UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ALBERTO DOS SANTOS CARVALHO JUNIOR

Otimização de Interconexões através de AIGs

Trabalho de Diplomação.

Prof. Dr. André Inácio Reis
Orientador

Porto Alegre, novembro de 2010.

AGRADECIMENTOS

Agradeço ao professor André Inácio Reis, meu orientador, pela paciência, compreensão e auxílio durante este trabalho, a minha querida mãe pelo carinho e apoio, ao professor Valter Roesler e a Andrea Krob pela confiança que sempre demonstraram no meu trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AIG	AND-Inverter Graph
ALU	Arithmetic Logic Unit
ASIC	Application-specific Integrated Circuit
BDD	Binary Diagram Decision
BLIF	Berkeley Logic Interchange Format
CAD	Computer Aided Design
CI	Circuito Integrado
FPGA	Field-Progammable Gate Array
FRAIGs	Functionally Reduced AIGs
NMF	Non-Monotone Factor
IC	Integrated Circuit
ISPD	International Symposium on Physical Design
RAM	Random-Access Memory
VLSI	Very Large Scale Integrated

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A tecnologia VLSI tem experimentado uma constante redução na dimensão de seus dispositivos (i.e. o tamanho mínimo do transistor), permitindo uma densidade que hoje ultrapassa a casa de centenas de milhões de transistores por chip. Essa contínua redução da dimensão dos dispositivos VLSI tem forte impacto sobre a tecnologia de várias formas. Em primeiro lugar, a densidade dos CIs cresce quadraticamente com a taxa de diminuição do tamanho dos transistores. Em segundo lugar, os dispositivos operam a uma velocidade maior, porém o atraso das interconexões permanece o mesmo, efeito de suas capacitâncias que não diminuem (THEIS, 2000). Esse atraso tem se tornado cada vez mais significativo.

Este trabalho propõe um novo fluxo de projeto de circuitos VLSI, orientado a otimização de interconexões. Utilizando uma representação em forma de AIG o circuito é posicionado através de diferentes posicionadores estado da arte e, uma otimização é aplicada duplicando os nodos da AIG nos caminhos críticos, então a AIG é novamente posicionada. O resultado é analisado quanto a não monotonicidade dos caminhos críticos otimizados.

Palavras-Chave: VLSI, Síntese Lógica, Posicionamento, Otimização do Atraso de Interconexões.

Interconnections Optimization through AIGs

ABSTRACT

VLSI technology development has provided a continuous reduction of the feature size of VLSI devices (i.e. the minimum transistor size), allowing a density that exceeds today the hundreds of millions of transistors per chip. This reduction in the size of VLSI devices has a strong impact on the technology in several ways. First, the density of ICs grows quadratically with the rate of decrease in the size of transistors. Second, the devices operate at higher speed, but the delay of interconnections remains the same, as a consequence of the fact that wire resistances and capacitances are not reduced in the same rate (THEIS, 2000). The wire delay has become more and more significant.

This work proposes a new design flow of circuits VLSI oriented to interconnection optimization. By using an AIG representation, the circuit is positioned through different state of the art placement tools; an optimization is applied by duplicating the nodes of AIG in critical paths, and then the AIG is positioned again. The result is analyzed of according the factor of non monotonicity of the critical paths being optimized.

Keywords: VLSI, Logical Synthesis, Placement, Interconnect Delay Optimization.

1 INTRODUÇÃO

1.1 Contexto

O desenvolvimento das tecnologias de fabricação de CIs tem escalado para transistores de tamanhos cada vez menores, permitindo uma densidade que hoje ultrapassa a casa de centenas de milhões de transistores por chip. Essa contínua redução da dimensão dos dispositivos VLSI (*Very Large Scale Integrated*) tem forte impacto sobre a tecnologia de várias formas. Primeiro, a densidade cresce quadraticamente com a taxa de diminuição do tamanho dos transistores. Segundo, os dispositivos operam a uma velocidade maior, porém o atraso das interconexões permanece o mesmo, efeito das capacitâncias e resistências de fio que não diminuem na mesma razão do tamanho mínimo dos transistores (THEIS 2000). Assim o atraso de interconexões se torna muito mais significativo.

De acordo com uma regra simples de escalonamento descrita em (BAKOGLU 1990), quando os dispositivos e interconexões são reduzidos em todas as suas três dimensões por um fator $\boldsymbol{S}$, o atraso intrínseco de *gate* é reduzido por um fator $\boldsymbol{S}$, o atraso das interconexões locais (conexões entre portas adjacentes) permanece o mesmo, mas o atraso de interconexões globais aumenta por um fator $\boldsymbol{S^2}$. Como resultado, o atraso das interconexões tem se tornado um fator dominante na determinação do desempenho do sistema. Em muitos dos sistemas atuais, cerca de 50% a 70% do ciclo de *clock* é consumido por atrasos de interconexões.

Devido a esse constante aumento de sua complexidade e da proliferação de seu uso, o projeto de circuitos VLSI tem uma crescente necessidade de ferramentas de CAD (*Computer Aided Design*) capazes de automatizar seu desenvolvimento, simplificando e acelerando o processo. Além disso, essas ferramentas devem ser capazes de auxiliar na otimização dos diferentes requisitos do circuito, tais como:

- Área total ocupada pelo circuito;
- Atraso de resposta ou *timing*: tempo de propagação de uma alteração na entrada do circuito até seu resultado na saída;
- Potência (energia) consumida pelo circuito;
- Testabilidade do circuito;
- Tempo de projeto, ou seja, o tempo destinado ao desenvolvimento do projeto.

A combinação de todos esses requisitos dentro de uma única função de custo é algo praticamente impossível de ser otimizado devido a sua complexidade. Para ajudar no tratamento de tal complexidade dois conceitos são utilizados: Hierarquia, descrevendo a

estrutura do projeto em diferentes níveis, e abstração, escondendo detalhes em cada nível da hierarquia. O uso de abstração permite-nos trabalhar sobre um determinado número de blocos em cada nível da hierarquia, e cada bloco composto por um determinado número de sub-blocos. A decomposição continua até que os blocos mais básicos da hierarquia sejam alcançados.

Porém uma hierarquia simples não seria suficiente para descrever o processo de desenvolvimento de um projeto VLSI. Para isso, existe um consenso geral de que há três domínios de projeto, cada um com sua hierarquia, são eles:

- Domínio comportamental: uma descrição das funcionalidades do circuito, onde parte, ou todo, é visto como uma caixa preta e a relação entre entradas e saídas é dada sem uma referência de como essa relação é implementada. Um sistema é descrito por algoritmos, descritos por elementos de memória, descritos através de uma lógica, que é descrita por relações de transferências;
- Domínio estrutural, uma descrição do circuito através de subcircuitos. Um processador é composto por ULAs, memória, etc., que são formadas portas lógicas, *flip-flops*, etc., que por sua vez são formadas por transistores;
- Domínio físico, uma descrição de como os sub-blocos do domínio estrutural estão dispostos fisicamente no chip.

Os três domínios são representados através do diagrama em Y de Gajski, apresentado na Figura 1.1, abaixo.

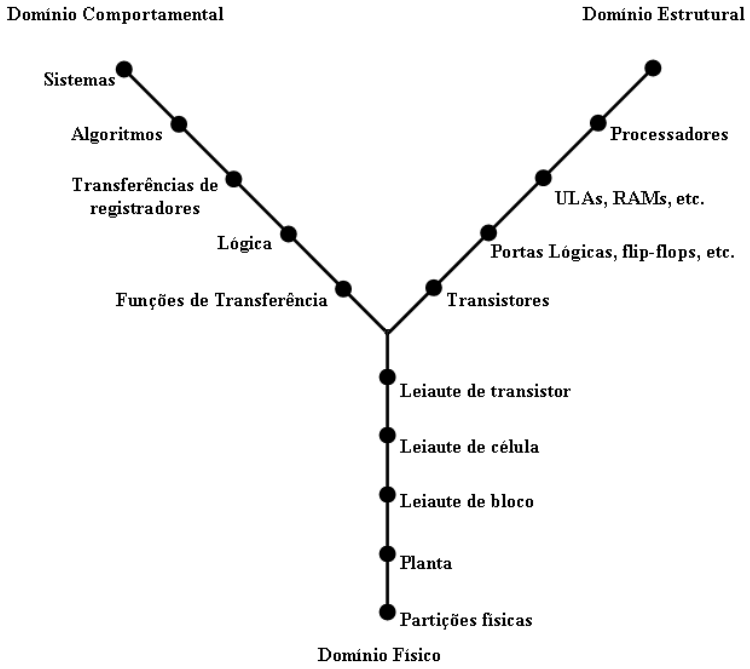

Figura 1.1: Diagrama em Y de Gajski para os três domínios de projeto.

Diferentes metodologias para o projeto de CIs foram desenvolvidas a partir do diagrama em Y de Gajski, genericamente classificadas como *full custom* ou *semi-custom*, sendo o termo *custom* uma referência ao grau de flexibilidade do layout.

A metodologia *full custom* tem como objetivo a otimização de parâmetros que nem sempre pode ser alcançada simultaneamente em abordagens mais automatizadas, como por exemplo, área, tempo de resposta e consumo de energia. São utilizadas para o

desenvolvimento de circuitos de propósito geral (microprocessadores, memórias), quando o número de chips justifica o tempo necessário para um layout altamente otimizado.

A metodologia *semi-custom* tem como objetivo acelerar o tempo de design do projeto, sendo mais utilizada para desenvolvimento de circuitos de aplicação específica (ASICs). *Gate array*, *standard cells* e *macro cell* são os três mais populares estilos de design *semi-custom*. O *gate array* é o mais restrito, todos os blocos tem o mesmo tamanho e somente podem ser associados a uma única *grid* do *layout*. *Standard cells* tem uma altura fixa, mas sua largura pode variar dependendo da funcionalidade do módulo. *Macro cells* podem ter diferentes formas e dimensões.

Na metodologia *standard cells* o projeto é um pouco mais simples do que na metodologia *full custom*, pois considera blocos de células com leiautes retangulares de mesma altura já definidos. Inicialmente, na síntese lógica, o circuito é dividido em vários blocos menores, cada bloco é mapeado para algum subcircuito (célula) pré-definido com funcionalidades equivalentes e características elétricas analisadas e especificadas. O conjunto de blocos predefinidos é denominado de biblioteca de células, normalmente composta de 500 a 1200 células, com uma descrição da dimensão, atraso e características elétricas das células. A etapa seguinte realiza o planejamento da área total ocupada pelo circuito (*floorplanning*), alocando o espaço necessário para os diferentes blocos que o compõem, e o planejamento das linhas de alimentação. Após isso é realizado o posicionamento das células em seus respectivos blocos (*placement*), o planejamento da árvore de *clock* do circuito, e o roteamento das interconexões que ligam as células entre e inter blocos. O circuito é verificado então quanto as restrições de fabricação (dependentes da tecnologia) e quanto aos requisitos especificados de atraso, potência e área. O fluxo da metodologia *standard cells* é apresentado na Figura 1.2 abaixo.

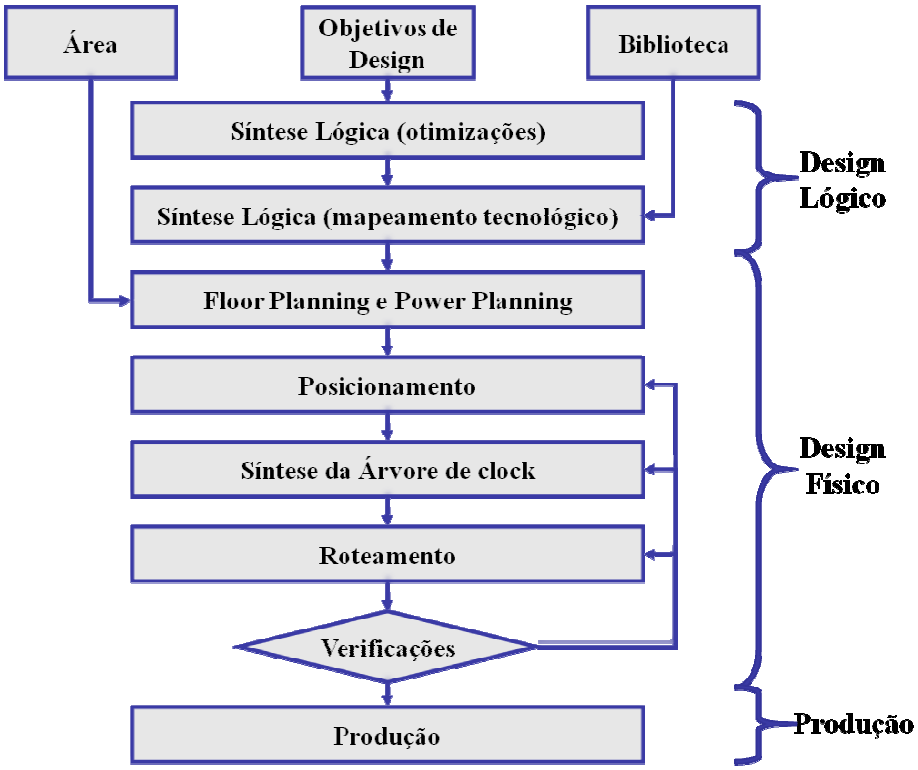

Figura 1.2: Fluxo de projeto para circuitos VLSI.

1.2 Motivação

Como citado anteriormente, em muitos sistemas atuais cerca de 50% a 70% do ciclo de *clock* é consumido por atrasos de interconexões. Um dos pontos críticos para redução dos atrasos das interconexões no circuito é o posicionamento das células, um bom posicionamento deve ter um roteamento possível além de atender aos requisitos de atraso e potência. Para as tecnologias mais recentes, o problema do posicionamento tornou-se ainda mais complexo, pois além do grande aumento do número de elementos a serem posicionados o atraso das interconexões tornou-se da mesma ordem de grandeza do atraso das células, fazendo com que a análise desse atraso faça parte do processo de posicionamento. Nessa situação os algoritmos de posicionamento devem levar em consideração múltiplas questões, como por exemplo, minimização das interconexões, congestionamento e potência. Grande parte dos posicionadores atuais considera como métrica para otimização das interconexões o comprimento total das mesmas (no caso de posicionadores que utilizam uma abordagem analítica), ou o número de interconexões que atravessam secções (no caso de posicionadores que utilizam a abordagem de particionamento *min-cut*). Em alguns casos o posicionamento detalhado ou um pós posicionamento direcionado a redução de atrasos é utilizado como forma de obter um melhor resultado.

Outra questão que se impõem em relação aos posicionadores atuais é o tempo utilizado para execução do posicionamento. Para muitos projetos mais complexos (com mais de 10^5 transistores), o tempo utilizado no processamento do posicionamento pode chegar a casa da dezena de horas em alguns posicionadores comerciais, para um resultado que nem sempre é satisfatório.

1.3 Objetivos

O objetivo desse trabalho é propor um novo fluxo para o projeto circuitos VLSI orientado a otimização de interconexões utilizando uma representação na forma de AIGs (AND-Inverter Graphs) e posicionadores estado da arte para o posicionamento dos nodos da AIG. Para tanto o circuito é descrito na forma de uma AIG, posicionado através de diferentes posicionadores, esse resultado é então analisado quanto não monotonicidade de seus caminhos críticos, e técnicas de otimização de atraso são utilizadas sobre o circuito posicionado, mais precisamente a duplicação de nodos nos caminhos críticos é a metodologia que será empregada. O resultado da otimização é então novamente posicionado, analisado e comparado de acordo com o fator de não monotonicidade dos caminhos críticos em relação ao posicionamento inicial. Como resultado do processo, obtemos um circuito, representado por uma AIG, posicionado, que pode ser utilizado na etapa de posicionamento, ou para mapeamento das células diretamente sobre o AIG ou como um guia para os requisitos de atraso e área no posicionador.

O uso do AIG tem por objetivo facilitar a manipulação e otimização do circuito, visto sua estrutura simplificada, e mesmo não sendo uma representação canônica do circuito pode ser reduzida funcionalmente (MISHCHENKO, A., et. al. 2004).

O novo fluxo proposto nesse projeto é apresentado na Figura 1.3 a seguir.

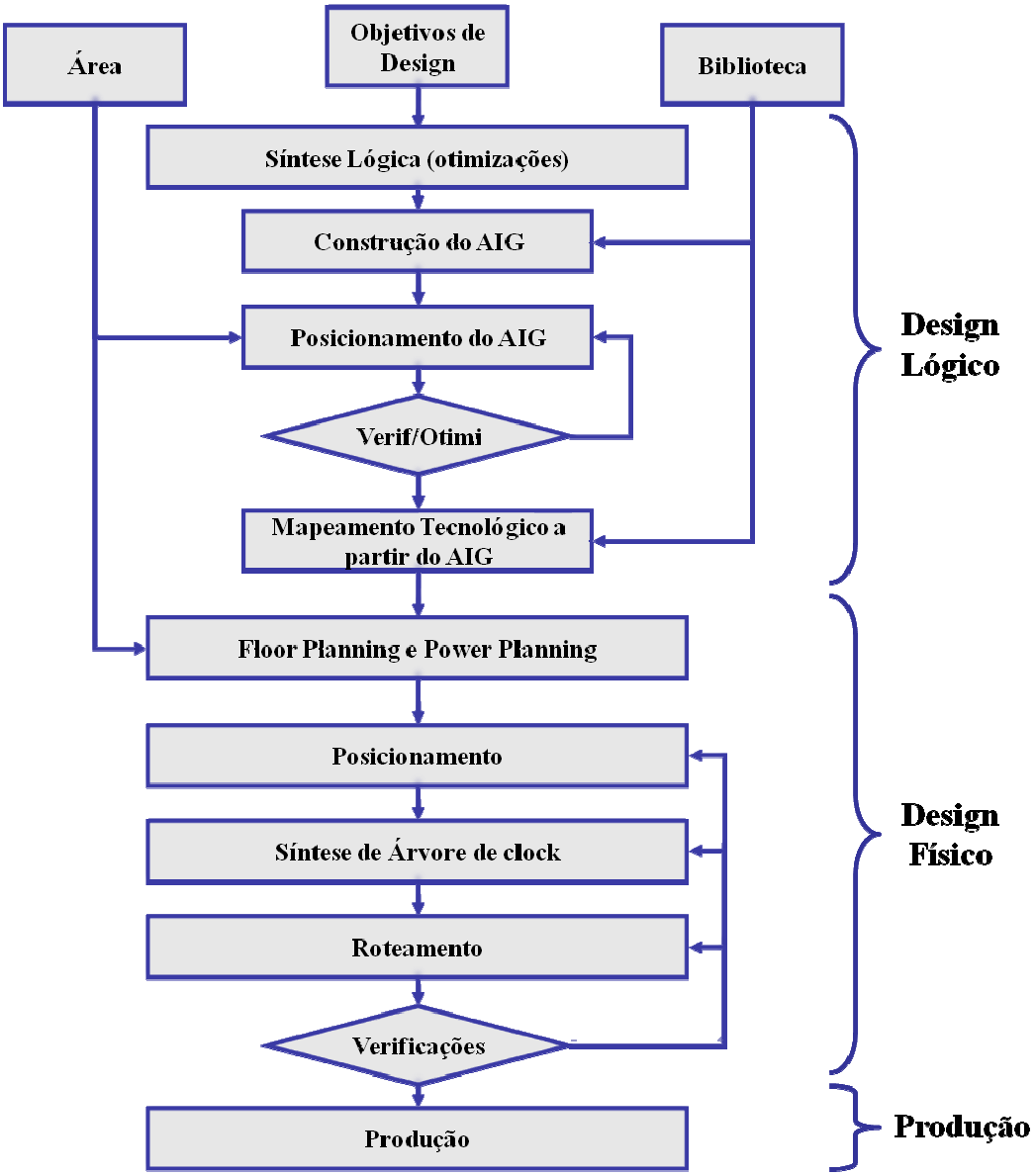

Figura 1.3: Fluxo proposto de projeto para circuitos VLSI.

1.4 Organização deste Trabalho

Este trabalho está organizado da seguinte forma: Na Seção dois são apresentadas conceitualmente as diferentes técnicas de particionamento e posicionamento presentes nos posicionadores que são utilizados, os posicionadores, técnicas de otimização de atraso, além de uma descrição mais detalhada da AIG; a seção três apresenta a descrição da ferramenta e fluxo de execução detalhado; a seção quatro descreve os experimentos realizados e apresenta seus resultados; na seção cinco são apresentados os trabalhos futuros; e na seção 6 as conclusões deste trabalho.

