UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ADRIEL MOTA ZIESEMER JUNIOR

Geração Automática de Partes Operativas de Circuitos VLSI

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Ricardo Augusto da Luz Reis
Orientador

Porto Alegre, setembro de 2007

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Ziesemer Junior, Adriel Mota

Geração Automática de Partes Operativas de Circuitos VLSI / Adriel Mota Ziesemer Junior. – Porto Alegre: PPGC da UFRGS, 2007.

90 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2007. Orientador: Ricardo Augusto da Luz Reis.

1. Geração automática. 2. Leiaute. 3. Parte operativa. 4. Células CMOS. 5. CAD. 6. Microeletrônica. I. Reis, Ricardo Augusto da Luz. II. Título.

“Só é útil o conhecimento que nos torna melhores”
— SÓCRATES

AGRADECIMENTOS

Gostaria de começar agradecendo primeiramente à minha família. Sou grato aos meus pais **Adriel** e **Nair** por todo apoio que sempre me deram, pela ótima educação, carinho, enfim, por tudo o que sou hoje. Agradeço à minha linda esposa **Angelina** por todo o seu amor, dedicação e paciência que tem tido em aceitar me dividir estes últimos dias com o computador no qual passo a maior parte do tempo escrevendo esta dissertação. Às minhas irmãs **Adriana** e **Luciana** cujos feitos alcançados com seus estudos serviram como referência durante boa parte da minha formação.

Dando prosseguimento, gostaria de agradecer a todas as pessoas que direta ou indiretamente tiveram alguma contribuição para a minha formação e que sem elas talvez eu não tivesse chegado aqui. Sou grato ao meu orientador **Ricardo Reis** pela oportunidade que me foi dada, por todas as suas contribuições técnicas ao longo deste trabalho e incentivos, tanto acadêmico quanto financeiro, que me permitiram em duas oportunidades participar de conferências no exterior. A todos os mestres que tive ao longo deste período cujos ensinamentos foram essenciais para o meu crescimento acadêmico. Outras pessoas que merecem um agradecimento especial são os meus colegas de laboratório que estiveram literalmente sempre ao meu lado e que me deram todo o apoio desde o início do mestrado. Agradeço aos meus amigos **Lucas Brusamarelo**, **Glauco Santos** e **Cristina Meinhart**, por terem sido uns dos primeiros a me receber e ajudar na integração com o grupo de pesquisa. **Lucas** foi meu colega em diversas disciplinas e também meu vizinho mais próximo de bancada durante muito tempo do mestrado. **Glauco**, atualmente o ancião da sala, sempre esteve disposto a ajudar oferecendo dicas, revisando papers e tirando dúvidas. **Cristina**, além de uma ótima amiga, foi parceira sempre fiel para uma partidinha de truco no final da tarde. Outros colegas que chegaram depois tiveram fundamental participação neste trabalho. **Cristiano Lazzari**, meu companheiro de Cadathlon e co-autor de diversos artigos, foi fonte inesgotável de dicas técnicas e autor de um conjunto maior ainda de ferramentas e algoritmos que foram muito bem utilizados. Meu amigo **Renato Hentschke**, um programador e pesquisador de primeira linha, capaz de trabalhar em diferentes projetos paralelamente e autor do famoso "sabe tudo", me ensinou muitas coisas no tempo em que trabalhamos juntos e foi companheiro de várias idas à Redenção durante os finais de semana em Porto Alegre.

Também devo ressaltar a importância de outros colegas: **Arnaldo Filho**, **Digeorgia Silva**, **Felipe Pinto**, **Guilherme Flach**, **Gustavo Neuberger**, **Gustavo Wilke**, **Lisane Brisolara**, **Rafael Zago**, **Sandro Sawicki**, **Thiago Assis** e **William Lautenschläger**. Os amigos são fundamentais para qualquer pessoa e foi muito bom ter vocês por perto durante este período.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ASIC	Circuito Integrado de Aplicação Específica (*Application Specific Integrated Circuit*)
CAD	Projeto Auxiliado por Computador (*Computer-Aided Design*)
CI	Circuito Integrado
CIF	Formato Intermediário Caltech (*Caltech Intermediate Format*)
CMOS	Semicondutor Metal-Óxido Complementar (*Complementary Metal-Oxide Semiconductor*)
DRC	Verificador de Regras de Projeto (*Design Rule Checker*)
DSP	Processamento Digital de Sinal (*Digital Signal Processing*)
E/S	Entrada e Saída
FIFO	Primeiro a entrar, primeiro a sair (*First In, First Out*)
FSM	Máquina de Estados Finitos (*Finite State Machine*)
GME	Grupo de Microeletrônica
GND	Terra ou suprimento de energia negativo (*Ground*)
ILP	Programação Linear com Inteiros (*Integer Linear Programming*)
LEF	Formato para Troca de Bibliotecas (*Library Exchange Format*)
MST	Menor Árvore de Expansão (*Minimum Spanning Tree*)
PLA	Array Lógico Programável (*Programmable Logic Array*)
PO	Parte Operativa
RTL	Nível de Transferência entre Registradores (*Register Transfer Level*)
SA	Resfriamento Simulado (*Simulated Annealing*)
SAT	(*Satisfiability*)
SPICE	Programa de simulação com ênfase em Circuitos Integrados (*Simulation Program with Integrated Circuit Emphasis*)
SOI	Silício Sobre Isolante (*Silicon On Insulator*)
TA	Aceitação por Limiar (*Threshold Accept*)
UFRGS	Universidade Federal do Rio Grande do Sul

VDD Suprimento de energia positivo

VHDL Linguagem de Descrição de Hardware VHSIC (*VHSIC Hardware Description Language*)

VHSIC Circuitos Integrados de Alto Desempenho (*Very High-Speed Integrated Circuit*)

VLSI Integração em Muito Larga Escala (*Very Large Scale Integration*)

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Tanto nos circuitos integrados para processamento de sinais digitais quanto em microprocessadores, a parte operativa é o núcleo onde a computação dos dados é realizada. A geração deste bloco costuma ser crítica para o desempenho global dos dispositivos. Ferramentas específicas para a geração de parte operativa costumam tirar proveito da regularidade estrutural do circuito para produzir leiautes mais densos e com melhor desempenho.

Este trabalho apresenta um novo fluxo de projeto para geração de parte operativa onde foi desenvolvido um gerador automático de leiaute de células CMOS com suporte à lógica não-complementar e um compilador de parte operativa. O uso destas duas ferramentas permite a rápida prototipação de uma biblioteca inteira de células lógicas otimizadas, para atender diferentes requisitos de desempenho, que em seguida são utilizadas para montagem de cada um dos blocos funcionais da parte operativa pelo compilador.

Comparações feitas com a ferramenta de síntese de células lógicas mostraram que a metodologia desenvolvida é capaz de produzir resultados similares em área e tempo de geração que métodos exatos e ainda possui a vantagem de suportar o uso de múltiplas métricas de qualidade durante o posicionamento dos transistores. As células geradas automaticamente apresentaram acréscimo de área médio de apenas 14% quando comparado às *standard-cells* e com resultado de atraso e consumo de potência muito próximos ou melhores. Circuitos de parte operativa foram gerados automaticamente pelo compilador e apresentaram na média, menor área, consumo de potência e atraso que circuitos gerados com um fluxo de síntese automático para *standard-cells*.

Palavras-chave: Geração automática, leiaute, parte operativa, células CMOS, CAD, microeletrônica.

Automatic Generation of Datapaths for VLSI Circuits

ABSTRACT

Datapath is the core where all the computations are performed in circuits for digital signal processing and also in microprocessors. The performance of the whole system is frequently determined by the implementation of the datapath. Tools dedicated for synthesis of this unit are called datapath compilers and use to take advantage on the structural regularity of the circuit to produce dense layouts and with good performance.

This work presents a new flow for datapath generation. An automatic cell synthesis tool with support to non-complementary logic is used in conjunction with a datapath compiler to achieve timing optimization and technology independence. The cell library produced as result of the synthesis process is used by the compiler to place the cells and generate each one of the datapath operators.

Comparisons with other cell sythesis tools shown that our approach was able to produce results comparable in area and generation time. Automatically generated cells were compared to standard-cell layouts and presented an average area overhead of just 14% while our circuits presented better or very close delay and power consumption. The datapaths produced by the compiler were compared to a traditional standard-cell based synthesis design flow and presented smaller area, delay and power consumption in average than this approach.

Keywords: automatic generation, layout , datapath, CMOS cells, CAD, microelectronic.

1 INTRODUÇÃO

Ferramentas de CAD são utilizadas em um número cada vez maior de áreas da computação com o objetivo de aumentar a produtividade. No projeto de circuitos VLSI, seu uso tem crescido em importância recentemente devido ao aumento da complexidade dos dispositivos e a necessidade de obter produtos que atendam ao *time-to-market*.

Devido à busca do mercado pela produção de dispositivos com um desempenho cada vez maior e capaz de efetuar um maior número de operações, fábricas de semicondutores se esforçam na tentativa de produzir circuitos integrados com transistores de tamanho cada vez menor, mais rápidos, e com menor consumo de potência. Graças às melhorias feitas no processo de fabricação dos semicondutores, atualmente já existem circuitos comerciais que contêm centenas de milhões de transistores dentro de um único *chip*, como indica a Figura 1.1. Por outro lado, cada melhoria no processo de fabricação, cria um novo conjunto de regras para projeto de circuitos integrados e o uso de ferramentas automáticas de CAD pode diminuir o tempo para refazer todo o leiaute.

1.1 Motivação

Parte operativa é o núcleo onde a computação dos dados é realizada tanto em processadores de sinais digitais como nos microprocessadores. O desempenho do sistema é freqüentemente determinado pelo projeto e implementação deste bloco.

O projeto de partes operativas em circuitos VLSI tem sido tradicionalmente feito à mão, uma vez que, a regularidade existente na topologia deste circuito, permite grande reaproveitamento no desenho dos leiautes. Com o aumento da complexidade de projeto dos circuitos de parte operativa, houve vários esforços na tentativa de produzir ferramentas para automatizar o processo de geração deste bloco (SUSIN, 1981; TALIERCIO; FOLETTO; LICCIARDI, 1991; AMMAR; GREINER, 1993). Compiladores de parte operativa são capazes de produzir leiautes igualmente densos, em um tempo de projeto muito menor do que se fossem desenhados à mão. Apesar disto, hoje em dia raramente a parte operativa tem sido desenvolvida como um bloco separado. Ao invés disto, cada vez mais um fluxo de otimização lógica, mapeamento, posicionamento e roteamento baseado em *standard-cells* tem sido usado para sua implementação, de forma conjunta com a parte de controle. Um motivo para isto é a simplicidade atual do uso de ferramentas que fazem síntese automática de ASICs a partir de uma descrição RTL e o seu alto grau de maturidade que permite a obtenção de resultados aceitáveis para uma gama de circuitos em um curto tempo de projeto. Outra explicação é a falta de integração dos compiladores de parte operativa com técnicas de otimização temporal uma vez que a maior parte dos compiladores tem por objetivo apenas a exploração da regularidade para gerar leiautes com área reduzida e ignora completamente a questão do atraso e dimensionamento dos

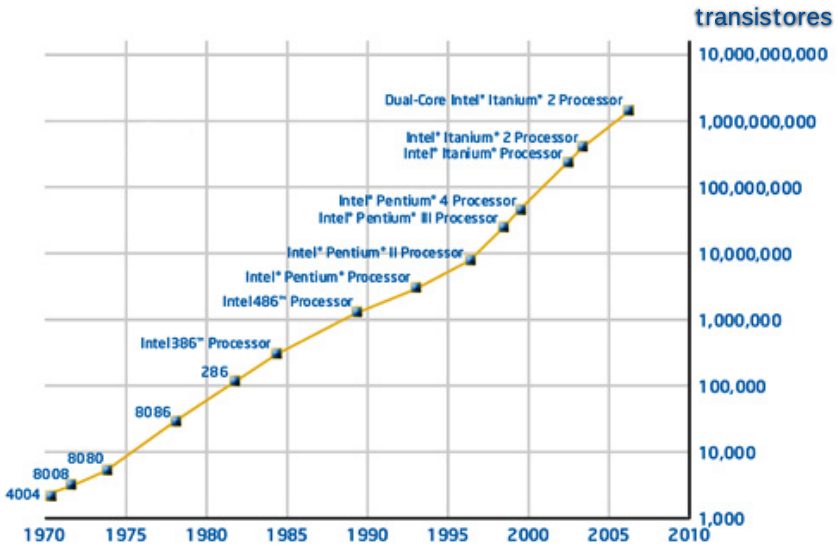


Figura 1.1: Aumento do número de transistores nos processadores Intel ao longo dos anos

transistores.

Neste trabalho foi atacado o problema de geração de partes operativas utilizando uma outra abordagem. Para associar otimização temporal com a geração de circuitos de parte operativa foram desenvolvidas duas ferramentas: um gerador automático de leiautes de células CMOS e um montador de parte operativa. O uso de bibliotecas de células pré-caracterizadas limita a sua portabilidade para diferentes tecnologias de fabricação e restringe a aplicação de otimizações para redução de atraso, potência e área devido à quantidade limitada de células em uma biblioteca. Com um gerador de células é possível criar rapidamente leiautes de células com diferentes redes e dimensionamento de transistores para diferentes tecnologias de fabricação. Isto permite que otimizações utilizando ferramentas para dimensionamento individual do *gate* dos transistores como a de Santos (SANTOS et al., 2003) possam ser utilizadas para realizar otimização temporal no caminho crítico do circuito.

Juntamente com o gerador de células, foi desenvolvido também um compilador de parte operativa. Esta ferramenta é responsável por instanciar, posicionar e conectar automaticamente as células que compõem cada bloco funcional que forma o leiaute da parte operativa. O uso de montadores especializados em conjunto com um gerador de leiautes de células permite que células lógicas mais adequadas possam ser instanciadas para implementar uma determinada funcionalidade quando comparado a um fluxo *standard-cell* onde células super-dimensionadas podem ser usadas no caso da inexistência de uma célula mais adequada. Isto pode causar um aumento no número de transistores do circuito e por conseqüência um aumento do seu consumo de energia.

1.2 Objetivos

O objetivo deste trabalho foi desenvolver um conjunto de ferramentas para geração automática de leiautes de partes operativas.

A estratégia utilizada suporta técnicas para otimização lógica e temporal através do uso de um gerador automático de leiautes de células lógicas em conjunto com um monta-

dor de parte operativa. O gerador aplica técnicas que permitem gerar células com lógica não-complementar e com um dimensionamento individual dos seus transistores. O montador permite a especificação dos operadores da parte operativa que devem ser implementados e também das células que devem ser utilizadas em cada um dos módulos.

1.3 Organização Deste Trabalho

Este trabalho está organizado da seguinte maneira:

O Capítulo 2 faz uma revisão sobre as abordagens utilizadas pelas diferentes ferramentas de geração de parte operativa existentes. Uma rápida revisão das técnicas utilizada nestes trabalhos é fornecida.

O Capítulo 3 faz um levantamento bibliográfico de ferramentas para geração automática de células CMOS. As diferentes etapas existentes em um fluxo de geração de células são explicadas e as técnicas existentes para cada uma das etapas são detalhadas.

A ferramenta para geração automática de leiautes de células desenvolvida neste trabalho é apresentada no Capítulo 4. O fluxo da ferramenta, bem como cada um dos algoritmos utilizados, são descritos.

O Capítulo 5 apresenta o montador de parte operativa desenvolvido neste trabalho. Seu funcionamento e recursos atualmente implementados são detalhados e exemplificados.

Resultados experimentais são reportados no Capítulo 6 para ambas as ferramentas. Uma comparação com leiautes obtidos por outras ferramentas conhecidas e com células *standard-cells* é realizada e os resultados obtidos discutidos.

Por fim, o Capítulo 7 apresenta as conclusões e as perspectivas de trabalhos futuros.

2 PARTE OPERATIVA

Este capítulo faz uma breve revisão sobre partes operativas e mostra algumas técnicas encontradas na literatura relacionadas a geração automática deste bloco.

2.1 Introdução

A maioria dos processadores encontrados tanto em computadores, controladores, quanto nas placas gráficas, podem ser divididos em parte operativa, controle, memória e entrada/saída (PATTERSON; HENNESSY, 1998) conforme mostra a Figura 2.1.

O controle é um circuito seqüencial (FSM) que pode ser implementado utilizando o modelo de máquina de estados de Moore ou de Mealy (GAJSKI, 1997) com lógica aleatória, PLAs ou memórias. Este bloco provê sinais de controle para todos os demais blocos e determina o que deve acontecer a cada ciclo de relógio.

Partes operativas (PO) são estruturas de múltiplos bits usadas para processamento digital de dados. Estruturalmente, uma PO normalmente é constituída de unidades de lógica e aritmética (ULAs), multiplexadores, registradores e barramentos que juntos são responsáveis pela realização da computação dos dados.

2.2 Abordagem geral

Ao contrário da parte de controle, a PO costuma ter um alto grau de regularidade na sua estrutura. A exploração desta regularidade permite aos projetistas construírem

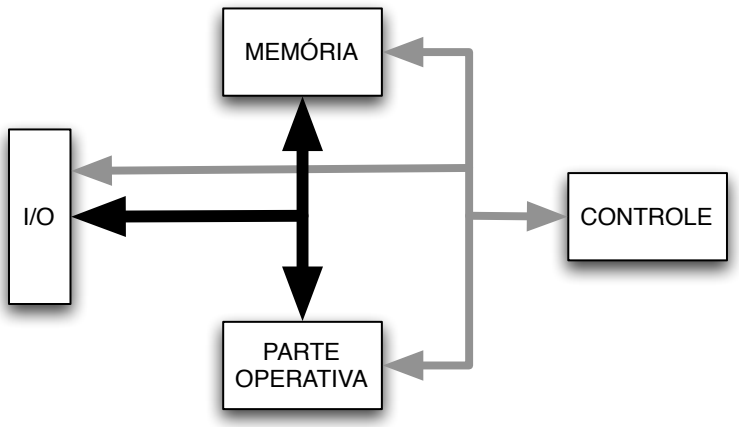


Figura 2.1: Unidades funcionais básicas de um processador

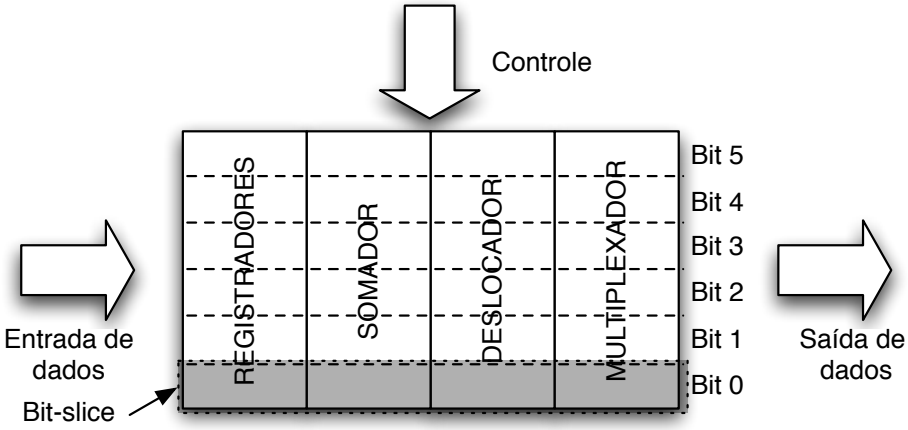


Figura 2.2: Exemplo de uma estrutura *bit-slice*

leiautes de PO de forma eficiente porque o fluxo de dados provê uma indicação óbvia para o posicionamento (RABAEY, 1996). A identificação desta regularidade também permite uma redução no esforço de projeto necessário para fazer a síntese e geração do leiaute dos blocos funcionais da PO, o que é traduzido em um menor tempo de projeto.

