HEURÍSTICAS PARA ROTEAMENTO DE FPGAS

Gustavo Rezende Carvalho
UFF
Niterói, Rio de Janeiro, Brasil
gcarvalho@ic.uff.br

Lucídio dos Anjos Formiga Cabral
UFPB
João Pessoa, Paraíba, Brasil
lucidiocabral@gmail.com

Antonio Carlos Cavalcanti
UFPB
João Pessoa, Paraíba, Brasil
accaval@hotmail.com

RESUMO

O presente artigo trata do problema de roteamento de circuitos para Field Programable Gate Arrays (FPGAs). Em função da natureza combinatória do problema, métodos heurísticos são comumente utilizados para gerar soluções de boa qualidade em um tempo computacionalmente aceitável. Neste contexto, um procedimento baseado na metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) com um procedimento de busca local baseado em ILS (Iterated Local Search) é proposto. O algoritmo foi testado em instâncias encontradas na literatura, benchmark MCNC, explorando os critérios de tempo crítico e números de trilhas, onde se mostrou capaz de melhorar 55% das instancias no critério de tempo critico.

Palavras chave: Roteamento, FPGA, GRASP

ABSTRACT

The present work deals with the Routing Circuits for Field Programable Gate Arrays (FPGAs). Due to the combinatorial nature of the problem, heuristics methods are commonly used to generate good quality solutions in an acceptable computational time. In this context, a procedure based on GRASP (Greedy Randomized Adaptive Search Procedure) with a procedure of local search based on ILS (Iterated Local Search) is proposed. The algorithm has been tested in benchmark problems found in the literature, MCNC, exploring timing-driven and channel-width criteria, being able to improve 55% of the benchmarks on timing-driven criteria.

KEYWORDS. Routing, FPGA, GRASP

Introdução

Com o surgimento em 1985 da tecnologia *Field-Programmable Gate Arrays* (FPGA), criada pelos co-fundadores da Xilinx, Ross Freeman e Bernard Vonderschmitt, o paradigma de hardware programável é criado. Projetada para competir com a *Complex Programmable Logic Device* (CPLDs), suas principais vantagens sobre suas concorrentes é ser reprogramavel através de uma linguagem de descrição de hardware (*hardware description language* – HDL) ou um design esquemático do circuito. Isto se torna possível pela utilização, em sua maioria, de tecnologia *Synchronous* RAM (SRAM) para configurar os blocos lógicos.

Para o primeiro FPGA comercial, a XC2064 da Xilinx, com apenas 64 blocos lógicos e LUTs de 3 entradas, não era uma tarefa difícil definir manualmente a programação do FPGA para implementar o circuito desejado, mas nem por isso deixava de ser trabalhoso. Para programar um FPGA, no mínimo é necessário ter o circuito desejado em mãos, mapeado logicamente, posicionar os pinos I/O do circuito e as funções lógicas nos blocos lógicos do FPGA e roteá-la, ou seja, configurar os *switchs* presentes no FPGA para definir as ligações do circuito.

Tendo em vista que automatizando este processo uma grande quantidade de tempo poderia ser poupada, conseqüentemente, ferramentas de CAD para FPGAs foram criadas. Um sistema de projeto típico deverá incluir suporte para os seguintes passos: entrada do projeto inicial, otimização lógica, simulação, mapeamento lógico, posicionamento e roteamento. Essas etapas interferem diretamente no desempenho do circuito implementado no FPGA e o tempo gasto em cada uma destas etapas é determinante no *time to market* dos projetos.

Sendo as trilhas o número de segmentos de fios de cada canal do FPGA, em 1997, Vaughn Betz propõe um desafio chamado de "*FPGA Place and Route Challenge*" (Desafio de posicionamento e roteamento de FPGA), no qual pagaria 1,00 US$ para cada trilha que utilizassem a menos que o programa VPR, para posicionar e rotear os circuitos do *benchmark* MCNC. Em função da natureza combinatória do problema de posicionar e rotear, NP-Difícil, podendo ser provado através de coloração de grafos, vários trabalhos foram publicados propondo métodos heurísticos para gerar soluções de boa qualidade em um tempo computacional aceitável.

A principal contribuição deste trabalho reside em propor um algoritmo de roteamento baseada nas metaheurísticas GRASP e ILS com o objetivo primordial de diminuir o tempo do caminho crítico do circuito roteado na FPGA. Outra questão abordada neste trabalho foi quanto ao *rip-up*, é que a seleção das *nets* a serem "ripadas" ou re-roteadas não foi bem explorada, pois a ordem em que as *nets* são roteadas interfere diretamente nos resultados e este estudo foi deixado em aberto por Keith So (SO, 2007).

O presente artigo está organizado da forma que se segue: na seção 1 descreve os fundamentos da FPGA, assim como arquiteturas e suas peculiaridades; a seção 2 dá uma breve explanação sobre a ferramenta VPR 5.0; a seção 3 descreve a metaheurística GRASP; a seção 4 descreve a metaheurísitca ILS; seção 5 demonstra o algoritmo proposto; na seção 6 encontram-se os resultados obtidos; e na ultima seção as conclusões.