2 TRABALHOS RELACIONADOS

2.1 Metodologias de Particionamento

Um Projeto VLSI pode conter centenas de milhões de transistores, o que torna a manipulação do circuito de forma única impossível, devido aos limites computacionais bem como de memória, o particionamento é absolutamente necessário. Um bom particionamento pode melhorar o desempenho do circuito além de reduzir os custos de leiaute.

Para o estudo do problema de particionamento utilizamos a notação de grafos, notação que será referenciada nas próximas seções. Sendo o grafo $G=G(V, E)$, em que G consiste de um conjunto V de vértices $V=\{v_1, v_2, v_3, ...\}$ e um conjunto E de arestas $E=\{e_1, e_2, e_3, ...\}$, cada aresta corresponde a um par de distintos vértices. Considerando agora um hipergrafo $H=H(N, L)$, em que H consiste de um conjunto N de vértices e um conjunto L de hiper-arestas, em que cada hiper-aresta corresponde a subconjunto N_i de distintos vértices com $|N_i| \geq 2$. Também é associada uma função peso $\omega: V \rightarrow \mathcal{N}$ a cada vértice ($\mathcal{N}$ é um conjunto de inteiros). Assim o circuito pode ser representado por um grafo ou por um hipergrafo, onde os vértices são os elementos do circuito e as arestas (ou hiper-arestas) correspondem as interconexões, além disso, o peso do vértice pode indicar o tamanho do elemento no circuito.

O particionamento *multi-way* de um grafo (ou hipergrafo) H é um conjunto não vazio de subconjuntos disjuntos $N=\{N_1, ..., N_r\}$, de forma que $\cup_{i=1}^{r} N_i = N$ e $N_i \cap N_i = 0$ para $i \neq j$. Um particionamento é aceitável se $b(i) \leq \omega(N_i) \leq B(i)$, onde $\omega(N_i)$ é a soma dos pesos dos vértices em N_i, $B(i)$ é o tamanho máximo da parte i e $b(i)$ é o tamanho mínimo, para $i=1,...,r$. Um caso especial ocorre com $r=2$, quando temos o biparticionamento.

2.1.1 Kernighan-Lin e Fiduccia-Mattheyses

Dado um grafo $G=G(V, E)$ sem peso, particionamos G dentro de dois grupos, V_1 e V_2, sendo que $|V_1| < \alpha|V|$ e $|V_2| < \alpha|V|$, onde α é o fator de balanço (tipicamente $\alpha=1/2$ nesse caso) e $|V|$ denota o número de vértices em V. Após identificar um par de vértices (v_a, v_b) com $v_a \in V_1$ e $v_b \in V_2$, cuja troca resulta em um decremento do custo da função de corte, ou um pequeno aumento caso nenhum decremento seja possível (na esperança de que o custo decremente nos próximos passos), os vértices v_a e v_b são bloqueados.

Há um valor máximo para a soma parcial $\sum_{i=1}^{k} g_i = \boldsymbol{Gain_k}$, onde $\boldsymbol{g_i}$ é o ganho do i-ésima troca. Se $\boldsymbol{Gain_k} > 0$, a redução do custo pode ser obtida movendo-se $\{\boldsymbol{v_{a1}}, ..., \boldsymbol{v_{ak}}\}$ para $\boldsymbol{V_2}$ e $\{ \boldsymbol{v_{b1}}, ..., \boldsymbol{v_{bk}}\}$ para $\boldsymbol{V_1}$. Após isso, o resultado é tratado como um particionamento inicial e o procedimento repete-se até que não exista um $\boldsymbol{k}$ tal que $\boldsymbol{Gain_k} > 0$.

Fidducia e Mattheyses aperfeiçoaram o algoritmo de KL, reduzindo sua complexidade, através da introdução dos seguintes requisitos:

- Somente um vértice pode ser deslocado por vez;
- Cada vértice tem um peso associado;
- Uma estrutura especial de dados é utilizada para selecionar os vértices a serem deslocados.

A estrutura especial de dados utilizada consiste de um *array* de ponteiros indexado por um conjunto $[-\boldsymbol{d_{max}}\cdot\boldsymbol{\omega_{max}}, \boldsymbol{d_{max}}\cdot\boldsymbol{\omega_{max}}]$ onde $\boldsymbol{d_{max}}$ corresponde ao máximo grau de um vértice no hipergrafo e $\boldsymbol{\omega_{max}}$ corresponde ao máximo custo da hiperaresta, assim os índices (positivo ou negativo) correspondem aos possíveis ganhos. Para duas partições $\boldsymbol{A}$ e $\boldsymbol{B}$, por exemplo, temos dois *arrays*, ***lista A*** e ***lista B***, e o movimento de um vértice altera custo no mínimo por $\boldsymbol{d_{max}}\cdot\boldsymbol{\omega_{max}}$.

2.1.2 Particionamento Multinível

Desenvolvidos para circuitos de larga escala, essa abordagem caracteriza-se por uma sucessiva construção de pequenos hipergrafos (*coarsened*). Então é realizada um biparticionamento do menor hipergrafo, essa bipartição é projetada para os próximos níveis e a cada nível um algoritmo iterativo é utilizado para refinar o particionamento.

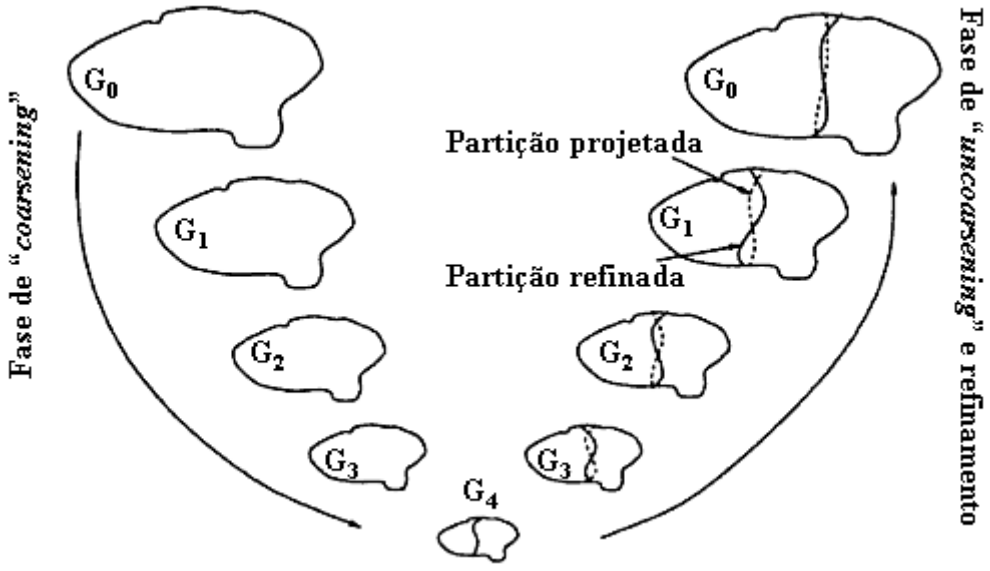

Figura 2.1: Fases do particionamento multinível em bisecções.

2.2 Metodologias de Posicionamento

A etapa de posicionamento é de fundamental importância na síntese física do circuito, pois o posicionamento das células influi diretamente no roteamento, densidade, comprimento total das interconexões, e congestionamento do circuito. Para as tecnologias mais recentes, o problema do posicionamento tornou-se ainda mais complexo, pois além do grande aumento do número de elementos a serem posicionados o atraso das interconexões tornou-se da mesma ordem de grandeza do atraso das células, fazendo com que a análise desse atraso faça parte do processo de posicionamento. Nessa situação os algoritmos de posicionamento devem levar em consideração múltiplas questões, como por exemplo, minimização das interconexões, além de outros problemas que fogem ao escopo deste trabalho, como é o caso do congestionamento e potência.

A classificação mais comum na literatura para algoritmos de posicionamento divide-os em duas classes: construtivos e iterativos. Os algoritmos construtivos geram um posicionamento completo a partir da descrição do *netlist* do circuito. Os algoritmos iterativos iniciam a partir de algum posicionamento já determinado e realizam alterações nas posições das células com o objetivo de alcançar um melhor resultado. Entre os algoritmos construtivos pode-se destacar o posicionamento aleatório (para posicionamento inicial), o posicionamento direcionado a forças e o baseado em particionamento (quadratura, *min-cut,* etc.). Entre os algoritmos iterativos destacam-se as meta-heurísticas, como o *Simulated Annealing*, Algoritmos Genéticos e Busca Tabu.

Na prática, nenhuma das duas abordagens alcança um resultado satisfatório em tempo de execução aceitável quando utilizada isoladamente. Pesquisadores têm tentado utilizar ambos os métodos (GOTO, p. 208-214, 1978), e as ferramentas comercias de maior êxito seguem esse caminho, iniciam com uma solução construtiva que é aperfeiçoada com o uso de um algoritmo iterativo.

Outra classificação adotada para identificar os algoritmos de posicionamento, divide-os em quatro tipos: posicionamento baseado em particionamento, posicionamento em quadratura, *Simulated Annealing*, posicionamento direcionado a forças. Todos os quatro tipos são apresentados a seguir.

2.2.1 Posicionamento baseado em Particionamento

Também denominados de posicionamento *min-cut*, tem como idéia central reduzir o número de interconexões sendo cortadas no particionamento do circuito, em uma sequência de sucessivas divisões (em bisecções ou quadrisecções).

2.2.1.1 Algoritmo de Breuer e Propagação de Terminal

O primeiro posicionamento *min-cut* foi proposto em (BREUER, p. 284-290, 1977), onde o objetivo era minimizar as *nets* cortadas por ambas as linhas, horizontal e vertical que particionavam a secção, o autor demonstra que a minimização dessas *nets* equivale a minimizar o comprimento das interconexões que delimitam a secção.

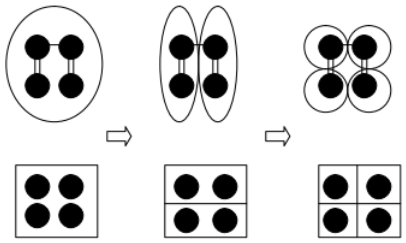

Figura 2.2: Posicionamento min-cut com bisecção (BREUER, p. 284-290, 1977).

O algoritmo proposto por Breuer foi aperfeiçoado por Dunlop e Kernighan (DUNLOP, p. 92-98 1985) utilizando um método de propagação de terminal. O método considera no particionamento, não somente as *nets* internas, mas também as externas a secção, inserindo um terminal "*dummy*" na posição onde *net* externa conecta-se a outro subcircuito. O terminal "*dummy*" é considerado no processamento das partições, resultando em um melhor particionamento em termos de comprimento de interconexões.

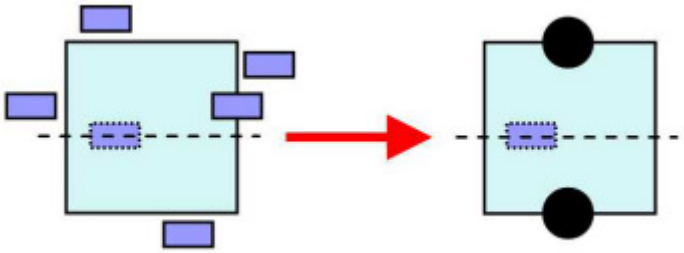

Figura 2.3: Propagação de terminal: A figura apresenta cinco terminais fixos sendo propagados, quatro acima e um abaixo da linha que secciona o bloco (DUNLOP, p. 92-98 1985).

2.2.1.2 Quadrisecção

A desvantagem dos métodos de particionamento é que, um bom resultado obtido no primeiro particionamento pode ser ruim para os particionamentos subseqüentes. Além disso, o posicionamento é essencialmente um problema de duas dimensões, enquanto que a solução recursiva de biparticionamento adota um método unidimensional. Suaris e Kedem (SUARIS, p. 4747-477, 1987) utilizam um particionamento em quadrisecções, realizando um procedimento em duas dimensões.

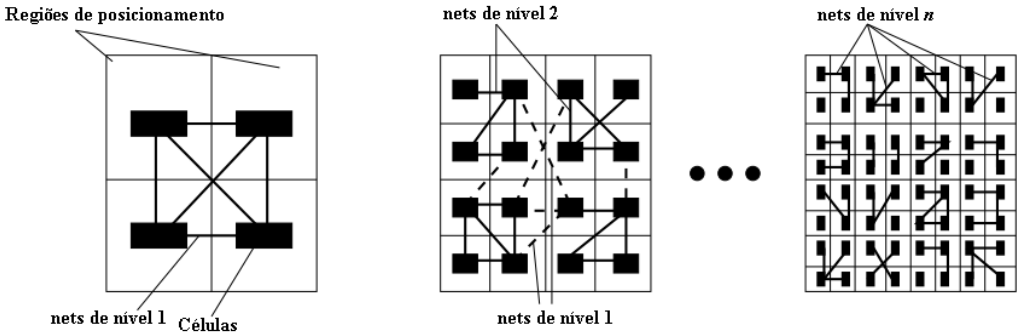

Figura 2.4: Particionamento *min-cut* com quadrisecções (SUARIS, p. 4747-477, 1987).

2.2.2 Posicionamento Quadrático

Para o posicionamento do circuito, a *netlist* geralmente é modelada por um hipergrafo, com cada nodo representando uma célula e cada aresta representando uma *net*. O HPWL é utilizado como uma estimativa do comprimento necessário da interconexão para roteamento de duas células. Porém a equação do HPWL é difícil de ser otimizada matematicamente. No posicionamento em quadratura os quadrados do comprimento e largura do perímetro formado pelas células (denominado de *quadratic wirelength*) são minimizados.

Sendo uma *net* $\boldsymbol{e_{i,j}}$ que conecta duas células $\boldsymbol{i}$ e $\boldsymbol{j}$, de localização $(\boldsymbol{x_i}, \boldsymbol{y_i})$ e $(\boldsymbol{x_j}, \boldsymbol{y_j})$ respectivamente, o *linear wirelength* é dado por $L_e = |x_i - x_j| + |y_i - y_j|$, o *quadratic wirelength* por $Q_e = (x_i - x_j)^2 + (y_i - y_j)^2$, o *linear wirelength* total do leiaute é $WL = \sum_e L_e$ e o *quadratic wirelength* total é $WL^2 = \sum_e Q_e$.

Quando uma *net* é compartilhada por mais de duas células, ela deve ser transformada em múltiplas *nets* de duas células para que o algoritmo possa processa-lá. Tradicionalmente um modelo "*clique*" é utilizado e uma net de k conexões é transformada para uma em C_k^2 *nets*. A Figura 2.4, abaixo, ilustra um *net* de qurato conexões e a transformação através do modelo "*clique*".

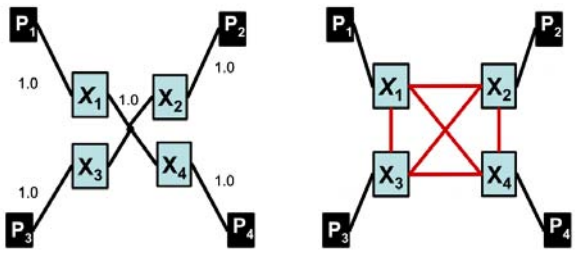

Figura 2.5: Transformação "*clique*" para *net* de quatro conexões(NAM, et. al., 2007).

Para a otimização da função *quadratic wirelength*, frequentemente é utilizada uma representação em forma de matriz: Utilizando um grafo $\boldsymbol{G=G(V, E)}$, como descrito na seção 2.1, com um comprimento não negativo $\boldsymbol{\omega(e)}$ associado a cada aresta. Todas as *nets* podem ser representadas por uma matriz de adjacência $\boldsymbol{A=(a_{ij})}$ com uma entrada $\boldsymbol{a_{ij}} = \boldsymbol{\omega}$ $(\boldsymbol{v_i}, \boldsymbol{v_i})$ se $(\boldsymbol{v_i}, \boldsymbol{v_j}) \in \boldsymbol{E}$ e $\boldsymbol{a_{ij}} = 0$ caso contrário. A posição de todos os vértices pode ser representada por um vetor *n*-dimensional $\boldsymbol{X=(x_i)}$ e $\boldsymbol{Y=(y_i)}$, sendo $(\boldsymbol{x_i}, \boldsymbol{y_i})$ as coordenadas do vértice $\boldsymbol{v_i}$.

Para o *quadratic wirelength* as dimensões $\boldsymbol{x}$ e $\boldsymbol{y}$ são independentes, assim é possível otimizá-las separadamente em cada direção. No caso da dimensão $\boldsymbol{x}$, o *quadratic wirelength* total é dado por:

$$\Phi_Q(X) = \frac{1}{2} X^T Q X + d^T X = \sum_{i,j=1}^{n} a_{ij}(x_i - x_j)^2 + d^T X \quad (2.1)$$

Assumindo que existem $\boldsymbol{n}$ células na *netlist*, e que $\boldsymbol{Q}$ denota a matriz Hessiana do sistema, que é essencialmente uma matriz de conectividade $\boldsymbol{nxn}$ da *netlist*. $\boldsymbol{Q}$ é simétrica e positiva. O termo opcional $\boldsymbol{d^T X}$ representa a conexão de células para I/O *pads* fixos, e o vetor $\boldsymbol{d}$ pode também capturar *offsets* de pinos. O objetivo da função é minimizado solucionando o sistema linear dado por:

$$QX + d = 0 \quad (2.2)$$

2.2.3 Simulated Annealing

Simulated Annealing é uma técnica para otimização de problemas em geral, especialmente útil quando o problema não é completamente conhecido. O algoritmo está baseado em uma analogia com a formação cristalina de alguns materiais. Quando o material é aquecido as moléculas próximas movimentan-se aleatoriamente trocando de posição, com a redução da temperatura os movimentos tendem a diminuir até cessar por completo alcançado uma estrutura cristalina.

O algoritmo de *simulated anneleaning* para posicionamento utiliza modificações (pertubações) aleatórias no estado atual de posicionamento, até que se atinja um estado aceitável, conceitualmente define-se um grafo de configuração onde cada vértice corresponde a uma solução possível, e uma aresta direcionada (v_i, v_j) representa um possível movimento da solução v_i para v_j. O processo de *annealing* movimenta a solução de um vértice a outro seguindo as arestas direcionadas do grafo de configuração. Independentemente da temperatura o algoritmo irá aceitar um movimento (v_i, v_j) se o custo $C(v_j)$ $C(v_i)$. Para evitar mínimos locais, o algoritmo aceita movimentos que aumentem o custo da solução quando a "temperatura é alta".

A melhor solução entre todas as visitadas pelo processo é armazenada, e quando o processo termina espera-se que ter um número suficiente de soluções examinadas para se obter uma solução de baixo custo. Tipicamente o número de soluções possíveis é uma função exponencial do tamanho do problema. Assim, o movimento de uma solução para outra é restrito a um pequeno número do total de soluções possíveis.

2.2.4 Algoritmos Direcionados a Força

A fim de minimizar o *wirelength*, células fortemente conectadas devem ser posicionadas próximas umas das outras, tanto quanto possível, como se houvesse uma força de atração entre as células. Deste modo o número de conexões entre duas células é utilizado para determinar a força de atração entre elas, o que pode ser modelado da seguinte forma:

$$F_{ij} = -c_{ij} d_{ij} \quad (2.3)$$

Onde c_{ij} é soma dos pesos das *nets* entre as duas células e d_{ij} é um vetor direcionado do centro de C_i até o centro de C_j. A magnitude desta força é, portanto, a distância (Euclideana ou de Manhattan) entre o centro das células. Por exemplo, na geometria Manhattan $|d_{ij}| = |x_i - x_j| + |y_i - y_j|$, onde (x_i, y_i) é o centro de C_i e (x_j, y_j) é o centro de C_j.

Algoritmos direcionados a força podem ser utilizados tanto na abordagem construtiva quanto na abordagem iterativa. No caso da abordagem construtiva o algoritmo direcionado a força realizam o posicionamento fazendo com que as células posicionadas mantenham um equilíbrio entre as forças que atuam sobre elas, o que é encontrado quando o vetor de somas das forças atuando em cada célula seja zero. Solução que pode ser obtido através de sistema não linear de equações. No caso da abordagem iterativa o algoritmo direcionado a força pode melhorar um posicionamento existente através da redução da força de atração entre as células.

2.3 Posicionadores Estado da Arte

Nesta seção são apresentados os posicionadores que serão utilizados no projeto. Essa descrição não tem a intenção de detalhar todas as características e forma de implementação dos posicionadores, mas apenas relatar quais e como as técnicas são utilizadas, uma descrição mais precisa pode ser encontrada em suas respectivas referências.

