-up e Re-routing***. Quando uma conexão não pode ser fisicamente realizada, o algoritmo 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"*.

A flexibilidade de roteamento em um FPGA é definido por dois parâmetros: f_c e f_s. O parâmetro f_s define a flexibilidade da matriz de chaves, indicando o número total de conexões oferecidas a cada segmento de fio conectado a mesma. Se o objetivo é reduzir a área de roteamento deve-se ter um fator f_s elevado, para possibilitar um maior número de conexões entre os canais de roteamento. Como a complexidade das matrizes de chaves é proporcional ao número de possibilidades de saídas para cada entrada, normalmente este parâmetro é limitado entre 3 e 6. O parâmetro f_c indica a flexibilidade do bloco lógico, representando o número total de trilhas, às quais um pino do bloco lógico pode se conectar.

2.1 VPR

O VPR ("*Versatile Place and Route*") é uma ferramenta de posicionamento, roteamento global e roteamento detalhado para FPGAs. Utiliza-se para posicionamento uma algoritmo *"Simulated Annealing"* [KIR83], cuja função custo 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]. É utilizado um algoritmo Maze para rotear

cada rede. Se uma rede ficar bloqueada, então é realizado “rip-up” e “re-route” até que a conexão bloqueada seja roteada (número de iterações definida pelo usuário).

Este programa modela os seguintes parâmetros da arquitetura do FPGA: o número de entradas e saídas do bloco lógico; em que lados do bloco lógico podem ser acessadas as entradas e as saídas; a equivalência lógica entre os pinos de entrada e saída; o 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 arquitetura da matriz de chaves, ou seja, quais as conexões que são permitidas (F_s); o número de trilhas que cada pino de entrada do bloco lógico pode se conectar (F_c); o valor de F_c para as saídas dos blocos lógicos; o valor de F_c para os *“PADS”* de E/S.

O circuito pode ser descrito em todos os formato aceitos pelo SIS [SEN92], utilizado para fazer a otimização lógica. O passo seguinte é o mapeamento tecnológico, realizado pelo software FlowMap [CON94]. O arquivo gerado pela etapa de mapeamento tecnológico é convertido para um formato *netlist* pelo software VPACK [BET97b]. O arquivo gerado por esta ferramenta é utilizado como entrada para o roteador VPR.

3. Implementação

O FPGA foi modelado com um grafo (Figura 1), onde os nodos são as matrizes de chaves e as arestas os canais de roteamento. Toda a implementação da estrutura de dados foi feita com o auxílio da biblioteca de estruturas LEDA (*“Library of Efficient Data types and Algorithms”*) [MEH89]. Os nodos não possuem nenhum tipo informação. As arestas contém todas as informações sobre o roteamento e informações utilizadas para que a frente de onda seja propagada através das arestas.