1. Fundamentos da FPGA

1.1 Arquiteturas de FPGAs

Todos os FPGAs são compostos de três componentes fundamentais: blocos lógicos (L), blocos de entrada/saída (E/S), e roteamento programável. Um circuito é implementado em um FPGA programando-se cada bloco lógico para gerar uma parte da lógica requerida pelo sistema, cada bloco (E/S) para agir como um pino (*pad*) de entrada ou de saída e o roteamento programável é configurado para fazer todas as conexões necessárias entre blocos lógicos e blocos E/S (CABRAL,2001).

1.2 Arquiteturas de Roteamento de FPGAs

A arquitetura de roteamento de um FPGA é a maneira através da qual são configuradas as chaves e segmentos de fios para realizar as interconexões dos blocos lógicos. Nesta seção apresenta-se uma arquitetura utilizada comercialmente como referência para o estudo das arquiteturas de roteamento de forma genérica, observando-se que cada fabricante possui um modelo particular. Os FPGAs comerciais podem ser classificadas em três grandes grupos, baseados em sua arquitetura de roteamento. Os FPGAs da XILINs, Lucent e Vanlis são baseadas em matrizes (*island-style*) enquanto os FPGAs da Actel são baseados em linhas e os da ALTERA são hierárquicas.

Em um modelo genérico de arquitetura de roteamento de FPGAs, segundo Torok (2001), podemos identificar várias características:

- Um segmento de fio é contínuo.
- Uma ou mais chaves podem ser ligadas a segmentos de fio.
- Cada final de um segmento de fio está tipicamente ligado a uma chave programável.
- Uma trilha é uma sucessão em linha de um ou mais segmentos de fio.
- Um canal de roteamento é um grupo de trilhas paralelas.

Levando as características anteriores em consideração, será admitido que todos os canais tenham o mesmo número de trilhas. O número de trilhas em cada canal é denotado por *W*. As trilhas em cada canal horizontal são numeradas de baixo para cima e, em cada canal vertical, da esquerda para a direita. Um número associado a uma trilha corresponde ao seu identificador. A flexibilidade do bloco *C*, denominado por *Fc*, é definida como o número de trilhas ao qual um pino lógico pode se conectar. A Figura 1 (b) ilustra um bloco *C*, onde observa-se as trilhas que passam através dele e podem ser conectadas aos pinos de blocos lógicos via um conjunto de *switches*. Na figura 1 (b), uma opção de roteamento é representado por um *X*, de modo que cada pino pode ser conectado para duas trilhas verticais, ou seja, *Fc = 2*. Já na figura 1 (c), existem três pinos lógicos no bloco *C* e cada um deles pode se conectar a qualquer uma das quatro trilhas, logo, conclui-se que a flexibilidade deste bloco é quatro.

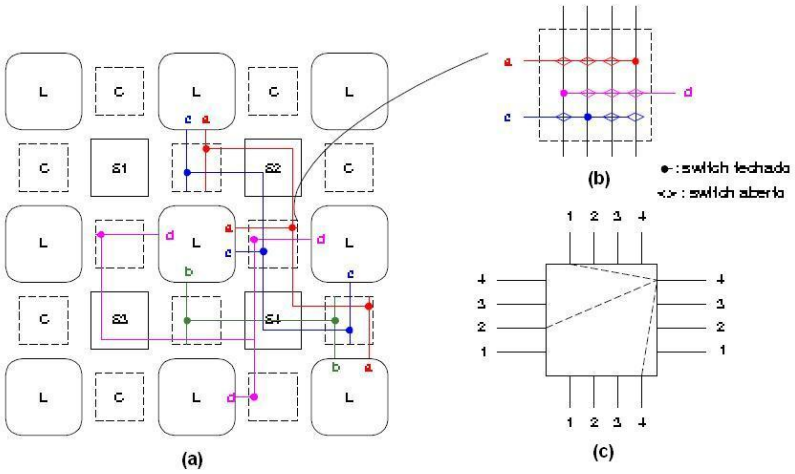

Figura 1: (a) Exemplo de roteamento. (b) Um bloco C. (c) Um bloco S.

Um bloco *S* é uma caixa retangular de *switches* (chaves programáveis) que conecta segmentos de fios em dois canais distintos. Dependendo da topologia, cada segmento de fio sobre um lado do bloco S pode ser conectado para um ou para todos, ou ainda para alguma fração dos segmentos de fios sobre cada um dos outros lados de um bloco *S*. A flexibilidade do bloco *S* é

dada pelo parâmetro *Fs*, que define o número de outros segmentos de fios aos quais um segmento em um bloco *S* pode se conectar.

Um exemplo de bloco *S* aparece na Figura 1 (c), onde cada linha tracejada representa um *switch* de roteamento programável – nesta figura, *Fs = 3*. A flexibilidade e a topologia adotada nos blocos *S* tem considerável impacto na roteabilidade das redes (*nets*). A flexibilidade do bloco S, denominada por *Fs*, é definida como o número de segmentos de fios ao qual cada segmento de fio pode conectar-se.