2.3.1 Capo

O Capo (ROY, et. al., 2005) realiza um posicionamento baseado em particionamento *min-cut* em bisecções. Durante o particionamento o Capo utiliza diferentes formas de distribuir as células nas secções, podendo distribuir uniformemente as células ou de forma desigual, tentando ou reduzir o *wirelength* no caso da distribuição uniforme ou reduzir o congestionamento no caso da distribuição não uniforme (NAM, et. al., 2007). Para o posicionamento detalhado o Capo utiliza diferentes técnicas para reduzir o HPLW, as técnicas utilizadas incluem um mecanismo de *sliding* com um otimizador *RowIroning* (baseado em um mecanismo de *branch-and-bound*) e um algoritmo guloso que realiza movimentos nas células de regiões com violação de densidade.

2.3.2 Dragon

O Dragon (TAGHAVI, et. al., 20005) realiza o posicionamento de forma hierárquica utilizando um posicionamento baseado em particionamento, porém diferentemente do Capo o particionamento é em quadrisecções. O procedimento do Dragon é dividido em duas etapas: posicionamento global (GP) e posicionamento detalhado (DP). Na etapa de posicionamento global, o circuito é sucessivamente particionado em quadrisecções, nessa etapa são permitidas sobreposições de células. A etapa seguinte (DP) utiliza esse resultado, corrigindo sobreposições de células e iterativamente aperfeiçoa-o utilizando heuristicas gulosas. Ainda na fase de posicionamento global, durante o particionamento, o Dragon utiliza *simulated annealing* para melhorar o particionamento tentando evitar mínimos locais. A versão mais recente do Dragon, de 2005, realiza uma melhoria no particionamento, redimensionando secções para permitir o posicionamento de blocos. A Figura 2.6 abaixo apresenta um fluxo de execução simplificado do Dragon.

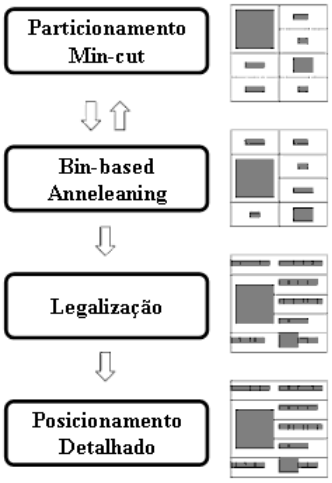

Figura 2.6: Fluxo de posicionamento no Dragon.

2.3.3 NTUPlace3

O NTUPlace3 (CHEN, et.al., 2008) utiliza um particionamento multinível e o posicionamento consiste de três etapas: posicionamento global, que distribui uniformemente e encontra a melhor posição para cada bloco; legalização, que remove sobreposições entre blocos e entre células; e posicionamento detalhado, onde a solução é refinada.

O posicionamento global é baseado em multinível e aplica uma técnica de *coarsening bottom-up* de dois estágios seguida de um *uncoarsening top-down*. A etapa de *coarsening* iterativamente "clusteriza" os blocos baseado em sua conectividade e tamanho, até que o problema alcance um tamanho mínimo. Na etapa de *uncoarsening* os blocos são "desclusterizados" e para refinar o posicionamento dos blocos é aplicado um modelo analítico para posicionamento global com uma função objetivo baseada em *log-sum-exp* como proposto em (NAYLOR, et. al., 2001).

Na etapa de legalização são removidas as sobreposições utilizando um sistema de prioridades baseado no tamanho e posição das células, também foi implementado um mecanismo de *look-ahead* para facilitar o processo de legalização.

A etapa de posicionamento detalhada é dividida em duas etapas, primeiramente são utilizados mecanismo de *swapping* e *matching* para reduzir o *wirelength*, logo após é aplicado um algoritmo de *sliding* para minimizar a densidade e o *overflow* em regiões de maior congestionamento.

2.3.4 mPL6

Como no NUTPlace3 o mPL6 (CHEN, et. al. 2006) realiza o posicionamento em três etapas, posicionamento global, legalização e posicionamento detalhado. O posicionamento global é realizado por um mecanismo multinível linear com um "*clustering*" baseado em uma aperfeiçoada heurística de "melhor escolha", uma redução de duas vezes no número de níveis em relação a versão anterior. A fase de legalização é dividida em duas etapas, com uma macro legalização baseado em um grafo discreto seguido de uma legalização baseada em um *scan* de tempo linear.

O posicionamento detalhado tem o objetivo de reduzir o *wirelength*, para isso é utilizada uma janela abrangendo uma ou mais linhas de células, todas as possíveis configurações de conexões de células são enumeradas, e após uma redução de *wirelength* através de *swapping* de células a janela é reduzida da sua metade. O processo continua até que nenhuma otimização seja possível.

2.4 Técnicas de Otimização de Atraso

Diversas técnicas de otimização de atraso já foram propostas para a etapa de posicionamento ou para refinamento do posicionamento. Alguns trabalhos dedicam parte ou mesmo todo posicionamento detalhado a um algoritmo direcionado a otimização do atraso no circuito de alguma forma, seja diminuindo caminho críticos ou reduzindo o congestionamento. Algumas dessas principais técnicas são apresentadas a seguir.

2.4.1 Dimensionamento de Gate

O dimensionamento de gate consiste basicamente na substituição de uma célula presente em um caminho crítico por uma célula com maior capacidade de corrente. Essa

técnica foi utilizada com sucesso (KANNAN, et. al., 1994), em conjunto com uma técnica de bufferização de interconexões alcançando reduções de 13 a 22% em relação a ferramentas comerciais. Porém essa abordagem causa grande impacto no posicionamento das células, pois a alteração da dimensão das células pode provocar o reposicionamento de grande parte do circuito além de alterar a área ocupada, fato que foi parcialmente tratado em (LI, et. al., 2005), onde a solução proposta impede grandes perturbações no posicionamento.

2.4.2 Bufferização

A bufferização tem por objetivo reduzir a carga de saída em células sobrecarregadas, ou seja, células com um excessivo *fanout*, devido a um grande número de interconexões ou outras células conectadas a sua saída. Para isso insere buffers entre a saída da célula sobrecarregada e as células conectadas a ela.

2.4.3 Realocação

Com o circuito já posicionado e roteado, o atraso das interconexões é estimado considerando a exata topologia do circuito, e as células pertencentes a caminhos críticos são movidas com o objetivo de reduzir o atraso nesses caminhos. Porém essa movimentação não é simples, e pode resultar em novos caminhos críticos, para evitar esse problema, é utilizado um limite para o aumento do custo total das interconexões. Em (AJAMI, et. al., 2001), experimentos demonstraram ganhos entre 9% e 14% para um aumento de 1 a 5% do circuito.

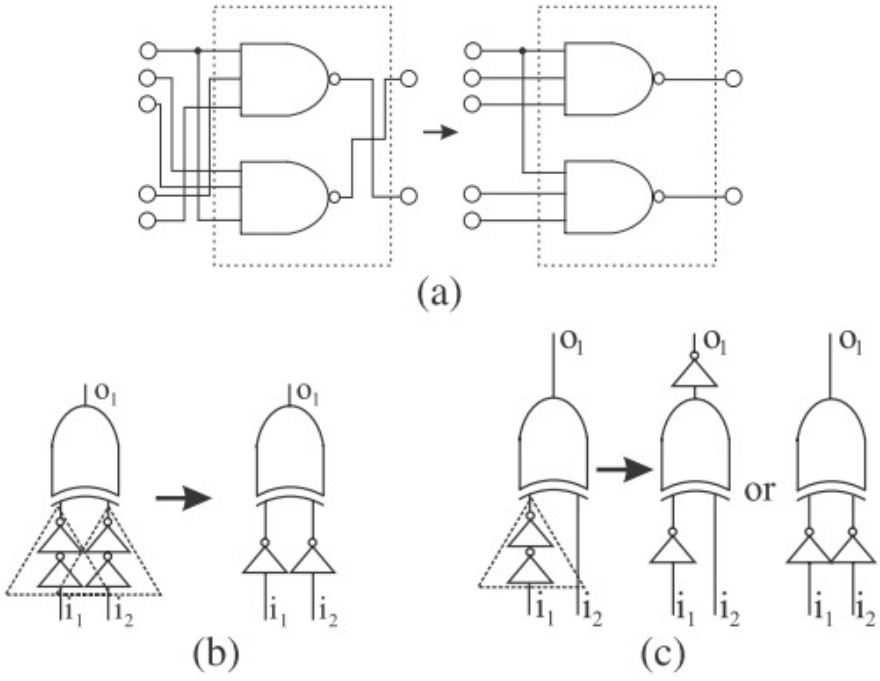

Figura 2.7: Exemplos de Realocação (AJAMI, et. al., 2001).

2.4.4 Transformação de Sinais

Explora as possíveis transformações ou replicações de sinais para obter uma redução de atrasos no circuito. Em (CHANG, et. al., 2007), espaços vazios são utilizados para duplicação de sinais, reduzindo o atraso das interconexões duplicadas. Os resultados

obtidos apresentaram em média uma redução média de 11% do atraso com uma aumento médio de 0,2% do *wirelength*.

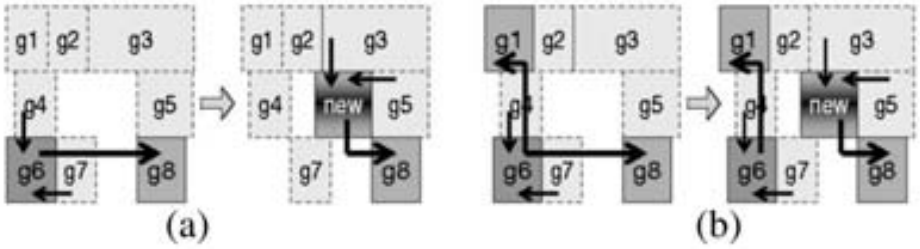

Figura 2.8: Redução de Atrasos com duplicação de sinais (CHANG, et. AL. 2007): em (a) uma nova célula substitui g6, reduzindo a interconexão para g8; em (b) a célula g6 é mantida e fornece sinal para g1 enquanto que a nova célula fornece sinal para g8.

2.4.5 Otimização de caminhos não monótonos

Caminhos não monótonos são caminhos percorridos pela interconexões em zigue-zague entre duas células, sua identificação pode permitir grandes ganhos em termos de *wirelength* e consequentemente redução de atrasos no circuito. Em (PLAZA, et. al., 2008) o fluxo proposto de pós posicionamento primeiro identifica os caminhos criticos não monótonos, e utilizando uma simulação funcional realiza transformações lógicas sobre o circuito, avaliando os novos parâmetros obtidos. Os resultados obtidos nos experimentos realizados apresentaram uma redução média de aproximadamente 11% com um aumento médio de cerca de 2% na área do circuito.

2.5 AIG

O AIG consiste de rede booleana composta de portas lógicas AND2 e inversores. Pode ser utilizada para manipular funções booleanas de larga escala em diversas aplicações como, por exemplo, verificação formal, síntese lógica e mapeamento tecnológico, além de apresentar a vantagem de ser menor e mais simples e mais de construir do que um BDD.

Graficamente, na AIG temos três tipos de vértices: vértices com apenas um terminal de valor “0” ou “1”, vértices sem entrada de arcos representando as entradas do circuito e vértices com duas entradas representando a função lógica AND2. Os arcos são direcionados e podem ter atributos inversores ou não (KUEHLMANN, et. al., 2002). A AIG pode ser criada por fatoração de soma de produtos e conversões das funções AND e OR através das regras de DeMorgan (MISHCHENKO, et. al., 2006).

A dimensão e o tempo necessário para gerar a AIG são proporcionais ao tamanho do circuito representado (KUEHLMANN, et. al., 2002). Além disso, uma AIG pode ser representada sem exceder três vezes o tamanho de um BDD correspondente, sendo que a computação booleana é mais eficaz em AIGs quando comparada a BDDs (MISHCHENKO, et. al., 2004).

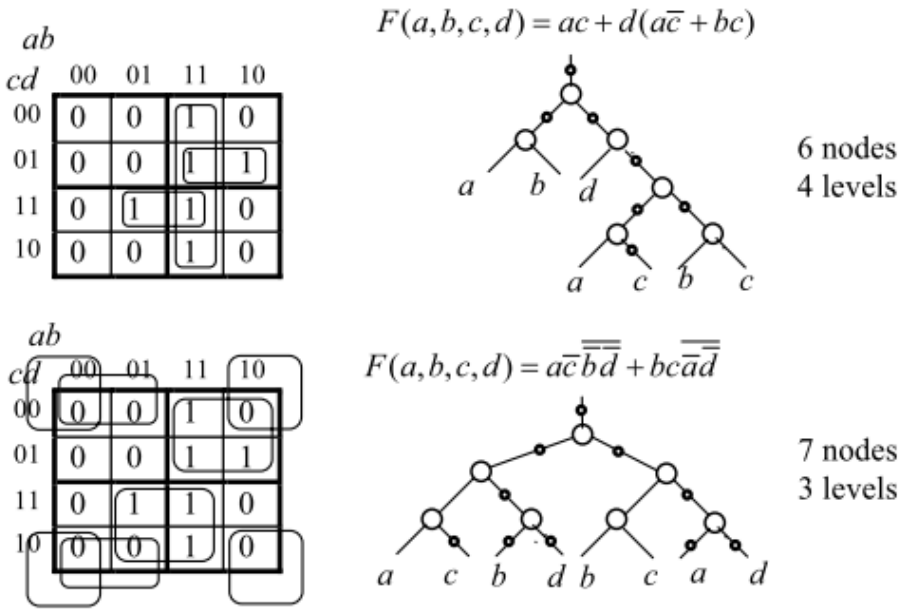

Figura 2.9: Representação de duas AIGs para a mesma função booleana (MISHCHENKO, et. al., 2004).

A AIG apresenta a desvantagem de não ser uma representação canônica do circuito, mas apesar disso pode ser reduzida funcionalmente (FRAIG), desde que respeitada certas condições: $f_{n1}(x) \neq f_{n2}(x)$ e $f_{n1}(x) \neq \overline{f_{n2}(x)}$ para dois nodos quaisquer $\boldsymbol{n_1}$ e $\boldsymbol{n_2}$, como demonstrado em (MISHCHENKO, et. al., 2004).

3 DESCRIÇÃO DO PROJETO

3.1 Objetivos

O objetivo deste trabalho é propor um novo fluxo para o projeto de circuitos VLSI baseados em *standard cells*. Neste novo fluxo uma etapa é adicionada logo após a síntese lógica. Nesta nova etapa o circuito é transformado em uma AIG que, após ser funcionalmente reduzida (FRAIG), é posicionada e otimizada em relação a suas interconexões. Após a otimização a AIG é novamente posicionada, e o resultado final é utilizado para o mapeamento de uma biblioteca em etapa de posicionamento final.

Para o posicionamento da AIG são utilizados quatro posicionadores estado da arte, disponíveis ou em código fonte ou como executáveis para livre uso acadêmico, os posicionadores utilizados são o Capo10, o Dragon2005, o mPL6 e o NTUplace3.

A otimização proposta neste trabalho utiliza a duplicação de nodos em caminhos críticos, tendo como parâmetro o fator de não monotonicidade (NMF) dos caminhos críticos do circuito. O fluxo da nova etapa proposta para o projeto baseado em *standard cells* é ilustrado na Figura 3.1 a seguir. A descrição de cada passo é apresentado na seção seguinte.

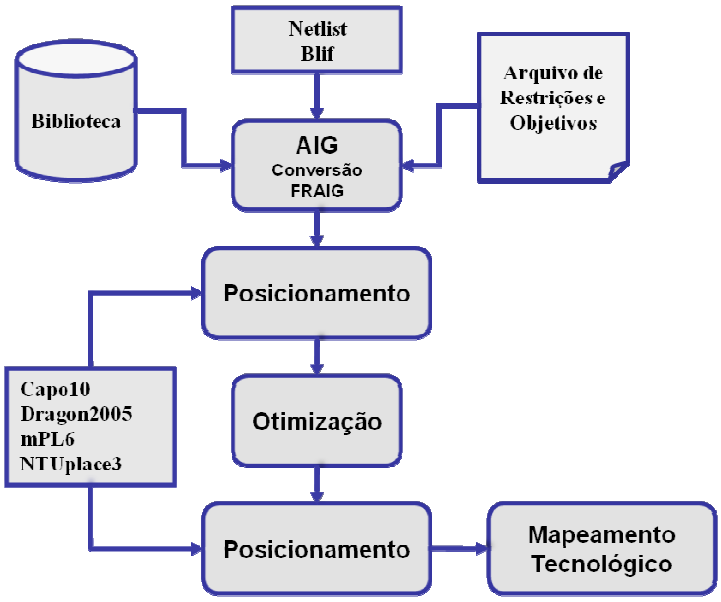

Figura 3.1: Fluxo de execução da ferramenta proposta.

3.2 Descrição da Ferramenta

3.2.1 AIG: Conversão e Redução Funcional

O primeiro passo é a conversão do circuito para uma descrição no formato AIG, nessa etapa a AIG também é funcionalmente reduzida. Para execução deste passo foi utilizada a biblioteca ABC (MISHCHENKO, A., 2005), biblioteca para síntese lógica e verificação. A ABC utiliza como sua estrutura interna de dados a AIG, suporta diversos formatos de entrada (aiger, baf, blif, bench e verilog por exemplo), realiza a redução funcional da AIG, além de executar o balanceamento e reescrita da AIG.

Neste trabalho foram utilizados apenas dois formatos de entrada: verilog, com a *netlist* e a correspondente biblioteca genlib (descrição das portas lógicas na *netlist*); ou o formato blif (*Berkeley Logic Interchange Format*), padrão de entrada e saída na ABC.

Apesar da ABC suportar diversos formatos de entrada e saída, ela não escreve a AIG diretamente nesses formatos. Quando um comando de *write* é executado o circuito, que internamente é representado por uma AIG, é convertido para uma soma de produtos. A forma de se conservar a AIG manipulada pela ABC é escrevendo-a diretamente no formato aiger(BIERE, A., 2006). Tal formato foi proposto em contraponto ao formato BAF, e fornece uma descrição textual mais compacta e facilmente compreensível. Porém tal formato não é suportado pelos posicionadores, que exigem uma descrição do circuito em LEF/DEF ou Bookshelf.

Portanto, após a conversão do circuito para uma AIG e sua síntese (redução funcional, balanceamento e reescrita), é gerado um arquivo intermediário no formato aiger. Tal arquivo é processado com o auxílio da AIGER (BIERE, A., 2006) um conjunto de bibliotecas e ferramentas para manipulação de AIGs *open source*. O resultado obtido é a descrição da AIG no formato blif, convertido para o formato bookshelf na etapa de posicionamento.

O fluxo detalhado da etapa de conversão do circuito para a AIG é a apresentado na Figura 3.2 abaixo.

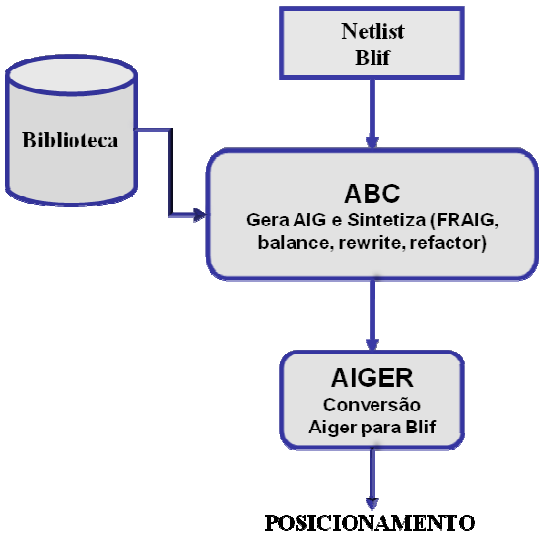

Figura 3.2: Fluxo detalhado da etapa de conversão do circuito para a AIG.

3.2.2 Posicionamento

Dentre os quatro posicionadores utilizados neste trabalho, dois (mPL6 e NTUPlace3) exigem como entrada o circuito descrito no formato Bookshelf (CALDWELL, et. al, 1999), em virtude disso, esse formato foi utilizado como padrão para todos os quatro posicionadores.

Como apresentado na etapa anterior, o resultado obtido é uma AIG descrita no formato blif. Para o posicionamento, essa descrição é convertida para o formato bookshelf com as ferramentas disponibilizadas para conversão de blif para bookshelf e geração de layout (PLACEMENT UTILITIES).