Tradicionalmente, POs são projetadas com o uso de estruturas chamadas de *bit-slice*. Estas estruturas exploram a regularidade existente ao longo dos diferentes *bits* da PO para gerar um leiaute compacto com conexões curtas e previsíveis. Em uma situação ideal, uma PO de n-*bits* é construída repetindo uma mesma *bit-slice* n vezes como mostra a Figura 2.2. As células dentro das *bit-slice* normalmente possuem todas a mesma altura, e são posicionadas de forma que os dados fluam em um sentido enquanto os sinais de controle cruzam ortogonalmente a eles.

Entretanto, nem todas as PO são compostas exclusivamente por blocos regulares. Circuitos reais apresentam blocos de largura diferente e/ou com lógica aleatória, que ferramentas de síntese de POs devem estar preparadas para lidar (AMMAR; GREINER, 1993).

Blocos funcionais de POs podem ser feitos com o posicionamento de módulos inteiros de n-bits otimizados (nos casos onde não for possível encontrar regularidade. Ex.: somadores/multiplicadores de alta performance) ou por módulos compostos pelo agrupamento de n células de 1 bit idênticos (nor, registrador, multiplexador, etc.). Os blocos funcionais costumam ser posicionados linearmente, em uma dimensão, enquanto células vizinhas (de *bits* diferentes) pertencentes ao mesmo bloco podem ter suas interconexões feitas por justaposição para facilitar o roteamento nas camadas superiores de metal. Estas células também freqüentemente são espelhadas no eixo horizontal para que possam compartilhar as trilhas de alimentação e o poço (que possui regras bastante restritivas).

Um roteamento sobre as células (*over-cells*) normalmente é utilizado para completar as interconexões do circuito sem precisar da inserção de espaços extras. Entretanto, quando utilizando tecnologias com número limitado de camadas de metais, muitas vezes o roteamento não pode ser completado e canais de roteamento fora da área das células são inseridos para realização de algumas conexões, especialmente entre células de diferentes *bits*. Barramentos são comumente utilizados para conectar as E/S dos operadores das unidades de lógica e aritmética aos elementos de memória da PO. Seu uso permite uma redução significativa do número de conexões internas quando comparado a outros méto-

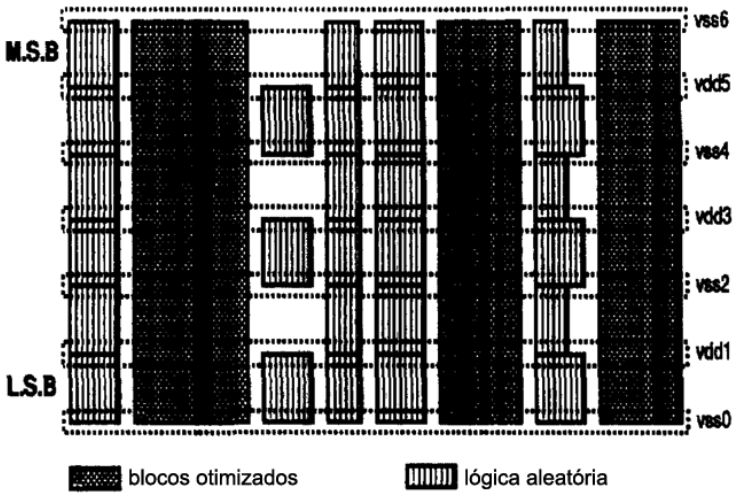


Figura 2.3: Parte operativa misturando blocos otimizados com blocos de lógica aleatória

dos para seleção de sinal. Por outro lado, o uso de multiplexadores provê uma solução mais simples e amigável à ferramentas de síntese (JAMIER, 1986).

2.3 Compiladores de Partes Operativas

Nesta seção é feita uma breve revisão sobre trabalhos anteriores de compiladores de partes operativas ou diretamente relacionados. As metodologias utilizadas em cada um destes trabalhos são apresentadas.

2.3.1 Um Compilador de Parte Operativa de Alta Densidade Misturando Lógica Aleatória e Blocos Otimizados

Em (AMMAR; GREINER, 1993), foi desenvolvida uma ferramenta capaz de utilizar blocos de lógica aleatória e blocos otimizados na mesma estrutura *bit-slice*, como ilustra a Figura 2.3, graças a uma biblioteca de células específica para PO. O leiaute das células é produzido de forma independente de tecnologia com o uso de uma ferramenta de conversão a partir de um leiaute simbólico. O compilador mantém uma biblioteca inteira de leiautes simbólicos de células lógicas e realiza a conversão para o leiaute da tecnologia utilizada de forma automática. A geração dos blocos otimizados tais como somadores, bancos de registradores e deslocadores (*barrel shifters*) é feita chamando os correspondentes módulos geradores com os parâmetros desejados. O posicionamento das células é realizado de forma semi-automática pelo compilador de PO. Um arquivo com a descrição estrutural simbólica do circuito fornece um posicionamento relativo das células dentro de cada bloco funcional. O compilador então calcula o ordenamento dos blocos funcionais automaticamente de forma a reduzir uma função de custo que leva em consideração o comprimento das conexões e o estouro da capacidade das trilhas de roteamento.

Outra característica deste compilador é o uso do conceito de terminais virtuais que são diferentes terminais de E/S localizados em diferentes posições de trilhas dentro das células e que são eletricamente equivalentes. Isto é feito para facilitar o acesso destes sinais às trilhas de roteamento e permitir que POs de maior complexidade possam ser produzidas.

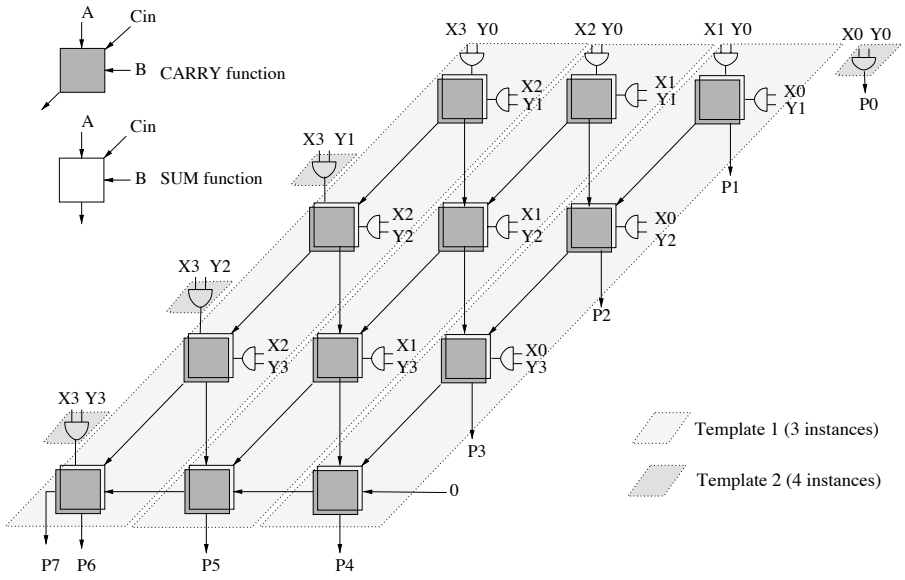


Figura 2.4: Extração de regularidade em um multiplicador 4x4

2.3.2 Uma Abordagem Geral para Extração da Regularidade em Circuitos de Parte Operativa

Na literatura existem diversos trabalhos que tentam extrair a regularidade da PO a partir de descrições provindas de diferentes níveis de abstração. Um destes trabalhos é (CHOWDHARY et al., 1998) onde são identificados automaticamente *templates* dentro de um bloco funcional descrito em HDL. O algoritmo monta um grafo com a estrutura do circuito e procura por sub-grafos com lógica idêntica para formar *templates* que possuam múltiplas instâncias e que juntos cubram todo o circuito.

Na Figura 2.4 são mostrados os *templates* encontrados pelo algoritmo no circuito de um multiplicador 4x4. Os *templates* encontrados são utilizados para formar uma hierarquia e facilitar a criação de *bit-slices* ou macro-células em POs de maior tamanho. Isto pode levar a realizações mais eficientes de leiautes com uma significante redução do esforço de projeto.

2.3.3 Desafios no desenvolvimento de CAD para projeto de Parte Operativa

Segundo um artigo publicado pela Intel (CHAN et al., 1999), POs de circuitos VLSI de alta performance, incluindo microprocessadores da Intel, costumam ser implementadas com o uso de estruturas *bit-slice* para manipulação simultânea de múltiplos bits de dados. O circuito e leiaute destas estruturas é mantido o mesmo em sua maior parte, em cada *bit-slice*, para conseguir a máxima performance, maior produtividade e maior densidade de transistores. Neste trabalho, foi feita uma revisão sobre as metodologias tradicionais na época para geração de POs e por fim foi proposto um novo fluxo de projeto automatizado de RTL até leiaute para suportar tecnologias de processo (fabricação de *chips*) mais recentes. Este fluxo é mostrado na Figura 2.5.

Um detalhe importante que foi ressaltado neste artigo foi a importância da etapa de geração de células dentro do fluxo de geração de PO. Segundo este documento, bibliotecas de células lógicas podem também ser utilizadas - como na tradicional síntese de partes de controle - mas que a densidade do leiaute, neste caso, pode não ser tão boa quando comparada com leiautes feitos com geração de células, uma vez que esta abordagem

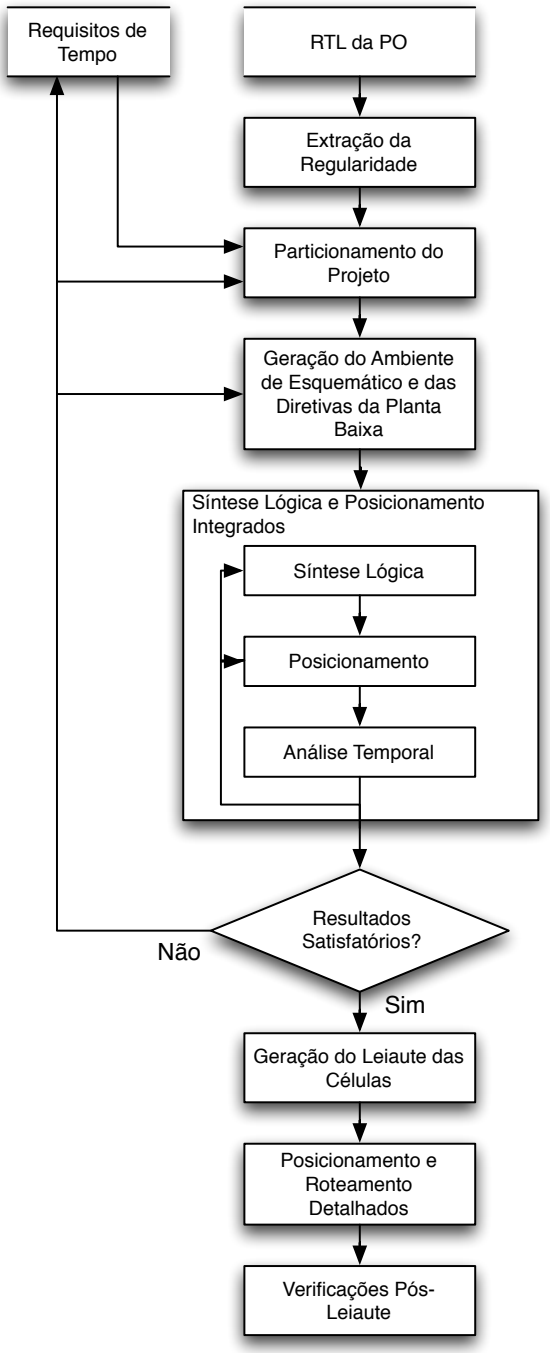


Figura 2.5: Fluxo automatizado para projeto de Partes Operativas proposto pela Intel em (CHAN et al., 1999)

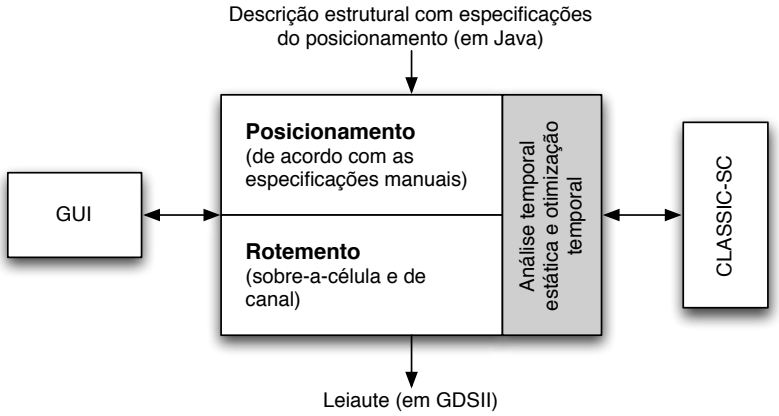


Figura 2.6: Fluxo de projeto do gerador de PO desenvolvido em (HU, 2000)

permite processar mais dispositivos em conjunto e tem a oportunidade de conseguir uma melhor otimização.

Outra questão que foi tratada neste trabalho foi quanto ao dimensionamento de transistores durante a etapa de síntese lógica. Segundo eles, após o mapeamento das células para uma determinada tecnologia, cada *gate* deve ser individualmente dimensionado para satisfazer os requerimentos de carga da saída. O dimensionamento dos *gates* é realizado iniciando pelas saídas primárias do circuito e atravessando o mesmo em direção às entradas primárias e que a carga de saída requerida esteja satisfeita para cada *gate* encontrado. Isto faz com que diferentes instâncias de uma mesma célula mapeada possam ser dimensionadas diferentemente dependendo da sua carga de saída no local onde for instanciada.

Por fim é informado que um gerador de leiautes de células chamado GeneSys (BASARAN et al., 1999) está sendo desenvolvido cuja principal característica é a possibilidade do projetista realizar intervenções manuais em qualquer etapa do fluxo de geração. Os resultados mostraram que com o uso desta ferramenta foi possível obter um ganho significativo de produtividade sobre o projeto manual para vários tipos de células, ao mesmo tempo em que atendia a todas as métricas de qualidade tais como densidade, confiabilidade, potência e atraso.

2.3.4 Um Compilador de Parte Operativa com Portabilidade Tecnológica

Em (HU, 2000), foi desenvolvido um compilador de PO com portabilidade tecnológica e suporte a inclusão de futuras extensões para análise temporal e otimizações. A ferramenta implementada utiliza o conceito de gerador de módulos que se caracteriza pelo posicionamento das células que compõem os blocos funcionais ser feito por algoritmos específicos para cada bloco, e que neste caso, foram descritos em classes da linguagem Java. Uma metodologia de roteamento mista *over-cell* (sobre as células) e de canal (para roteamento de redes entre diferentes *bits*) em 3 camadas de metal foi utilizada. O suporte a diferentes tecnologias é feito através da integração com a ferramenta comercial CLASSIC-SC da empresa Cadabra que é responsável pela geração automática do leiaute das células utilizadas pelo compilador. O fluxo de geração da ferramenta é mostrado na Figura 2.6. A etapa sombreada não foi abordada até o momento da publicação do trabalho, apenas o suporte para sua futura integração foi provido.

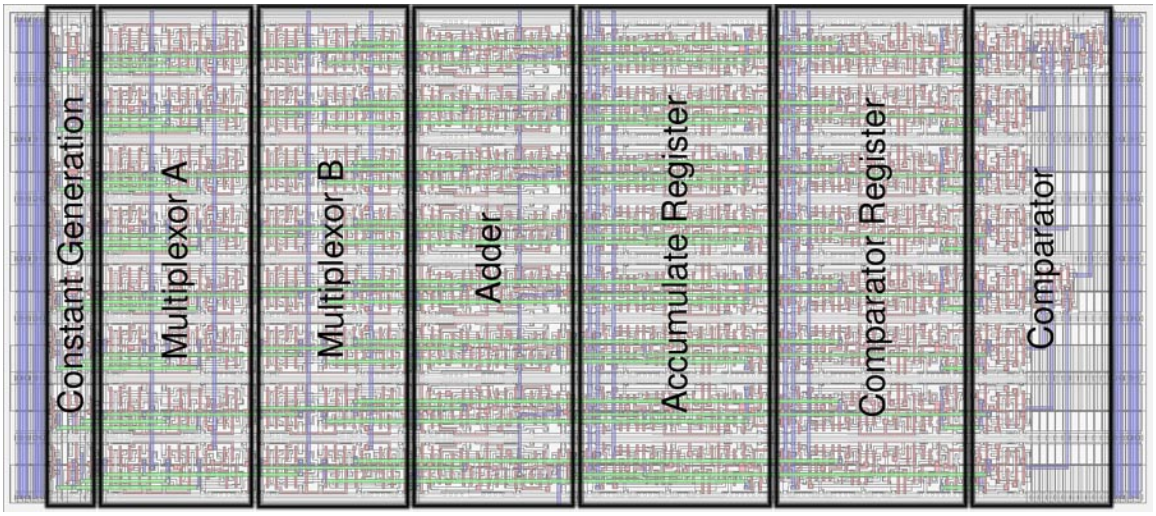


Figura 2.7: Leiaute da PO de um contador de 8 bits produzido automaticamente pela ferramenta Spongepaint em (BATTEN, 2007)

2.3.5 Spongepaint Datapath Generator

Neste trabalho foi desenvolvido um gerador de PO semi-automático chamado Spongepaint (BATTEN, 2007). O objetivo deste projeto foi o de construir um gerador flexível, capaz de suportar roteamento por barramentos e posicionamento de células-folha provindas de uma biblioteca. A automatização é feita por algoritmos especiais chamados *builders* que são responsáveis por realizar o posicionamento e o roteamento interno das células que compõem cada bloco funcional. Segundo o autor, esta técnica garante uma maior generalização destes algoritmos de forma que possam ser reutilizados em diversos projetos de POs. A Figura 2.7 mostra um exemplo de leiaute de PO gerado automaticamente pela ferramenta. Os operadores produzidos pelos *builders* aparecem em destaque.

Segundo o autor, novos *builders* podem ser facilmente incluídos para geração de outros operadores através da codificação do algoritmo de geração do operador diretamente no código fonte da ferramenta. Um *framework* (plataforma de software) foi desenvolvido para facilitar este processo.

A desvantagem da utilização deste método sozinho é que o suporte a outras tecnologias fica limitado à existência de uma biblioteca pronta com os leiautes das células que serão utilizadas no projeto da PO.

2.4 Conclusão

Uma pequena revisão dos métodos para geração de PO encontrados na literatura foi apresentada neste capítulo. Esta revisão teve o objetivo de descobrir como implementações estado-da-arte destes geradores abordam o problema e que técnicas são utilizadas.