1.3 Posicionamento

Algoritmos de posicionamento determinam qual bloco lógico dentro de um FPGA deverá implementar cada uma das funções lógicas requeridas pelo circuito. As metas de otimização são:

- Posicionar, de forma aglutinada, blocos lógicos conectados minimizando assim o comprimento total de fio requerido (*wirelength-driven placement*).
- Posicionar blocos lógicos de modo a balancear a densidade de fiação ao longo dos canais do FPGA (*routability-driven placement*).
- Maximizar a velocidade do circuito (*timing-driven placement*).

Atualmente as três principais classes de posicionadores em uso são *min-cut* (*partitioning-based*) (HUANG,1997), analítico (ALPERT, 1997), (KAHNG, 1992) e um destaque maior para o *simulated annealing* por ter sido utilizado nos artigos mais recentes, (BETZ,1997), (LUU,2009), (SWARTZ,1995). No caso do *simulated annealing*, tem-se a vantagem de ser possível adicionar facilmente novas metas de otimização ou restrições ao posicionador.

1.4 Roteamento

Após as localizações (posições) para todos os blocos lógicos terem sido definidas, um roteador determina quais *switches* programáveis devem ser ligadas para conectar todos os pinos de entrada e saída do circuito. É usual representar a arquitetura de roteamento de um FPGA através de um grafo GA=(N,D) orientado (dirigido) (EBELING, 1995), (NAG, 1994). Cada fio e cada pino de bloco lógico torna-se um nó $n \in N$, no grafo de recursos de roteamento e cada

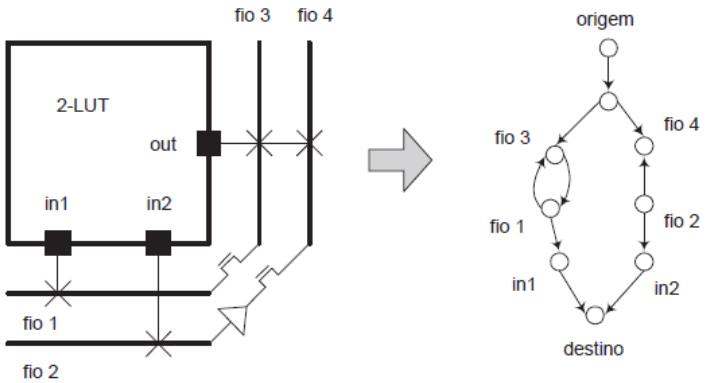

Figura 2: Modelando o roteamento detalhado FPGA como um grafo orientado.

switch torna-se um arco orientado $a \in D$ (para *switches* unidirecionais, tais como *buffers*) ou um par de arcos orientados (para *switches* bidirecionais, tais como transistores de passagem) entre dois nós apropriados. A figura 2 mostra o grafo de roteamento de recursos correspondendo a uma porção de uma FPGA cujos blocos lógicos contém uma única LUT com 2-entradas e 1-saída.

Alguns trabalhos têm representado FPGAs como grafos não orientados (ALEXANDER, 1994), mas uma representação de grafo orientado é necessária se *switches* direcionais, do tipo *buffers tri-state* e multiplexadores, para serem modelados corretamente.

Freqüentemente blocos lógicos FPGA têm pinos logicamente equivalentes. Por exemplo, todos os pinos de entrada de uma LUT são logicamente equivalentes. Isto significa que um roteador pode completar uma dada conexão usando qualquer um dos pinos de entrada de uma LUT. Esta equivalência lógica é modelada no grafo de roteameto de recursos adicionando-se nós fontes (*sources*), nos quais todas as redes começam e nós destinos (*sinks*) onde todos os terminais de rede terminam. Vale salientar que existe um nó fonte para cada conjunto de pinos de saída logicamente equivalentes e um arco do nó fonte para cada um destes pinos de saída. De forma similar, existe um nó destino para cada conjunto de pinos de entrada logicamente equivalente e um arco de cada um desses pinos de entrada para um nó destino. Portanto, cada rede de sinal consiste em um único nó de G_A, nó origem do sinal, junto com um subconjunto de nós correspondente aos seus destinos (*sinks*).

Rotear uma rede de sinal (*net*) corresponde a encontrar uma árvore neste grafo de recurso de roteamento entre os nós que representam os pinos de blocos lógicos a serem conectados, de tal forma que todos os nós destinos possam ser alcançados no nó origem. Quando o número de fios (recursos de roteamento) de um FPGA é limitado deve-se buscar por árvores as mais curtas possíveis. É importante ressaltar que o roteamento de uma rede (*net*) não use recursos de roteamento que outra rede (*net*) necessite.

Uma meta adicional de otimização é construir redes sobre ou próximas do caminho crítico, roteando-as usando caminhos mais curtos e recursos de roteamento mais rápidos. Roteadores que tentam aperfeiçoar desempenho desta maneira são chamados de *timing-drive* e desde que grande parte do atraso em um FPGA se deve ao roteamento programável, torna-se essencial aplicar um roteador deste tipo para que se obtenha circuitos com bons desempenhos (CABRAL, 2001).