O formato bookshelf descreve o circuito com conjunto de seis arquivos, um arquivo inicial (*filename*.aux) com a lista ordenada dos outros, um *filename*.nets com a lista de nets, um *filename*.nodes com a lista de nodos e respectivas dimensões, *filename*.pl com as posições de cada nodo, um *filename*.scl com a descrição das trilhas para as células e um arquivo *filename*.wts com a largura de nodos e nets. Na conversão blif para bookshelf são gerados os arquivos .aux, .nets, .nodes. Para a dimensão dos nodos é utilizado a seguinte proporção: inversores com dimensões 1 (x=1, y=1); portas lógicas AND2 com dimensões 3 e 1 (x=3, y=1); e latchs com dimensões 6 e 1 (x=6, y=1).

Com essa descrição inicial uma ferramenta para geração de layout é utilizada para completar o conjunto de arquivos bookshelf, fornecendo os arquivos .wts, .pl e .scl. O número de trilhas gerado no arquivo .scl corresponde a área retangular necessária para o nodos presentes no circuito. Os passos até a geração do formato bookshelf completo para o posicionamento são apresentados na Figura 3.3 a seguir.

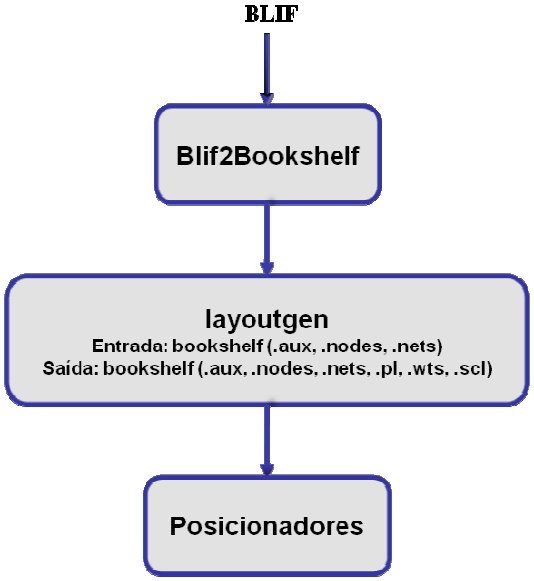

Figura 3.3: Fluxo detalhado da geração dos arquivos no formato bookshelf.

O circuito, devidamente descrito no formato bookshelf é então posicionado utilizando uma das quatro opções disponíveis: Capo, Dragon, mPL6 ou NTUPlace3. Para cada posicionador utilizou-se os recursos disponíveis para obter o melhor resultado possível no posicionamento:

- Para o Capo foram utlizados os seguintes parâmetros: -f filename.aux -num 5 -save -plot filename.plt. -f define o arquivo de entrada; -num define o número de vezes que o posicionamento é executado, por ser um posicionamento baseado

em heurísticas o resultado nem sempre é o mesmo e essa opção permite executar o Capo n vezes, e o melhor dentre esses n posicionamentos é escolhido; -save salva o posicionamento em um arquivo out.pl; e -plot gera o circuito posicionado no formato gnuplot;

- Para o Dragon foram utilizados os seguintes parâmetros: -f filename.aux -dis -pc; -f define o arquivo de entrada; -dis gera o circuito posicionado no formato .disp; e -pc salva o posicionamento no arquivo .pl;
- Para o mPL6 foram utilizados os seguintes parâmetros: -d filename.aux -mvcycle 1 -cluster_ratio 0.25 -target_density 1.0; -d define o arquivo de entrada; -mvcycle define o número de ciclos no posicionamento; -cluster_ratio define uma proporção entre o nível mais refinado e o posicionamento global; -target_density define a densidade esperada no posicionamento. Os valores utilizados são os mesmos valores recomendados para um posicionamento de alta qualidade;
- Para o NTUPlace3 foram utilizadas os seguintes parâmetros: -aux filename.aux -util 1.0; -aux define o arquivo de entrada; e -util define a densidade esperada no posicionamento.

O resultado obtido é posicionamento do circuito descrito nos mesmos arquivos iniciais. Esse resultado é então processada na etapa de otimização, descrita na subseção a seguir.

3.2.3 Otimização

Na etapa de otimização a AIG posicionada é analisada. Essa análise utiliza o fator de não monotonicidade (NMF) como parâmetro da qualidade do posicionamento em relação as interconexões entre seus nodos, e uma tabela com os NMFs de cada caminho crítico é criada.

Os caminhos críticos são identificados como os caminhos com o maior número de nodos na AIG, e para cada caminho crítico é criada uma entrada na tabela. Cada entrada da tabela armazena então o NMF do caminho, bem como NMF de cada nodo para o nodo inicial do caminho. Para estender essa análise e a otimização proposta para um número maior de caminhos da AIG, um parâmetro foi adicionado a esta etapa. Tal parâmetro define uma relação de aceitação de um caminho como crítico, desde que o caminho tenha um tamanho (número de nodos) até n% menor que o maior caminho da AIG.

O cálculo do fator de não monotonicidade segue o método proposto originalmente (PLAZA, et. al., 2008), expresso pela seguinte fórmula:

$$NMF = \frac{1}{c_{ideal}(x_i, x_k)} \sum_{n=1}^{k-1} c(x_n, x_{n+1}) \,. \quad (3.1)$$

Em que $\boldsymbol{c_{ideal}(x_i,x_k)}$ é o custo linear (distância linear) do nodo inicial $\boldsymbol{i}$ (de posição x_i) ao nodo final $\boldsymbol{k}$ (de posição x_k), e $c(\boldsymbol{x_n,x_{n+1}})$ é o custo do linear do nodo $\boldsymbol{n}$ (de posição x_n) até o nodo $\boldsymbol{n+1}$ (de posição x_{n+1}).

Realizada a análise inicial, essas informações são armazenadas para uma comparação posterior, e a otimização proposta é executada sobre cada caminho analisado.

A otimização consiste na duplicação de nodos com *fanout* maior que 1 nos caminhos analisados. Desta forma temos uma redução da dimensão das nets nos caminhos críticos, permitindo uma maior liberdade para o posicionador, possibilitando assim uma redução dos caminhos não monótonos. A duplicação de nodos também tem como vantagem o fato de não aumentar a profundidade da AIG, além de não aumentar o número de caminhos críticos, pois o que ocorre é uma divisão de nets, a única desvantagem é o aumento de área.

A duplicação de nodos é realizada apenas sobre nodos intermediários, nodos iniciais (entradas ou barreiras temporais) e finais (barreiras temporais ou saídas), não são duplicados. A não duplicação de entradas e saídas é auto-explicativa, já a não duplicação de barreiras temporais tem por objetivo não aumentar a complexidade da árvore de clock do circuito. A figura abaixo ilustra a duplicação de nodos reduzindo o fator de não monotonicidade.

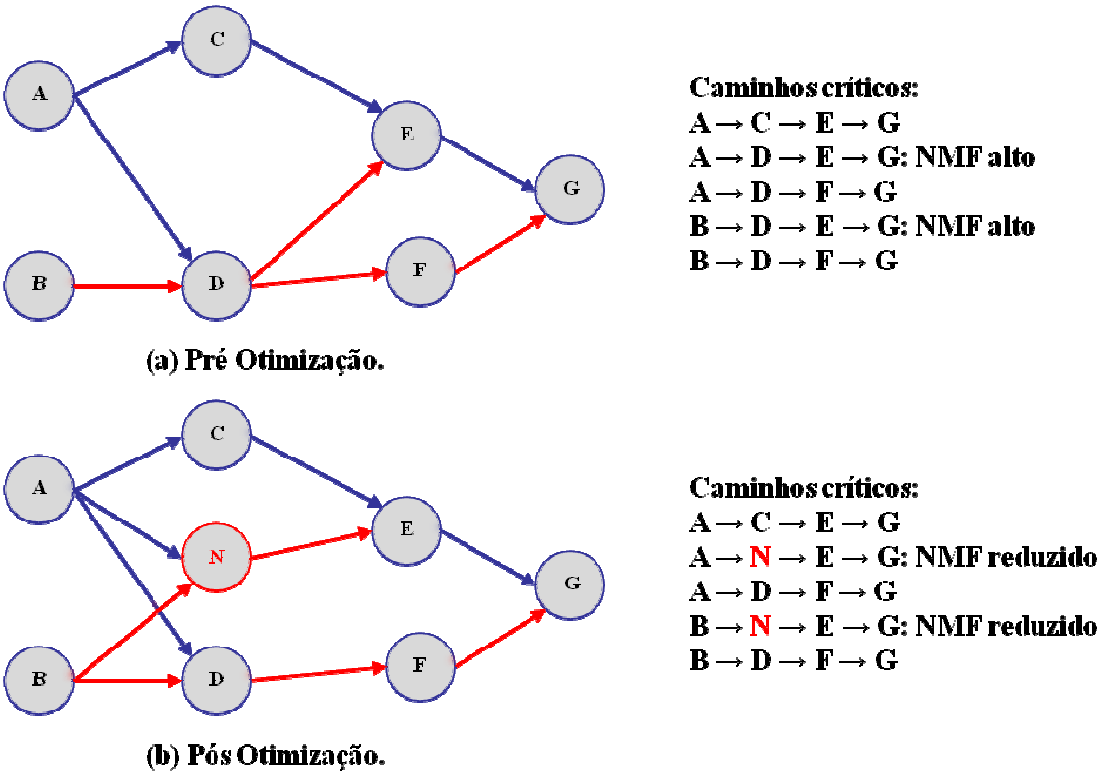

Figura 3.4: Exemplo do uso da duplicação de nodos para redução de caminhos não monótonos. (a) Caminhos iniciados nos nodos ‘A’ ou ‘B’ em direção a ‘G’ tem um ponto crítico no nodo ‘D’, aumentando consideravelmente seu NMF. (b) Com a duplicação do nodo ‘D’ pode-e obter uma redução considerável no fator de monotonicidade dos caminhos críticos.

O algoritmo utilizado na duplicação de nodos é apresentado abaixo na Figura 3.5, tem como entrada o grafo que descreve a AIG e a tabela gerada na análise dos caminhos críticos. O algoritmo percorre cada nodo presente em cada caminhos crítico na tabela, verifica se o nodo tem *fanout* maior que 1, caso afirmativo o nodo é duplicado. Para evitar múltiplas duplicações de um mesmo nodo, após a duplicação, tanto o novo nodo quanto o nodo origem, são marcados como novos no grafo. Assim a duplicação de um nodo candidato somente ocorrerá se, cumprida a exigência de *fanout* maior que 1, o nodo ainda não tenha sido duplicado.

```
TablePath table_path;
GraphNode graph;

…

for each path in table_path
  for each node in path
    if fanout ( node ) > 1 and !node.new
      duplic_node ( graph, node )
```

Figura 3.5: Algoritmo utilizado para duplicação de nodos.

Cada nodo duplicado é uma cópia exata do original, recebendo as mesmas dimensões e posicionamento. Os nodos filhos são divididos, sempre que possível, na mesma proporção. O critério para divisão é a proximidade dos nodos, ou seja, um nodo duplicado recebe o seu primeiro nodo aleatoriamente, mas os demais (se houver) são repassados de acordo com a proximidade do primeiro. A Figura 3.6 abaixo ilustra essa divisão.

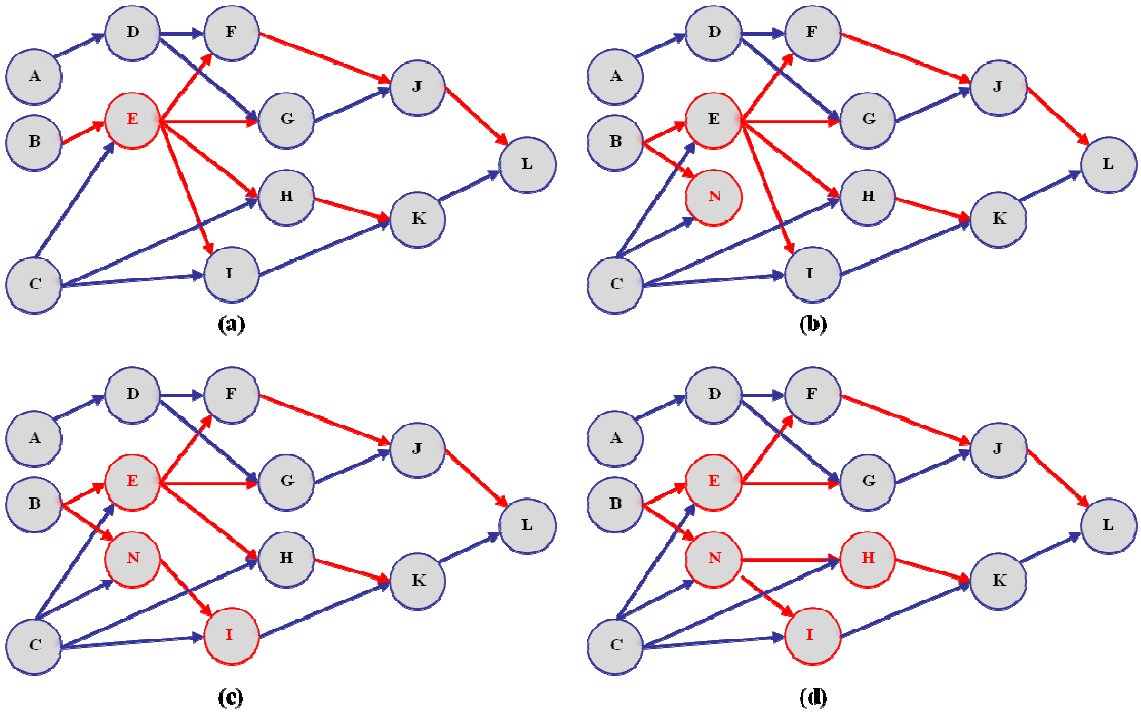

Figura 3.6: Exemplo de duplicação de nodos com divisão de nodos filhos. (a) Nodo identificado como ‘E’ é selecionado para duplicação. (b) Nodo ‘E’ é duplicado dando origem ao nodo ‘N’. (c) Nodo ‘N’ recebe o primeiro nodo filho, ‘I’, aleatoriamente de ‘E’. (d) Nodo ‘N’ recebe o segundo nodo, ‘H’, de acordo com a posição do nodo ‘I’.

Após a duplicação dos nodos a área do circuito, descrita em função das trilhas declaradas no arquivo *filename*.scl, é redimensionada na proporção do aumento de nodos na AIG, o resultado da otimização é então novamente enviado para o posicionador.

3.2.4 Reposicionamento e Análise do Resultado

A AIG otimizada é posicionada novamente, e o resultado é analisado segundo os mesmo critérios da analise pré otimização. Uma segunda tabela de caminhos críticos é gerada e os valores médios de cada tabela são calculados, fornecendo a comparação apresentada nas tabelas abaixo.

Tabela 3.1: Exemplo de tabela para comparação das dimensões do circuito pré o pós otimização.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	Nnodes_pre	Nnodes_pos	
Número de Nets	Nnets_pre	Nnets_pos	
Número de pinos	Nnpins_pre	Nnpins_pos	
Área	Area_pre	Area_pos	
Número de caminhos críticos	Npaths_pre	Npaths_pos	
Número NMF calculados	NnodesAn_pre	NnodesAn_pos	

Para todos os posicionadores, a otimização aplicada é a mesma. Portanto, para um mesmo benchmark todos os quatros posicionadores apresentarão igual aumento de nodos, nets, pinos e área, somente alterando seus fatores de não monotonicidade. Por isso a Tabela 3.1 acima é apresentada para cada benchmark executado de forma separada, a tabela 3.2, a seguir, apresenta o ganho percentual obtido em termos de NMF, por cada posicionador. Ganhos positivos representam um aumento do NMF, ganhos negativos representam uma redução.

Tabela 3.2: Exemplo de tabela para comparação de NMFs médios pré o pós otimização.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos				
NMF médio nos nodos				

O NMF do caminho é calculado entre o nodo inicial e o nodo final de cada caminho crítico, já o NMF nos nodos é calculado a partir do nodo inicial do caminho até cada nodo presente no mesmo.

Como já explicado, a duplicação de nodos não aumenta a profundidade do circuito, também não aumenta o número de caminhos críticos, portanto o número de nodos analisados pré e pós otimização deve permanecer o mesmo.

4 EXPERIMENTOS

4.1 Descrição

Para análise da otimização proposta foi utilizado o conjuto de benchmarks ITC'99 benchmarks desenvolvidos no grupo de CAD da Politecnico di Torino (ITC'99). Com netlists descritos em diversos formatos. O formato de entrada utilizado nos testes foi o blif.

Os testes seguem o fluxo proposto no capítulo 3, sendo o parâmetro que estende a análise dos caminhos críticos, descrito na seção, fixado em 20% para todos os posicionadores e benchmarks, ou seja, serão analisados os caminhos até 20% menores que o maior caminho presente no circuito.

Os resultados mais significativos são apresentados nas tabelas abaixo, divididos por benchmark. Lembrando que o objetivo principal da utilização dos quatro posicionadores não é a comparação entre o desempenho dos mesmos, mas sim a identificação de quais metodologias de posicionamento apresentam melhor resultado com a otimização proposta.

4.2 Resultados obtidos

Tabela 4.1: Benchmark B02 (FSM that recognizes BCD numbers). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	25	29	16%
Número de Nets	24	28	16.7%
Número de pinos	65	77	18.5%
Área	74	85	14.9%
Número de caminhos críticos	20	20	xxx
Número NMF calculados	94	94	xxx

Tabela 4.2: Benchmark B02 (FSM that recognizes BCD numbers). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	161%	-35.3	-33.5%	157%
NMF médio nos nodos	66.8	-22.5	-5.99	22.5

Tabela 4.3: Benchmark B01 (FSM that compares serial flows). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	68	86	26.5%
Número de Nets	62	80	29%
Número de pinos	166	219	31.9%
Área	183	228	24.6%
Número de caminhos críticos	74	74	xxx
Número NMF calculados	393	393	xxx

Tabela 4.4: Benchmark B01 (FSM that compares serial flows). Comparação do fator NMF pré e pós duplicação de nodos para os quatro posicionadores.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	-47.8%	-36.4%	-55.3%	2.68%
NMF médio nos nodos	-10.5%	-14.1%	-29.7%	-22.1%

Tabela 4.5: Benchmark B06 (Interrupt handler). Comparação das dimensões do circuito pré e pós duplicação de nodos

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	68	86	26.5%
Número de Nets	62	80	29%
Número de pinos	166	219	31.9%
Área	183	228	24.6%
Número de caminhos críticos	74	74	xxx
Número NMF calculados	393	393	xxx

Tabela 4.6: Benchmark B06 (Interrupt handler). Comparação do fator NMF pré e pós duplicação de nodos para os quatro posicionadores.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	-25.2%	-22.2%	-19.2%	-29.8%
NMF médio nos nodos	-18%	-0.514%	-9.44%	-20.5%

As tabelas de 4.1 até 4.6 apresentam os resultados da otimização proposta para os três menores benchmarks testados. Nas três situações o resultado mostrou-se bastante irregular alternando significativos ganhos em alguns casos e desempenho completamente adverso em outros.

Para o benchmark B02, o menor de todos, a duplicação de nodos causou um aumento na área do circuito aproximadamente em 15% e alcançou um bom resultado na redução do NMF médio dos caminhos críticos com os posicionadores Dragon e mPL6, porém com NTUPlace3 e Capo o resultado foi bastante ruim. Nos benchmarks B01 e B06 o número de nodos aumenta e os resultados apresentaram-se um pouco mais homogêneos, em B01 o aumento da área foi de aproximadamente 20% com uma redução do NMF médio em torno de 50% em alguns casos, em B06 o aumento da área foi de aproximadamente 25% com uma redução do NMF em torno de 20% a 50% para todos os posicionadores.