Como resultado desta pesquisa, concluiu-se que a maior parte dos compiladores de parte operativa, tratam o problema de geração de células e montagem da PO de forma separada. Eles utilizam bibliotecas prontas de células, ou então geradas automaticamente com o uso de ferramentas de síntese de células lógicas, como entrada para o compilador de parte operativa. Algumas ferramentas especiais auxiliam o processo da escolha do melhor conjunto de células que deve integrar a biblioteca. O compilador então procura realizar o posicionamento das células da forma mais regular possível. Este posicionamento pode

ser feito com o uso de algoritmos que tentam extrair a regularidade da PO ou também por geradores de módulos que possuem no seu código a informação do posicionamento ideal para geração de cada bloco funcional. O ordenamento dos blocos funcionais ao longo da PO possui grande influência no grau de utilização das trilhas de roteamento. As duas abordagens encontradas nos compiladores de parte operativa pesquisados foi: utilizar um arquivo de descrição de PO com informação relativa do posicionamento destes blocos, ou então realizar um posicionamento automático destes blocos tentando minimizar uma função de custo. Por fim, as interconexões podem ser feitas de forma automática, com o uso de roteadores similares aos de um fluxo *standard-cell*, ou por algoritmos específicos para a geração de cada módulo.

A metodologia escolhida neste trabalho para geração automática de circuitos de PO foi utilizando um compilador de PO em conjunto com um gerador automático de leiautes de células CMOS para permitir a realização de otimizações elétricas nas células utilizadas pelo compilador e também suportar independência tecnológica. O suporte a geração de circuitos regulares (registrador, deslocador,etc.) é feito por algoritmos de posicionamento especializados e os demais (somador logarítmico, multiplicadores Wallace, raiz quadrada, etc.), através de posicionadores convencionais, com o objetivo de reduzir o comprimento das conexões.

3 GERAÇÃO AUTOMÁTICA DE LEIAUTE

Este capítulo aborda os principais métodos encontrados na literatura para geração de leiautes, com atenção especial à geração de células CMOS. Uma breve revisão dos trabalhos de síntese física anteriormente desenvolvidos no grupo de microeletrônica da UFRGS também é apresentada.

3.1 Introdução

A forma tradicional de projetar um circuito integrado requer que o projetista primeiramente defina uma biblioteca de células lógicas e um modelo comportamental da funcionalidade de um circuito integrado. Estas bibliotecas tipicamente incluem portas lógicas fundamentais como inversor, NOR, NAND, XOR, mas também podem ter elementos seqüenciais como latches e flip-flops. Freqüentemente também encontramos nestas bibliotecas diferentes versões de leiautes para a mesma célula, diferenciando-se apenas no dimensionamento dos seus transistores. Bibliotecas típicas costumam ter três de células para cada porta lógica: uma para menor área, uma para baixo consumo e uma para melhor desempenho. Isto é feito para que células diferentes, que implementam a mesma lógica, possam ser usadas atender diferentes requisitos de área, potência e atraso.

Normalmente, bibliotecas de células são feitas de forma manual por um projetista experiente. Por este processo ser extremamente demorado e suscetível a erros, diversos trabalhos foram feitos na tentativa de automatizar parcialmente, ou totalmente, este procedimento.

3.2 Geração de Leiaute Segundo a Metodologia TRANCA / UFRGS

Diversas ferramentas para síntese física de CIs vem sendo desenvolvidas na UFRGS, incluindo algumas específicas para geração de partes operativas (SUSIN, 1981; JAMIER, 1986; CARRO, 1989). Por outro lado, estes trabalhos já se encontram tecnologicamente desatualizados e não são capazes de suportar muito dos recursos oferecidos atualmente pelas *foundrys* para fabricação de circuitos integrados.

Lazzari (LAZZARI, 2003) fez um resumo dos trabalhos que haviam sido desenvolvidos até o ano de 2001 seguindo a metodologia TRANCA (REIS, 1987). Usando como base a linha do tempo publicada por ele, uma versão atualizada pode ser vista na Figura 3.1.

Em (WILKE et al., 2002) foi desenvolvida uma ferramenta para verificação temporal com base na simulação de vetores flutuantes e traço de caminhos para identificar o atraso crítico de circuitos combinacionais. Este trabalho, em conjunto com outros que foram

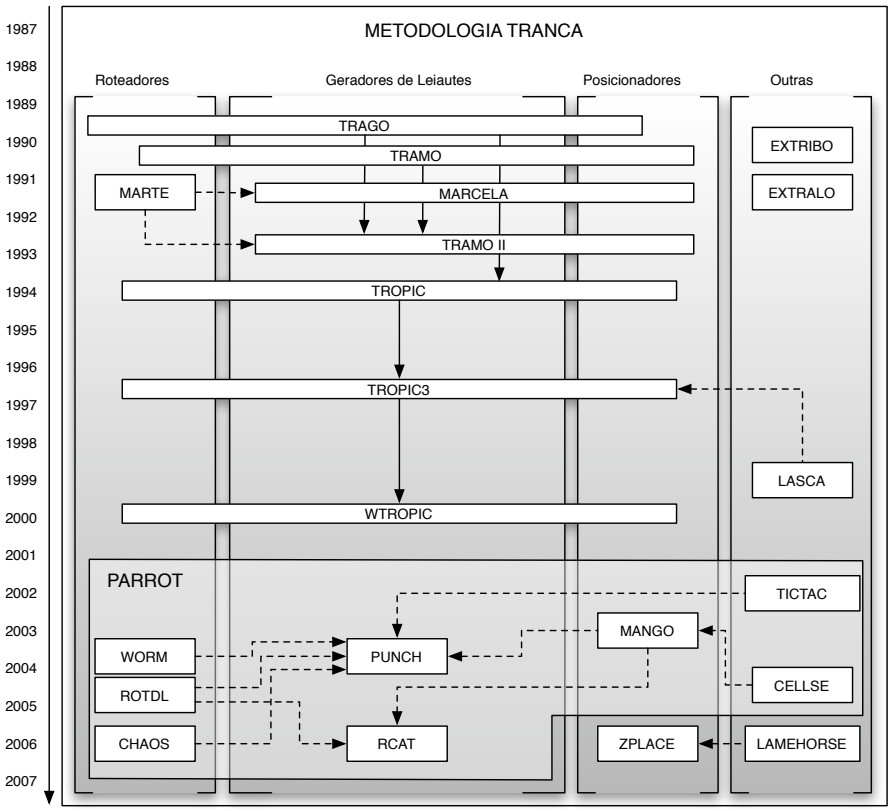


Figura 3.1: Linha do tempo das ferramentas de síntese física segundo a metodologia TRANCA

desenvolvidos sobre este tema receberam o nome de TICTAC.

Um posicionador chamado MANGO PARROT foi apresentado em (HENTSCHKE, 2002). Esta ferramenta produz como resultado um posicionamento relativo das células ao longo das bandas do circuito com o objetivo de reduzir o comprimento total das conexões. Entre as características desta ferramenta está a capacidade de realizar o posicionamento sem a existência prévia de uma biblioteca com o tamanho das células. A área ocupada por cada célula é estimada de acordo com a quantidade de transistores existentes e a tecnologia utilizada.

Em (LAZZARI, 2003), foi desenvolvida a ferramenta PARROT PUNCH com objetivo de aproveitar as ferramentas existentes na época e criar um fluxo de geração de leiaute com foco na redução de atraso e potência. A principal característica de PUNCH é a geração de leiautes *on-the-fly* sem a necessidade de uma biblioteca de células. O leiaute é gerado como se o circuito inteiro fosse uma célula gigante, sem hierarquia, como mostra a Figura 3.2.

Em conjunto com a ferramenta PUNCH, foi desenvolvido um roteador em malha (*maze router*) chamado WORM para realizar as interconexões do circuito. As duas ferramentas possuem uma forte integração de forma que uma falha do roteador em completar o roteamento do circuito, faz com que o gerador inserira automaticamente espaços nas regiões problemáticas para a próxima tentativa do roteador.

O fluxo de geração de leiaute criado com o uso destas ferramentas recebeu o nome de PARROT. Diversas ferramentas foram desenvolvidas posteriormente e integradas à este fluxo.

Um novo roteador mais robusto chamado ROTDL foi desenvolvido em (FLACH;

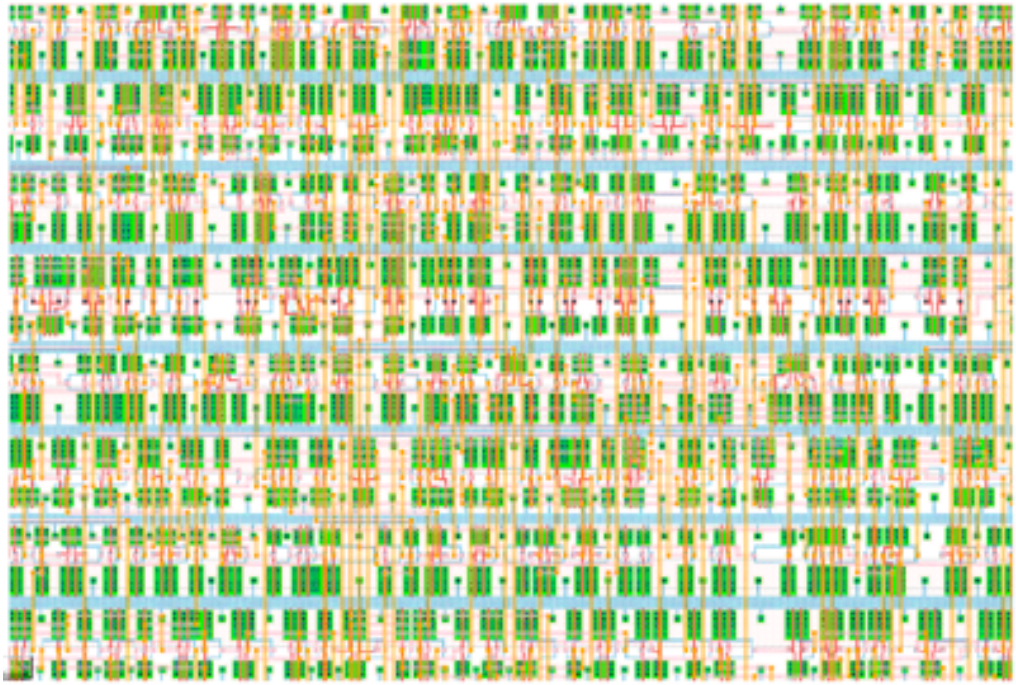


Figura 3.2: Exemplo de leiaute produzido pela ferramenta PUNCH

HENTSCHKE; REIS, 2004). Apesar de não ter ainda uma integração tão forte com o gerador de leiautes, esta ferramenta é capaz de rotear em menor tempo, circuitos de mesma complexidade que o WORM.

Como a estimativa do tamanho das células feito pelo posicionador MANGO era muito simples e tinha uma grande margem de erro, em (ZIESEMER et al., 2006) foi apresentado um gerador de estimativas de tamanho de células chamado CELLSE. Com esta ferramenta foi possível estimar as dimensões das células de forma mais automatizada e precisa. Os leiautes produzidos por PUNCH apresentaram como resultado um leiaute com uma relação de aspecto mais previsível, com um menor número de redes não roteadas e comprimento médio das conexões.

O roteador ChAOS (SANTOS; JOHANN; REIS, 2006) surgiu como uma alternativa para roteamento convergente e ágil uma vez que é implementado com algoritmos de baixa complexidade. A convergência é garantida pela inserção de espaços (que correspondem às *filler-cells* das *standard-cells*) e a agilidade é garantida por um pré-planejamento global baseado em árvores de expansão e um roteamento detalhado baseado no Greedy Channel Router. Os resultados mostraram que apesar de apresentar tempos de execução muito inferiores ao WORM e ROTDL, poderia haver um pequeno aumento da área final devido aos espaços inseridos.

Em (MEINHARDT, 2006) foi desenvolvido um gerador de leiautes regulares baseado em matrizes de células chamado RCAT. Esta ferramenta foi integrada ao fluxo PARROT e tinha como principal característica a previsibilidade da suas características elétricas.

Dando início às pesquisas na área de circuitos 3D, Hentschke (HENTSCHKE, 2007) desenvolveu um posicionador quadrático com refinamento local iterativo utilizando o método de *Threshold Accept* (DUECK; SCHEUER, 1990) para redução de *wirelength* (comprimento das conexões) 3D com observância do caminho critico. Este posicionador, que recebeu o nome de ZPLACE, também realiza posicionamento em circuitos 2D e é capaz de suportar circuitos com centenas de vezes mais células do que o MANGO PARROT.

Em (SAWICKI et al., 2007), a ferramenta LAMEHORSE foi desenvolvida para realização do particionamento e posicionamento dos PADs e pinos de I/O em circuitos 3D. O posicionamento destes elementos assume fundamental importância para o correto funcionamento dos algoritmos de posicionamento quadráticos uma vez que são eles que servirão de suporte para as células ao longo das diferentes *tiers* (camada de *chips*). Além disto, um correto posicionamento pode contribuir significativamente para redução do wirelength e atraso das interconexões do circuito.

As ferramentas ZPLACE e LAMEHORSE, por tratar de problemas recentes, não estão ainda integradas a um fluxo de síntese física.

3.3 Métodos para Geração Automática de Leiautes de Células

Lefebvre (LEFEBVRE; MARPLE; SECHEN, 1997), fez um interessante estudo sobre as abordagens mais comuns de geradores de leiaute e conseguiu classificá-las em 3 categorias distintas:

Geradores procedurais de leiaute podem ser baseados em alguma linguagem de descrição ou simbólicos. O leiaute é descrito de uma forma independente de regras de desenho e o gerador transforma a descrição fornecida no leiaute da célula correspondente de acordo com as regras tecnológicas utilizadas. Cada célula necessita ter seu próprio código gerador e que precisa ser desenvolvido por um projetista experiente para uma melhor eficácia. Como desvantagem, geradores procedurais tendem a não serem amigáveis à drásticas mudanças na arquitetura da célula e tecnologias de interconexão.

Re-compactação de leiautes pré-existentes provê uma solução mais elegante ao problema uma vez que uma simples ferramenta genérica pode processar uma grande variedade de células. Entretanto, uma biblioteca pronta de células precisa ser previamente desenvolvida à mão e também possui a desvantagem de não tolerar muito bem mudanças arquiteturais na célula.

Síntese de células é a geração do leiaute tendo como ponto de partida o *netlist* da rede de transistores de cada célula. Este método costuma ser completamente flexível quanto à arquitetura da célula e não requer nenhuma informação pré-existente do leiaute. Entretanto, o problema de criar leiautes competitivos em qualidade com os feitos-à-mão não é um problema trivial e a complexidade deste processo tem sido o maior empecilho para o seu sucesso comercial.

Dentre os três métodos, Lefebvre defende a síntese de células para geração de leiautes devido à sua flexibilidade, particularmente no contexto dos futuros avanços no estado-da-arte da síntese de circuitos digitais.

A metodologia escolhida para este trabalho foi a síntese de leiaute de células por este ser o único método de geração completamente automático capaz de gerar células com diferentes dimensionamento de transistores e uma fácil portabilidade tecnológica.

3.4 Síntese de Leiaute de Células

Síntese de células consiste em mapear uma rede de transistores dimensionados no leiaute de célula correspondente, de acordo com uma arquitetura de célula pretendida. A

metodologia utilizada em implementações estado-da-arte começa com o usuário provendo um *netlist*, um *template* (gabarito) e a tecnologia de fabricação a ser utilizada. O *netlist* possui uma lista dos transistores que formam o circuito, com seus respectivos tamanhos, seus terminais de entrada/saída e interconexões. O gabarito descreve a forma física do leiaute das células tal como: altura, largura das linhas de alimentação, grade de roteamento, etc. De uma forma geral, o mesmo gabarito costuma ser usado para construção de todas as células da mesma biblioteca.

A saída de um sistema para síntese de células é uma coleção de leiautes resultantes da solução sucessiva de três subproblemas: posicionamento de transistores, roteamento e compactação.

O leiaute resultante costuma ser analisado por um projetista de leiaute, para verificar sua qualidade e se necessário, fazer pequenos ajustes à mão para corrigir/melhorar suas características, antes da célula ser utilizada para constituir uma biblioteca.

3.4.1 Especificação da Célula

O primeiro passo para a geração automática, é o projetista fornecer a especificação da célula. A especificação pode ser provida em 2 níveis de abstração diferentes: físico e lógico. O *netlist* lógico especifica a função lógica que a célula deve executar e não entra em detalhes da sua constituição interna (transistores, conexões, etc.). O *netlist* físico possui correspondência de um para um entre seus componentes e os transistores implementados no leiaute final e por esta razão, é o preferido para ser utilizado como entrada em uma ferramenta de síntese de células lógicas.

Para gerar circuitos a partir de um *netlist* lógico, é necessário primeiramente mapear a célula para um arranjo de transistores que realize a função especificada obtendo-se assim o seu *netlist* físico. Há várias otimizações que podem ser feitas durante o mapeamento para obter células com diferentes características de consumo estático/dinâmico, delay e área. A ferramenta TABA (REIS; ROBERT; REIS, 1998) é um exemplo de ferramenta capaz de transformar o *netlist* lógico no físico automaticamente.

3.4.2 Estilos de Leiautes

Estilo de leiaute é a organização interna da célula e que normalmente serve como guia para todo o processo de síntese. Algumas das definições freqüentemente encontradas no projeto de ferramentas de síntese de células são: regras para posicionamento de transistores P e N (número e posição das bandas), regras para roteamento entre transistores (camadas e regiões que podem ser utilizadas), posição das interfaces da células, posição e forma das linhas de alimentação, etc.

Diversos estilos de leiautes visando a minimização da área e da complexidade dos algoritmos de posicionamento e roteamento através da definição de regiões específicas para alocação dos transistores P e N foram propostos na literatura.

Uehara e vanCleemput (UEHARA; VANCLEEMPUT, 1981) propuseram o estilo de leiaute chamado *linear-matrix* para geração de circuitos de lógica CMOS complementar. Neste estilo, as células são formadas por duas linhas horizontais de transistores PMOS e NMOS paralelas à alimentação e com os polisilícios dos *gates* perpendiculares como ilustrado na Figura 3.3. Agrupados desta forma, transistores com o mesmo sinal podem ser conectados diretamente com difusão ao invés de conexões com contatos e metais. Isto permite que as células tenham uma maior densidade de transistores e, ao mesmo tempo, melhorem suas características elétricas devido à uma menor capacitância de acoplamento entre os transistores.

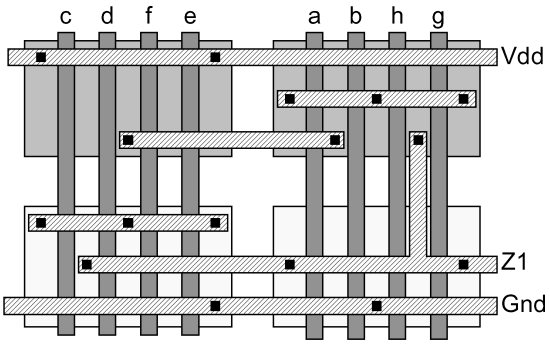


Figura 3.3: Estilo de leiaute linear-matrix

Estilos baseados no *linear-matrix* são freqüentemente referenciados como 1D (ou de uma dimensão) pois os transistores são posicionados seqüencialmente, lado a lado, mudando apenas o seu ordenamento e orientação. As variações mais freqüentes encontradas no estilo *linear-matrix* são quanto à posição e nível de metal utilizado nos canais de alimentação da célula, método de roteamento interno e posição dos pinos de entrada e saída.

O estilo 1-1/2D proposto em (REKHI; TROTTER; LINDER, 1995) define regiões para posicionamento dos transistores de forma similar ao *linear-matrix* mas permite que transistores P possam ser posicionados na região de difusão N e vice-versa. Esta técnica permite um melhor aproveitamento de área principalmente nos casos de circuitos com grande discrepância no número de transistores PMOS e NMOS.