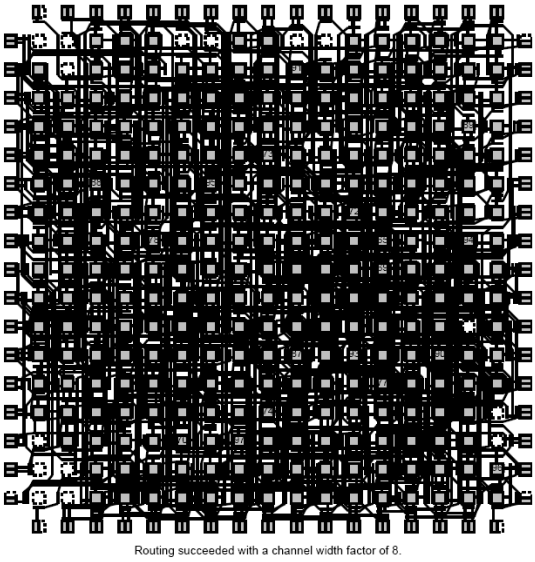

Figura 3: Exemplo de roteamento de FPGA (Imagem retirada da ferramenta VPR).

2. VPR

O Versatile Place and Route (VPR) é uma ferramenta de CAD para alocação e roteamento de FPGA's, publicado por Vaughn Betz e Jonathan Rose no ano de 1997. Em 2009, é lançada a versão 5.0 do VPR e vem suprir a necessidade de aceitar como entrada os tipos de arquiteturas de

FPGAS modernas desenvolvidas ao longo dos anos. O VPR passou a ser um conjunto de ferramentas que incluem quatro novos significantes recursos (LUU, 2009):

- Suporta uma gama de arquiteturas *single-drive* (ALTERA, 2008), (LEMIEUX, 2004) e (XILINX, 2008).
- Aceita arquiteturas heterogêneas, ou seja, que possua *hard-blocks*.
- Provê modelos elétricos otimizados de uma grande gama de arquiteturas em diferentes processos tecnológicos, incluindo negociação de a*rea-delay* para cada arquitetura.
- Um conjunto de testes de regressão para checar funcionalidades e a qualidade dos resultados das saídas das ferramentas.

Seu código é aberto e está disponível no site http://www.eecg.utoronto.ca/vpr e foi utilizado, neste trabalho, sua ferramenta de alocação e o mecanismo de rotear nets do tipo *timing-driven* assim como o verificador de roteamento.

3. GRASP

A metaheurística Greedy Randomized Adaptive Search Procedure (GRASP) foi inicialmente desenvolvida por Feo e Resende (FEO, 1995) e consiste em um procedimento iterativo de duas etapas, a saber: (i) fase de construção, na qual uma solução viável é gerada; (ii) fase de busca local, cuja vizinhança da solução construída é pesquisada até que se encontre um ótimo local.

4. ILS

Suponha que uma solução ótima local tenha sido encontrada por um algoritmo de busca local. Ao invés de reiniciar o mesmo procedimento a partir de uma solução completamente nova, a metaheurística ILS aplica repetidamente uma busca local às soluções iniciais obtidas através de perturbações das soluções ótimas locais previamente visitadas (STÜTZLE, 1998).

Segundo Lourenço et al. (2002), a idéia essencial da ILS está atribuída ao fato desta metaheurística realçar seu foco em um subespaço menor, ao invés de considerar o espaço completo de soluções, definido por aquelas que são ótimos locais de um dado procedimento de otimização.

5. Algoritmo

O algoritmo proposto consiste em um procedimento de busca adaptativo, guloso e randomizado (Greedy Randomized Adaptive Search Procedure - GRASP) que utiliza o método de *rip-up* com ordenação seletiva e re-roteamento. Para realização da busca local, usamos *rip-up* e re-roteamento baseado na metaheurística ILS.

5.1 Algoritmo Proposto

O algoritmo possui duas etapas: a primeira corresponde à construção gulosa randomizada e a segunda é relativa à fase de refinamento, em que se utiliza a metaheurística ILS. O pseudocódigo encontra-se ilustrado na Figura 4, onde s* representar a melhor solução e α um parâmetro que define o grau aleatoriedade da escolha da Lista de Candidatos Restrita - LCR.

```
Procedimento GRASP/ILS(Max_iteracoes, semente, α)
1. CarregarDados( );
2. para k = 1,..., Max_Grasp_iteracoes faça
3.     s ← Construção_Gulosa_Aleatoria(semente, α);
4.     s ← ILS(N(.), f(.), s);
5.     Atualizar_Solucao(s, s*);
6. fim;
7. retornar s*;
Fim GRASP/ILS
```