Tabela 4.7: Benchmark B09 (Serial to serial converter). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	176	214	21.6%
Número de Nets	175	213	21.7%
Número de pinos	476	590	23.9%
Área	528	624	18.2%
Número de caminhos críticos	800	800	xxx
Número NMF calculados	9920	9920	xxx

Tabela 4.8: Benchmark B09 (Serial to serial converter). Comparação do fator NMF pré e pós duplicação de nodos para os quatro posicionadores.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	-13.3%	-29.5%	-48.2%	9.89%
NMF médio nos nodos	16.4%	-10.5%	-32.1%	3.9%

Tabela 4.9: Benchmark B03 (Resource arbiter). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	198	218	10.1%
Número de Nets	612	214	10.3%
Número de pinos	515	575	11.7%
Área	568	622	9.51%
Número de caminhos críticos	474	474	xxx
Número NMF calculados	4884	4884	xxx

Tabela 4.10: Benchmark B03 (Resource arbiter). Comparação do fator NMF pré e pós duplicação de nodos para os quatro posicionadores.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	-17.7%	-26.8%	-6.83	-7.42%
NMF médio nos nodos	-20.8%	-4.01%	-11.9%	-10.3%

Tabela 4.11: Benchmark B08 (Find inclusions in sequences of numbers). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	196	261	33.2%
Número de Nets	192	257	33.9%
Número de pinos	517	712	37.7%
Área	541	718	32.7%
Número de caminhos críticos	516	516	xxx
Número NMF calculados	6472	6472	xxx

Tabela 4.12: Benchmark B08 (Find inclusions in sequences of numbers). Comparação do fator NMF pré e pós duplicação de nodos para os quatro posicionadores.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	-0.393%	-6.54%	-30.3%	-32.1%
NMF médio nos nodos	21.4%	-15.5%	-0.142%	-34.4%

Tabela 4. 13 Benchmark B10: Benchmark B10 (Voting system). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	223	286	28.3%
Número de Nets	217	280	29%
Número de pinos	597	786	31.7%
Área	600	778	29.7%
Número de caminhos críticos	388	388	xxx
Número NMF calculados	4026	4026	xxx

Tabela 4. 14: Benchmark B10 (Voting system). Comparação do fator NMF pré e pós duplicação de nodos para os quatro posicionadores.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	-27.7%	-5.38%	-29.2%	11.5%
NMF médio nos nodos	23.8%	4.29%	: -14.4%	5.31%

As tabelas de 4.7 até 4.14 apresentam os resultados da otimização proposta para os benchmarks testados com um tamanho entre 100 e 300 nodos, ao total quatro. Nesses casos o aumento de área variou entre 10% e 30% aproximadamente e a redução do NMF médio dos caminhos críticos mostrou-se novamente irregular variando muito de posicionador para posicionador em cada teste realizado.

Para o benchmark B09, com 176 nodos iniciais, o aumento da área ficou próximo de 20% enquanto que a redução do NMF variou de aproximadamente 50% com o mPL6 até um aumento de cerca de 10% para o NTUPlace3. Para os benchmarks B08 e B10 o aumento da área foi consideravelmente alto, em torno de 30%, para uma redução no NMF de até 30% em alguns casos, nesses dois casos o NTUPlace3 apresentou uma variação muito grande na redução do NMF médio dos caminhos críticos, -32.1% para B08 e +11.05% para B10. E o benchmark B03 apresentou o melhor resultado até aqui, com um amento de área inferior a 10%, a redução do NMF médio nos caminhos críticos foi acima de 25% com o Dragon e com o Capo chegou a 17.7%.

Tabela 4.15: Benchmark B13 (Interface to meteo sensors). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	359	381	6.13%
Número de Nets	349	371	6.3%
Número de pinos	930	996	7.1%
Área	1010	1080	6.11%
Número de caminhos críticos	81	81	xxx
Número NMF calculados	873	873	xxx

Tabela 4.16: Benchmark B13 (Interface to meteo sensors). Comparação do fator NMF pré e pós duplicação de nodos para os quatro posicionadores.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	-15.1%	-17.7%	-6.12%	-26.5%
NMF médio nos nodos	-0.895%	-3.76%	3.48%	-11.3%

Tabela 4.17: Benchmark B07 (Count points on a straight line). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	452	524	15.9%
Número de Nets	444	516	16.2%
Número de pinos	1239	1455	17.4%
Área	1280	1470	14.9%
Número de caminhos críticos	27215	27215	xxx
Número NMF calculados	670710	670710	xxx

Tabela 4.18: Benchmark B07 (Count points on a straight line). Comparação do fator NMF pré e pós duplicação de nodos para os quatro posicionadores.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	31.8%	-3.08%	-5.88%	0.326%
NMF médio nos nodos	2.61%	-12.2%	-8.41%	0.27%

Tabela 4.19: Benchmark B11 (Scramble string with variable cipher). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	573	636	11%
Número de Nets	567	630	11.1%
Número de pinos	1627	1816	11.6%
Área	1560	1720	9.86%
Número de caminhos críticos	545	545	xxx
Número NMF calculados	11217	11217	xxx

Tabela 4.20: Benchmark B11 (Scramble string with variable cipher). Comparação do fator NMF pré e pós duplicação de nodos para os quatro posicionadores.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	-55.1%	-34.5%	72.1%	-10.1%
NMF médio nos nodos	-54.1%	-18.3%	6.62%	-16.6%

Tabela 4.21: Benchmark B04 (Compute min and max). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	661	736	11.3%
Número de Nets	653	728	11.5%
Número de pinos	1806	2031	12.5%
Área	1820	2010	10.5%
Número de caminhos críticos	474	474	xxx
Número NMF calculados	4884	4884	xxx

Tabela 4.22: Benchmark B04 (Compute min and max). Comparação do fator NMF pré e pós duplicação de nodos para os quatro posicionadores.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	-35.7%	-26.7%	-50.8%	-21.8%
NMF médio nos nodos	-32.9%	-5.62%	-56%	-18%

As tabelas de 4.15 até 4.22 apresentam os resultados da otimização proposta para os benchmarks testados com um tamanho entre 300 e 700 nodos, ao total quatro. Nesses casos o aumento de área variou menos ficando em torno de 10% e 15% para ganhos significativos na redução do NMF médio nos caminhos críticos em quase todos os testes.

Para o benchmark B13 o aumento foi o menor entre todos os testes, 6,11% para uma redução do NMF médio nos caminhos críticos acima de 6% com o mPL6 e acima de 15% com os outros posicionadores. Os benhmarks B04 e B11, apesar do aumento de área semelhante, em torno de 10%, apresentaram resultados bastante distintos em relação a redução do NMF médio nos caminhos críticos, enquanto que para B04 em todos os casos a redução do NMF foi significativa, entre 20% e 50%, para B11 o ganho no NMF oscilou de uma redução em 50% com o Capo até um aumento acima de 70% com o mPL6. Para o benchmark B11, com o maior aumento de área nesses quatro testes, a redução do NMF apresentou resultados ruins para todos os casos, chegando até a um aumento acima de 30% com o Capo.

Tabela 4.23: Benchmark B12 (1-player game (guess a sequence)). Comparação das dimensões do circuito pré e pós duplicação de nodos.

Parâmetro	Pré-Otimização	Pós-Otimização	Relação (%)
Número de Nodos	1214	1361	12.1%
Número de Nets	1208	1355	12.2%
Número de pinos	3383	3824	13%
Área	3470	3870	11.4%
Número de caminhos críticos	4668	4668	xxx
Número NMF calculados	71137	71137	xxx

Tabela 4.24: Benchmark B12 (1-player game (guess a sequence)). Comparação do fator NMF pré e pós duplicação de nodos para os quatro posicionadores.

Parâmetro	Capo	Dragon	mPL6	NTUPlace
NMF médio nos caminhos críticos	-18.8%	-29.3%	3.57%	-28%
NMF médio nos nodos	-11.45%	-8.68%	-1.06%	-23.6%

4.3 Análise dos Resultados

Os testes realizados apresentaram grandes oscilações nos resultados obtidos entre os benchmarks testados, mesmo nos casos em que o número de nodos é semelhante (casos de B04 e B07), e também, para alguns casos, entre posicionadores no mesmo benchmark (caso de B11).

Oscilações entre benchmarks podem ser explicadas pelas características de cada circuito. A proporção de caminhos críticos em relação ao número caminhos do circuito pode explicar o aumento de área considerável em alguns casos (em torno de 30%), pois

com uma maior proporção de caminhos críticos o número de duplicações pode aumentar significativamente. Circuitos com uma menor proporção de caminhos críticos tendem a apresentar um menor número de duplicações e se é dada uma prioridade a esses caminhos a tendência é uma redução significativa do NMF médio. Outro fator que pode influenciar no resultado é a dimensão das nets, quando uma net apresenta um número elevado de nodos a simples duplicação do nodo de entrada pode não ser suficiente para reduzir o NMF médio do caminho, pois a tendência é de que o caminho não possa ser linearizado, causando um aumento do fator NMF.

As oscilações entre os resultados obtidos por diferentes posicionadores para um mesmo benchmark podem ser explicadas pelo algoritmo implementado por cada posicionador, tendo mais ou menos dificuldade em tratar o aumento de nodos em um mesmo nível.

Independente das oscilações em grande parte dos testes os resultados foram satisfatórios, demonstrando que a duplicação de nodos pode representar ganhos significativos na redução dos NMF no circuito e consequentemente reduzir o custo médio das interconexões.

5 TRABALHOS FUTUROS

Para integração da otimização implementada neste trabalho como uma ferramenta completa no novo fluxo proposto para o projeto de circuitos VLSI baseados em *standard cells*, algumas etapas devem ser concluídas, fornecendo então uma solução completa e totalmente integrada ao novo fluxo de projeto. Essas etapas incluem a integração de um posicionador e o mapeamento tecnológico.

5.1 Integração de um Posicionador

A integração de um posicionador eliminaria etapas de conversão de formato, além de permitir o uso das estruturas de dados otimizadas para AIGs (MISHCHENKO, A., et. al. 2004). Desta forma pode-se obter um significativo ganho de tempo no processamento do posicionamento da AIG.

Assim sendo, o posicionador a ser integrado deve ser capaz de trabalhar internamente com as estrutura de dados utilizadas para a AIG. Como tal ferramenta não existe, é necessária a sua implementação, ou então a adaptação de um posicionador as estruturas de dados da AIG.

Com a integração de um posicionador a solução, um refinamento da otimização pode ser realizado. Tal refinamento passa por uma análise mais criteriosa dos nodos a serem duplicados, além do *fanout* do nodo questões como sua posição e o NMF dos nodos conectados a sua saída podem ser analisadas. A inserção de buffers em situações em que a duplicação de nodos não resulta em uma diminuição no NMF também é uma alternativa que pode ser analisada.

5.2 Mapeamento Tecnológico

O mapeamento tecnológico completa a solução proposta, possibilitando a substituição de uma etapa complexa de posicionamento por uma etapa de validação e refinamento do resultado obtido no mapeamento tecnológico.

O mapeamento segue o processo proposto em (MISHCHENKO, et al., 2004), com uma varredura da AIG posicionada em busca de padrões cujas funções lógicas são conhecidas e presentes na biblioteca. O objetivo é cobrir todo o grafo usando os padrões conhecidos com uma solução ótima segundo uma função de custo. Nesse caso a função custo deve garantir a prioridade de mapeamento aos padrões que englobam nodos posicionados próximos um ao outro, mantendo-se assim a solução obtida após o posicionamento do AIG.

No mapeamento tecnológico podem ocorrer sobreposições e regras de projeto podem ser infligidas. Essas situações ser tratadas na etapa posterior, onde a validação e refinamento da solução substituem o posicionamento.

6 CONCLUSÃO

A contínua redução da dimensão dos dispositivos VLSI tem aumentado consideravelmente a complexidade do projeto de circuitos VLSI, a densidade de transistores cresce quadraticamente com a redução do tamanho do seu tamanho, exigindo cada vez mais das ferramentas utilizadas para automatização do projeto. Além disso, o atraso das interconexões permanece o mesmo, tornando-se um fator determinante do desempenho do sistema. Cabe então as ferramentas de CAD tratar mais este fator.

Toda essa complexidade tem aumentado o tempo de desenvolvimento do projeto, principalmente nas etapas de síntese física. O aumento da densidade de transistores em conjunto com a não redução dos atrasos das interconexões exige cada vez mais esforço no posicionamento e roteamento do circuito para que possa se alcançar os requisitos exigidos. Uma alternativa é diminuir a complexidade nessas etapas, dividindo o problema com etapas anteriores.

Portanto esse foi o principal foco deste trabalho. Para tanto a solução proposta engloba a utilização de uma representação mais simples e otimizada do circuito, a AIG, posicionada e otimizada em uma nova etapa no fluxo do projeto de circuitos VLSI baseados em *standard cells*.

Nesta nova etapa, a solução proposta utiliza a AIG que representa o circuito para um pré-posicionamento, seguido da sua otimização (que tem por objetivo principal reduzir os atrasos das interconexões) e o mapeamento tecnológico sobre a AIG posicionada finalizando o processo. O fato de a ferramenta produzir um AIG pré posicionado permite ao mapeamento tecnológico tomar decisões que já levam em conta o posicionamento da estrutura de dados usada no mapeamento. Como resultado, tem-se a simplificação da etapa de posicionamento, que passa a ser apenas um processo de validação e refinamento da solução obtida na etapa anterior.

A otimização proposta para redução dos atrasos das interconexões é a duplicação de nodos com *fanout* maior que um em caminhos críticos, está idéia baseia-se em três premissas: a) a duplicação de nodos não aumenta a profundidade do circuito; b) a duplicação de nodos não aumenta o número de caminhos críticos; c) com a redução das dimensões dos transistores pode-se conceber um aumento de área em razão de um ganho no atraso do circuito.

Para os experimentos foram utilizados quatro posicionadores estado da arte, todos ou com código aberto ou executável de distribuição livre. O objetivo do uso de diferentes posicionadores foi determinar qual a metodologia apresenta melhor resultado

em conjunto com a otimização proposta. O parâmetro utilizado para análise dos resultados foi o fator de não monotonicidade médio nos caminhos críticos.

Os resultados, apesar de algumas oscilações, de modo geral apresentaram uma redução considerável do NMF médio nos caminhos críticos, demonstrando que a duplicação de nodos pode ser uma boa alternativa para redução das interconexões, e consequentemente redução do atraso, no projeto de CIs.

Contudo, a solução proposta ainda não está completa, necessitando cumprir aind algumas etapas, já relacionadas no capítulo anterior, que podem proporcionar um ganho ainda mais significativo na redução dos atrasos das interconexões, além de acelerar o tempo de projeto de CIs baseado em *standard cells*. A integração de um posicionador direcionado ao posicionamento de AIGs, principalmente, pode materializar o fluxo proposto no capítulo como uma alternativa aos métodos atuais para projeto de CIs e alcançar todos os objetivos aqui propostos.

REFERÊNCIAS

AJAMI, A. H.; PEDRAM M., Post-Layout Timing-Driven Cell Placement Using an Accurate Net Length Model with Movable Steiner Points. **Proceedings of the 2001 Conference on Asia South Pacific Design Automation**, p. 595-600, 2001.

BAKOGLU, H. B., Circuits, Interconnections, and Packaging for VLSI, Addison-Wesley, 1990. 978-0201060089.

BARDEEN, J.; Brattain, W., The Transistor, a Semiconductor Triode. **Phys. Rev**., vol. 74, p. 230, Jul. 1948.

BEESON, R.; RUEGG, H., New Forms of All Transistor Logic. **ISSCC Digest of Technical Papers**, p. 10–11, Fev. 1962.

BIERE, A., 2006. AIGER: format, library and set of utilities for And-Inverter Graphs (AIGs), Tech. Report [S.l.:s.n]. Disponível em: http://fmv.jku.at/aiger/FORMAT.aiger. Acesso em: nov. 2010.

BJESSE, P.; BORALV, A.; DAG-aware circuit compression for formal verification. **Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference,** p. 42-49, 7-11 Nov. 2004.

BREUER, M. A., A Class of Min-cut Placement Algorithms. **Design Automation Conference IEEE/ACM**, p. 284-290, 1977.

CALDWELL, A., KAHNG, A. B., MARKOV, I., 1999. Placent Formats, rev 1.2 [S.l.:s.n]. Disponível em: http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Placement/plFormats.html. Acesso em: nov. 2010.

CHAN, T. F., et. al. 2006. mPL6: Enhanced Multilevel Mixed-Size Placement", In: ISPD'06, Ab. 2006.

CHANG, C. W.; et. al. 2000. Fast Post-placement Rewiring Using Easily Detectable Functional Symmetries. **Annual ACM IEEE Design Automation Conference**, p.286-289, 2000.

CHANG, C. W.; MARKOV, I. L.; BERTACCO, V. 2007. Safe Delay Optimization for Physical Synthesis. **ASP-DAC'07**, p. 628-633, 2007.

CHEN, T.-C. ; JIANG, Z.-W. ; HSU, T.-C. ; CHEN, H.-C ; CHANG Y.-W ; 2008. NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints. [S.l.:s.n]. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, Vol.27, No.7.

DUNLOP, A. E.; KERNIGHAN, B. W., A Procedure for Placement of Standard Cell VLSI Circuits. **IEEE Transactions on Computer Aided Design**, Vol. 4, No. 1, p. 92-98, Jan, 1985.

FLYNN, M. J.; HUNG, P.; RUDD, K. W., Deep-Submicron Microprocessor Design Issues, **IEEE Micro 1**, 2009.

GAJSKI, D. D.; KUHN, R. H., New VLSI Tools. **IEEE Computer**, Vol. 16, No. 12, p. 11-14.

GEREZ, S. H., Algorithms for VLSI Design Automation. Baffins Lane: John Wiley & Sins Ltd., 1998. ISBN 0-471-98489-2.

GOTO, S.; KUH, E. S., An Approach to the Two-Dimensional Placement Problem in Circuit Layout. **IEEE Transactions on Circuits and Systems**, Vol. 25, No. 3, p. 208-214, 1978.

HARRIS, J., Direct-Coupled Transistor Logic Circuitry in Digital Computers. **ISSCC Digest of Technical Papers**, p. 9, Fev. 1956.

HWANG, C.; PEDRAM, M., Timing-Driven Placement Based on Monotone Cell Ordering Constraints. **Proceedings of the 2006 Conference on Asia South Pacific Design Automation**, p. 201-206, 2006.

ITC'99 Benchmarks. Disponível em: http://www.cad.polito.it/downloads/tools/itc99.html. Acesso em: nov. 2010.

KANNAN, L. N., SUARIS, P. R. e FANG, H., A Methodology and Algorithms for Post-placement Delay Optimization. **Annal ACM IEEE Design Automation Conference**, p. 327-332.

KEUTZER, K.; NEWTON, A. R.; SHENOY, N, The Future of Logic Synthesis and Physical Design in Deep-submicron Process Geometries. **Internacional Symposium on Physical Design**, p. 327-332, 1994.

KUEHLMANN, A. et. al., Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, Vol. 21, No. 12, p. 1377-1394, 2002.

LI, D.; KoH, C. K.; MADDEN, P., Floorplan Management: Incremental Placement for Gate Sizing and Buffer Insertion. **Proceedings of the 2005 Conference on Asia South Pacific Design Automation**, p. 349-354, 2005.

MISHCHENKO, A., et. al. 2004. FRAIGs: Functionally Reduced AND-INV Graphs. **International Conference on Computer Aided Design**, 2004.

MISHCHENKO, A., 2005.ABC: A System for Sequential Synthesis and Verification. Berkeley Logic Synthesis and Verification Group. Disponível em <http://www.eecs.berkeley.edu/~alanmi/abc/abc.htm>. Acesso em: nov 2010.

MISHCHENKO, A., et. al. 2006. Improvements to Combinational Equivalence Checking. **International Conference on Computer Aided Design**, p. 836-843, 2004.

NAM, G. J.; CONG, J., Modern Circuit Placement. New York: Springer Science, 2007. IABN 978-0-387-36837-5.

NAYLOR, W. C., Donelly, R. and Sha, L. (2001) "Non-linear optimization system and method for wire length and dealy optimization for an automatic electric circuit placer", US patent 6,301,693.

NORMAN, R.; LAST, J.; HAAS, I. Solid-State Micrologic Elements. **ISSCC Digest ofTechnical Papers**, p. 82–83, Fev. 1960.

PLACEMENT UTILITIES. Disponível em: http://vlsicad.eecs.umich.edu/BK/PlaceUtils. Acesso em nov. 2010.

PLAZA, S. M., MARKOV, I. L., BERTACCO, V., Optimizing Non-Monotonic Interconnect using Functional Simulation and Logic Restructuring. **ISPD'08**, Ab. 2008.

RABAEY, J. M.; CHANDRAKASAN, A. P.; NICKOLIC, B., Digital Integrated Circuits. 2th ed. s.l.: Paperback, 2003.

ROY, J. A.; PAPA, D. A.; SAURABH, N. A.; CHAN, H. H,; NG, A. N.; LU, J. F., MARKOV, I. L. Capo: Robust and Scalable OpenSource MinCut Floorplacer. **ISPD'05.** Ab. 2005, San Francisco, California, USA.

SCHOCKLEY, W., The Theory of pn Junctions in Semiconductors and pn-Junction Transistors. **BSTJ**, vol. 28, p. 435, 1949.

SUARIS, P. R.; KEDEN, G., Quadrisection: A New Approach to Standard Cell Layout. **International Conference on Computer Aided Design IEEE/ACM**, p. 474-477, Nov. 1987.