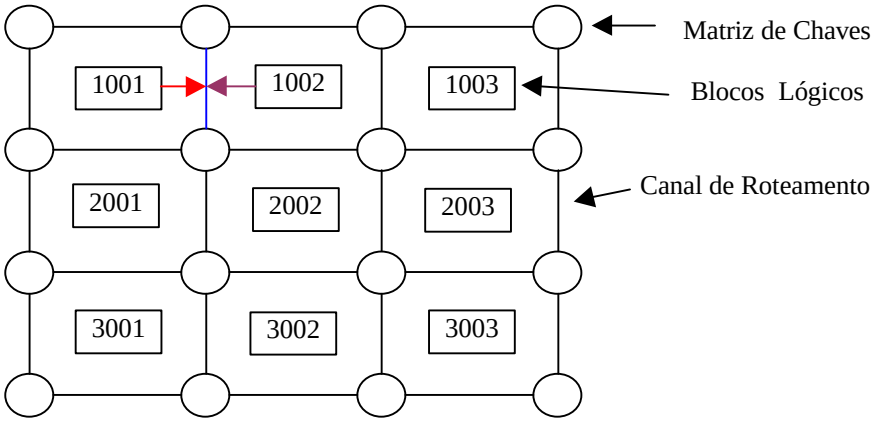


Figura 1 - FPGA modelado como um grafo

As arestas têm na sua estrutura a lista *tracks*, com dimensão definida no arquivo de tecnologia, que corresponde ao número de trilhas disponível em cada canal. Esta estrutura contém as redes utilizadas no canal. Quando a lista *tracks* está com todas trilhas ocupadas, o canal é dito *congestionado*. Dentre as informações armazenadas na aresta, há duas variáveis que indicam sua posição no FPGA. Por exemplo, o canal indicado na figura acima conterá os valores 1001 e 1002, respectivamente. Este canal, *1001-1002*, é interpretado como o canal na linha 1, entre as colunas 1 e 2. Se o bloco 1001 tem um pino na borda leste, a rede conectada a este pino será alocada à aresta 1001-1002; se este mesmo bloco tem um pino na borda sul, a rede será alocada à aresta 1001-2001.

A entrada do roteador compreende dois arquivos: a descrição da arquitetura alvo (configuração) e o arquivo com o circuito (dados). O arquivo de configuração que serve para

modelar o FPGA permite parametrizar a quantidade de linhas e de colunas de blocos lógicos, o número de trilhas por canal, o número repetições do roteamento caso o circuito não seja totalmente roteado e a descrição do bloco lógico (em qual borda os sinais serão conectados). O arquivo que contém o circuito a ser roteado é descrito no formato *.lca* (formato proprietário da Xilinx – ver figura Figura 9), sendo a única informação necessária a lista de redes. Este arquivo é gerado após mapeamento e posicionamento, não possuindo as informações de roteamento. As redes são colocadas em uma lista de *strings* e ordenadas de forma decrescente a partir da rede de maior fan-out. Isto é feito com o objetivo de rotear as redes que conectam um maior número de blocos lógicos antes das demais, pois no início do roteamento não há obstáculos.

3.1 Implementação sem modelagem das matrizes de chaves

O software em sua primeira versão não considera as restrições existentes nas matrizes de chaves, ou seja, $F_s=\infty$ (Figura 2a). Em decorrência disso, a rede entrando pela trilha 1 na matriz de chaves, no lado oeste, poderá sair da matriz de chaves em qualquer outra trilha nos demais lados. Esta configuração não corresponde à realidade dos FPGAs comerciais atuais. Na seção seguinte será apresentado o algoritmo com restrição no parâmetro F_s (Figura 2b).