Estilo 2D permitem arranjos vertical e horizontal de transistores. Enquanto em algumas abordagens são geradas múltiplas linhas de transistores 1D para formar o leiaute 2D como no exemplo da ferramenta CLIP (GUPTA; HAYES, 1997) mostrada na Figura 3.4 (a). Existem outras que não são baseadas em linhas de transistores e que permitem que os mesmos sejam posicionados de várias formas diferentes, como no exemplo da ferramenta TEMPO (RIEPE; SAKALLAH, 2003) mostrada na Figura 3.4. Neste último caso, os transistores não obedecem nenhum estilo pré-determinado e o seu posicionamento é feito através de heurísticas que procuram por uma solução de menor custo. Este tipo de posicionador costuma ser mais freqüentemente utilizado para circuitos analógicos e para famílias lógicas como Cascade Voltage Switch Logic (CVSL), Pass Transistor Logic (PTL) e domino CMOS onde não há muita regularidade entre as conexões dos transistores PMOS e NMOS, e nem equivalência entre seus *gates*.

3.4.3 Posicionamento dos Transistores

A área ocupada pela célula tem relação direta com o posicionamento de seus transistores porque existem diversas otimizações que podem ser feitas de acordo com o seu posicionamento. A Figura 3.5 mostra um exemplo de possíveis otimizações entre dois transistores. Caso estes transistores não possuam nenhum sinal em comum, eles precisam ser posicionados separadamente ocupando grande área como mostrado em (a). Caso eles possuam alguma rede em comum, otimizações visando minimizar a área ocupada e melhorar as características elétricas do circuito podem ser feitas. Este é o caso de (b) onde o *source*/dreno de um transistor pertence a mesma rede do *source*/dreno do outro transistor e de (c) onde ambos possuem o sinal de *gate* em comum.

Efeitos como rotação, translação e espelhamento também podem ser explorados de forma a aproveitar melhor toda a área da célula. Entretanto, quanto maior o grau de liberdade sobre o posicionamento dos transistores, maior o espaço de soluções que o algoritmo

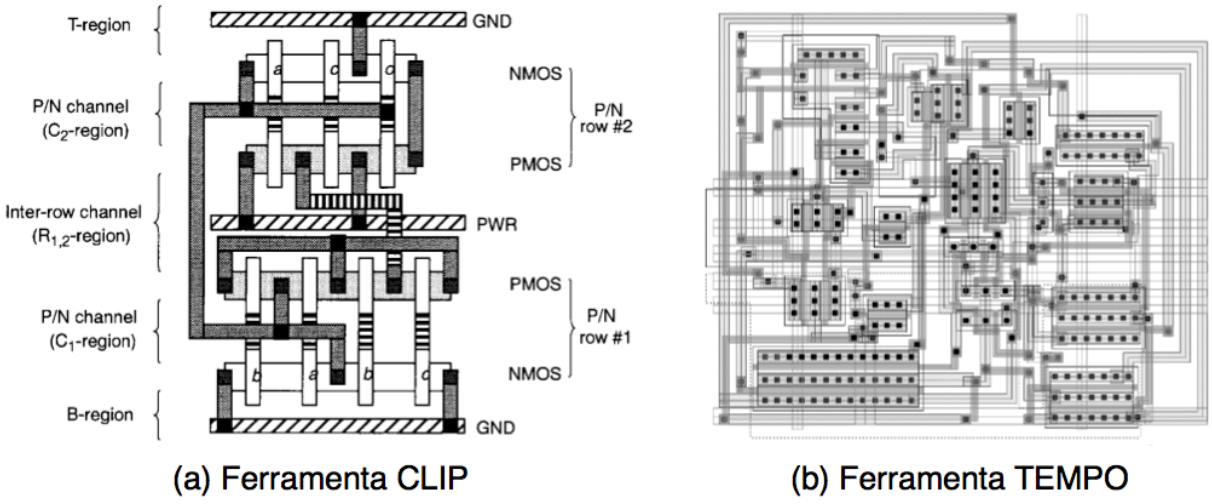


Figura 3.4: Estilos de leiautes 2D

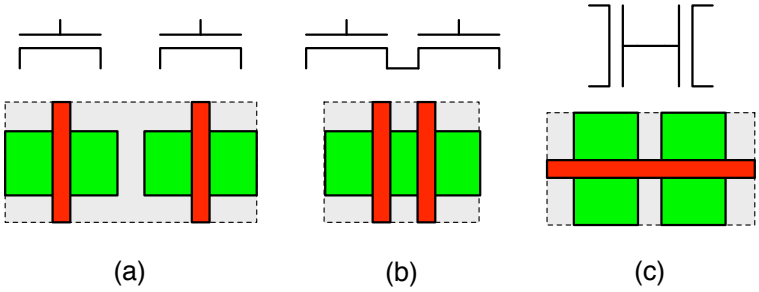


Figura 3.5: Exemplo de otimização elétrica e de área entre dois transistores

de posicionamento deve pesquisar, o que reflete em um aumento da complexidade e do tempo de execução do algoritmo. Outro problema, é que uma distribuição arbitrária dos transistores ao longo do leiaute também costuma dificultar a etapa seguinte de roteamento da célula.

3.4.3.1 Algoritmos para Posicionamento de Transistores

A escolha do algoritmo de posicionamento de transistores é altamente dependente do estilo de leiaute adotado.

Quando Uehara (UEHARA; VANCLEEMPUT, 1981) propôs o estilo *linear-matrix*, ele também sugeriu o algoritmo do caminho de Euler para encontrar um posicionamento dos transistores que minimize o número de quebras na difusão. Entretanto este método não era ótimo e se limitava a geração apenas de circuitos duais, ou seja, onde é possível formar pares de transistores com o mesmo sinal nas duas difusões P e N.

Um método exato para minimizar altura e largura de células CMOS 1D foi proposto pela primeira vez por Maziasz e Hayes em (MAZIASZ; HAYES, 1991). Este método considera a ocupação dos canais de roteamento para tentar minimizar a altura da célula.

Gupta desenvolveu em (GUPTA; HAYES, 1996) um método de minimização da largura de células CMOS no estilo 2D (células 1D com altura de duas ou mais bandas) usando programação linear com inteiros (ILP). A principal contribuição deste trabalho foi propor uma técnica para agrupar transistores em série para poder tratar de células com mais de 20 transistores ao custo de uma pequena perda em otimalidade.

Dando prosseguimento a este trabalho, em (GUPTA; HAYES, 1997, 2000) foi apresentada a ferramenta CLIP que é capaz de realizar pela primeira vez otimização na largura e na altura das células no estilo 2D. Com o uso de um resolvedor de ILP, a ferramenta é capaz de encontrar a solução ótima dentro da forma como o problema de posicionamento foi elaborado. Ainda neste trabalho, CLIP foi estendido para agrupar transistores e funcionar de forma hierárquica. A nova ferramenta foi chamada de HCLIP e é capaz de produzir resultados com tempo de execução até três ordens de grandeza menor que CLIP e produzindo os mesmos resultados em aproximadamente 80% dos casos.

Finalmente em (GUPTA; HAYES, 1998), Gupta apresentou a ferramenta FCLIP que diferencia-se de CLIP no fato de conseguir unir a técnica de folding com posicionamento usando ILP.

Em (IIZUKA; IKEDA; ASADA, 2004), Iizuka desenvolveu um algoritmo também exato de posicionamento no estilo 1D utilizando a técnica de satisfiability (SAT). O posicionamento e roteamento dos transistores é primeiro transformado num problema de SAT e depois resolvido com um resolvedor de SAT. Os resultados obtidos foram similares aos de Gupta mas com garantia de convergência no roteamento da célula.

Todos estes métodos fazem pares de transistores P e N e não podem ser aplicados à células que possuem transistores das redes P e N não duais.

A Tabela 3.1 (IIZUKA; IKEDA; ASADA, 2005b) lista o número de células com redes P e N não duais encontradas em bibliotecas *standard-cells* de tecnologias recentes. Em uma biblioteca, não somente flip-flops ou buffers three-state, mas também circuitos originalmente duais freqüentemente possuem redes não-duais como resultado de técnicas para fazer o transistor caber na altura da célula como o folding. Além disto, a biblioteca também pode utilizar células com lógica de transistores de passagem (PTL), lógica dinâmica/dominó, etc. que podem ser altamente irregular, com complexo compartilhamento de difusão e roteamento pouco trivial. Por esta razão, é desejável que ferramentas atuais de geração automática de células também sejam capazes de gerar circuitos com outros

Tabela 3.1: Número de células não-duais em uma biblioteca *standard-cell* comercial

Tecnologia	Número total de células	Número de células não-duais	%
130 nm A	527	274	52%
130 nm B	578	294	51%
90 nm A	462	90	19%
90 nm B	340	176	52%

tipos de redes além das série/paralelo complementar.

Diversos métodos para posicionamento de células com redes arbitrárias e número diferente de transistores P e N foram propostos.

Lib (HSIEH et al., 1990) identifica clusters fortemente conectados, e forma pares de transistores P e N dentro do cluster para posicionar transistores no estilo 1D. Um algoritmo de folding também foi desenvolvido para tratar transistores maiores que a altura da banda de difusão.

Em (GUPTA; THE; HAYES, 1996), Gupta descreve uma ferramenta chamada XPRESS para produzir leiautes no estilo 1D com suporte a dimensionamento individual dos transistores com o uso de folding. Esta ferramenta identifica conjuntos de transistores fortemente conectados para formar subconjuntos. Um algoritmo exato é utilizado para encontrar a melhor cobertura de subconjuntos que minimize o número de quebras na difusão (GAPs) e o número de trilhas de roteamento horizontais. Segundo o autor do trabalho, esta ferramenta foi utilizada ativamente dentro da Intel na época para produção de leiautes de bibliotecas de células para PO.

Em (GURUSWAMY et al., 1997) foi apresentada a ferramenta CELLERITY da Motorola para síntese automática de bibliotecas de *standard-cell* no estilo 2D. O sistema desenvolvido suporta células de altura simples (uma banda) e de altura dupla (duas bandas). Um algoritmo de folding é aplicado nos transistores antes da realização do posicionamento, modificando o *netlist* inicialmente fornecido. O posicionamento é feito com *Simulated Annealing* onde a função de custo tem o objetivo de minimizar: o número de quebras de difusão, o comprimento total das conexões, a densidade de canal e o desalinhamento entre *gates*, *sources* e drenos dos transistores. Os resultados obtidos com este trabalho deram origem a uma patente (MAZIASZ; GURUSWAMY; RAMAN, 2001) que apresenta detalhes da metodologia desenvolvida.

Um posicionador automático para células de partes operativas é apresentado por Serdar em (SERDAR; SECHEN, 2001). Este algoritmo é capaz de posicionar os transistores sem um estilo previamente definido e em várias orientações diferentes. Para isto, o método proposto neste artigo sugere a representação dos blocos que representam os transistores a serem posicionados através de uma estrutura O-tree (utilizada originalmente em (GUO; CHENG; YOSHIMURA, 1999) para *floorplaning*). Nesta representação, a ordem dos elementos na árvore indica a posição relativa dos blocos como pode ser observado na Figura 3.6. O autor divide o processo de posicionamento dos transistores em duas etapas: posicionamento global, usando *Simulated Annealing*, e posicionamento detalhado, feito por modificações desenvolvidas por eles no algoritmo O-tree. Estas modificações no posicionamento detalhado visam permitir ao algoritmo lidar com cadeias de transistores em diferentes orientações e a adequação do leiaute às regras de desenho da tecnologia. As principais características do posicionador desenvolvido por Serdar são: capacidade de

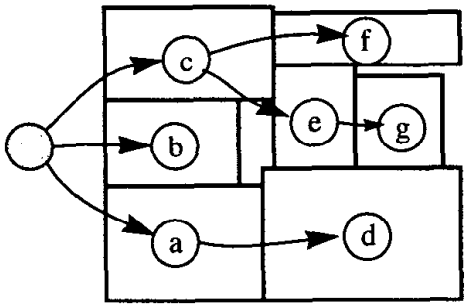
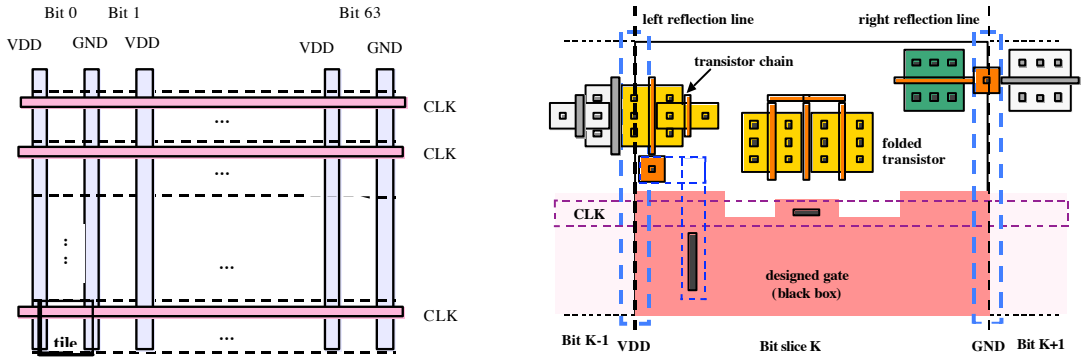


Figura 3.6: Representação O-tree de um posicionamento



(a) Vista global da estrutura da parte operativa

(b) Possibilidades para posicionamento de transistores

Figura 3.7: Posicionador automático de células de partes operativas (SERDAR; SECHEN, 2001)

lidar com largura fixa de célula, posicionamento de transistores na linha de reflexão (para conectar com células adjacentes usando difusão) e formato de célula não retangular. O roteamento detalhado é realizado em 4 camadas de metal pela ferramenta comercial ITools (ITOOLS, 2007).

A Figura 3.7 (a) mostra a estrutura regular da parte operativa adotada por Serdar. Em (b), aparece em destaque um exemplo de célula para a estrutura mostrada e as alternativas de posicionamento dos transistores.

Em (RIEPE; SAKALLAH, 2003), Riepe desenvolveu para a Magma um gerador de leiaute de células 2D chamado TEMPO. A técnica utilizada por ele foi fazer agrupamentos de transistores que formam uma seqüência ininterrupta de terminais de dreno e fonte e depois aplicar *Simulated Annealing* para posicionar os grupos de forma a minimizar a função de custo que utiliza uma combinação ponderada dos custos de roteamento e posicionamento (área, perímetro, violação da relação de aspecto, etc.). O restante do fluxo é realizado com a ajuda de um roteador detalhado chamado Anagram-II e o compactador Masterport, ambos providos por terceiros.

IIzuka propôs em (IIZUKA; IKEDA; ASADA, 2005a) modificações na sua ferramenta de síntese para suportar circuitos com redes não duais de transistores. Para reduzir o problema do tempo de execução do algoritmo para células com número elevado de transistores, o autor fez uso de uma hierarquia similar à utilizada em (GUPTA; THE; HAYES, 1996). Como resultado, obteve-se uma melhora considerável no tempo de execução em troca de um pequena piora no resultado do posicionamento. A Figura 3.8 mostra o estilo de leiaute, a formulação do posicionamento de transistores e do roteamento interno da

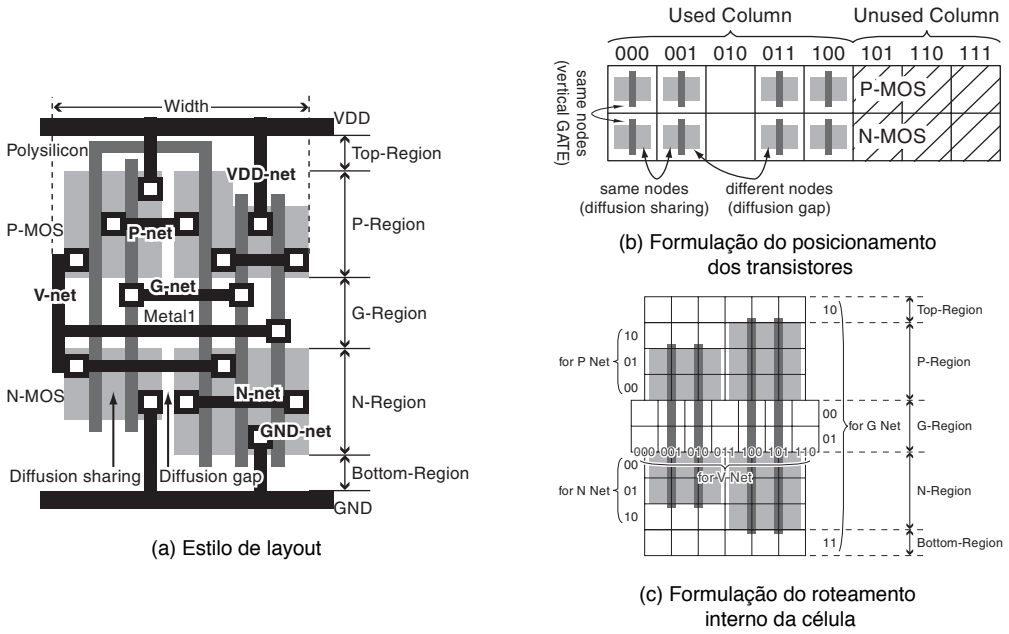


Figura 3.8: Posicionamento de transistores usando SAT (IIZUKA; IKEDA; ASADA, 2005a)

célula (c) utilizado neste trabalho.

3.4.4 Posicionamento dos *Ties*

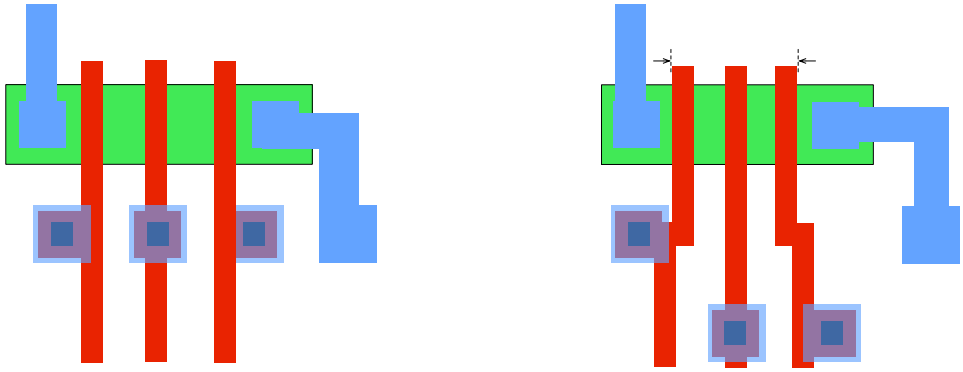
Para evitar o *latch-up*, diversas tecnologias de fabricação exigem a colocação regular de *ties* ao longo do leiaute. Os *ties* são contatos conectados a alimentação que polarizam o poço/substrato e evitam a ocorrência de curto-circuito nos transistores. Normalmente, os *ties* são posicionados em regiões específicas da célula, sob as linhas de alimentação, previamente à realização do posicionamento dos transistores. Entretanto, existem alguns trabalhos no sentido de encontrar um posicionamento dos *ties* que traga um melhor aproveitamento da área da célula como em (MAZIASZ; GURUSWAMY; RAMAN, 2001). O método utilizado neste trabalho foi o de posicionar os *ties* somente após a realização da compactação da célula. Um algoritmo varre o leiaute a procura de regiões disponíveis para a colocação dos *ties* e por fim os conecta a alimentação.