Figura 4: Procedimento GRASP/ILS

5.1.1 Procedimento Construtivo

O procedimento construtivo consiste em rotear as *nets* através da LCR (Lista de Candidatos Restrita) usando como critério tempo de atraso das nets para construí-la, diferentemente do VPR, no qual as *nets* são roteadas na ordem em que são dispostas no arquivo de entrada de *netlist*. Também foi testado outro parâmetro para construir a LCR: o critério das *nets* com o maior número de terminações, cujos resultados não foram promissores.

```
Procedimento Construção_Gulosa_Aleatoria(semente, α)
1.para i = 1,..., max_tentativas faça
2.    para k = 1,..., max_iteracoes faça
3.       boleano viavel ← TRUE;
4.       para j = 1,..., Numero_de_Nets faça
5.          escolha_Gulosa_da_Net(α)
6.          viavel ← Roteia_net( );
7.          se (!viavel) break;
8.          senão rip-up( );
9.                penaliza-recursos-em-comum( );
10.      fim;
11.      se (viavel) break;
12.   fim;
13.   atualizar_Solucao(s, s*);
14.fim;
15.para i = 1,..., max_tentativas faça
16.    para k = 1,..., max_iteracoes faça
17.       boleano viavel ← TRUE;
18.       para j = 1,..., Numero_de_Nets faça
19.          escolhe_ net_com_maior_delay(s*)
20.          viavel ← Roteia_net( );
21.          se (!viavel) break;
22.          senão rip-up( );
23.                penaliza-recursos-em-comum( );
24.       fim;
25.       se (viavel) break;
26.     fim;
27.     Atualizar_Solucao(s, s*);
28.fim;
29.retornar s*;
Fim Construção_Gulosa_Aleatoria
```

Figura 5: Procedimento Construtivo

Após a seleção da *net* na LCR, o roteamento dos pares *source-sink* de cada *net* são realizados através do procedimento de roteamento *timing-driven* encontrado no VPR 5.0 (LUU, 2009). Assim que todos os pares forem roteados, uma segunda *net* é selecionada na LCR e seus

pares *source-sink* são roteados e assim sucessivamente para todas as *nets*. Caso as *nets*, no final deste procedimento, estiverem compartilhando algum recurso em comum, significa que o roteamento ainda não é viável. Os recursos compartilhados são penalizados e as *nets* são novamente roteadas até um máximo de *max_Iteracoes* tentativas.

De *max_tentativas* de roteamento é retirado o melhor roteamento viável. Foi observado que, ao final do roteamento, tínhamos uma lista de *nets* ordenadas por atraso. Retirar as penalidades dos recursos de roteamento e tentar rotear esta lista novamente mostrou melhoria em vários testes. Observado isso, um procedimento *guloso* de roteamento *timing-driven* foi adicionado ao algoritmo para rotear esta lista. Nesta etapa, o *max_iteracoes* tem um valor maior que o *max_Iteracoes* da etapa de roteamento com LCR. Ao se obter um roteamento viável, caso este roteamento seja melhor, ele passa ser a melhor solução obtida, caso não seja, sua lista de nets, no final do roteamento, é ordenado por atraso e utilizado para uma nova tentativa de roteamento. Este processo é repetido *max_tentativas* vezes.

5.1.2 Procedimento de Busca Local (ILS)

A fase de busca local, responsável por refinar a solução inicialmente obtida na fase de construção, é efetuada pelo algoritmo baseado na metaheurística ILS. Consiste basicamente em re-rotear as *nets* da melhor solução da fase de construção utilizando o mesmo roteador *timing-driven*, porém uma pequena perturbação na lista de *nets* é que causa a convergência a soluções na vizinhança da solução obtida na fase de construção.

Na Figura 6, podemos observar um exemplo de perturbação. Na primeira tabela, temos a lista de nets ordenadas por tempo de atraso obtido da melhor solução da fase de construção. A perturbação consiste em colocar a net de maior atraso no final da lista de nets a serem roteadas. Com a nova lista, tenta-se rotear utilizando o método *timing-driven*. Em caso de não haver melhorias, utilizamos a lista de nets da primeira tabela de atraso e causamos uma nova perturbação. Colocamos a segunda net com maior atraso para o final da lista e tenta-se rotear novamente.