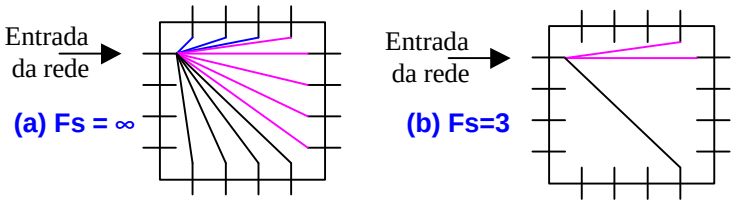


Figura 2 – Matriz de chaves com $F_s=\infty$ e $F_s=3$

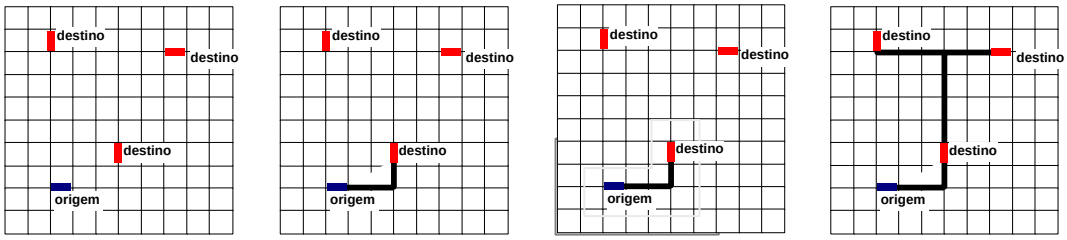
O software inicia com a leitura dos arquivos de configuração e de dados (.lca), ordena as redes e inicializa o grafo que modela o FPGA. Após essas tarefas estarem concluídas, tem início o algoritmo de roteamento, apresentado pelo pseudo-código da Figura 3.

```
Caminha no grafo()
{
 enquanto lista de redes não vazia
   {
    procura a posição da origem da rede_i
    adiciona os pontos de destino da rede_i à lista de destinos

    /* Roteamento da origem ao destino mais próximo */
    destino_k = Caminho Mínimo (origem, lista de destinos)

    se existe caminho
       Retro-propagação (origem, destino_k)
    senão
      { adiciona a rede_i à lista de redes não roteadas
        reinicia o processo com a próxima rede
      }
    retira o destino_k da lista de destinos

    /* Roteamento dos demais pontos da rede, através da propagação de aresta */
    enquanto lista de destinos não vazia
      {
       destino_k = Caminho Mínimo Modificado (lista de destinos)
       se existe caminho
          Retro-propagação Modificada(destino_k)
       senão
         { adiciona a rede a lista de redes não roteadas
           reinicia o processo com a próxima rede
         }
       retira destino_k da lista de destinos
      }
    }
}
```

Figura 3 – Função de caminhamento no grafo

O algoritmo apresentado realiza a conexão rede a rede, e insere as redes não roteadas em uma lista. Para a realização do “rip-up” e “re-route” deve-se reordenar as redes e relançar o roteamento, tantas vezes quantas foram especificadas no arquivo de configuração.

É importante ressaltar que ao contrário do algoritmo MAZE, que propaga à partir de um ponto, *nosso algoritmo propaga à partir da aresta*. A Figura 4 apresenta a propagação de aresta em uma rede com mais de 2 pontos. Inicialmente é realizada a propagação da origem até o destino mais próximo. Após, na fase de retro-propagação, é realizada a conexão entre estas 2 arestas. Uma vez efetuada esta conexão, a origem para o segundo (e demais) destinos passa a ser as arestas da conexão parcial já efetuada. *Esta é uma importante contribuição do nosso trabalho, pois garante a conexão de redes multi-ponto com o caminho mínimo (não garante-se o caminho ótimo).*



(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 a partir da aresta

(d) roteamento final para os 4 pontos da rede

Figura 4 – Roteador Maze com “propagação de aresta”

As Figura 5 e Figura 6 os algoritmos para encontrar o caminho mínimo entre uma origem ou rede parcialmente roteada à um destino. A rotina “caminho mínimo” busca o destino a partir da aresta *origem*. A retro-propagação marcará as arestas entre a origem e o destino mais próximo, fazendo *routing=1*. A rotina “caminho mínimo modificado” busca o próximo destino a partir de todas as arestas marcadas com *routing=1*.

```
Caminho Mínimo (Origem, lista de destinos)
{ /* Origem: é uma aresta e Destino: lista de arestas */

 G[origem].counter=0
 G[origem].visited=1

 se origem ∈ {lista de destinos}
    retorna origem /* encontrou caminho quando há pinos que compartilham mesmo canal */

 para todas as arestas (e,G) adjacentes à origem
   se e não foi visitada && existem trilhas vagas em e
     { G[e].counter = G[origem].counter + 1
       G[e].visited = 1
       adiciona e à lista de arestas
       se e ∈ {lista de destinos}
          retorna e  /* encontrou caminho */
     }

 enquanto (lista de arestas não vazia)
   { retira k da lista de arestas /* k torna-se aresta corrente */
     para todas as arestas (e,G) adjacentes à k
       se e não foi visitada && existem trilhas vagas em e
         { G[e].counter = G[k].counter + 1
           G[e].visited = 1
           adiciona e à lista de arestas
           se e ∈ {lista de destinos}
                 retorna e  /* encontrou caminho */
         }
    }
}                                        /* e , k indicam aresta, G indica grafo */
```