Tecnologias mais recendes baseadas em SOI dispensam a necessidades destes contatos o que pode trazer uma economia de área em algumas células.

3.4.5 Portas de Entrada/Saída da Célula

As portas entrada/saída da célula são os meios responsáveis pela comunicação com as demais células e interfaces do circuito. O interfaceamento pode ser realizado através da colocação de pinos para camadas superiores de metal - como no caso das bibliotecas de *standard-cells* recentes para tecnologias com 2 ou mais camadas de metal - ou também através de interfaces posicionadas no limite das células para conexões por justaposição - muito utilizado em tecnologias mais antigas onde não existiam muitos níveis de metais e as conexões eram feitas através de canais de roteamento e também em circuitos especiais onde o posicionamento das células obedece alguma regra preestabelecida.

De uma forma geral, quando utiliza-se bibliotecas de células em tecnologias recentes, as conexões de alimentação são feitas exclusivamente por justaposição e as demais



(a) Posicionamento de pinos ineficiente (b) Posicionamento de pinos otimizado

Figura 3.9: Influência do posicionamento das portas de E/S no leiaute da célula

interfaces da célula são posicionadas no seu interior para serem acessadas através de roteamento *over-cell* nas camadas superiores de metal. Estas regiões normalmente são alinhadas à uma grade de roteamento para facilitar a realização das interconexões do circuito e garantir a obediência às regras de desenho da tecnologia.

O posicionamento das interfaces também costuma ter grande influência na área final da célula e na facilidade de interconexão destas com as demais. Um posicionamento ineficiente dos pinos de entrada e saída da célula pode levar a um aumento da área do circuito e também inserir obstáculos para o roteamento externo conforme estudo feito em (LEFEBVRE; MARPLE; SECHEN, 1997). A Figura 3.9 mostra uma situação onde o posicionamento das portas da célula levou a um aumento da área de difusão dos transistores o que produz um aumento da resistência e das capacitâncias parasitas associadas.

3.4.6 Roteamento

O roteamento interno da célula, assim como o posicionamento, é altamente dependente do estilo de leiaute escolhido.

Roteamento de canal é uma das técnicas mais utilizadas em leiautes 1D. Nesta metodologia, é definido um canal entre as difusões P e N onde são feitas as conexões entre os transistores. Entretanto, esta solução falha em aproveitar os recursos existentes fora desta região como roteamento em metal sobre os transistores e em polisilício junto à alimentação. Alguns trabalhos como em (ONG; LI; LO, 1989), utilizam algoritmos específicos em combinação com o roteamento de canal para completar o roteamento nestas regiões. Outros reportam o uso de *maze routers* (roteadores em malha) com suporte a obstáculos. Este algoritmo tende a ser mais flexível em termos de estilo de leiaute e estrutura do circuito, mas apresenta maior dificuldade de ajuste para que possa produzir um leiaute com melhor qualidade. Em (POIRIER, 1989), Poirier descreve o uso de A* para melhorar o desempenho em relação a outros trabalhos que utilizam Lee (LEE, 1961).

De forma a tentar convergir para uma solução, alguns roteadores completam o roteamento pela adição automática de novas trilhas. Esta inserção pode ser disparada pela detecção de uma solução não factível ou *time-out* (tempo limite).

3.4.7 Compactação

Após os elementos estarem devidamente posicionados e o circuito roteado, o leiaute é finalmente compactado. Compactação é o processo de geração do leiaute da célula a partir de uma especificação inicial de acordo com as regras de desenho especificadas. O

termo compactação também pode ser usado para a transformação de um leiaute (mais antigo) em um novo com diferentes regras de projeto.

Grande parte dos compactadores são capazes de compactar em apenas uma direção de cada vez (MARPLE; SMULDERS; HEGEN, 1988). Entretanto, existem alguns trabalhos que reportam o uso de compactadores bi-dimensionais que realizam a compactação da altura e largura da célula simultaneamente (SHIGEHIRO et al., 1994). Os algoritmos encontrados para resolver este tipo de problema, em geral envolvem técnicas de teoria de grafos, *simulated annealing*, e programação linear. Uma revisão dos métodos de compactação pode ser encontrada em (BOYER, 1988).

Compactadores baseados em grafos são os mais comumente encontrados. Neste método, os nós representam os elementos do leiaute ou bordas dos polígonos, e os arcos representam regras de desenho ou restrições de conexão entre os elementos/bordas. Normalmente as regras de distância mínima da tecnologia são utilizadas para estabelecer valores para os arcos.

As principais funções de minimização definidas durante a compactação normalmente se referem à altura e largura das células, e também ao comprimento das conexões (em especial das críticas). Algumas restrições também são muitas vezes necessárias para adequar o resultado ao estilo de leiaute especificado tais como: pinos alinhados à grade de roteamento, dimensões (altura e largura) múltiplas desta grade, etc.

3.5 Conclusão

Este capítulo fez uma breve revisão sobre geração automática de leiaute de células lógicas para tecnologia CMOS. As etapas básicas de um fluxo de geração e a forma como diversos trabalhos tratam cada um dos problemas associados foi apresentada.

Durante esta revisão, foi visto que células com redes não dual de transistores P e N representam até metade do total de células de uma biblioteca *standard-cell* e por esta razão os geradores de células para tecnologias recentes devem estar preparados para suportar esta restrição. Além disto, também é desejável que o gerador suporte um dimensionamento individual dos transistores para fins de otimização de atraso e potência. Todas as ferramentas apresentadas que estão sendo usadas comercialmente ou para geração de células de parte operativa possuem estas características.

Com base nestas conclusões, o desenvolvimento da ferramenta de síntese de células CMOS partiu do princípio que deveria suportar no mínimo estas duas restrições para que pudesse ser utilizada em conjunto com o gerador de parte operativa. A ferramenta de síntese desenvolvida é apresentada no Capítulo 4.

4 DESENVOLVIMENTO DE UM GERADOR AUTOMÁTICO DE LEIAUTES DE CÉLULAS CMOS

Neste capítulo é apresentada a ferramenta de síntese de células lógicas CMOS desenvolvida neste trabalho para ser utilizada em conjunto com um compilador de parte operativa. Esta ferramenta foi elaborada a partir de uma seleção dos algoritmos e estratégias reportados no Capítulo 3 e recebeu o nome de CellGen.

4.1 Introdução

Projeto de células de alta performance e baixo consumo de energia exige que um dimensionamento individual da largura (BORAH; OWENS; IRWIN, 1995) e comprimento (GUPTA et al., 2004) do canal dos transistores tenha que ser feito e respeitado durante o processo de síntese de leiaute. Além disto, células utilizadas em projeto de PO freqüentemente possuem uma rede de transistores não complementar devido ao uso de células otimizadas, de diferentes famílias lógicas, ou como resultado da aplicação de folding em seus transistores para atender os requisitos de altura da célula.

Para automatizar o processo de desenvolvimento do leiaute de células de POs, foi desenvolvido uma ferramenta de síntese de células cujas características incluem o suporte à células com diferentes redes de transistores e a obediência ao dimensionamento informado na sua especificação.

As células automaticamente geradas seguem um padrão que as permitem constituir uma biblioteca de células e posteriormente serem utilizadas para integrar um fluxo de geração com posicionamento e roteamento automático, similar ao fluxo para *standard-cells*.

4.2 Formulação do problema

A especificação da estrutura interna das células é feita no formato SPICE. Este formato possui para cada célula informada, a lista de seus transistores, o comprimento e a largura dos *gates*, as redes às quais pertencem e seus pinos de entrada/saída. O gerador recebe este arquivo como entrada e gerar o leiaute de cada uma das células de acordo com as regras da tecnologia especificada e de um *template* que define os parâmetros de geração. O leiaute resultante é salvo em formato CIF e LEF.

A Figura 4.1 mostra o fluxo de projeto da ferramenta.

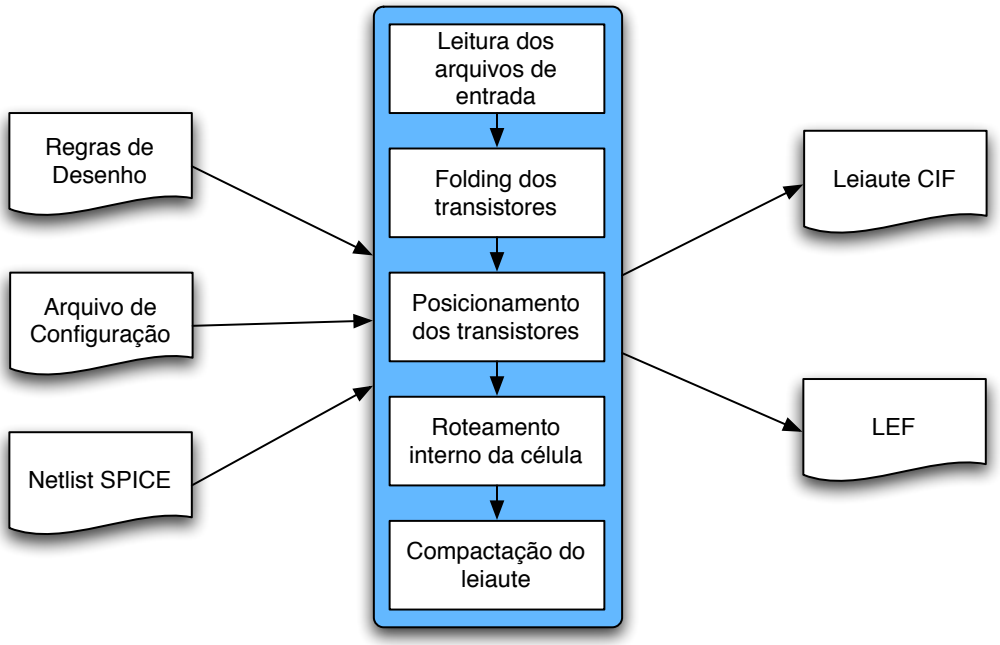


Figura 4.1: Fluxo de projeto do gerador de células

4.3 Estilo de Leiaute

O estilo de leiaute escolhido para ser utilizado no gerador de células é baseado fundamentalmente no estilo 1D proposto inicialmente por Uehara (UEHARA; VANCLEEMPUT, 1981). A Figura 4.2 mostra um exemplo de célula que ilustra o estilo de leiaute escolhido. Suas características estão listadas abaixo:

- Suporte à circuitos com diferentes redes de transistores individualmente dimensionados;
- Estilo de leiaute em 1D formado por duas linhas horizontais de difusão paralelas, uma P e outra N, com *gates* dos transistores posicionados na vertical;
- Poço posicionado e dimensionado de forma a permitir o espelhamento das células entre bandas pares e ímpares.
- Roteamento interno da célula feito exclusivamente com: polisilício, metal 1 e difusão (para conexão de transistores adjacentes que possuam o mesmo sinal nos extremos);
- Trilhas centrais para roteamento vertical e horizontal na região entre as difusões P e N utilizando polisilício e metal 1.
- Trilhas adicionais para roteamento sobre os transistores P/N utilizando metal 1;
- Roteamento em polisilício na região de fora dos transistores junto à alimentação quando houver espaço suficiente.
- Linhas de alimentação localizada nos limites inferior e superior da célula em metal 1 com conexão com as células vizinhas por justaposição;

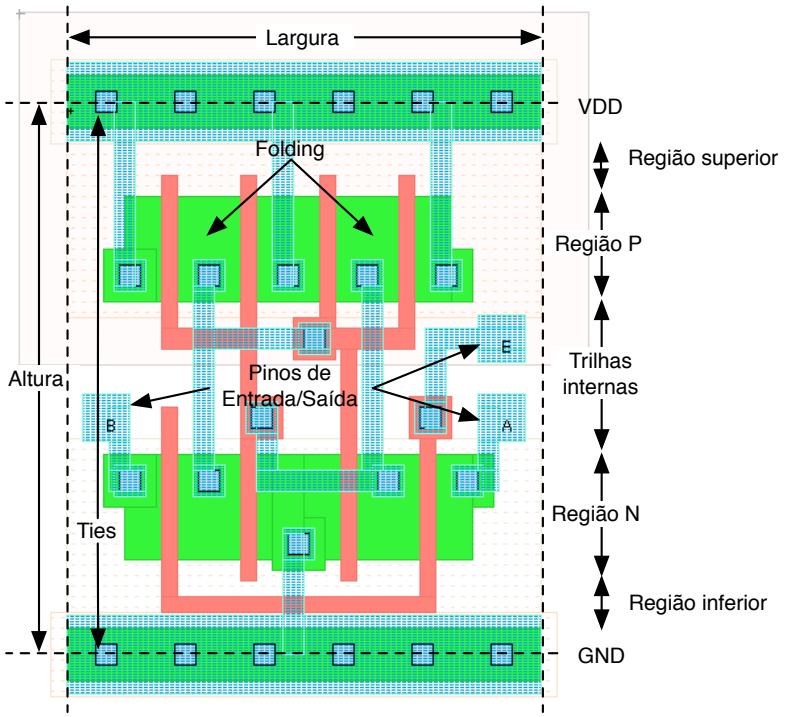
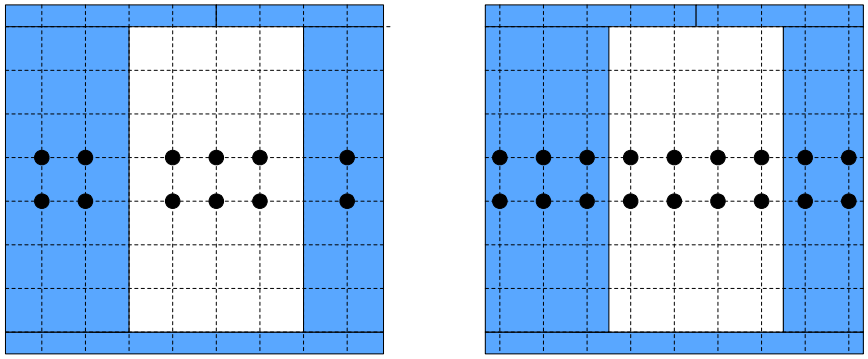


Figura 4.2: Estilo de leiaute 1D utilizado no gerador

- *Ties* posicionados sob as linhas de alimentação nos limites da célula e alinhados à grade de roteamento (para evitar problemas de DRC com células vizinhas);
- Contato único para conexão com as áreas ativas dos transistores;
- Pinos de entrada/saída da célula posicionados sobre as trilhas centrais e alinhados à grade de roteamento.
- Uso de jogs nas trilhas internas para reduzir o espaçamento entre *gates* e a largura da célula.
- Largura máxima das difusões calculada de acordo com a altura da célula e a posição das trilhas.
- Utilização de GAPs sempre que não houver continuidade na rede de transistores ao longo das difusões.
- Nenhum re-ordenamento na rede dos transistores é realizado.

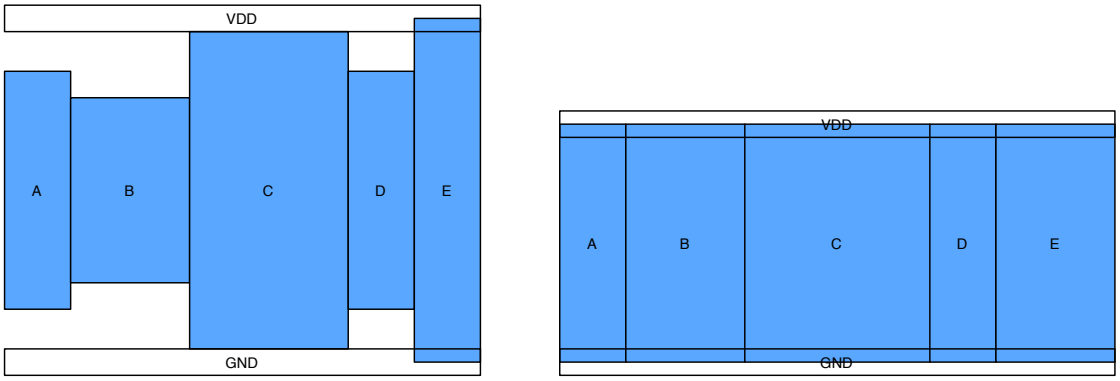
O alinhamento dos pinos de E/S da célula a uma grade de roteamento garante a conformidade de suas conexões às camadas superiores de metal com as regras de desenho da tecnologia utilizada além de também permitir que algoritmos de menor complexidade possam ser utilizados para realização do roteamento com as demais células do circuito. Esta técnica tem sido muito utilizada atualmente, principalmente para projeto de bibliotecas de células e por esta razão também foi definida no estilo de leiaute proposto.

Para garantir que os pinos fiquem alinhados à grade de roteamento, é necessário fazer com que as células também sejam posicionadas sobre a grade. Foram encontradas duas formas de realizar este posicionamento. A Figura 4.3 (a) mostra a forma de posicionamento sem *offset* no eixo X enquanto (b) mostra a forma de posicionamento com *offset*.



(a) Sem offset (b) Com offset em X

Figura 4.3: Opções de alinhamento das células à grade de roteamento



(a) Sem folding (b) Com folding

Figura 4.4: Diferença de altura na banda de acordo com a aplicação do método de folding nas células

O estilo escolhido foi o com *offset* por ser mais eficiente quanto ao número de conexões de E/S que podem ser inseridas dentro da célula. A existência de *offset* em Y não tem impacto no número de conexões em E/S uma vez que não pode haver pinos para os sinais de E/S sobre a alimentação conforme o estilo de leiaute proposto.

4.4 Folding

Dimensionamento de transistores é essencial para a produção de circuitos com alta performance. Várias ferramentas recentes são capaz de realizar um dimensionamento individual dos transistores com o objetivo de reduzir o atraso e o consumo de energia (SANTOS et al., 2005). Leiautes produzidos no estilo 1D com transistores de diferentes alturas tendem a desperdiçar área uma vez que a altura de cada linha de difusão P e N é calculada de acordo com o seu transistor mais largo como pode ser visto na Figura 4.4.

Para resolver este problema, um dos métodos mais utilizados é o folding de transistores. Este método consiste em quebrar os transistores de maior tamanho em transistores menores, conectados em paralelo, com o objetivo de manter a altura da célula reduzida à custa de um pequeno acréscimo na largura. De acordo com Gupta (GUPTA; HAYES, 1998), o problema de folding pode ser classificado como posicionamento estático/dinâmico com folding estático/dinâmico:

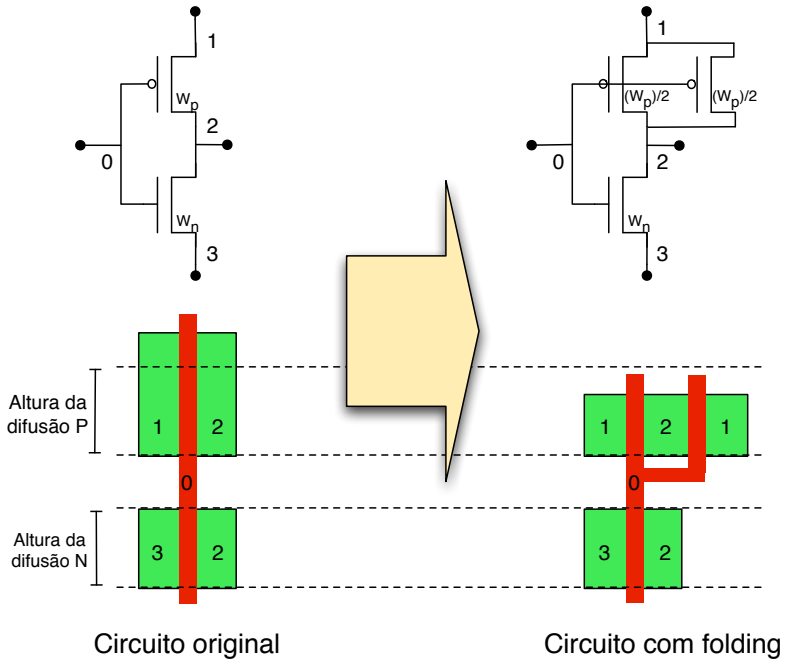


Figura 4.5: Método de folding aplicado à um transistor maior que a altura da linha de difusão

1. Posicionamento e folding estático: Dado um posicionamento de transistores pré-especificado e os limites no tamanho dos transistores (limites para o folding), quebrar os transistores e determinar suas orientações de forma a preservar o posicionamento e minimizar a área.