SWADE, D., Redeeming Charles Babbage's Mechanical Computer. **Scientific American**, p. 86–91, Fev. 1993.

TAGHAVI, T.; YANG, X.; CHOI, B.-K. (2006) "Dragon2005: Large-Scale Mixed-size Placement Tool", In: ISPD'05, San Francisco,California, USA, April.

THEIS, T. N., The Future of Interconnection Technology. **IBM Journal of Reasearch and Development**, Vol. 44, No. 3, 2000.

VEDOVELLI, E. **Otimização de Interconexões Através de Posicionamento e Síntese Lógica**. 2009. 45 f. Projeto de Diplomação (Graduação em Engenharia da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

ANEXO A <ARTIGO TG1: OTIMIZAÇÃO DE INTERCONEXÕES ATRAVÉS DE AIGS>

Otimização de Interconexões através de AIGs

Alberto dos Santos Carvalho Jr.[1], André Inácio Reis[2]

[1,2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

ascjunior@inf.ufrgs.br, andreis@inf.ufrgs.br

Abstract. *The VLSI technology has experienced a constant reduction of the feature size of VLSI devices (i.e. the minimum transistor size), which has a strong impact on the design of ICs in various ways. First, the drastic increase in density. Second, the devices operate at higher speed, but the delay of their interconnections remains the same. This paper proposes a new tool for optimization of interconnects in VLSI circuits. Using a representation in form of AIG, the circuit is positioned through state of the art placers, optimized for their critical paths and then repositioned, the process is repeated until it reach an acceptable outcome.*

Resumo. A tecnologia VLSI tem experimentado uma constante redução na dimensão de seus dispositivos (i.e. o tamanho mínimo do transistor), o que tem forte impacto sobre o projeto de ICs de várias formas. Primeiro, o aumento drástico na densidade. Segundo, os dispositivos operam a uma velocidade maior, porém o atraso de suas interconexões permanece o mesmo. Este trabalho propõe uma nova ferramenta para otimização de interconexões em circuitos VLSI. Utilizando uma representação em forma de AIG o circuito é posicionado através de posicionadores estado da arte, otimizado em relação a seus caminhos críticos e então reposicionado, o processo se repete até que se alcance um resultado aceitável.

1. Introdução

O desenvolvimento das tecnologias de fabricação de ICs tem escalado para transistores de tamanhos cada vez menores, permitindo uma densidade que hoje ultrapassa a casa de centenas de milhões de transistores por chip. Essa contínua redução da dimensão dos dispositivos VLSI (Very Large Scale Integrated) tem forte impacto sobre a tecnologia de várias formas. Primeiro, a densidade cresce quadraticamente com a taxa de diminuição do tamanho dos transistores. Segundo, os dispositivos operam a uma velocidade maior, porém o atraso das interconexões permanece o mesmo, efeito de suas capacitâncias que não diminuem, Theis (2000), esse atraso torna-se muito mais significativo.

De acordo com uma regra simples de escalonamento descrita em Bakoglu (1990), quando os dispositivos e interconexões são reduzidos em todas as suas três dimensões por um fator S, o atraso intrínseco de *gate* é reduzido por um fator S, o atraso das interconexões locais (conexões entre portas adjacentes) permanece o mesmo, mas o atraso de interconexões globais aumenta por um fator S^2. Como resultado, o atraso das interconexões tem se tornado um fator dominante na determinação do desempenho do sistema. Em muitos dos sistemas atuais, cerca de 50% a 70% do ciclo de *clock* é consumido por atrasos de interconexões.

Um dos pontos críticos para redução dos atrasos das interconexões no circuito é o posicionamento das células, um bom posicionamento deve ter um roteamento possível além de atender aos requisitos de atraso e potência. Para as tecnologias mais recentes, o problema do posicionamento tornou-se ainda mais complexo, pois além do grande aumento do número de elementos a serem posicionados o atraso das interconexões tornou-se da mesma ordem de grandeza do atraso das células, fazendo com que a análise desse atraso faça parte do processo de posicionamento. Nessa situação os algoritmos de posicionamento devem levar em consideração múltiplas questões, como por exemplo, minimização das interconexões, congestionamento e potência. Grande parte dos posicionadores atuais considera como métrica para otimização das interconexões o comprimento total das mesmas (no caso de posicionadores que utilizam uma abordagem analítica), ou o número de interconexões que atravessam secções (no caso de posicionadores que utilizam a abordagem de particionamento *min-cut*). Em alguns casos o posicionamento detalhado ou um pós posicionamento direcionado a redução de atrasos é utilizado como forma de obter um melhor resultado.

Outra questão que se impõem em relação aos posicionadores atuais é o tempo utilizado para execução do posicionamento. Para muitos projetos mais complexos (com mais de 10^5 transistores), o tempo utilizado no processamento do posicionamento pode chegar a casa da dezena de horas em alguns posicionadores comerciais, para um resultado que nem sempre é satisfatório.

Este trabalho propõe uma nova ferramenta de otimização de interconexões utilizando uma representação do circuito na forma de AIGs (AND-INV *Graphs*) e, posicionadores estado da arte. O uso da AIG tem por objetivo facilitar a manipulação e otimização do circuito, visto sua estrutura simplificada, e mesmo não sendo uma representação canônica do circuito pode ser reduzida funcionalmente (FRAIG – *Functionally Reduced* AIG). O circuito descrito na forma de uma AIG é posicionado e analisado quanto aos requisitos de atraso, verificando-se seus caminhos críticos, e técnicas de otimização de atraso são aplicadas (mais precisamente a duplicação de nodos nos caminhos críticos é a metodologia empregada). O resultado da otimização é novamente posicionado, processo repetido até que se obtenham os requisitos de atraso ou tenha-se alcançado um limite de iterações estipulado. Como resultado, obtemos uma AIG posicionada, que pode ser utiliza na etapa de posicionamento, ou para mapeamento das células diretamente sobre a AIG ou como um guia para etapa de posicionamento.

Este trabalho está organizado da seguinte forma: Na seção dois é apresentada uma descrição geral sobre projetos VLSI; na seção três são revisadas algumas técnicas de particionamento e posicionamento presentes nos posicionadores utilizados neste trabalho; a seção quatro apresenta os posicionadores; na seção cinco são revisadas algumas técnicas de otimização de atraso, além de uma descrição mais detalhada da AIG; a seção seis apresenta uma descrição do projeto da nova ferramenta; a seção sete apresenta as conclusões obtidas; e por fim a seção oito traz um resumo de trabalhos futuros para a ferramenta proposta.

2. Projetos VLSI

Devido ao constante aumento de sua complexidade e da proliferação de seu uso, o projeto de ICs tem a crescente necessidade de ferramentas de CAD (*Computer Aided Design*) capazes de automatizar seu desenvolvimento, simplificando e acelerando o processo. Além disso, essas ferramentas devem ser capazes de auxiliar na otimização dos diferentes requisitos do circuito, tais como: (a) área total ocupada pelo circuito; (b)

atraso de resposta ou *timing*: tempo de propagação de uma alteração na entrada do circuito até seu resultado na saída; (c) potência (energia) consumida pelo circuito; (d) testabilidade do circuito; (e) tempo de projeto, seja, o tempo destinado ao desenvolvimento do projeto.

A combinação de todos esses requisitos dentro de uma única função de custo é algo praticamente impossível de ser otimizado devido a sua complexidade, Gerez (1998). Para ajudar no tratamento de tal complexidade dois conceitos são utilizados: Hierarquia, descrevendo a estrutura do projeto em diferentes níveis, e abstração, escondendo detalhes em cada nível da hierarquia. O uso de abstração permite-nos trabalhar sobre um determinado número de blocos em cada nível da hierarquia, com cada bloco composto por um determinado número de sub-blocos. A decomposição continua, até que os blocos mais básicos da hierarquia sejam alcançados. Porém uma hierarquia simples não seria suficiente para descrever o processo de desenvolvimento de um projeto VLSI. Para isso, existe um consenso geral de que há três domínios de projeto, cada um com sua hierarquia, são eles:

- Domínio comportamental: descrição das funcionalidades do circuito, onde parte, ou todo, é visto como uma caixa preta e a relação entre entradas e saídas são dadas sem uma referência de como essa relação é implementada. Um sistema é descrito por algoritmos, descritos por elementos de memória, descritos através de uma lógica, que é descrita por relações de transferências;

- Domínio estrutural, uma descrição do circuito através de subcircuitos. Um processador é composto por ULAs, memória, etc., que são formadas portas lógicas, *flip-flops*, etc., que por sua vez são formadas por transistores;

- Domínio físico, uma descrição de como os subcircuitos do domínio estrutural estão dispostos fisicamente no chip.

Os três domínios são representados através do diagrama em Y de Gajski and Kuhn (1998), apresentado na Figura 1, a seguir.

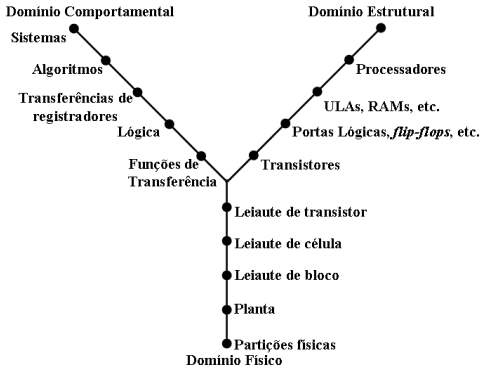

Figura 1:Diagrama em Y de Gajski para os três domínios de projeto.

Diferentes metodologias para o projeto de ICs foram desenvolvidas a partir do diagrama em Y de Gajski, genericamente classificadas como *full custom* ou *semi-custom*, sendo o termo *custom* uma referência ao grau de flexibilidade do leiaute. A metodologia *full custom* tem como objetivo a otimização de parâmetros que nem sempre pode ser alcançada simultaneamente em abordagens mais automatizadas (por exemplo, área, *timing*, potência). É utilizada para o desenvolvimento de circuitos de propósito geral (microprocessadores, memórias), quando o número de chips justifica o tempo necessário para um leiaute altamente otimizado. A metodologia *semi-custom* tem

como objetivo acelerar o tempo de design do projeto, sendo mais utilizada para desenvolvimento de circuitos de aplicação específica (ASICs – *Application-specific Integrated Circuit*). *Gate array*, *standard cell* e *macro cell* são os três mais populares estilos de design *semi-custom*. O *gate array* é o mais restrito, todos os blocos tem o mesmo tamanho e somente podem ser associados a uma única *grid* do *layout*. *Standard cells* tem uma altura fixa, mas sua largura pode variar dependendo da funcionalidade do módulo. *Macro cells* podem ter diferentes formas e dimensões.

A metodologia *standard cell* considera blocos de células com leiautes retangulares de mesma altura já definidos. Inicialmente, na síntese lógica, o circuito é dividido em vários blocos menores, cada bloco é mapeado para algum subcircuito (célula), pré-definido, com funcionalidades equivalentes e características elétricas analisadas e especificadas. O conjunto de blocos predefinidos é denominado de biblioteca de células. A etapa seguinte realiza o planejamento da área total ocupada pelo circuito (*floorplanning*), alocando o espaço necessário para os diferentes blocos que o compõem, e o planejamento das linhas de alimentação. Após isso é realizado o posicionamento das células (*placement*), o planejamento da árvore de *clock* do circuito, e o roteamento das interconexões que ligam as células entre e inter blocos. O circuito é verificado então quanto as restrições de fabricação (dependentes da tecnologia) e quanto aos requisitos especificados de atraso, potência e área. O fluxo da metodologia *standard cell* é apresentado na Figura 2, abaixo.

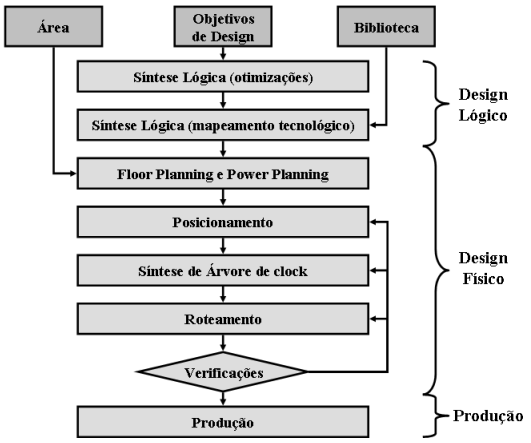

Figura 2: Fluxo da Metodologia Standard Cell.

3. Técnicas de Particionamento e Posicionamento VLSI

3.1. Técnicas de Particionamento

Para o estudo do problema utilizamos a notação de grafos, notação que será referenciada nas próximas subseções. Sendo o grafo $G=G(V, E)$, em que G consiste de um conjunto V de vértices $V=\{v_1, v_2, v_3, ...\}$ e um conjunto E de arcos $E=\{e_1, e_2, e_3, ...\}$, cada arco corresponde a um par de vértices distintos. Sendo o hipergrafo $H=H(N, L)$, em que H consiste de um conjunto N de vértices e um conjunto L de hiperarcos, com cada hiperarco correspondendo a subconjunto N_i de distintos vértices e $|N_i| \geq 2$. Também é associada uma função peso $\omega: V \rightarrow \mathcal{N}$ a cada vértice ($\mathcal{N}$ é um conjunto de inteiros). Assim o circuito pode ser representado por um grafo ou por um hipergrafo, onde os vértices são os elementos do circuito e os (hiper)arcos correspondem as interconexões, além disso, o peso do vértice pode indicar o tamanho do elemento no circuito.

As técnicas de particionamento presentes nos posicionadores utilizados neste trabalho são apresentadas a seguir.

3.1.1. Kenrighan-Lin e Fiduccia-Mattheyses

Dado um grafo $G=G(V, E)$ sem peso, particionamos G dentro de dois grupos, V_1 e V_2, sendo que $|V_1| < \alpha|V|$ e $|V_2| < \alpha|V|$, onde α é o fator de balanço (tipicamente α=1/2 nesse caso) e $|V|$ denota o número de vértices em V. Após identificar um par de vértices (v_a, v_b) com $v_a \in V_1$ e $v_b \in V_2$, cuja troca resulta em um decremento do custo da função de corte, ou um pequeno aumento caso nenhum decremento seja possível (na esperança de que o custo decremente nos próximos passos), os vértices v_a e v_b são bloqueados.

Há um valor máximo para a soma parcial $\sum_{i=1}^{k} g_i = Gain_k$, onde g_i é o ganho do i-ésima troca. Se $Gain_k > 0$, a redução do custo pode ser obtida movendo-se $\{v_{a1}, ..., v_{ak}\}$ para V_2 e $\{ v_{b1}, ..., v_{bk}\}$ para V_1. Após isso, o resultado é tratado como um particionamento inicial e o procedimento repete-se até que não exista um k tal que $Gain_k > 0$.

Fidducia e Mattheyses aperfeiçoaram o algoritmo de KL, reduzindo sua complexidade, através da introdução dos seguintes requisitos: (a) Somente um vértice pode ser deslocado por vez; (b) cada vértice tem um peso associado; (c) Uma estrutura especial de dados é utilizada para selecionar os vértices a serem deslocados que consiste de um *array* de ponteiros indexado por um conjunto $[-d_{max}\cdot\omega_{max}, d_{max}\cdot\omega_{max}]$ onde d_{max} corresponde ao máximo grau de um vértice no hipergrafo e ω_{max} corresponde ao máximo custo da hiperaresta, assim os índices (positivo ou negativo) correspondem aos possíveis ganhos. Para duas partições A e B, por exemplo, temos dois *arrays*, ***lista A*** e ***lista B***, e o movimento de um vértice altera custo no mínimo por $d_{max}\cdot\omega_{max}$.

3.1.2 Particionamento Multinível

Desenvolvida para circuitos de larga escala, essa abordagem caracteriza-se por uma sucessiva construção de pequenos hipergrafos (*coarsening*). Então é realizado um biparticionamento do menor hipergrafo, essa bipartição é projetada para os próximos níveis (*uncoarsening*) e a cada nível um algoritmo iterativo é utilizado para refinar o particionamento.

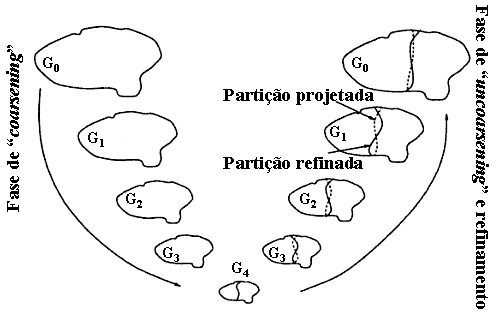

Figura 3: Fases do Particionamento Multinível em bisecções.

3.2 Técnicas de Posicionamento

A classificação mais comum na literatura para algoritmos de posicionamento divide-os em duas classes: construtivos e iterativos. Os algoritmos construtivos geram um posicionamento completo a partir da descrição da *netlist* do circuito. Os algoritmos iterativos iniciam a partir de algum posicionamento já determinado e realizam

alterações nas posições das células com o objetivo de alcançar um melhor resultado. Entre os algoritmos construtivos pode-se destacar o posicionamento aleatório (para posicionamento inicial), o posicionamento direcionado a forças e o baseado em particionamento (quadratura, *min-cut,* etc.). Entre os algoritmos iterativos destacam-se as meta-heurísticas, como o *Simulated Annealing*, Algoritmos Genéticos e Busca Tabu. Na prática, nenhuma das duas abordagens alcança um resultado satisfatório em tempo de execução aceitável quando utilizada isoladamente. Pesquisadores têm tentado utilizar ambos os métodos Goto and Kuh (1978), e as ferramentas comercias de maior êxito seguem esse caminho, iniciam com uma solução construtiva que é aperfeiçoada com o uso de um algoritmo iterativo.

Outra classificação adotada para identificar os algoritmos de posicionamento, divide-os em quatro tipos: posicionamento baseado em particionamento, posicionamento em quadratura, *Simulated Annealing*, posicionamento direcionado a forças. Todos os quatro tipos são apresentados a seguir.

3.2.1 Posicionamento baseado em Particionamento

Também denominados de posicionamento *min-cut*, tem como idéia central reduzir o número de interconexões sendo cortadas no particionamento do circuito, em uma sequência de sucessivas divisões (em bisecções ou quadrisecções).

3.2.1.1. Algortimo de Breuer e Propagação de Terminal

O primeiro posicionamento *min-cut* foi proposto por Breuer (1977), onde o objetivo era minimizar as *nets* cortadas por ambas as linhas, horizontal e vertical que particionavam a secção, o autor demonstra que a minimização dessas *nets* equivale a minimizar o comprimento das interconexões que delimitam a secção.

O algoritmo proposto por Breuer foi aperfeiçoado por Dunlop and Kernighan (1985) utilizando um método de propagação de terminal. O método considera no particionamento, não somente as *nets* internas, mas também as externas a secção, inserindo um terminal "*dummy*" na posição onde *net* externa conecta-se a outro subcircuito. O terminal "*dummy*" é considerado no processamento das partições, resultando em um melhor particionamento em termos de comprimento de interconexões.

3.2.1.2. Quadrisecção

A desvantagem dos métodos de particionamento é que, um bom resultado obtido no primeiro particionamento pode ser ruim para os particionamentos subseqüentes. Além disso, o posicionamento é essencialmente um problema de duas dimensões, enquanto que a solução recursiva de biparticionamento adota um método unidimensional. Suaris e Kedem (1987) utilizam um particionamento em quadrisecções, realizando um procedimento em duas dimensões.

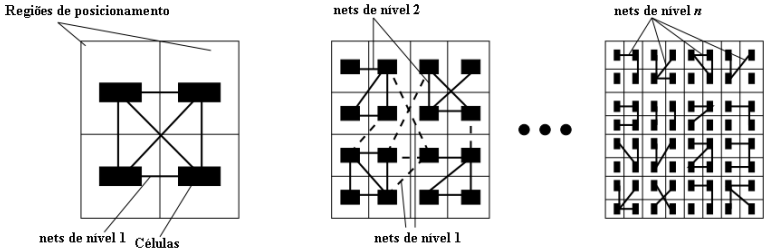

Figura 4: Particionamento em Quadrisecções.

3.2.2. Posicionamento baseado em Quadratura

Para o posicionamento do circuito, a *netlist* geralmente é modelada por um hipergrafo, com cada nodo representando uma célula e cada aresta representando uma *net*. O HPWL é utilizado como uma estimativa do comprimento necessário da interconexão para roteamento de duas células. Porém a equação do HPWL é difícil de ser otimizada matematicamente. No posicionamento em quadratura os quadrados do comprimento e largura do perímetro formado pelas células (denominado de *quadratic wirelength*) são minimizados.