Figura 5 – Algoritmo da função Caminho Mínimo

```
Caminho Mínimo Modificada (lista de destinos)
{
  para todas as arestas (e,G)
    se g[e].routing = 1          /* busca as arestas em roteamento */
      {
       G[e].counter = 0
       G[e].visited = 1
       se e ∈ {lista de destinos}
            retorna e  /* encontrou caminho */

       para todas as arestas k adjacentes a e
         se k não foi visitada && existem trilhas vagas em k
           { G[k].counter = G[e].counter + 1
             G[k].visited = 1
             adiciona k à lista de arestas
             se k ∈ {lista de destinos}
                 retorna e  /* encontrou caminho */
           }
      }
  enquanto (lista de arestas não vazia)
    { retira k da lista de arestas /* k aresta corrente */
      para todas as arestas e adjacentes a k
        se e não foi visitada && existem trilhas vaga em e
         { G[e].counter = G[k].counter + 1
           G[e].visited = 1
           Adiciona e a lista de arestas
           se e ∈ {lista de destinos}
                retorna e  /* encontrou caminho */
         }
     }
}                                        /* e , k indicam aresta, G indica grafo */
```

Figura 6 - Algoritmo da função Caminho Mínimo Modificado

A última fase do roteamento é a retro-propagação. A retro-propagação é a propagação do(s) destino(s) para a origem, garantindo que o menor caminho (distância) entre dois pontos seja realizado, desde que este caminho tenha sido encontrado. As Figura 7 e Figura 8, apresentam o pseudo-código das duas funções de retro-propagação: entre origem e destino (2 pontos) e entre as aresta e o destino (multi-ponto). A função “retro-propagação” tem como ponto de partida o destino, e recursivamente busca arestas de nível anterior, até que seja alcançada a origem. A função “retro-propagação modificada” também parte do destino, porém busca recursivamente arestas de nível anterior, até encontrar uma aresta pertencente à conexão já parcialmente efetuada.

```
Retro-propagação (origem, destino, flag, nome da rede) /* flag tem valor 0 na 1ª iteração */
{
 se não flag
   { G[destino].used_tracks++
     G[destino].routing = 1
     G[destino].lista de redes = nome da rede
     flag ++
   }

  se (origem = destino)
    retorna 0                    /** CHEGOU **/

  para todas as arestas e adjacentes ao destino
   se G[e].counter = (G[destino].counter-1) && existem trilhas vagas && G[e].visited = 1
     {
      G[e].used_tracks++
      G[e].routing = 1
      G[e].lista de redes  =  nome da rede
      Retro-propagação (origem, e, 1, nome da rede)
     }
}                                                    /* e  indica aresta , G indica grafo */
```

Figura 7 – Algoritmo da função Retro-propagação

```
Retro-propagação Modificada (destino, flag, nome da rede)
{
  se não flag
   { G[destino].used_tracks++
     G[destino].routing = 1
     G[destino].lista de redes = nome da rede
     flag ++
   }

  para todas as arestas e adjacentes a destino
   se G[e].counter = (g[destino].counter-1) && existem trilhas vagas && G[e].visited = 1
     {
      G[e].used_tracks++
      G[e].lista de redes  =  nome da rede
      se G[e].routing
        retorna 0
      senão
       {
        G[e].routing = 1
        Retro-propagação Modificada (e, 1, nome da rede)
       }
     }
}                                        /* e  indica aresta, G indica grafo */
```

Figura 8 – Algoritmo da função Retro-propagação Modificada.

3.2 Versão com modelagem das matrizes de chaves

Nesta versão do software PMaze foram inseridas restrições nas matrizes de chaves e o método de propagação teve de ser alterado. A restrição imposta à matriz de chaves é que uma rede ao longo de todo o seu caminho deverá ocupar sempre a mesmo trilha. Se uma rede entrar pelo lado oeste da matriz de chaves no canal, só poderá sair pela mesma trilha no outros 3 lados (sul, norte e leste). Para que isso aconteça o parâmetro F_s deve ser igual a 3. Este é o valor que estamos usando neste trabalho.