2. Posicionamento estático com folding dinâmico: Dado um posicionamento dos transistores, determinar o número de quebras e orientação de cada transistor de forma a minimizar a área;

3. Posicionamento dinâmico com folding estático: Dado os limites para o folding, quebrar os transistores e determinar a sua posição e orientação de forma a minimizar a área da célula.

4. Posicionamento e folding dinâmicos: Para cada transistor, determinar o número de quebras, sua posição e orientação tal que a área total seja minimizada.

Apesar desta classificação originalmente ter sido proposta para posicionamento de leiautes no estilo 2D, o mesmo se aplica para 1D.

A metodologia escolhida neste trabalho foi a de número 3, com folding estático e posicionamento dinâmico. A razão para esta escolha se deve por determinar o posicionamento após o folding possuir vantagens em área sobre o posicionamento estático. Além disto, como as linhas de difusão determinam limites para folding dos transistores P e N, isto elimina a necessidade do folding ser calculado dinamicamente.

Segundo esta metodologia, o problema pode ser definido como: Dada uma rede de transistores e os limites de altura das linhas de difusões P e N, substituir os transistores da rede com altura maior do que o limite estabelecido, por transistores menores, paralelos, de forma que a soma da suas larguras sejam igual à largura do transistor original.

Uma ilustração deste método é apresentada na Figura 4.5.

A execução do algoritmo de folding garante que todos os transistores possuam largura menor ou igual que o estabelecido para suas respectivas linha de difusão. Desta forma, nenhuma outra modificação na rede de transistores precisa ser feita nas etapas seguintes do fluxo.

4.5 Posicionamento

A função da etapa de posicionamento é encontrar um arranjo de transistores dentro da área da célula que atenda a um determinado objetivo. O objetivo mais freqüentemente encontrado nos algoritmos de posicionamento para o estilo 1D é o de encontrar o ordenamento e orientação dos transistores nas bandas P e N de forma que o número de quebras de difusões (GAPs) seja mínimo e que haja uma correspondência entre os sinais de *gate* dos transistores verticalmente alinhados nas duas difusões.

Diversos métodos exatos são capazes de encontrar a solução ótima (MAZIASZ; HAYES, 1991; IIZUKA; IKEDA; ASADA, 2004; GUPTA; HAYES, 1998) ou quase-ótima (IIZUKA; IKEDA; ASADA, 2005b) para este problema em tempo aceitável. Entretanto, estes métodos não consideram o uso de outras métricas de qualidade que podem levar a soluções ainda melhores tais como comprimento total das conexões e densidade do canal. Segundo (VAHIA; CIESIELSKI, 1999), o uso de métodos aleatórios tem tido grande aceitação entre pesquisadores devido a sua habilidade de lidar com funções de custos multidimensionais. Cellerity (GURUSWAMY et al., 1997) foi a primeira ferramenta a incorporar todas estas métricas num posicionador feito com *Simulated Annealing*.

O posicionador desenvolvido neste trabalho pode utilizar como heurística o algoritmo de caminho de Euler, o qual é capaz de encontrar rapidamente, e de forma exata, o posicionamento com o menor número de quebras de difusões, e também um método heurístico que faz uso de *Threshold Accept* para posicionar transistores com uma estrutura de rede mais complexa e de forma a incluir outras métricas de qualidade durante o posicionamento.

4.5.1 Especificação do Problema

O problema de posicionamento de transistores em 1D pode ser definido como: dada uma descrição da rede de transistores de uma célula, distribuir os transistores em duas listas distintas P e N, de acordo com o seu tipo, e encontrar um ordenamento e orientação dos transistores que minimize a área e melhores suas características elétricas. Para tanto, deve-se considerar que transistores vizinhos pertencentes a mesma lista e que compartilham o sinal da difusão podem ser posicionados sem a existência de GAPs, e que transistores de mesmo índice tendem a ser posicionados verticalmente alinhados durante a etapa de compactação da célula.

Células com quantidade diferente de transistores P e N são preenchidas com transistores *dummies* para que ambas as listas fiquem com o mesmo tamanho e todos os transistores tenham o seu par na lista oposta.

Os detalhes da implementação dos dois métodos de posicionamento desenvolvidos são descritos à seguir.

4.5.2 Caminho de Euler

Um método simples para encontrar um ordenamento ótimo para os transistores com o objetivo de reduzir o número de GAPs nas linhas de difusão é através do algoritmo de caminho de Euler (UEHARA; VANCLEEMPUT, 1981). O método consiste em encontrar

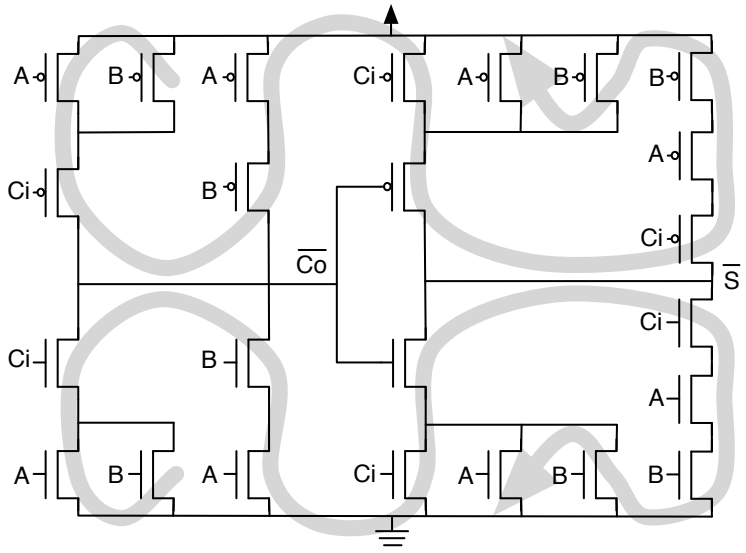


Figura 4.6: Exemplo de caminho de Euler em um somador completo

um caminho comum ininterrupto - com o mesmo ordenamento dos sinais de *gate* - para ambas as redes pull-up e pull-down que passe por cada transistor apenas uma vez. A estrutura de dados utilizada para execução deste algoritmo foi um grafo não orientado onde as arestas representam os transistores e os nós, as redes. Um exemplo de caminho de Euler na rede de transistores de um somador completo é mostrado na Figura 4.6.

Nem sempre é possível encontrar um único caminho de Euler para todo o grafo. Neste caso, a estratégia utilizada foi encontrar o menor número de sub-caminhos de Euler que juntos percorram todas as arestas do grafo apenas uma vez. Sempre que um sub-caminho termina, um GAP é gerado e um novo caminho é percorrido começando por qualquer outro par de arestas que ainda não tenham sido visitadas.

Uma limitação deste algoritmo é que apenas *netlist* de células que possuam o mesmo número de transistores nas redes P e N são posicionados com sucesso - como no caso da classe dos circuitos CMOS estático complementar. Alguns circuitos como multiplexadores e portas XNOR feitos com lógica de transistores de passagem também puderam ser posicionados utilizando caminho de Euler, mas somente quando possuíam esta característica na rede de seus transistores.

Um outro problema que surge com a utilização deste algoritmo em conjunto com a técnica de *folding* é que até mesmo circuitos CMOS estáticos poderão ter suas redes desbalanceadas e deixarem de ser amigáveis à realização do posicionamento com o caminho de Euler.

Devido à estas restrições, um novo posicionador precisou ser desenvolvido.

4.5.3 Threshold Accept

A heurística *Threshold Accept* (DUECK; SCHEUER, 1990) foi proposta para ser uma evolução do algoritmo conhecido como *Simulated Annealing* (KIRKPATRICK; GELATT; VECCHI, 1983). Segundo os autores, o método é mais simples e apresenta resultados aparentemente superiores ao *Simulated Annealing*.

O algoritmo começa com uma solução inicial qualquer (que pode ser aleatória) e executa perturbações com a finalidade de encontrar soluções próximas de acordo com o valor

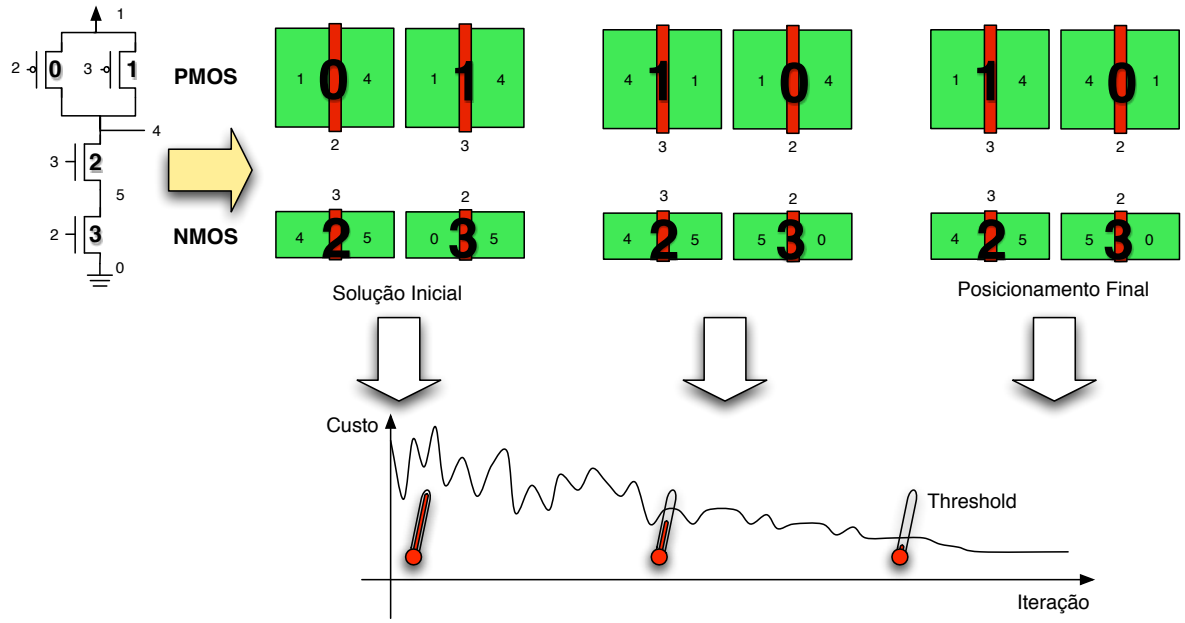


Figura 4.7: Exemplo de execução do algoritmo de posicionamento usando *Threshold Accept*

do *threshold*. Este valor define o quanto soluções piores devem ser aceitas e diminui ao longo das iterações. Por fim, quando o *threshold* for igual a zero, o algoritmo torna-se guloso e aceita somente perturbações que levem a soluções de igual ou menor custo que a atual. O algoritmo termina quando nenhuma melhora é detectada após várias iterações.

A Figura 4.7 mostra um exemplo de uma possível execução do algoritmo de *Threshold Accept* (TA) sobre um circuito, e a melhoria do posicionamento conforme o *threshold* diminui.

A versão do algoritmo de TA utilizada neste trabalho é a mesma de (HENTSCHKE, 2007). Hentschke desenvolveu um *template* em C++ que implementa as funções de minimização do algoritmo e exige apenas o desenvolvimento das funções de avaliação de custo e perturbação da estrutura que possui a solução do problema que deve ser minimizado. Este *template* caracteriza-se por utilizar um escalonador de *threshold* adaptativo para acelerar o decréscimo do *threshold* nos casos onde a taxa de aceitação for muito alta - o que pode indicar que o algoritmo fica alternando entre soluções aleatórias sem convergir para um decréscimo no custo - e diminuir a velocidade, quando a taxa de aceitação for menor.

Os dois métodos que precisaram ser implementados são descritos a seguir.

4.5.3.1 Função de Custo

A função de custo retorna uma medida qualitativa da solução ao longo das iterações do algoritmo de TA. Para solucionar o problema de posicionamento 1D de transistores, foram definidas 4 métricas para determinar a qualidade do posicionamento:

- Largura da célula - Este índice tem o objetivo de produzir um posicionamento com um menor número de GAPs e também de alinhar verticalmente os GAPs pertencentes a ambas difusões - o que em geral tem um impacto menor na largura da célula e melhora o seu roteamento interno. A largura da célula é dada pelo número máximo de elementos utilizados por cada linha de difusão. Cada transistor insere 3 elemen-

tos diferentes representando os seus 3 terminais (*source*, *gate* e *drain*) na sua respectiva difusão. A inexistência de GAPs entre transistores adjacentes faz com que apenas um elemento seja necessário para representar os terminais de *source*/dreno de ambos transistores. A ocorrência de GAP entre transistores de apenas uma difusão (na P ou na N), insere um elemento extra na posição correspondente da difusão oposta para que os próximos transistores permaneçam alinhados.

- Número de GAPs - É o número total de quebras de difusão existentes nas difusões P e N somados. Esta medida faz-se necessária, uma vez que GAPs verticalmente alinhados contribuem apenas uma vez para o aumento da largura da célula.

- *Gate Mismatches* - Quando os *gates* de dois transistores pertencentes a difusões diferentes são posicionados de forma alinhada, eles podem ser conectados de forma eficiente utilizando apenas uma conexão vertical de polisilício. Desta forma, o valor desta métrica é dado pelo número de transistores com sinal de *gate* diferente do seu equivalente na difusão oposta para cada posicionamento fornecido.

- Comprimento das conexões - Esta métrica realiza uma estimativa do tamanho final de todas as conexões dentro da célula. O tamanho de cada conexão é definido como a largura, em número de elementos (*gates*, difusões, GAPs), de cada rede da célula. Desta forma, um valor menor para esta métrica, deverá produzir como resultado um leiaute com conexões de menor tamanho. As conexões pertencentes às redes de alimentação não entram no cálculo por não adicionarem conexões horizontais dentro da célula. Na Figura 4.7 o posicionamento final teve um custo menor que o intermediário graças à este fator, o que tende a produzir posicionamentos com maior quantidade de conexões com a alimentação e com melhores características elétricas.

O modelo de representação de posicionamento utilizado e o processo de conversão para uma representação abstrata do leiaute da célula pré-roteamento é mostrado na Figura 4.8. Durante esta conversão, dois elementos extras são adicionados às extremidades da célula para facilitar o roteamento na etapa seguinte do fluxo.

Algumas métricas possuem uma contribuição maior do que as demais para a qualidade geral do posicionamento. Uma forma de contornar isto foi ponderando a contribuição de cada uma durante o calculo do valor final da função de custo. O peso usado para este cálculo foi experimentalmente determinado e o resultado é dado pela seguinte fórmula:

$$W = 4Wgm + 2Wg + 3Ww + Wwl$$

onde Wgm é o número de *gate mismatches*, Wg é o número de GAPs, Ww é a largura da célula (em número de elementos) e Wwl é o comprimento total das conexões.

4.5.3.2 Função de Perturbação

A função de perturbação tem o objetivo de expandir o espaço de soluções de forma que novas opções sejam experimentadas a cada iteração do algoritmo de TA.

Para encontrar incrementalmente novas soluções, o algoritmo desenvolvido modifica o posicionamento fornecido movendo a cada iteração, um conjunto contínuo de transistores de ambas as listas P e N ou uma de cada vez, conforme exemplo da Figura 4.9. Para cada lista selecionada, uma janela de tamanho menor ou igual ao número de transistores existentes na difusão é calculada. Dentro desta janela, dois tipos de movimentos com

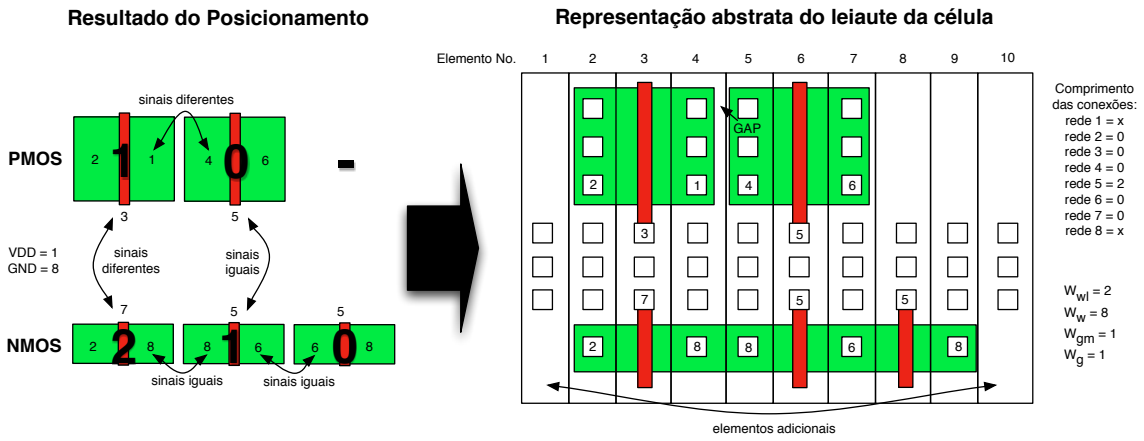


Figura 4.8: Exemplo de conversão de um posicionamento na estrutura de dados de roteamento com os custos associados

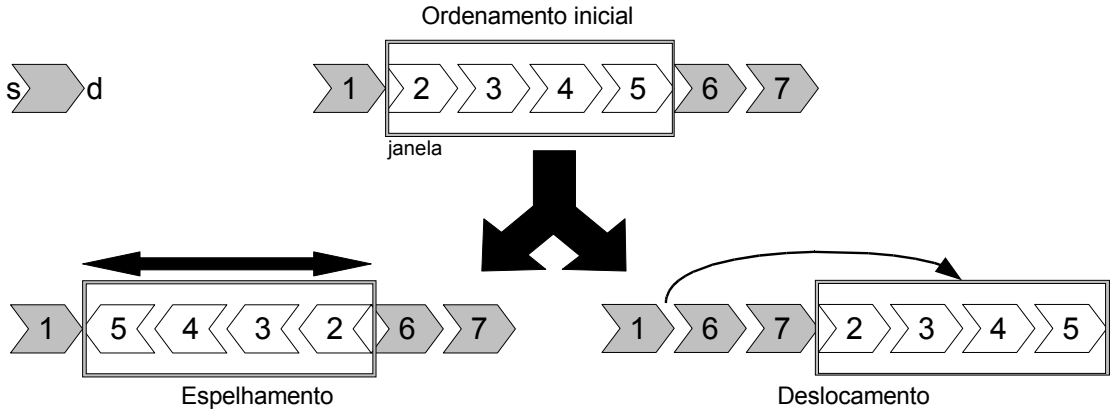


Figura 4.9: Movimentos implementados na função de perturbação do algoritmo de posicionamento de transistores

iguais chances de execução podem ser escolhidos. O primeiro movimento realiza a operação de deslocamento movendo todos os transistores da janela em direção ao início ou ao final da lista de transistores. O segundo movimento realiza a operação de espelhamento onde todos os transistores dentro da janela têm sua ordem e orientação invertida. Todos os parâmetros são aleatoriamente selecionados para uma melhor procura dentro do espaço de soluções.