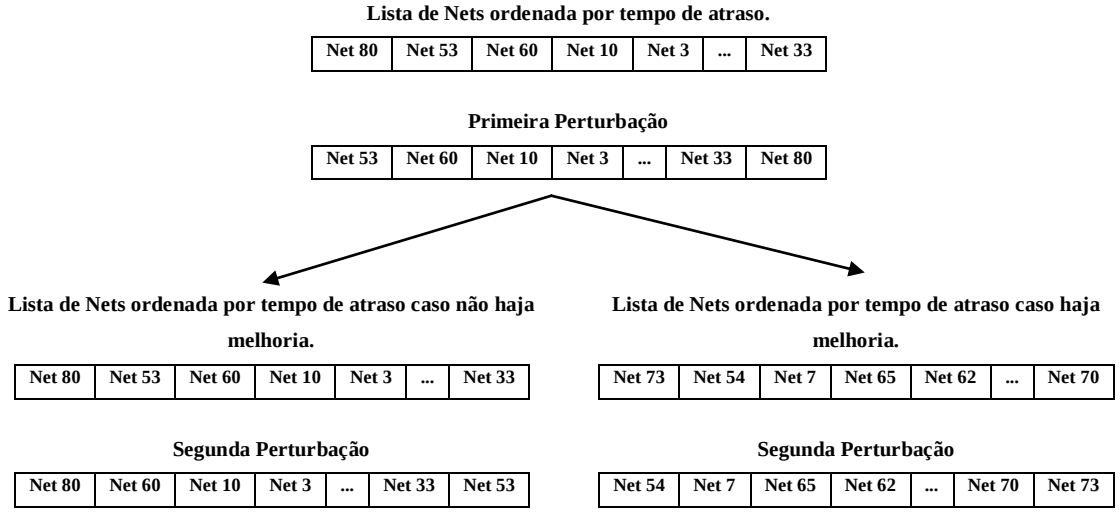

Figura 6: Exemplo de perturbações com melhor e sem melhoria

Quando houver melhoria, obtemos uma nova lista de *nets* ordenadas e recomeça o processo de perturbação e re-roteamento sobre esta lista. Sempre que houver uma melhoria, o processo recomeça. Caso não haja melhorias, o processo termina após "*ils_iteracoes*" iterações.

```
Procedimento ILS(N(.), f(.), r, s)
para i = 1,..., ils_iteracoes faça
  para k = 1,..., max_iteracoes faça
    boleano viavel ← TRUE;
    perturbação( ils_iteracoes );
    para j = 1,..., Numero_de_Nets faça
      escolhe_ net_com_maior_delay(s*)
      viavel ← Roteia_net( );
      se (!viavel) break;
      senão rip-up( );
        penaliza-recursos-em-comum( );
    fim;
    se (viavel) break;
  fim;
  atualizar_solucao(s, s*);
  se( s==s* ) i = 1;
fim;
Fim ILS
```

Figura 7: Procedimento de busca local (ILS)

5.1.3 Critério de Parada

Uma das peculiaridades do caminho crítico nos circuitos roteados para FPGA é que se o roteamento nunca fizer um caminho no eixo contrário à direção do ponto de destino, o caminho pode ser considerado ótimo.

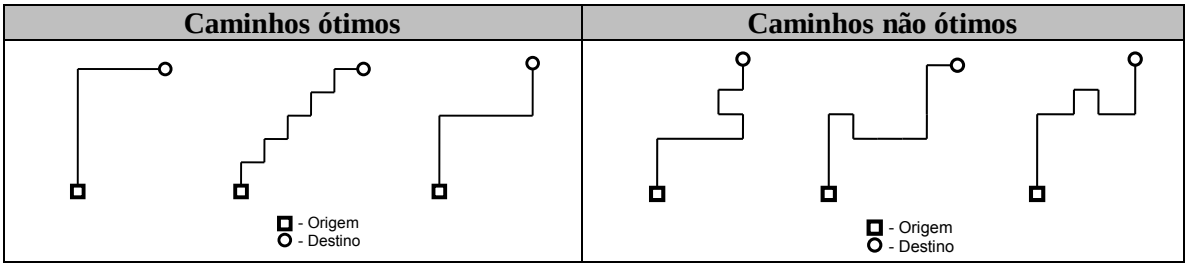

Figura 8: Exemplos de caminhos ótimos e não ótimos

Observado que na primeira iteração do roteador *timing-driven* o roteamento é feito com baixa penalidade nas trilhas, permitindo que os caminhos se sobreponham, eles atingem nesta etapa valores próximos ao ótimo em relação ao caminho crítico. Para ser mais exato, em média, 0,01% pior que o melhor caminho crítico permitido no circuito, considerando um *placement* específico para o circuito. Tendo este valor em mãos, se qualquer roteamento viável obtido durante a execução do algoritmo atingir valores iguais ou menores a este a busca é interrompida.

6. Resultados

6.1 Benchmark

Os resultados foram obtidos através do roteamento dos 20 circuitos do benchmark MCNC na arquitetura de FPGA do tipo Island Style, mapeada tecnologicamente para células compostas por uma LUT-4 (Lookup table de quatro entradas) e um flip-flop, conforme definido em BETZ (1997).

Neste trabalho todos os dados experimentais utilizados para alimentar o sistema em estudo são modelos não reais e em todas as simulações realizadas foram considerados os parâmetros default do VPR. Para comparação de tempo crítico é verificado o menor tamanho de trilha necessária para rotear os circuitos do benchmark MCNC, o VPR foi executado em uma única iteração fixando o número de trilhas para o valor mínimo conhecido.

Circuito	Trilhas	Circuito	Trilhas	Circuito	Trilhas	Circuito	Trilhas
alu4	11	**Des**	10	**ex1010**	11	**s38417**	8
apex2	12	**Diffeq**	8	**frisc**	15	**s38584**	8
apex4	13	**Dsip**	8	**misex3**	11	**seq**	12
bigkey	9	**Elliptic**	12	**pdc**	19	**spla**	15
clma	14	**ex5p**	14	**s298**	8	**tseng**	8

Tabela 1: Número de trilhas obtidas pela configuração default do VPR 5.0

O processo de alocação foi realizado pela ferramenta de alocação do VPR 5.0. Os arquivos que resultaram deste processo foram utilizados tanto para o roteamento da ferramenta VPR 5.0 como para o roteamento da heurística GRASP/ILS.