Na propagação, ao invés de utilizar apenas uma trilha no nível seguinte, utiliza-se todas as trilhas disponíveis. Para isto há dois vetores: *propag* (propagação) e *vagas* (trilhas vagas). A origem da rede sendo roteada preenche o vetor *propag* com todas as posições disponíveis. No nível seguinte (1) será feita uma operação lógica e (and) entre o *propag(0)* e o *vagas(1)*, preenchendo-se assim o vetor *propag(1)*. Este procedimento é feito até ser encontrado o destino. Cabe salientar que esse algoritmo é para redes com apenas 2 pontos, o algoritmo para redes com mais de 2 pontos, está em desenvolvimento.

Na retro-propagação o destino poderá conter mais de uma opção para alocação de trilha. Optou-se por escolher a primeira trilha livre no vetor *propag(n)* e realizar a retro-propagação como no modelo sem restrições de chaves.

4. Resultados

A Tabela 2 apresenta os resultados obtidos com a primeira versão do nosso roteador (Pmaze) nos mesmo benchmarks utilizados pelos softwares VPR e SEGA (Fs=3). Cabe salientar que o PMaze não estava utilizando a modelagem de chaves (Fs=∞). Os circuitos *apex2* e *bigkey*, foram roteados com um número de trilhas por canal menor que o utilizado pelo VPR. Os resultados desta tabela foram obtidos através da execução do programa PMaze em uma estação Sun Ultra-Sparc. O maior circuito, *clma*, contendo 8445 redes foi roteado em um FPGA com as dimensões de 92x92 blocos lógicos.

Nome do Arquivo	Nº Trilhas	Ltotal*	CPU (hh:mm,ss)	R & R **	Dimensão	%RR ut. ***	Nº de Redes
Alu4	10	18056	0:22:25,26	3	40x40	55,05%	1536
Apex2	**10**	26068	0:54:54,19	9	44x44	65,83%	1916
Apex4	12	18285	0:6:31,98	2	36x36	57,20%	1271
Bigkey	**6**	16059	5:0:54,90	19	54x54	45,06%	1936
Clma	12	111129	19:57:43,91	7	92x92	54,12%	8445
Des	7	20803	2:45:53,71	23	63x63	36,85%	1847
Diffeq	8	13783	0:13:23,87	3	39x39	55,22%	1562
Dsip	7	13587	2:11:23,32	5	54x54	32,68%	1599
Elliptic	10	39721	4:15:63,54	4	61x61	52,51%	3735
Ex1010	10	60978	4:41:24,75	7	68x68	64,98%	4608
Ex5p	12	16112	0:11:10,65	4	33x33	59,83%	1072
frisc	11	47618	4:18:42,20	9	60x60	59,14%	3576
misex3	10	18369	0:16:35,75	4	38x38	61,97%	1411
pdc	16	81562	1:58:9,97	3	68x68	54,32%	4591
s298	7	18401	2:48:10,62	9	44x44	66,38%	1935
s38417	8	56802	5:33:26,47	5	81x81	53,45%	6435
s38584.1	9	54471	16:23:21,41	3	81x81	45,56%	6485
seq	11	23970	0:16:4,57	3	42x42	60,33%	1791
spla	13	55045	1:1:36,15	3	61x61	55,98%	3706
tseng	6	9158	0:19:4,8	7	33x33	68,02%	1099
Total	**195**		**73:36:46,43**				

Número Total de trilhas utilizadas / ** Número de Repetições ("Rip-up and Re-route) / * Porcentagem de Recursos de Roteamento Utilizado*