Sendo uma *net* $\boldsymbol{e_{i,j}}$ que conecta duas células $\boldsymbol{i}$ e $\boldsymbol{j}$, de localização ($\boldsymbol{x_i}$, $\boldsymbol{y_i}$) e ($\boldsymbol{x_j}$, $\boldsymbol{y_j}$) respectivamente, o *linear wirelength* é dado por $L_e = | x_i - x_j | + | y_i - y_j |$, o *quadratic wirelength* por $Q_e = (x_i - x_j)^2 + (y_i - y_j)^2$, o *linear wirelength* total do leiaute é $WL = \sum_e L_e$ e o *quadratic wirelength* total é $WL^2 = \sum_e Q_e$.

Quando uma *net* é compartilhada por mais de duas células, ela deve ser transformada em múltiplas *nets* de duas células para que o algoritmo possa processa-lá. Tradicionalmente um modelo "*clique*" é utilizado e uma net de k conexões é transformada para uma em C_k^2 *nets*.

Para a otimização da função *quadratic wirelength*, frequentemente é utilizada uma representação em forma de matriz: Utilizando um grafo $\boldsymbol{G=G(V, E)}$, como descrito na seção 3.1, com um comprimento não negativo $\boldsymbol{\omega(e)}$ associado a cada aresta. Todas as *nets* podem ser representadas por uma matriz de adjacência $\boldsymbol{A=(a_{ij})}$ com uma entrada $\boldsymbol{a_{ij} = \omega}$ ($\boldsymbol{v_i}$, $\boldsymbol{v_i}$) se ($\boldsymbol{v_i}$, $\boldsymbol{v_j}$) $\in \boldsymbol{E}$ e $\boldsymbol{a_{ij}} = 0$ caso contrário. A posição de todos os vértices pode ser representada por um vetor *n*-dimensional $\boldsymbol{X=(x_i)}$ e $\boldsymbol{Y=(y_i)}$, sendo ($\boldsymbol{x_i}$, $\boldsymbol{y_i}$) as coordenadas do vértice $\boldsymbol{v_i}$. Para o *quadratic wirelength* as dimensões $\boldsymbol{x}$ e $\boldsymbol{y}$ são independentes, assim é possível otimizá-las separadamente em cada direção. No caso da dimensão $\boldsymbol{x}$, o *quadratic wirelength* total é dado por:

$$\Phi_Q(X) = \frac{1}{2} X^T Q X + d^T X = \sum_{i,j=1}^{n} a_{ij}(x_i - x_j)^2 + d^T X \quad (3.1)$$

Assumindo que existem $\boldsymbol{n}$ células na *netlist*, e que $\boldsymbol{Q}$ denota a matriz Hessiana do sistema, que é essencialmente uma matriz de conectividade $\boldsymbol{nxn}$ da *netlist*. $\boldsymbol{Q}$ é simétrica e positiva. O termo opcional $\boldsymbol{d^T X}$ representa a conexão de células para I/O *pads* fixos, e o vetor $\boldsymbol{d}$ pode também capturar *offsets* de pinos. O objetivo da função é minimizado solucionando o sistema linear dado por: $QX + d = 0$ (3.2)

3.2.3. Simulated Annealing

Simulated Annealing é uma técnica para otimização de problemas em geral, especialmente útil quando o problema não completamente conhecido. O algoritmo está baseado em uma analogia com a formação cristalina de alguns materiais. Quando o material é aquecido as moléculas próximas movimentan-se aleatoriamente trocando de posição, com a redução da temperatura os movimentos tendem a diminuir até cessar por completo alcançado uma estrutura cristalina.

Simulated anneleaning para posicionamento utiliza modificações aleatórias no estado atual de posicionamento, até que se atinja um estado aceitável. Conceitualmente define-se um grafo de configuração onde cada vértice corresponde a uma solução possível, e um arco direcionado ($\boldsymbol{v_i}$, $\boldsymbol{v_j}$) representa um possível movimento da solução $\boldsymbol{v_i}$

para v_j. O processo de *annealing* movimenta a solução de um vértice a outro seguindo os arcos direcionados do grafo de configuração. Independentemente da temperatura o algoritmo irá aceitar um movimento (v_i, v_j) se o custo $C(v_j) \leq C(v_i)$. Para evitar mínimos locais, o algoritmo aceita movimentos que aumentem o custo da solução quando a "temperatura é alta". A melhor solução entre todas as visitadas pelo processo é armazenada, e quando o processo termina espera-se que ter um número suficiente de soluções examinadas para se obter uma solução de baixo custo. Tipicamente o número de soluções possíveis é uma função exponencial do tamanho do problema. Assim, o movimento de uma solução para outra é restrito a um pequeno número do total de soluções possíveis.

3.2.4. Posicionamento Direcionado a Forças

A fim de minimizar o *wirelength*, células fortemente conectadas devem ser posicionadas próximas umas das outras, tanto quanto possível, como se houvesse uma força de atração entre as células. Deste modo o número de conexões entre duas células é utilizado para determinar a força de atração entre elas, o que pode ser modelado da seguinte forma: $F_{ij} = -c_{ij}d_{ij}$, onde c_{ij} é soma dos pesos das *nets* entre as duas células e d_{ij} é um vetor direcionado do centro de C_i até o centro de C_j. A magnitude desta força é, portanto, a distância (Euclideana ou de Manhattan) entre o centro das células. Por exemplo, na geometria Manhattan $|d_{ij}|=| x_i - x_j |+| y_i - y_j |$, onde ($x_i$, y_i) é o centro de C_i e (x_j, y_j) é o centro de C_j.

Algoritmos direcionados a força podem ser utilizados tanto na abordagem construtiva quanto na abordagem iterativa. No caso da abordagem construtiva o algoritmo direcionado a força realizam o posicionamento fazendo com que as células posicionadas mantenham um equilíbrio entre as forças que atuam sobre elas, o que é encontrado quando o vetor de somas das forças atuando em cada célula seja zero. Solução que pode ser obtido através de sistema não linear de equações. No caso da abordagem iterativa o algoritmo direcionado a força pode melhorar um posicionamento existente através da redução da força de atração entre as células.

4. Posicionadores Estado da Arte

Nesta seção são apresentados os posicionadores que serão utilizados no projeto. Essa descrição não tem a intenção de detalhar todas as características e forma de implementação dos posicionadores, mas apenas relatar quais e como as técnicas são utilizadas, uma descrição mais precisa pode ser encontrada em suas respectivas referências.

4.1 Capo

O Capo, Roy et. al. (2005), realiza um posicionamento baseado em particionamento *min-cut* em bisecções. Durante o particionamento o Capo utiliza diferentes formas de distribuir as células nas secções, podendo distribuir uniformemente as células ou de forma desigual, tentando ou reduzir o *wirelength* no caso da distribuição uniforme ou reduzir o congestionamento no caso da distribuição não uniforme Nam et. al. (2007). Para o posicionamento detalhado o Capo utiliza diferentes técnicas para reduzir o HPWL (*Half-Perimeter WireLength*), as técnicas utilizadas incluem um mecanismo de *sliding* com um otimizador *RowIroning* (baseado em um mecanismo de *branch-and-bound*) e um algoritmo guloso que realiza movimentos nas células de regiões com violação de densidade.

4.2 Dragon

O Dragon, proposto em Wang, Yang and Sarrafzadeh (2000), realiza o posicionamento de forma hierárquica utilizando um posicionamento baseado em particionamento, porém diferentemente do Capo o particionamento é em quadrisecções. O procedimento do Dragon é dividido em duas etapas: posicionamento global (GP) e posicionamento detalhado (DP). Na etapa de posicionamento global, o circuito é sucessivamente particionado em quadrisecções, nessa etapa são permitidas sobreposições de células. A etapa seguinte (DP) utiliza esse resultado, corrigindo sobreposições de células e iterativamente aperfeiçoa-o utilizando heuristicas gulosas. Ainda na fase de posicionamento global, durante o particionamento, o Dragon utiliza *simulated annealing* para melhor o particionamento tentando evitar mínimos locais. A versão mais recente do Dragon, Taghavi, Yang, and Choi (2006), realiza um melhoria no particionamento, redimensionando secções para permitir o posicionamento de blocos.

4.3 NTUPlace3

O NTUplace3, proposto por Chen et. al. (2008), utiliza um particionamento multinível e o posicionamento consiste de três etapas: posicionamento global, que distribui uniformemente e encontrando a melhor posição para cada bloco; legalização, que remove sobreposições entre blocos e entre células; e posicionamento detalhado, onde a solução é refinada.

O posicionamento global é baseado em multinível e aplica uma técnica de *coarsening bottom-up* de dois estágios seguida de um *uncoarsening top-down*. A etapa de *coarsening* iterativamente "clusteriza" os blocos baseado em suas conectividades e tamanhos, até que o problema alcance um tamanho mínimo. Na etapa de *uncoarsening* os blocos são "desclusterizados" e para refinar o posicionamento é aplicado um modelo analítico para posicionamento global com uma função objetivo baseada em *log-sum-exp* como proposto em Naylor, Donelly and Sha (2001).

Na etapa de legalização são removidas as sobreposições utilizando um sistema de prioridades baseado no tamanho e posição das células, também foi implementado um mecanismo de *look-ahead* para facilitar o processo de legalização. A etapa de posicionamento detalhada é dividida em duas etapas, primeiramente são utilizados mecanismo de *swapping* e *matching* para reduzir o *wirelength*, logo após é aplicado um algoritmo de *sliding* para minimizar a densidade e o *overflow* em regiões de maior congestionamento.

4.4 mPL6

Como no NUTplace3 o mPL6 proposto por Chan et. al. (2006) realiza o posicionamento em três etapas, posicionamento global, legalização e posicionamento detalhado. O posicionamento global é realizado por um mecanismo multinível linear com um "*clustering*" baseado em uma aperfeiçoada heurística de "melhor escolha", uma redução de duas vezes no número de níveis em relação a versão anterior. A fase de legalização é dividida em duas etapas, com uma macro legalização baseado em um grafo discreto seguido de uma legalização baseada em um *scan* de tempo linear.

O posicionamento detalhado tem o objetivo de reduzir o *wirelength*, para isso é utilizada uma janela abrangendo uma ou mais linhas de células, todas as possíveis configurações de conexões de células são enumeradas, e após uma redução de

wirelength através de *swapping* de células a janela é reduzida da sua metade. O processo continua até que nenhuma otimização seja possível.

5. Técnicas de Otimização de Atraso e AIG

5.1 Técnicas de Otimização de Atraso

Diversas técnicas de otimização de atraso já foram propostas para a etapa de posicionamento ou para refinamento do posicionamento. Alguns trabalhos dedicam parte ou mesmo todo posicionamento detalhado a um algoritmo direcionado a otimização do atraso no circuito de alguma forma, seja diminuindo caminho críticos ou reduzindo o congestionamento. As principais técnicas são apresentadas a seguir.

5.1.1 Dimensionamento de Gate

Consiste basicamente na substituição de uma célula presente em um caminho crítico por uma célula com maior capacidade de corrente. Essa técnica foi utilizada com sucesso em Kannan, Suaris and Fang (1994), em conjunto com uma técnica de bufferização de interconexões alcançando reduções de 13 a 22% em relação a ferramentas comerciais. Porém essa abordagem causa grande impacto no posicionamento das células, pois a alteração da dimensão das células pode provocar o reposicionamento de grande parte do circuito além de alterar a área ocupada, fato que foi parcialmente tratado em LI, et. al., 2005, onde a solução proposta impede grandes perturbações no posicionamento.

5.1.2 Bufferização

Basicamente tem por objetivo reduzir a carga de saída em células sobrecarregadas, ou seja, células com um excessivo *fanout*, devido a um grande número de interconexões ou outras células conectadas a sua saída. Para isso insere buffers entre a saída da célula sobrecarregada e as células conectadas a ela.

5.1.3 Realocação

Com o circuito já posicionado e roteado, o atraso das interconexões é estimado considerando a exata topologia do circuito, e as células pertencentes a caminhos críticos são movidas com o objetivo de reduzir o atraso nesses caminhos. Porém essa movimentação não é simples, e pode resultar em novos caminhos críticos, para evitar esse problema, é utilizado um limite para o aumento do custo total das interconexões. Em Ajami and Pedram (2001), experimentos demonstraram ganhos entre 9% e 14% para uma aumento de 1 a 5% do circuito.

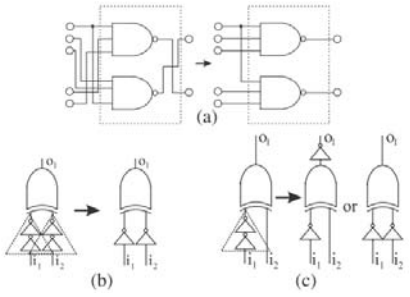

Figura 5: Exemplos de realocação.

5.1.4. Transformação de Sinais

Explora as possíveis transformações ou replicações de sinais para obter uma redução de atrasos no circuito. Em Chang, Markov and Bertacco (2007), espaços vazios são

utilizados para duplicação de sinais, reduzindo o atraso das interconexões duplicadas. Os resultados obtidos apresentaram em média uma redução média de 11% do atraso com uma aumento médio de 0,2% do *wirelength*.

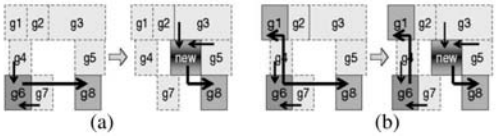

Figura 6: Redução de Atrasos com duplicação de sinais: em (a) uma nova célula substitui g_6, reduzindo a interconexão para g_8; em (b) a célula g_6 é mantida e fornece sinal para g_1 enquanto que g_6 fornece sinal para g_8.

5.1.5. Otimização de Caminhos não Monótonos

Caminhos não monótonos são caminhos percorridos pela interconexões em zigue-zague entre duas células, sua identificação pode permitir grandes ganhos em termos de *wirelength* e consequentemente redução de atrasos no circuito. Em Plaza, Markov and Bertacco (2008) o fluxo proposto de pós posicionamento primeiro identifica os caminhos críticos não monótonos, e utilizando uma simulação funcional realiza transformações lógicas sobre o circuito, avaliando os novos parâmetros obtidos. Os resultados obtidos nos experimentos realizados apresentaram uma redução média de aproximadamente 11% com um aumento médio de cerca de 2% na área do circuito.

5.2 AIG

O AIG consiste de rede booleana composta de duas entradas portas AND2 e inversores. Pode ser utilizado para manipular funções booleanas de larga escala em diversas aplicações como, por exemplo, verificação formal, síntese lógica e mapeamento tecnológico, além de apresentar a vantagem de ser menor e mais simples e mais de construir do que um BDD.

Graficamente, no AIG temos três tipos de vértices: vértices com apenas um terminal de valor “0” ou “1”, vértices sem entrada de arcos representando as entradas do circuito e vértices com duas entradas representando a função lógica AND2. Os arcos são direcionados e podem ter atributos inversores ou não, Kuehlmann et. al. (2002). A AIG pode ser criada por fatoração de soma de produtos e conversões das funções AND e OR através das regras de DeMorgan, Mishchenko et. al. (2006).

A dimensão e o tempo necessário para gerar o AIG é proporcional ao tamanho do circuito representado, Kuehlmann et. al. (2002). Além disso, uma AIG pode ser representada sem exceder três vezes o tamanho de um BDD correspondente, sendo que a computação booleana é mais eficaz em AIGs quando comparada a BDDs, Mishchenko et. al. (2004).

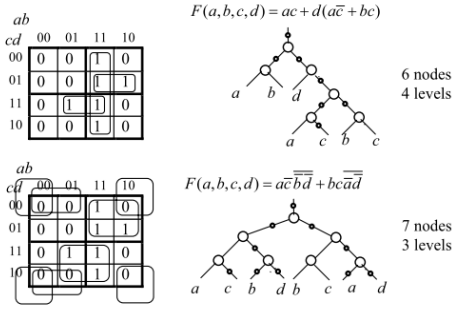

Figura 7: Representação de duas AIGs para a mesma função booleana.

A AIG apresenta a desvantagem de não ser uma representação canônica do circuito, mas apesar disso pode ser reduzida funcionalmente (FRAIG), desde que respeitada certas condições: $f_{n1}(x) \neq f_{n2}(x)$ e $f_{n1}(x) \neq \overline{f_{n2}(x)}$ para dois nodos quaisquer n_1 e n_2, como demonstrado em (MISHCHENKO, et. al., 2004).

6. Descrição da Ferramenta Proposta

O objetivo deste trabalho é criar uma nova ferramenta para otimização de interconexões em projetos VLSI, através do uso de uma representação em forma de AIG do circuito e, utilizando posicionadores estado da arte. O fluxo da ferramenta é apresentado na Figura 8, podendo ser o posicionador umas das seguintes opções: Capo10, Dragon2005, mPL6 ou NTUplace3. As entradas, saídas e limites da ferramenta são descritos na próxima seção.

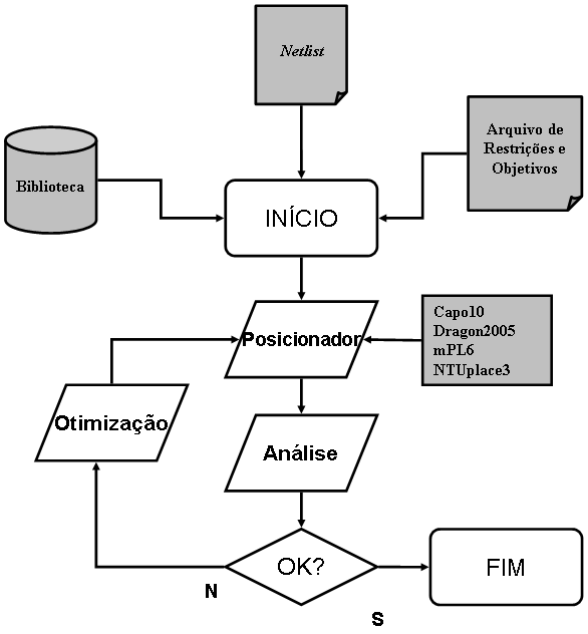

Figura 8: Fluxo de Execução da Ferramenta Proposta.

6.1. Entradas

Para que a ferramenta possa executar o posicionamento do AIG e a otimização das interconexões, são necessárias as seguintes informações:

- *Netlist* do circuito. Inicialmente a ferramenta irá trabalhar diretamente com o *netlist* do circuito, gerando internamente a AIG. A utilização de uma AIG gerada externamente deverá ser adicionada em uma etapa posterior;

- Biblioteca de Células: A ferramenta depende da biblioteca de células correspondente a tecnologia do circuito a ser posicionado. A partir da biblioteca a ferramenta extrai as informações das células para o posicionamento da AIG, e em uma etapa posterior, para o mapeamento tecnológico;

- Arquivo de Restrições e Objetivos: As restrições e objetivos para o circuito a ser posicionado devem ser fornecidas em formato SDC (*Synopsis® Design Constraints*). Dentre as restrições pode-se especificar uma área retangular para o posicionamento do circuito, como objetivo pode-se indicar um intervalo mínino de clock que o circuito deve ser capaz de alcançar.

6.2. Fluxo de Execução da Ferramenta

A Figura 8 apresenta o fluxo de execução da ferramenta. Tendo todas as informações necessárias a AIG é criada a partir da *netlist*, então com os dados obtidos da biblioteca de células é realizada uma estimativa de área mínima necessária para o posicionamento do AIG, caso essa área seja maior que a área especificada no arquivo de restrições a ferramenta determina uma nova área suficientemente grande para posicionar a AIG. O passo seguinte é posicionar a AIG utilizando um dos posicionadores referenciados na seção anterior, inicialmente todos os posicionadores serão utilizados para que possa se determinar qual obtém o melhor resultado posicionando a AIG. Após o posicionamento o circuito é analisado quanto aos requisitos de atraso e, caminhos críticos tem seus nodos duplicados na AIG para evitar caminhos não-monotônicos, o novo AIG é então enviado novamente para o posicionador, o processo é repetido então tantas vezes quanto necessário para que se obtenha os resultados especificados no arquivo de restrições e objetivos, ou até que se alcance um limite pré determinado de iterações.

Como resultado a ferramenta apresenta a AIG posicionada, resultado que posteriormente pode ser utilizado para mapeamento tecnológico diretamente sobre a AIG ou então como um parâmetro dos limites do circuito.

7. Conclusão

Este trabalho propõe uma nova ferramenta para otimização de interconexões em projetos VLSI, baseada em uma descrição do circuito na forma de uma AIG.