6.2 Testes Timing-Driven

A tabela 2 ilustra os resultados obtidos com o VPR 5.0 e pela heurística GRASP/ILS obtidos sobre o *benchmark* MCNC, onde "Melhor" é o melhor resultado obtido naquele circuito entre 10 execuções, "Média" é a média das 10 execuções, "Tempo" é o tempo em que o VPR 5.0 demorou para encontrar a solução e "Tempo Médio" é a media de tempo do algoritmo GRASP/ILS para obter os resultados nas 10 execuções. Os valores das soluções estão expressos em nanosegundos e do tempo de execução estão expressos em segundos. Cinco valores de α

	Critical Path							
	VPR 5.0		GRASP/ILS ALFA 0.2					
Circuito	**Critical Path**	**Tempo**	**Melhor**	**Var %**	**Média**	**Var %**	**|Tempo Médio**	**Var %**
alu4	103,68 ns	22 s	78,73 ns	24,07	82,36 ns	20,56	682 s	-3000,00
apex2	94,46 ns	15 s	94,46 ns	0,00	94,97 ns	-0,54	166 s	-1006,67
apex4	105,03 ns	16 s	82,25 ns	21,68	89,50 ns	14,79	666 s	-4062,50
bigkey	76,64 ns	12 s	74,28 ns	3,08	74,46 ns	2,85	45 s	-275,00
clma	207,59 ns	103 s	206,95 ns	0,30	208,36 ns	-0,37	1636 s	-1488,35
des	91,12 ns	27 s	90,98 ns	0,15	91,02 ns	0,11	344 s	-1174,07
diffeq	62,62 ns	5 s	62,59 ns	0,05	62,60 ns	0,04	216 s	-4220,00
dsip	56,86 ns	28 s	56,85 ns	0,01	57,39 ns	-0,93	1074 s	-3735,71
elliptic	143,01 ns	46 s	108,41 ns	24,19	108,84 ns	23,89	218 s	-373,91
ex5p	93,46 ns	11 s	72,60 ns	22,32	74,89 ns	19,87	371 s	-3272,73
ex1010	189,98 ns	58 s	186,06 ns	2,07	186,10 ns	2,04	1296 s	-2134,48
frisc	134,43 ns	40 s	133,80 ns	0,47	133,93 ns	0,37	861 s	-2052,50
misex3	95,57 ns	12 s	73,83 ns	22,75	79,12 ns	17,21	535 s	-4358,33
pdc	150,98 ns	56 s	150,97 ns	0,01	151,44 ns	-0,30	613 s	-994,64
s298	141,31 ns	9 s	130,75 ns	7,47	132,15 ns	6,48	901 s	-9911,11
s38417	99,64 ns	40 s	99,03 ns	0,62	99,40 ns	0,24	1633 s	-3982,50
s38584	114,76 ns	91 s	114,76 ns	0,00	114,76 ns	0,00	522 s	-473,63
seq	90,54 ns	15 s	78,95 ns	12,80	80,10 ns	11,54	364 s	-2326,67
spla	139,96 ns	37 s	120,45 ns	13,94	129,32 ns	7,60	1620 s	-4278,38
tseng	55,26 ns	4 s	54,65 ns	1,11	56,97 ns	-3,09	121 s	-2925,00
TOTAL	2246,88 ns	647 s	2071,34 ns	7,81	2107,66 ns	6,20	13884 s	-2045,90

Tabela 2: Resultados *Critical Path* do GRASP/ILS com α 0.2

foram utilizados, sendo eles: 0.1, 0.2, 0.3, 0.4 e 0.5.

O primeiro teste foi realizado com o α=0.1. Comparando o campo "Média" com os resultados do VPR 5.0, o roteador GRASP/ILS apresentou um melhoria de 5,14% nos caminhos críticos dos circuitos MCNC em relação ao VPR 5.0. Foi o α que apresentou a 4ª melhor média entre os resultados. O α=0.2 apresentou um melhoria de 6,20%, sendo o melhor resultado apresentado; α=0.3 obteve 5,61%, sendo o segundo melhor resultado; α=0.4 com 4,99% de melhoria obtendo o pior resultado; α=0.5 com 5,54% obtendo o terceiro melhor resultado.

Na tabela 2, com $\alpha = 0.2$, pode-se observar bons resultados nas instâncias "alu4", "apex4", "elliptic", "ex5p", "misex", "seq" e "spla", com um melhoramento que varia de 12,80% até 24,19%. Nas instâncias "bigkey", "ex1010", "s298" e "tseng" houve um melhoramento não muito expressivo que varia de 1,11% até 7,47%. Os circuitos restantes: "apex2", "clma", "des", "diffeq", "dsip", "frisc", "pdc", "s38417", não havia como melhorar os resultados, pois estão pertos de um roteamento considerado possivelmente ótimo no critério *Critical Path* para a quantidade de trilhas que foram roteados, apresentando assim uma variação que vai de 0,00% até 0,62%.

Quanto ao tempo, conseguiu-se tempo competitivo em poucas instâncias devido ao critério de parada que foi adicionado ao algoritmo. O critério de parada obteve maiores ocorrências para as instâncias que obtiveram resultados pertos do melhor valor conhecido e durante a fase de construção, ou seja, as instâncias que tiveram variação entre 0,00% e 0,62%. Nos outros casos, o tempo foi muito superior ao VPR 5.0 chegando, no pior caso, há quase 70 vezes mais tempo. Esta característica se repetiu para todos os α's.