***Tabela 2 – Tabela de resultados do PMaze roteando os* Benchmarks (F_s=∞)**

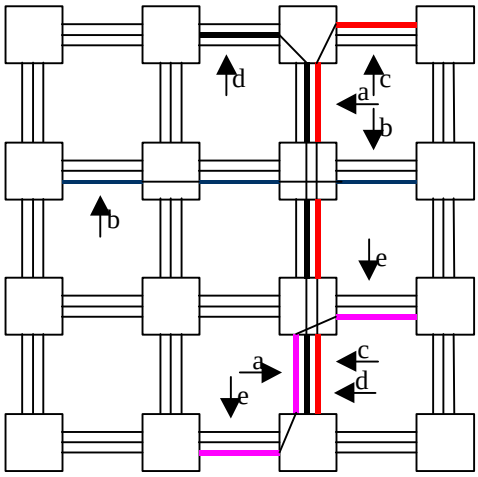
Observa-se nesta primeira implementação que o algoritmo básico funciona corretamente, com um baixo número de repetições (rip-up and re-route). Devido ao fato de não se considerar a modelagem das chaves nesta primeira versão, o número mínimo de trilhas foi atingido, conseguindo-se até mesmo reduzir este em dois casos. A inserção das restrições nas chaves irá aumentar o número de trilhas e o número de repetições, devido ao menor número de opções de roteamento. Cabe ainda a salientar que o tempo de CPU deve ser drasticamente reduzido em versões posteriores do roteador, pois uma ferramenta de CAD deve fornecer ao projetista o resultado rapidamente, para que seja possível explorar o maior espaço de soluções possível. Este elevado tempo de CPU deve-se principalmente à forma de implementação do algoritmo de busca de caminhos no grafo.

A versão do roteador com as matrizes de chaves modeladas, através do parâmetro de flexibilidade (f_s=3) está em desenvolvimento. O problema para redes com apenas dois pontos foi solucionado, validando-o com um circuito de teste com 5 redes de dois pontos, auto-excludentes. O arquivo de dados é apresentado na Figura 9.

```
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
```

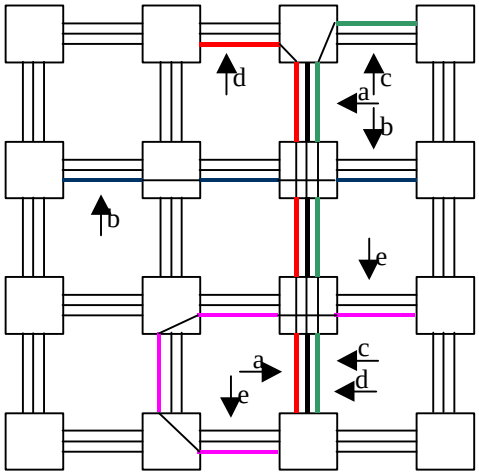
Figura 9 – Arquivo de dados do nosso circuito de teste

Se este circuito for roteado como está apresentado (da rede ***e*** à rede ***a***), haverá bloqueio da rede ***a***. Isso ocorre porque ao rotearmos as redes ***e***, ***d***, ***c*** haverá congestão do canal 3002-3003 (terceira linha, entre a segunda e terceira coluna). Para isto, são necessárias 4 repetições (rip-up e re-route) para que o circuito seja roteado com sucesso. A Figura 10(a) ilustra o circuito com bloqueio, e a Figura 10 (b) ilustra a solução do problema, onde a rede ***e*** foi roteada com um comprimento maior que o mínimo devido a uma congestão no canal.



Roteando-se a seguinte ordem: **e** (trilha 0), **d** (trilha 1) e **c** (trilha 2) ocorre um bloqueio no canal 3002-3003, impossibilitando a realização da conexão **a**.

(a)



A solução correta é rotear as 3 redes **a**, **c** e **d** primeiro e depois realizar o roteamento da rede **e**, através de um caminho alternativo, com comprimento maior que o mínimo.

(b)

Figura 10 – Roteamento com modelagem de matriz de chaves

5. Conclusão

Este artigo apresentou a implementação de um algoritmo para roteamento de FPGAs tendo como base o algoritmo MAZE, com número de iterações de "rip-up e re-route" parametrizável. Nossas principais contribuições são: (1) eficiente modelagem da arquitetura de FPGAs utilizando a biblioteca LEDA; (2) método de roteamento para redes multi-ponto baseado em propagação de aresta; (3) método de roteamento (em desenvolvimento) para matriz de chaves com restrições.

O trabalho presente consiste em validar o método de propagação com restrição de chaves para redes multi-ponto, e comparar os resultados obtidos com o software VPR. Uma vez o sistema validado, tem-se como trabalho futuro a paralelização do algoritmo, utilizando programação multi-thread. O paralelismo será a nível de redes e não a nível de algoritmo. A proposta é particionar o FPGA em quadrantes, roteando-se em paralelo as redes que são exclusivas a cada quadrante.