4.5.4 Notas

O método de caminho de Euler, apesar de extremamente rápido na maneira como foi implementado, possui grande dificuldade na adoção de métricas além das de número de GAPs e número de *gates* desalinhado. Outro problema que surge é como modificar o algoritmo para tratar de transistores *dummies* (sem par) sem aumentar significativamente a complexidade do posicionador. Diversas tentativas foram feitas ao longo deste trabalho, mas todas sem sucesso.

O algoritmo de TA possui a facilidade de poder agrupar diversas medidas de qualidade e também de suportar virtualmente qualquer estrutura de célula. Apesar do tempo de execução do posicionamento ser, de uma forma geral, maior do que utilizando o algoritmo de caminho de Euler, esta situação se inverte quando a célula necessita mais de 2 GAPs.

Isto se deve à complexidade elevada do algoritmo que tem de testar todas as possibilidades de quebras de difusão para encontrar o ordenamento ótimo dos transistores.

Os melhores resultados do algoritmo de TA foram obtidos quando o algoritmo era executado com um *threshold* elevado (para forçar a criação de uma solução inicial aleatória). Uma tentativa de utilizar o resultado do algoritmo de caminho de Euler como solução inicial e em seguida executar o TA com um *threshold* inicial reduzido foi realizada, mas os resultados logo mostraram que o algoritmo tende a convergir rapidamente para mínimos locais e os resultados não eram tão bons quando comparados com a solução inicial aleatória.

4.6 Roteamento

Para realizar o roteamento, primeiramente é necessário converter o posicionamento fornecido em uma estrutura que contenha uma representação das partes do circuito que precisam ser conectadas e os lugares por onde podem passar conexões de acordo com o estilo de leiaute definido. A estrutura escolhida neste trabalho foi um grafo onde os nós representam os terminais dos transistores, sinais de entrada/saída e pontos de articulação das redes de roteamento, e as arestas representam os nós que devem ser conectados durante a etapa compactação do leiaute.

A Figura 4.10 mostra o modelo de grafo de roteamento utilizado e o que cada conexão representa na célula. Os nós terminais dos transistores (*gate*, fonte e dreno), sinais de alimentação e interfaces da célula são marcadas com a rede à qual pertencem para que o algoritmo de roteamento possa conectar todos os nós pertencentes à mesma rede sem a ocorrência de conflitos. Uma restrição que precisou ser definida foi marcar os nós da região das trilhas conectados aos terminais de *gate* dos transistores como pertencentes à mesma rede para evitar uma violação das regras de desenho entre a extensão do *gate* e uma outra rede que viesse a ocupar o nó.

O resultado do roteamento do grafo representa a forma como este deve ser feito na célula. A existência ou não de conexões entre dois nós, assume diferentes significados no momento da geração do leiaute:

- Conexões entre dois nós de mesma camada indicam que um caminho deve ser traçado com o respectivo material entre estes dois pontos;

- Conexões entre nós das camadas de metal 1 e polisilício ou difusão indicam a existência de um contato;

- Ligação de um nó sobre as difusões com VDD ou GND faz com que um caminho em metal 1 para a alimentação tenha que ser criado;

- Um nó de metal 1 na região de trilhas, conectado ao nó de interface representa a inserção de um pino de entrada/saída da célula neste ponto.

Pesos diferentes foram atribuídos às arestas para priorizar o uso de conexões que possuam menor resistividade e/ou que tenham menor impacto de área. De uma forma geral, arestas que representam vias/contatos possuem peso maior que as de polisilício que por sua vez possuem peso maior que as de metais.

Para efetuar o roteamento da célula, foi utilizado uma versão modificada do algoritmo de roteamento baseado em negociação de congestionamento chamado PathFinder detalhado a seguir.

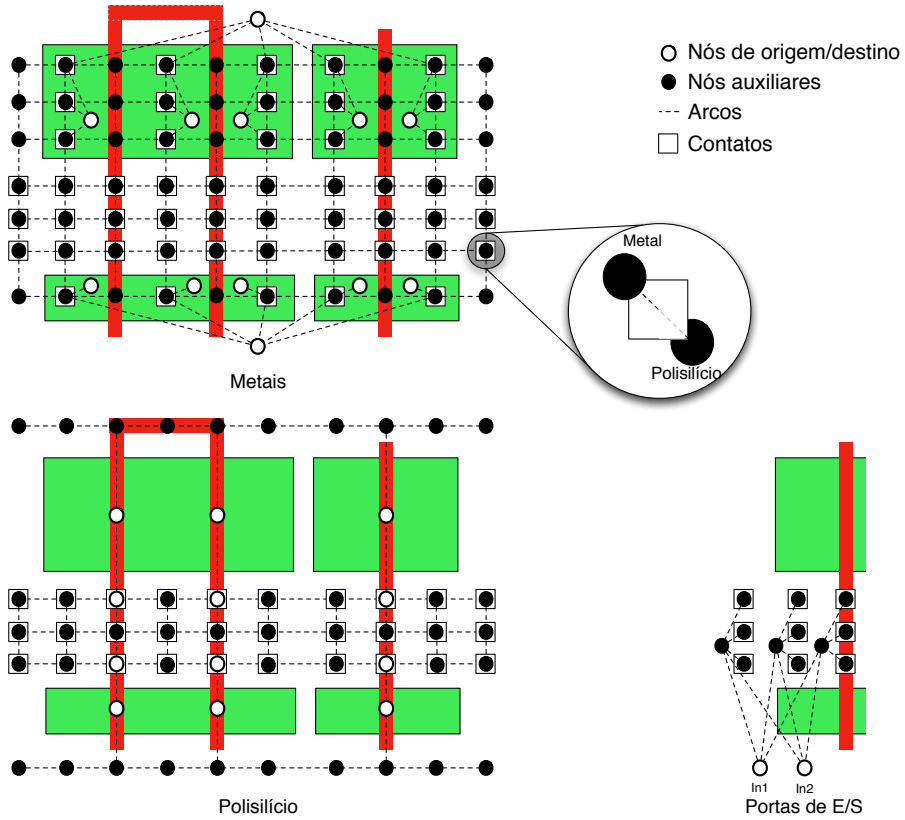


Figura 4.10: Modelo de grafo utilizado para roteamento interno da célula

4.6.1 O algoritmo para roteamento PathFinder

O algoritmo PathFinder (MCMURCHIE; EBELING, 1995) foi originalmente concebido para a realização de roteamento em FPGAs mas cuja técnica pode ser aplicada para solução de outros problemas de roteamento. Por ser um algoritmo baseado na negociação de congestionamento, ele possui como característica a capacidade das redes competirem por nós que estejam congestionados. Redes com maior dificuldade em traçar caminhos alternativos para realizar suas conexões vencem a disputa com as demais, produzindo, segundo o autor, um resultado melhor globalmente - quando comparado à outros algoritmos estilo *rip-up and re-route* - e próximo ao ótimo.

O roteador é composto de duas partes semi-independentes: um roteador de sinal e um roteador global. O roteador de sinal realiza a conexão entre os nós pertencentes à mesma rede - o algoritmo desenvolvido utiliza o método de Lee (LEE, 1961) para realizar a procura de sinais. Já o roteador global chama o roteador de sinal diversas vezes para conectar todas às redes do circuito e ajusta o custo de compartilhamento dos nós baseado na demanda por sinais àquele recurso.

O roteamento de sinal é feito fornecendo uma lista de nós origem e uma rede de destino. O algoritmo executa uma busca em largura, expandindo primeiro os nós de menor custo acumulado em relação à origem, até encontrar o primeiro nó pertencente à rede desejada. O roteamento de redes com grau maior que dois é feito conectando dois nós quaisquer na primeira iteração, e utilizando a lista dos nós obtidos como resultado, como origem para as iterações seguintes.

O custo de usar um determinado nó n durante uma iteração do roteador de sinal é dado pela fórmula:

$Cn = (Bn + Hn) * Pn$

onde Bn é o custo base da aresta que chega ao nó n, Hn é o histórico de congestionamento de n em iterações passadas e Pn é a quantidade de sinais atualmente utilizando n.

Durante a primeira iteração, Pn é inicializado com 1, assim nenhuma penalidade é imposta pelo uso de n independente de quantos sinais ocupem o nó. Nas iterações seguintes, Pn é incrementado gradualmente, dependendo de quantos sinais ocupam n, fazendo com que algumas redes desistam e encontrem uma outra rota de menor custo.

A chave do algoritmo é o fator Hn. A cada iteração que n é compartilhado, Hn é incrementado lentamente. Seu efeito é de permanentemente aumentar o custo da utilização dos nós congestionados de forma que rotas alternativas sejam tentadas.

A métrica Pn é importante para acelerar a execução do algoritmo. Ela insere um fator de ordem na realização das conexões que serve como um critério de desempate entre redes que disputam o mesmo nó e também diminui as chances de duas ou mais conexões desistirem do nó ao mesmo tempo. Entretanto, um acréscimo muito abrupto de Pn, pode fazer com que as redes desistam muito rapidamente de nós congestionados, eliminando a competição, e tornando o algoritmo demasiadamente sensível à ordem de realização das conexões tal como o esquema de rip-up and re-route padrão.

O roteamento termina quando todas as redes forem roteadas sem que ocorra nenhum conflito entre elas ou, em caso de falha, quando atingir um número limite de tentativas sem sucesso.

4.6.2 Notas

Esta implementação do algoritmo de roteamento de sinal, apesar de extremamente rápida e capaz de produzir resultados razoáveis, não garante a obtenção de resultados ótimos, ou próximos disto.

Para solucionar este problema, encontra-se em fase de desenvolvimento a inclusão da técnica de Iterated 1-Steiner (KAHNG; ROBINS, 1990) que, segundo o autor, deve permitir a obtenção de resultados com comprimento de conexões em média 11% menores. A heurística A* também deve ser usada em conjunto para compensar o aumento da complexidade do algoritmo.

Contatos redundantes na difusão costumam melhorar as características elétricas e também de fabricação das células. O número ideal de contatos varia de acordo com a tecnologia utilizada e a posição na célula. A rotina de inserção de contatos extras não chegou a ser finalizada a tempo mas está sendo desenvolvida como um pós-processamento ao roteador. O objetivo é inserir automaticamente contatos na difusão e os conectar a suas respectivas redes sempre que não houver obstrução no nível de metal 1.

4.7 Compactação

A compactação é o último passo para geração do leiaute do circuito. As geometrias que compõem o desenho da célula finalmente são instanciadas e posicionadas de acordo com o resultado obtido nas etapas de posicionamento e roteamento. Além disto, restrições são adicionadas para tornar o leiaute compatível com o estilo de leiaute definido e algoritmos de minimização são utilizados para diminuir a largura da célula e o tamanho das conexões.

O resultado das etapas anteriores de geração é suficiente para prover uma informação relativa da posição de cada polígono que forma o leiaute da célula. Isto permite que este

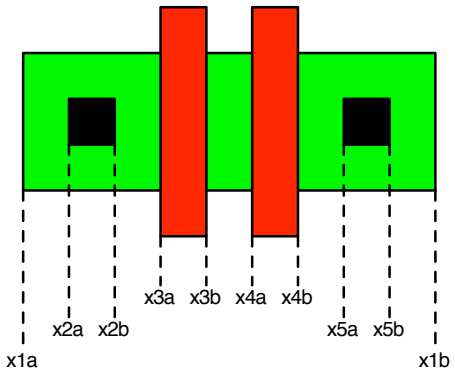


Figura 4.11: Variáveis criadas para compactação com programação linear

problema possa ser modelado como um conjunto de equações lineares que pode ser resolvido de forma simples através de um resolvedor de equações lineares. A técnica utilizada neste trabalho para resolver o problema de compactação de leiaute com programação linear com inteiros (ILP) é explicada a seguir.

4.7.1 Programação Linear com Inteiros

Tendo pronto o posicionamento dos transistores e seu roteamento, já é possível instanciar as estruturas que formam o leiaute do circuito.

Inicialmente, estas estruturas são desprovidas de posição e tamanho, apenas sua camada é definida. Em seguida, uma variável é associada a cada lado dos elementos geométricos do leiaute que necessitam ser compactados (borda esquerda e direita para compactação horizontal e limite superior e inferior para compactação vertical). Uma vez que a posição relativa dos polígonos é conhecida, é possível descrever a relação entre as variáveis na forma de equações lineares. As regras que definem os espaçamentos entre estas variáveis são descritas no arquivo de tecnologia. A Figura 4.11 ilustra um pequeno exemplo onde as equações necessárias para compactar o seu leiaute são descritas a seguir:

```
01:  x2a - x1a >= EDFCT;
02:  x2b - x2a =  WCT;
03:  x3a - x2b >= SCTP1;
04:  x3b - x3a =  WP1;
05:  x4a - x3b >= SP1P1
06:  x4b - x4a =  WP1;
07:  x5a - x4b >= SCTP1;
08:  x5b - x5a =  WCT;
09:  x1b - x5b >= EDFCT;
```

onde EDFCT é a envoltória mínima de difusão no contato, WCT é a largura do contato, SCTP1 é o espaçamento mínimo entre contato e polisilício, WP1 é a largura do *gate* e SP1P1 é o espaçamento mínimo entre *gates*.

O resolvedor de equações lineares utilizado para solucionar o sistema é o LPSolve (LPSOLVE, 2007). Ele consegue resolver este tipo de sistema com variáveis contínuas de forma extremamente rápida e com complexidade linear.

Entretanto, existem situações especiais que exigem o uso de variáveis discretas como no caso do posicionamento das portas de entrada/saída na grade de roteamento. A Figura

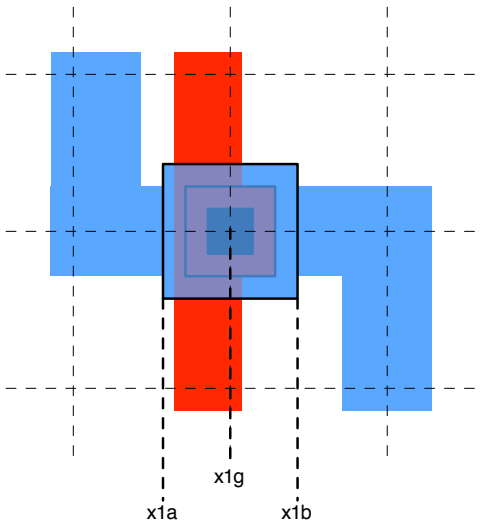


Figura 4.12: Alinhamento das portas de entrada/saída das células à grade de roteamento

4.12 ilustra um exemplo de leiaute de uma porta de interface de uma célula onde o código utilizado para resolver o problema de alinhamento à grade é descrito a seguir:

```
01:  x1g - hGrid = Grid x1gpos;
02:  x1g - x1a =  HPWidth;
03:  x1b - x1g =  HPWidth;
04:  int: x1gpos;
```

onde hGrid é a metade da largura do *pitch* do Grid (devido ao *offset* da grade horizontal na célula) e HPWidth é a metade da largura de metal 1 necessária para comportar uma via para metal 2.

Forçando x1gpos a ser uma variável inteira, a equação da linha 01 obriga que x1g fique em uma posição múltipla do Grid de roteamento. As linhas 02 e 03 determinam as laterais da porta em relação ao ponto central x1g.

O problema do uso de variáveis discretas é que a complexidade do resolvedor em solucionar o sistema se torna exponencial. No caso da existência de um número elevado destas variáveis, o problema pode se tornar muito custoso computacionalmente e não terminar num tempo aceitável. Esta limitação não chegou a ser um problema grave para a geração de células pois o número de variáveis inteiras não costuma ser grande e a compactação termina em poucos segundos mesmo para o caso de células com muitos pinos de entrada/saída.

4.7.1.1 A função objetivo

Um projetista ao desenhar um circuito, possui uma lista de objetivos que ele tem como prioridade de minimização. Os objetivos mais comuns são:

- *Área* : Em uma biblioteca de células, normalmente todas as células possuem a mesma altura. Desta forma, a minimização da largura é a única minimização que pode ser aplicada, a menos que todas as células sejam redimensionadas para uma nova altura.

- *Difusões* : Por possuir uma resistência alta, a redução das difusões dos transistores é importante para melhorar as suas características elétricas.

- *Roteamento em Polisilício e Metal* : O metal possui uma resistência consideravelmente inferior à resistência do polisilício e por esta razão as conexões feitas em polisilício costumam ter uma prioridade maior de minimização do que as de metal.

A largura da célula mostrada na Figura 4.11 pode ser minimizada adicionando a variável mais a direita do desenho na função objetivo do resolvedor:

```
10:  min: x1b;
```

A maneira encontrada para priorizar a minimização de determinados objetivos em detrimento de outros (de maneira similar à forma que um projetista faria) no problema de geração de células, foi através da atribuição de pesos às variáveis de minimização. Isto é feito multiplicando o peso da variável pelo seu valor na função de minimização. Desta forma, para dar maior prioridade na minimização das conexões em polisilício sobre as de metal, bastaria colocar:

```
min: 2 PWIDTH + MWIDTH;
```

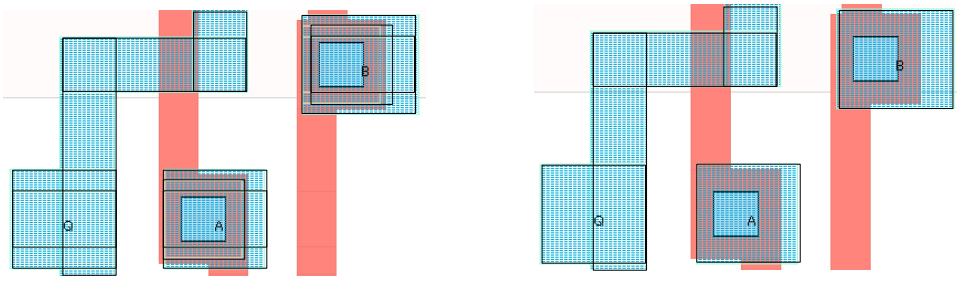
onde PWIDTH é a largura da conexão em polisilício e MWIDTH é a largura da conexão em metal.

O compactador desenvolvido neste trabalho consegue realizar compactação nos eixos X e Y simultaneamente. De uma forma geral, todas as posições X dos elementos do leiaute são determinados pelo resultado da resolução do sistema de equações lineares. A compactação no eixo Y é feita somente para determinar a posição das linhas de roteamento em polisilício nas regiões superior e inferior da célula para que fiquem o mais próximas possíveis dos transistores às quais estão conectadas.

4.7.2 Otimização pós-compactação

Após a geração do leiaute, muita informação redundante é armazenada à estrutura de dados com o leiaute do circuito. Para economizar memória e também diminuir o tamanho do arquivo de saída da ferramenta, foi desenvolvido um algoritmo para unir retângulos pertencentes a uma mesma camada que estejam sobrepostos ou justapostos. Cada retângulo pertencente a mesma camada, é comparado a todos os demais a procura de algum outro que satisfaça estas condições. Para cada ocorrência encontrada, o algoritmo elimina a redundância e começa novamente a partir do primeiro retângulo da camada atual. O algoritmo só termina quando nenhuma nova redundância é encontrada e todas as camadas terem sido pesquisadas.