Conclusões

Este artigo tratou do Problema de Roteamento de Circuitos Integrados Baseados em FPGAs. Para resolvê-lo, um algoritmo baseado na metaheurísticas GRASP e ILS foi proposto. É a primeira vez que estas metaheurísticas foram utilizadas para solucionar este problema.

Para construção da solução inicial, foi utilizado o método proposto pelo GRASP, com a utilização de uma lista de candidatos restrita (LCR) para gerar soluções variadas. O método de roteamento de nets utilizado é o *timing-driven* encontrado no VPR 5.0 (LUU,2009). Na etapa de busca local, um algoritmo baseado na metaheurística ILS foi utilizado onde a perturbação consistia em colocar uma das nets com maior tempo de atraso no final da lista de nets ordenadas pelo critério do atraso para em seguida tentar roteá-las.

O algoritmo desenvolvido foi testado no benchmark MCNC e comparado com resultados obtidos pelo VPR 5.0. O critério utilizado para comparação dos resultados foi o tamanho do caminho crítico. Nos testes, dos 20 circuitos, o algoritmo foi capaz de melhorar 18 instâncias, sendo três delas não tão significativas, nos melhores casos, ganhos de até 25,7% e empatar nas outras duas instâncias. Na média, conseguiu melhorar em 6,20% os resultados obtidos pelo VPR 5.0.

Como trabalhos futuros sugerem-se: a incorporação de procedimentos mais eficientes para reduzir a dependência do fator α para geração da solução inicial; implementação de outros mecanismos de perturbação; o desenvolvimento de estratégias paralelas para o algoritmo proposto com o objetivo de reduzir o tempo de execução.

Referências

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

Alpert, C., Chan, T., et al, D. Huang. "Quadratic Placement Revisited". In *Design Automation Conference*, páginas 752-757, 1997.

Altera Corporation. Stratix IV device handbook version SIV5V1-1.1, July 2008. http://www.altera.com/literature/hb/ stratix-iv/stratix4_handbook.pdf.

Betz, V. The "FPGA Place and Route Challenge". http://www.eecg.toronto.edu/~vaughn/challenge/challenge.html. ,1997

Betz, V., e Rose, J. "VPR: A new packing, placement and routing tool for FPGA research." In Proceedings of the 7th International Workshop on Field Programmable Logic, 1997.

Cabral, L. A. F. "Paralelizando a Fase de Roteamento de Circuitos Baseados em FPGAs". Tese – Universidade Federal do Rio de Janeiro, 2001.

Chen, C.P., Chu, C. C. N., e Wong, D. F. "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation," in Proc. Int. Conf. Computer-Aided Design, páginas 614–621, 1997.

Ebeling, C., McMurchie, l., Hauck, S. A., e Burns S. "Placement and Routing Tools for the Triptych FPGA," IEEE Trans. on VLSI, Dec. Páginas 473 -482, 1995.

Feo, T.A. e Resende, M.G.C. "Greedy Randomized Adaptive Search Procedures." Journal of Global Optimization, 6, páginas 109-133, 1995.

Huang D., Kahng, A. "Partitioning-Based Standard-Cell Global Placement with an exact Objective". In *ACM Symp on Physical Design*, páginas 18-25, 1997.

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

Lemieux, G. et al. "Directional and single-driver wires in FPGA interconnect." In IEEE International Conference on Field-Programmable Technology, páginas 41–48, December 2004.

Lourenço, H. R., Martin, O. C., Stutzle, T. "Iterated Local Search. In: Fred Glover e Gary A. Kochenberger (eds.)", Handbook of Metaheuristics, páginas 321-353. Kluwer Academic Publishers, Norwell, MA, 2002.

Luu, J., Kuon, L., Jamieson, P., Campbell, T., Ye, A., Fang, W. M. e Rose, J. "VPR 5.0: FPGA CAD and Architecture Exploration Tools with Single-Driver Routing, Heterogeneity and Process Scaling" (2008) Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays, California, USA, páginas 133-142, 2009.

Nag, S., Rutenbar, R. "Performance-Driven Simultaneous Place and Route for Row-Based FPGAs". In *ICCAD*, páginas 332-338, 1995.

Resende, M. G. C. e Ribeiro, C. C. "Greedy randomized adaptive search procedures." In: Fred Glover e Gary A. Kochenberger (eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Norwell, MA, páginas 219-250, 2005.

So, k. "Solving Hard Instances of FPGA Routing with a Congestion-Optimal Restrained-Norm Path Search Space" International Symposium on Physical Design, Austin, Texas, páginas 151-158, 2007.

Stützle, T. G. "Local Search Algorithms for Combinatorial Problems." Tese (doutorado) Technische Universit¨at Darmstadt, Alemanha, 1998.

Swartz, W., Sechen, C. "Timing Driven Placement for Large Standard Cell Circuits". In *Design Automation Conference*, páginas 211-215, 1995.

Torok, D. L. "Projeto Visando a Prototipação do Protocolo de Acesso ao Meio em Redes ethernet." Porto Alegre: PUC, 2001.

Xilinx. Virtex-5 user guide, March 2008. UG190 (v4.0) http://www.xilinx.com/support/documentation/ user_guides/ug190.pdf.