6. Bibliografia

[ALE95] M. J. Alexander e G. Robins. *"New Performance-Driven FPGA Routing Algorithms"*. DAC, 1995, p. 562 – 567.

[ALT98] Home-Page da Altera – http://www.altera.com/

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

[BET97a] V. Betz e J. Rose, *"VPR: A New Packing, Placement and Routing Tool for FPGA Research"*, International Workshop on Field Programmable Logic and Applications, 1997. Disponível para Download em http://www.eecg.toronto.edu/~jayar/software.html.

[BET97b] V. Betz. *"VPR and VPACK User's Manual (Version 3.99)"*. Disponível para Download: http://www.eecg.toronto.edu /~vaughn, Mai 1997, p. 17.

[BRO92] S. D. Brown, J. Rose e Z. G. Vranesic. *"A detailed Router for Field-Programmable Gate Arrays"*. IEEE Trans on CAD, Mai 1992, pp. 620-628.

[CON94] J. Cong e Y. Ding. *"FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Look-Up Table Based FPGA Designs"*. IEEE Trans. Computer Aided Design, Jan. 1994, pp. 1 – 12.

[EBE95] C. Ebeling, L. McMurchie, S. A. Hauck e S. Burns. *"Placement and Routing Tools for Triptych FPGA"*. IEEE Trans. on VLSI, Dec. 1995, pp. 473-482.

[KIR83] S. Kirkpatrick, C. D. Gellat, Jr. e M. P. Vecchi. *"Optimization by Simulated Annealing"*. Sciense, 13 Mai 1983, pp. 671 – 680.

[LEE95] Y-S. Lee, A. C.-H. Wu. *"A Performance and Routability Driven Router for FPGAs Considering Path Delays"*. IEEE/ACM Design Automation Conference, 1995, p. 557 – 561.

[LEM93] G. G. F. Lemieux e S. D. Brown. *"A Detailed Routing Algorithm for Allocating Wire Segments in Field-Programmable Gate Arrays"*. 4th ACM/SIGDA Physical Design Workshop, 1993.

[LEM97] G. G. F. Lemieux, S. D. Brown, D. Vranesic. *"On Two-Step Routing for FPGAs"*. ISPD, 1997, p. 60 – 66.

[MEH89] K. Mehlhorn e S. Näher. *"LEDA, a Library of Efficient Data Types and Algorithms"*. Procedings of 14th Symposyum on Mathematical Foundations of Computer Science, LNCS Vol. 379, pp. 88-106, 1989.

[ROS90] J. Rose. *"Parallel Global Routing for Standard Cells"*. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 9, Nº 10, pp. 1085 – 1095, Out. 1990.

[SEN92] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R.Stephan, R. K. Brayton, e A. L. Sangiovanni-Vincentelli. *"SIS: A System for Sequencial Circuit Synthesis"*. Technical Report UCB/ERL M92/14, Laboratório de Pesquisa em Eletrônica, Universidade da Califórnia, Berkeley, CA 94720, Mai 1992.

[THA94] S. Thakur, D. F. Wong e S. Muthukrishnan. *"Algorithms for a Switch Module Routing Problem"*. IEEE/ EURO-DAC '94, European Design Automation Conference with EURO-VHDL '94, Set. 94.

[WU94] Y.-L. Wu, M. Marek-Sadowska. *"An Efficient Router for 2-D Field Programmable Gate Arrays"*. EDAC, 1994, p. 412 – 416.

[WU95] Y.-L. Wu, M. Marek-Sadowska. *"Orthogonal Greedy Coupling – A New Optimization Approach to 2-D FPGA Routing"*. DAC, 1995, p. 568 – 573.

[XIL98] Home-Page da Xilinx - http://www.xilinx.com/

*The author Fernando Moraes would like to gratefully acknowledge the continued support of the "Conselho Nacional de Desenvolvimento Científico e Tecnológico" (**CNPq**). Research grant 522939/96-1.*