O uso de AIGs tem como objetivo facilitar a manipulação do circuito durante sua otimização, além de acelerar o posicionamento, em virtude da regularidade proporcionada por esta representação (composta apenas por portas AND2 e inversores). O resultado obtido é uma AIG, representação do circuito, posicionada que pode ser utilizada para um posterior mapeamento tecnológico. Desta forma, minimizamos o custo da etapa de posicionamento, que passa a ter apenas a tarefa de mapeamento tecnológico e validação do circuito, além de termos uma estrutura de fácil manipulação, a AIG, para otimização do circuito.

8. Trabalhos Futuros

As próximas etapas no desenvolvimento da ferramenta proposta neste trabalho incluem o mapeamento tecnológico sobre a AIG posicionada, resultado obtido até aqui, além da integração da ferramenta proposta como parte de um dos posicionadores utilizados, sendo escolhido aquele que apresentar melhor resultado e viabilidade para a integração.

9. Referências

Ajami, A. H. and Pedram, M. (2001) “Post-Layout Timing-Driven Cell Placement Using an Accurate Net Length Model with Movable Steiner Points” In: Proceedings of the 2001 Conference on Asia South Pacific Design Automation, p. 595-600, Yokohama, Japan.

Bakoglu, H. B. (1990) Circuits, Interconnections, and Packaging for VLSI, Addison-Wesley.

Breuer, M. A. (1977) “A Class of Min-cut Placement Algorithms”, In: Design Automation Conference IEEE/ACM, p. 284-290, 1977.

Chan, T. F., et. al. (2006) “mPL6: Enhanced Multilevel Mixed-Size Placement”, In: ISPD’06, Ab. 2006.

Chang, K.-H., Markov, I. L. and Bertacco, V. (2007) “Safe Delay Optimization for Physical Synthesis”, In: ASP-DAC’07, p. 628-633.

Chen, T.-C., et. al. (2008) “NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints” In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.27, No.7.

Dunlop, A. E. and Kernighan, B. W. (1985) “A Procedure for Placement of Standard Cell VLSI Circuits” In: IEEE Transactions on Computer Aided Design, Vol. 4, No. 1, p. 92-98, January.

Flynn, M. J., Hung, P. and Rudd, K. W. (2009) “Deep-Submicron Microprocessor Design Issues” In: IEEE Micro 1.

Gajski, D. D. and Kuhn, R. H. (1998) “New VLSI Tools”, In: IEEE Computer, Vol. 16, No. 12, p. 11-14.

Gerez, S. H. (1998) “Algorithms for VLSI Design Automation”, Baffins Lane: John Wiley & Sins Ltd..

Goto, S. and Kuh, E. S. (1978) “An Approach to the Two-Dimensional Placement Problem in Circuit Layout” In: IEEE Transactions on Circuits and Systems, Vol. 25, No. 3, p. 208-214.

Kannan, L. N., Suaris, P. R. and Fang, H. (1994) “A Methodology and Algorithms for Post-placement Delay Optimization”, In: Annal ACM IEEE Design Automation Conference, p. 327-332.

Kuehlmann, A. et. al. (2002) “Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification”, In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 21, No. 12, p. 1377-1394.

Mishchenko, A., et. al. (2004) “FRAIGs: Functionally Reduced AND-INV Graphs”, In: International Conference on Computer Aided Design.

Mishchenko, A., et. al. (2006) “Improvements to Combinational Equivalence Checking”, In: International Conference on Computer Aided Design, p. 836-843.

Nam, G. J. and Cong, J. (2007) “Modern Circuit Placement”, New York: Springer Science.

Nam, G. J. (2007) ISPD 2005/2006 Placement Contest Updates. ISPD. [Online]. http://www.ispd.cc/slides/archives/ispd2007/slides07/placement-updates.pdf.

Naylor, W. C., Donelly, R. and Sha, L. (2001) “Non-linear optimization system and method for wire length and dealy optimization for an automatic electric circuit placer”, US patent 6,301,693.

Plaza, S. M., Markov, I. L. and Bertacco, V. (2008) “Optimizing Non-Monotonic Interconnect using Functional Simulation and Logic Restructuring”, In: ISPD’08, April.

Rabaey, J. M., Chandrakasan, A. P. and Nickolic, B. (2003), Digital Integrated Circuits. 2th ed. s.l., Paperback.

Roy, J. A., et. al. (2005) “Capo: Robust and Scalable Open-Source Min-Cut FloorPlacer” In: Proceedings of the 2005 International Symposium on Physical Desing, p. 224-226.

Suaris, P. R. and Keden, G.(1987) “Quadrisection: A New Approach to Standard Cell Layout”, In: International Conference on Computer Aided Design IEEE/ACM, p. 474-477, November.

Taghavi, T., Yang, X. and Choi, B.-K. (2006) “Dragon2005: Large-Scale Mixed-size Placement Tool”, In: ISPD’05, San Francisco,California, USA, April.

Theis, T. N. (2000) “The Future of Interconnection Technology”, In: IBM Journal of Reasearch and Development, Vol. 44, No. 3.

Vedovelli, E. (2009) “Otimização de Interconexões Através de Posicionamento e Síntese Lógica”, 45 f. Projeto de Diplomação (Graduação em Engenharia da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

Wang, M., Yang, X. and Sarrafzadeh, M. (2000) “Dragon2000: Fast standard-cell placement for large circuits”, In: International Conference on Computer-Aided Design, p. 260–263.

ANEXO B <ESPECIFICAÇÃO DO FORMATO BLIF>

BERKELEY LOGIC INTERCHANGE FORMAT (BLIF)

University of California
Berkeley

The goal of BLIF is to describe a logic-level hierarchical circuit in textual form. A circuit is an arbitrary combinational or sequential network of logic functions. A circuit can be viewed as a directed graph of combinational logic nodes and sequential logic elements. Each node has a two-level, single-output logic function associated with it. Each feedback loop must contain at least one latch. Each net (or signal) has only a single driver, and either the signal or the gate which drives the signal can be named without ambiguity.

In the following, angle-brackets surround nonterminals, and square-brackets surround optional constructs.

Models

A model is a flattened hierarchical circuit. A BLIF file can contain many models and references to models described in other BLIF files. A model is declared as follows:

decl-model-name is a string giving the name of the model.

decl-input-list is a white-space-separated list of strings (terminated by the end of the line) giving the formal input terminals for the model being declared. If this is the first or only model, then these signals can be identified as the primary inputs of the circuit. Multiple *.inputs* lines are allowed, and the lists of inputs are concatenated.

decl-output-list is a white-space-separated list of strings (terminated by the end of the line) giving the formal output terminals for the model being declared. If this is the first or only model, then these signals can be identified as the primary outputs of the circuit. Multiple *.outputs* lines are allowed, and the lists of outputs are concatenated.

decl-clock-list is a white-space-separated list of strings (terminated by the end of the line) giving the clocks for the model being declared. Multiple *.clock* lines are allowed, and the lists of clocks are concatenated.

command is one of:

<logic-gate>	<generic-latch>	<library-gate>
<model-reference>	<subfile-reference>	<fsm-description>
<clock-constraint>	<delay-constraint>	

Each *command* is described in the following sections.

The BLIF parser allows the *.model, .inputs, .outputs, .clock* and *.end* statements to be optional. If *.model* is not specified, the *decl-model-name* is assigned the name of the BLIF file being read. It is an error to use the same string for *decl-model-name* in more than one model. If *.inputs* is not specified, it can be inferred from the signals which are not the outputs of any other logic block. Similarly, *.outputs* can be inferred from the signals which are not the inputs to any other blocks. If any *.inputs* or *.outputs* are given, no inference is made; a node that is not an output and does not fanout produces a warning message.

If *.clock* is not specified (e.g., for purely combinational circuits) there are no clocks. *.end* is implied at end of file or upon encountering another *.model*.

Important: the first model encountered in the main BLIF file is the one returned to the user. The only *.clock, clock-constraint,* and *timing-constraint* constructs retained are the ones in the first model. All subsequent models can be incorporated into the first model using the *model-reference* construct.

Anywhere in the file a `#' (hash) begins a comment that extends to the end of the current line. Note that the character `#' cannot be used in any signal names. A `\' (backslash) as the last character of a non-comment line indicates concatenation of the subsequent line to the current line. No whitespace should follow the `\'.

Logic Gates

A *logic-gate* associates a logic function with a signal in the model, which can be used as an input to other logic functions. A *logic-gate* is declared as follows:

output is a string giving the name of the gate being defined.

in-1, in-2, ... in-n are strings giving the names of the inputs to the logic gate being defined.

single-output-cover is, formally, an n-input, 1-output PLA description of the logic function corresponding to the logic gate. {0, 1, -} is used in the n-bit wide ``input plane" and {0, 1} is used in the 1-bit wide ``output plane". The ON-set is specified with 1's in the ``output plane," and the OFF-set is specified with 0's in the ``output plane." The DC-set is specified for primary output nodes only, by using the construct *.exdc*.

A sample *logic-gate* with its *single-output-cover*:

In a given row of the *single-output-cover*, ``1" means the input is used in uncomplemented form, ``0" means the input is complemented, and ``-" means not used. Elements of a row are ANDed together, and then all rows are ORed.

As a result, if the last column (the ``output plane") of the *single-output-cover* is all 1's, the first n columns (the ``input plane") of the *single-output-cover* can be viewed as the truth table for the logic gate named by the string *output*. The order of the inputs in the *single-output-cover* is the same as the order of the strings *in-1, in-2, ..., in-n* in the *.names* line. A space between the columns of the ``input plane" and the ``output plane" is required.

External Don't Cares

External don't cares are specified as a separate network within a model, and are specified at the end of the model specification. Each external don't care function, which is specified by a *.names* construct, must be associated with a primary output of the main model and specified as a function of the primary inputs of the main model (hierarchical specification of external don't cares is currently not supported).

The external don't cares are specified as follows:

exdc indicates that the following *.names* constructs apply to the external don't care network.

output is a string giving the name of the primary output for which the conditions are don't cares.

in-1, in-2, ... in-n are strings giving the names of the primary inputs which the don't care conditions are expressed in terms of.

single-output-cover is an n-input, 1-output PLA description of the logic function corresponding to the don't care conditions for the output.

Flip flops and latches

A *generic-latch* is used to create a delay element in a model. It represents one bit of memory or state information. The *generic-latch* construct can be used to create any type of latch or flip-flop (see also the *library-gate* section). A *generic-latch* is declared as follows:

input is the data input to the latch.

output is the output of the latch.

type is one of {fe, re, ah, al, as}, which correspond to ``falling edge,'' ``rising edge,'' ``active high,'' ``active low,'' or ``asynchronous.''

control is the clocking signal for the latch. It can be a *.clock* of the model, the output of any function in the model, or the word ``NIL'' for no clock.

init-val is the initial state of the latch, which can be one of {0, 1, 2, 3}. ``2'' stands for ``don't care'' and ``3'' is ``unknown.'' Unspecified, it is assumed ``3.''

If a latch does not have a controlling clock specified, it is assumed that it is actually controlled by a single global clock. The behavior of this global clock may be interpreted differently by the various algorithms that may manipulate the model after the model has been read in. Therefore, the user should be aware of these varying interpretations if latches are specified with no controlling clocks.

Important: All feedback loops in a model must go through a *generic-latch*. Purely combinational-logic cycles are not allowed.

Library Gates

A *library-gate* creates an instance of a technology-dependent logic gate and associates it with a node that represents the output of the logic gate. The logic function of the gate and its known technology dependent delays, drives, etc. are stored with the *library-gate*. A *library-gate* is one of the following:

name is the name of the *.gate* or *.mlatch* to instantiate. A gate or latch with this name must be present in the current working library.

formal-actual-list is a mapping between the formal parameters of *name* (the terminals of the *library-gate*) and the actual parameters of the current model (any signals in this model). The format for a *formal-actual-list* is a white-space-separated sequence of assignment statements of the form:

All of the formal parameters of *name* must be specified in the *formal-actual-list* and the single output of *name* must be the last one in the list.

control is the clocking signal for the mlatch, which can be either a *.clock* of the model, the output of any function in the model, or the word ``NIL" for no clock.

init-val is the initial state of the mlatch, which can be one of {0, 1, 2, 3}. ``2" stands for ``don't care" and ``3" is ``unknown." Unspecified, it is assumed ``3."

A *.gate* refers to a two-level representation of an arbitrary input, single output gate in a library. A *.gate* appears under a technology-independent interpretation as if it were a single *logic-gate*.

A *.mlatch* refers to a latch (not necessarily a D flip flop) in a library. A *.mlatch* appears under a technology-independent interpretation as if it were a single *generic-latch* and possibly a single *logic-gate* feeding the data input of that *generic-latch*.

*.gate*s and *.mlatch*es are used to describe circuits that have been implemented using a specific library of standard logic functions and their technology-dependent properties. The library of *library-gate*s must be read in before a BLIF file containing *.gate* or *.mlatch* constructs is read in.

Model (subcircuit) references

A *model-reference* is used to insert the logic functions of one model into the body of another. It is defined as follows:

model-name is a string giving the name of the model being inserted. It need not be previously defined in this file, but should be defined somewhere in either this file, a *.search* file, or a master file that is *.search*ing this file. (see *.search* below)

formal-actual-list is a mapping between the formal terminals (the *decl-input-list, decl-output-list,* and *decl-clock-list*) of the called model *model-name* and the actual parameters of the current model. The actual parameters may be any signals in the current model. The format for a *formal-actual-list* is the same as its format in a *library-gate*.

A *.subckt* construct can be viewed as creating a copy of the logic functions of the called model *model-name*, including all of *model-name*'s *generic-latch*es, in the calling model. The hierarchical nature of the BLIF description of the model does not have to be preserved. Subcircuits can be nested, but cannot be self-referential or create a cyclic dependency.

Unlike a *library-gate*, a *model-reference* is not limited to one output.

The formals need not be specified in the same order as they are defined in the *decl-input-list, decl-output-list,* or *decl-clock-list*; elements of the lists can be intermingled in any order, provided the names are given correctly. Warning messages are printed if elements of the *decl-input-list* or *decl-clock-list* are not driven by an actual parameter or if elements of the *decl-output-list* do not fan out to an actual parameter. Elements of the

decl-clock-list and *decl-input-list* may be driven by any logic function of the calling model.

Subfile References

A *subfile-reference* is:

file-name gives the name of the file to search.

A *subfile-reference* directs the BLIF reader to read in and define all the models in file *file-name*. A *subfile-reference* does not have to be inside of a *.model*. *subfile-reference*s can be nested.

Search files would usually be used to hold all the subcircuits referred to in *model-reference*s, while the master file merely searches all the subfiles and instantiates all the subcircuits it needs.

A *subfile-reference* is not equivalent to including the body of subfile *file-name* in the current file. It does not patch fragments of BLIF into the current file; it pauses reading the current file, reads *file-name* as an independent, self-contained file, then returns to reading the current file.

The first *.model* in the master file is always the one returned to the user, regardless of any *subfile-reference*s than may precede it.

Finite State Machine Descriptions

A sequential circuit can be specified in BLIF logic form, as a finite state machine, or both. An *fsm-description* is used to insert a finite state machine description of the current model. It is intended to represent the same sequential circuit as the current model (which contains logic), but in FSM form. The format of an *fsm-description* is:

num-inputs is the number of inputs to the FSM, which should agree with the number of inputs in the *.inputs* construct for the current model.

num-outputs is the number of outputs of the FSM, which should agree with the number of outputs in the *.outputs* construct for the current model.

num-terms is the number of ``<input> <current-state> <next-state> <output>" 4-tuples that follow in the FSM description.

num-states is the number of distinct states that appear in ``<current-state>" and ``<next-state>" columns.

reset-state is the symbolic name for the reset state for the FSM; it should appear somewhere in the ``<current-state>" column.

input is a sequence of *num-inputs* members of {0, 1, -}.

output is a sequence of *num-outputs* members of {0, 1, -}.

current-state and *next-state* are symbolic names for the current state and next state transitions of the FSM.

latch-order-list is a white-space-separated sequence of latch outputs.

code-mapping is newline separated sequence of:

num-terms and *num-states* do not have to be specified. If the *reset-state* is not given, it is assigned to be the first state encountered in the ``<current-state>" column.

The ordering of the bits in the *input* and *output* fields will be the same as the ordering of the variables in the *.inputs* and *.outputs* constructs if both an *fsm-description* and logic functions are given.

latch-order-list and *code-mapping* are meant to be used when both an *fsm-description* and a logical description of the model are given. The two constructs together provide a correspondence between the latches in the logical description and the state variables in the *fsm-description*. In a *code-mapping*, *symbolic-name* consists of a symbolic name from the ``<current-state>'' or ``<next-state>'' columns, and *encoded-name* is the pattern of bits ({0, 1}) that represent the state encoding for *symbolic-name*. The *code-mapping* should only be given if both an *fsm-description* and logic functions are given. *.latch-order* establishes a mapping between the bits of the *encoded-name*s of the *code-mapping* construct and the latches of the network. The order of the bits in the encoded names will be the same as the order of the latch outputs in the *latch-order-list*. There should be the same number of bits in the *encoded-name* as there are latches if both an *fsm-description* and a logical description are specified.

If both *logic-gate*s and an *fsm-description* of the model are given, the *logic-gate* description of the model should be consistent with the *fsm-description*, that is, they should describe the same circuit. If they are not consistent there will be no sensible way to interpret the model, which should then cause an error to be returned.

If only the *fsm-description* of the network is given, it may be run through a state assignment routine and given a logic implementation. A sole *fsm-description*, having no logic implementation, cannot be inserted into another model by a *model-reference*; the state assigned network, or a network containing both *logic-gate*s and an *fsm-description* can.

Clock Constraints

A *clock-constraint* is used to set up the behavior of the simulated clocks, and to specify how clock events (rising or falling edges) occur relative to one another. A *clock-constraint* is one or more of the following:

cycle-time is a floating point number giving the clock cycle time for the model. It is a unitless number that is to be interpreted by the user.

event-percent is a floating point number representing a percentage of the clock cycle time at which a specific *.clock_event* occurs. Fifty percent is written as ``50.0.''

event-1 through *event-n* are one of the following:

where *rise-fall* is either ``r'' or ``f'' and stands for the rising or falling edge of the clock and *clock-name* is a clock from the *.clock* construct. The apostrophe between *rise-fall* and *clock-name* is a seperator, and serves no purpose in and of itself.

before and *after* are floating point numbers in the same ``units'' as the *cycle-time* and are used to define the ``skew'' in the clock edges. *before* represents maximum amount of time before the nominal time that the edge can arrive; *after* represents the maximum amount of time after the nominal time that the edge can arrive. The nominal time is *event-percent*% of the *cycle-time*. In the unparenthesized form for the *clock-event*, *before* and *after* are assumed ``0.0.''

All events, *event-1 ... event-n*, specified in a single *.clock_event* are to be linked together. A routine changing any one edge should also modify the occurrence time of all the related clock edges.

Delay Constraints

A *delay-constraint* is used to specify parameters to more accurately compute the amount of time signals take to propagate from one point to another in a model. A *delay-constraint* is one or more of :

rise, *fall*, *drive*, and *load* are all floating point numbers giving the rise time, fall time, input drive, and output load.

in-name is a primary input and *out-name* is a primary output.

before-after can be one of {b, a}, corresponding to ``before'' or ``after,'' and *event* has the same format as the unparenthesized form of *event-1* in a *clock-constraint*.

.area sets the area of the model to be *area*.

.delay sets the delay for input *in-name*. *phase* is one of ``INV,'' ``NONINV,'' or ``UNKNOWN'' for inverting, non-inverting, or neither. *max-load* is a floating point number for the maximum load. *brise, drise, bfall,* and *dfall* are floating point numbers giving the block rise, drive rise, block fall, and drive fall for *in-name*.

.wire_load_slope sets the wire load slope for the model.

.wire sets the wire loads for the model from the list of floating point numbers in the *wire-load-list*.

.input_arrival sets the input arrival time for the input *in-name*. If the optional arguments are specified, then the input arrival time is relative to the *event*.

.output_required sets the output required time for the output *out-name*. If the optional arguments are specified, then the output required time is relative to the *event*.

.input_drive sets the input drive for the input *in-name*.

.max_input_load sets the maximum load that the input *in-name* can handle.

.output_load sets the output load for the output *out-name*.

.default_input_arrival, .default_output_required, .default_input_drive, .default_output_load set the corresponding default values for all the inputs/outputs whose values are not specifically set.

There is no actual unit for all the timing and load numbers. Special attention should be given when specifying and interpreting the values. The timing numbers are assumed to be in the same ``unit'' as the *cycle-time* in the *.cycle* construct.