Esta otimização permitiu reduzir o número de linhas necessárias para descrever uma célula NAND2 de 101 para 73 linhas, o que representou uma diminuição de aproximadamente 28% no tamanho do arquivo. A Figura 4.13 mostra o resultado do uso desta otimização na região das trilhas desta célula.



(a) Leiaute original (b) Leiaute otimizado

Figura 4.13: Remoção de elementos redundantes no leiaute de uma célula NAND com 2 entradas

5 DESENVOLVIMENTO DE UM COMPILADOR DE PARTE OPERATIVA

Este capítulo apresenta o compilador de parte operativa Jungle Parrot, desenvolvido neste trabalho. Detalhes do seu projeto como descrição dos arquivos de entrada, posicionamento das células e roteamento são informados.

5.1 Formulação do problema

O compilador/montador de PO é uma ferramenta que recebe como entrada uma descrição de PO contendo os módulos que devem ser gerados com suas interconexões, e uma biblioteca de leiautes de células CMOS. O montador realiza o posicionamento das células da biblioteca e o seu roteamento de forma a implementar a lógica correspondente à descrição recebida e por fim o gera o leiaute do circuito correspondente como resultado.

O fluxo do montador desenvolvido é mostrado na Figura 5.1.

5.2 Formato para especificação de Parte Operativa

Foram estudadas diversas formas para especificação da estrutura de uma PO. Taliercio (TALIERCIO; FOLETTO; LICCIARDI, 1991) propõe o uso de esquemáticos para esta função. Já Matsumoto (MATSUMOTO et al., 1990) utiliza um leiaute simbólico hierárquico da rede de transistores.

O tipo de descrição utilizado neste trabalho é através de um arquivo textual que contém a lista dos módulos que devem ser gerados junto com os parâmetros de geração de forma similar ao formato desenvolvido por Batten (BATTEN, 2007). Cada módulo gerado é associado a um label. O roteamento entre os módulos é feito através de comandos que conectam as portas de interface dos módulos à uma rede externa. Estas portas são acessadas através do label dos módulos.

O código necessário para produzir a PO cujo diagrama é mostrado na Figura 5.2 é listado a seguir:

```
OPTIONS_SECTION
circuit_name  somador
rules_file    tech_035.rul
set_grid      1.5 1.5
supply_size   2.5
cif_file      somador.cif
cellsnet_file cellgen.sim
```

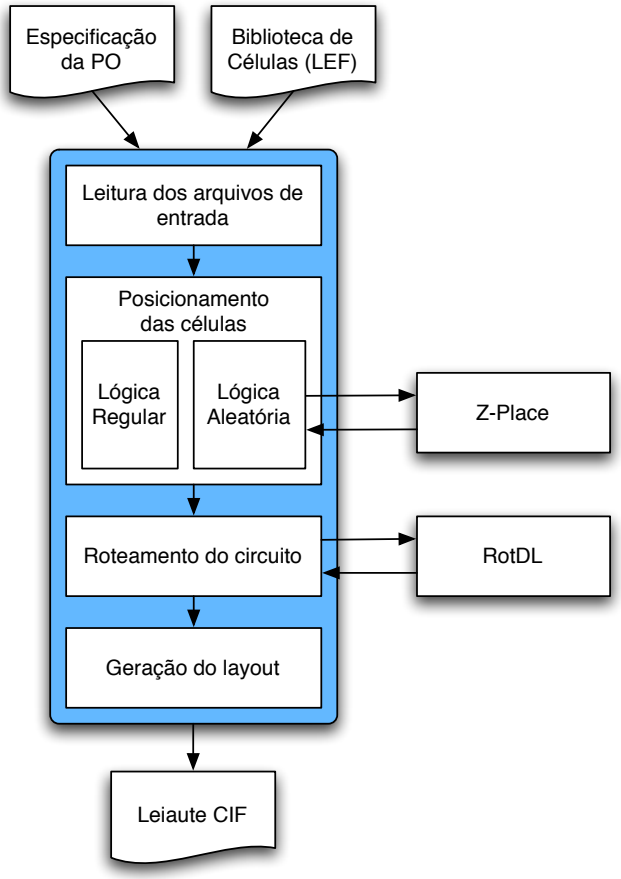


Figura 5.1: Fluxo da ferramenta de geração de Parte Operativa

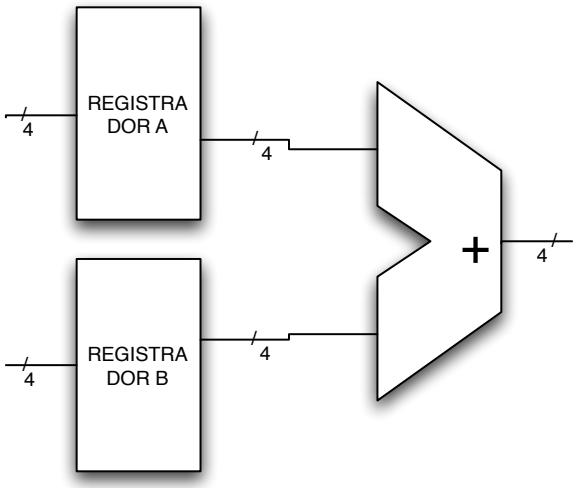


Figura 5.2: Exemplo de circuito de uma parte operativa

```
cellslib_file cellgen.lef
cellsHeight   9
nrRows        4
show_grid     yes

BUILDERS_SECTION
powerline  2
register   4 rega dp1
register   4 regb dp1
adder      4 add  add31

NETLIST_SECTION
connect    rega.q 0 3 to add.a 0 3
connect    regb.q 0 3 to add.b 0 3
```

A seção OPTIONS define a configuração do gerador de PO. Os parâmetros suportados são listados na Tabela 5.1.

Tabela 5.1: Parâmetros de configuração do gerador de parte operativa

Parâmetro	Tipo de dado	Descrição
circuit_name	string	Nome do circuito
rules_file	string	Nome do arquivo que contem as regras de desenho
set_grid	float float	Largura do pitch horizontal e vertical da grade de roteamento em micro metro
supply_size	float	Largura da linha de alimentação horizontal em micro metro
cif_file	string	Nome do arquivo de saída contendo o leiaute da PO
cellslib_file	string	Nome do arquivo contendo a biblioteca de células
cellsHeight	unsigned int	Altura das células da biblioteca em passos da grade de roteamento
nrRows	unsigned int	Número de bandas no leiaute do circuito
show_grid	yes/no	Imprime a grade de roteamento em metal 4

A seção BUILDERS descreve e configura os módulos que deverão ser produzidos no leiaute da PO desejada. O ordenamento dos módulos listados nesta seção reflete diretamente na ordem com que as células que implementam estes módulos aparecerão no leiaute final.

Até o presente momento, os seguintes geradores de módulos já foram implementados:

- *POWERLINE p1* - Desenha uma linha de alimentação vertical em metal 2 no circuito para fornecer corrente às linhas horizontais que alimentam as células em metal 1. O parâmetro p1 especifica a largura das linhas de VDD e GND em metal 2.
- *BITLOGIC nr_bits label p1* - Cria um operador lógico (AND, OR, XOR) utilizando a célula p1 da biblioteca.
- *REGISTER nr_bits label p1* - Gera um registrador utilizando a célula p1.

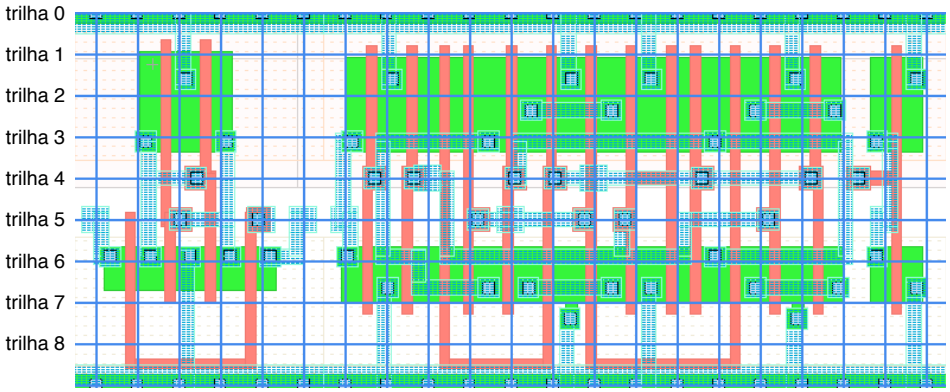


Figura 5.3: Ordenamento das trilhas de roteamento dentro da célula

- *RCADDER nr_bits label p1* - Chama o método para geração de um somador ripple-carry utilizando a célula p1.
- *CARRYSAVE nr_bits label p1 p2 p3* - Gera um multiplicador do tipo carry-save. Os parâmetros p1, p2 e p3 representam as células do somador completo, meio somador e nand2 respectivamente que devem ser utilizadas.
- *MUX nr_bits label p1* - Gera um multiplexador 2:1 com a célula p1.
- *SHIFTER nr_bits label p1* - Gera um deslocador utilizando a célula p1.
- *RANDOMLOGIC p1* - Gera um bloco de lógica aleatória de acordo com a especificação do *netlist* SPICE fornecido. O parâmetro p1 define a percentagem de espaços em branco que deve ser adicionado à área das células antes de efetuar o posicionamento.

Por fim, a seção NETLIST especifica as interconexões entre os pinos de entrada/saída dos módulos gerados. Os comandos aceitos até o presente momento são:

- *CREATENET p1 p2 p3 p4* - Cria um barramento em metal 3 que atravessa a PO horizontalmente. O parâmetro p1 identifica o barramento através de um label. A largura do barramento é indicada por p2 e p3. A trilha da grade de roteamento por onde o barramento deve passar é apontada pelo parâmetro p4 conforme ordenamento mostrado na Figura 5.3.
- *CONNECT p1 p2 p3 TO p4 p5 p6* - Conecta dois intervalos de pinos pertencentes à redes diferentes. Os parâmetros p1 e p4 indicam respectivamente as redes de origem e destino enquanto o intervalo é representado por [p2,p3] e [p5, p6].

Uma ilustração com o estilo de posicionamento das células produzido pelos geradores de módulos e o fluxo de sinais ao longo da PO para o circuito de exemplo é mostrada na Figura 5.4.

5.3 Posicionamento

O posicionamento das células pode ser produzido de duas formas distintas: de forma determinística pelo gerador dos módulos ou heurística através de um algoritmo de posicionamento.

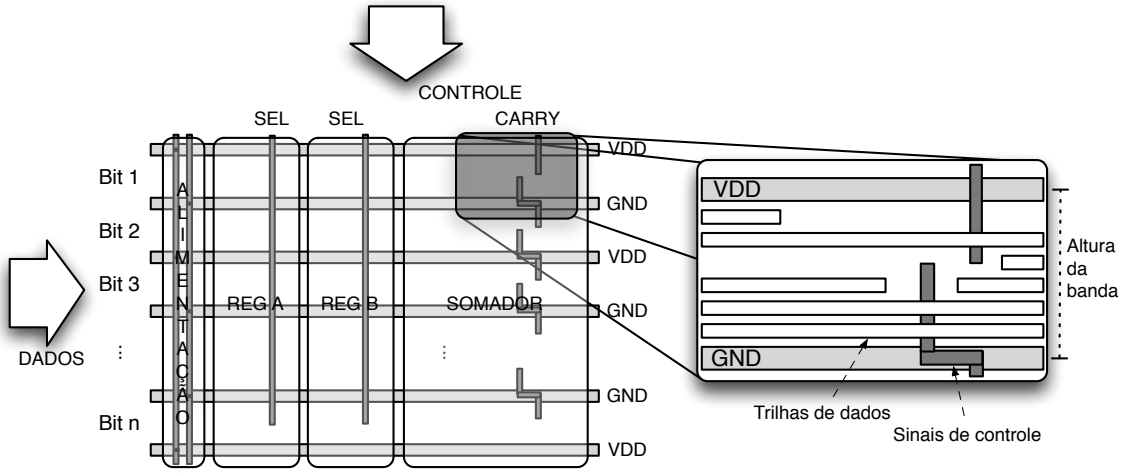


Figura 5.4: Estilo de leiaute do gerador de Parte Operativa

As células instanciadas pelos geradores de módulos são posicionadas automaticamente pelo compilador. Como a maioria das funções comumente encontradas em POs possui uma estrutura regular (registradores, multiplexadores, somadores, etc.) , algoritmos relativamente simples podem ser utilizados para esta função. Muitas vezes a mesma célula é instanciada na mesma posição mudando apenas a sua banda.

Os módulos de lógica aleatória são posicionados de forma automática utilizando a ferramenta Z-Place (HENTSCHKE et al., 2006) que realiza um posicionamento quadrático das células do circuito. O posicionamento obtido é interpretado e as células são posicionadas dentro da PO de acordo com as coordenadas indicadas pelo posicionador.

Em ambos os casos, o posicionamento é feito de forma espelhada ao longo das bandas para que as células possam compartilhar as linhas de alimentação e o poço. O espelhamento é feito posicionando as células pertencentes à bandas pares invertidas no eixo Y com relação às células das bandas ímpares.

Após o posicionamento, várias linhas de metal 1 atravessando horizontalmente todo o circuito são criadas para garantir a continuidade das linhas de alimentação.

5.4 Roteamento

O roteamento é realizado através de um roteador externo chamado RotDL (FLACH; HENTSCHKE; REIS, 2004). Este roteador caracteriza-se por necessitar de uma grade de roteamento para poder funcionar de forma adequada. O que o gerador de PO faz é produzir um arquivo com a descrição da posição dentro desta grade de cada um dos pinos que necessitam ser conectados junto com a rede a qual pertencem. Em seguida, o roteador é executado e, caso consiga realizar todas as conexões, um arquivo no formato CIF é produzido com o resultado do roteamento.

Devido a forma como o posicionamento das células foi planejado, o fluxo dos sinais dentro da PO segue um padrão bem definido. Os sinais de dados normalmente atravessam o circuito na horizontal passando por todos operadores pertencentes a um mesmo *bit* enquanto os sinais de controle cruzam o circuito na vertical atingindo todos os bits de um mesmo operador. Internamente aos operadores o fluxo dos dados intermediários (internos ao processamento do operador) pode se dar nos dois sentidos.

O roteador produz como resultado um leiaute que em seguida é agregado ao resultado

do compilador para formar o leiaute completo do circuito roteado.

5.5 Geradores de módulos

De uma forma geral, partes operativas são constituídas de um conjunto de blocos funcionais e cada bloco funcional é constituído de um conjunto de células. Os geradores de módulos oferecem uma maneira prática e eficiente de adicionar novos blocos funcionais à uma PO. Os geradores descrevem o bloco na sua forma estrutural informando a posição de suas células, a forma como devem ser conectadas, e os sinais das suas interfaces. O compilador de parte operativa interpreta esta descrição e produz o leiaute correspondente de forma automática.

O código que produz um registrador parametrizado, com n bits de largura é mostrado abaixo:

```
void DesignMng::Register(int n, string label, string cell){
  map<string,string> tmp;
  int p, start=getNextPosGrid();
  for(int c=0; c<n; c++){
    p=intToStr(c);
    tmp.clear();
    tmp["D"] = label+".D."+p;
    tmp["C"] = label+".C";
    tmp["Q"] = label+".Q"+p;
    tmp["QN"] = label+".QN"+p;
    insertCellInst(label+"."+p, cell, tmp);
    placeCell(label+"."+p,start,cellsHeight*c,c\%2,false);
  }
}
```

Com o uso deste gerador, o projetista pode instanciar quantos registradores ele desejar no seu projeto a partir do arquivo de descrição de PO.

Esta simplicidade na descrição do código dos geradores permitiu que módulos mais complexos como o do multiplicador carry-save fossem descritos utilizando apenas 48 linhas de código em linguagem C++.

6 RESULTADOS

Este capítulo apresenta os resultados obtidos pelas ferramentas de geração automática de células (CellGen) e de geração de parte operativa (Jungle Parrot) descritas nos Capítulos 4 e 5 respectivamente. Todos os circuitos apresentados neste capítulo foram desenvolvidos para uma tecnologia de 0,35μm com 3 camadas de metal.

6.1 Gerador Automático de Células CMOS

Uma comparação dos leiautes produzidos automaticamente com o resultado obtido por diferentes métodos, sejam gerados automaticamente ou feitos-à-mão, foi realizada. O resultado encontrado em cada uma das comparações é discutido em detalhes a seguir.

6.1.1 Comparação com Iizuka

O gerador de células desenvolvido por IIzuka (IIZUKA; IKEDA; ASADA, 2004) utiliza satisfabilidade booleana para realizar o posicionamento de células complementares. Esta ferramenta possui um estilo de leiaute similar ao desenvolvido neste trabalho, mas não utiliza técnicas avançadas de otimização para fazer a compactação do leiaute como por exemplo inserção de *dog-legs* para redução do espaçamento entre *gates*. Por outro lado tem-se a garantia da obtenção de um posicionamento com o menor número de GAPs possível.

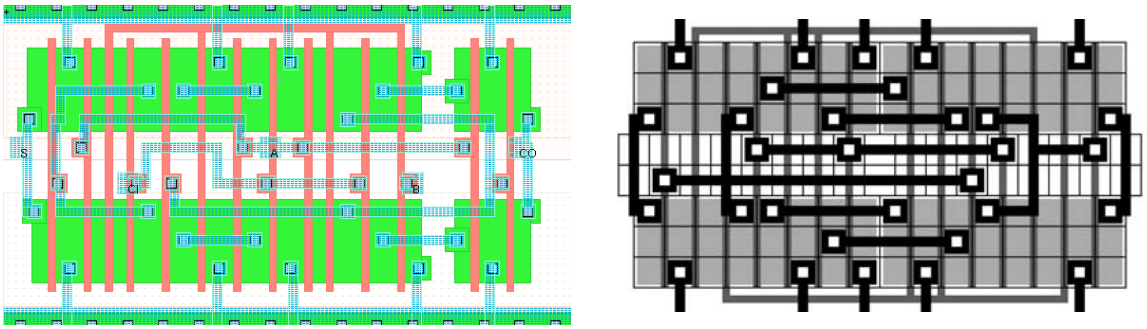
Neste artigo, Iizuka comparou o seu trabalho com Gupta e Hayes em (GUPTA; HAYES, 1996) e provou que o seu método é capaz de produzir circuitos com igual ou menor largura e num tempo de execução até uma ordem de grandeza inferior. Além disto, os leiautes produzidos também apresentaram resultados próximos mas ligeiramente inferiores em área que os gerados pela ferramenta comercial ProGenesys v4.2 (PROGENESYS, 2007).

A Tabela 6.1 mostra uma comparação feita com base nos dados divulgados por IIzuka neste trabalho.

O principal que se pode perceber com estes dados, é que o número de GAPs encontrados pelo nosso gerador foi sempre mínimo, mesmo utilizando um método de posicionamento estocástico. Os resultados de tempo de execução de IIzuka não consideram o tempo gasto com a compactação das células e servem apenas para comparar a diferença de complexidade entre os dois algoritmos uma vez que foram executados em plataformas de hardware diferentes: Iizuka utilizou uma estação UltraSPARC-III 750MHz com 2Gb de RAM enquanto todos os testes com a ferramenta CellGen foram executados em um Dual 2 GHz PowerPC G5 com 1Gb de RAM. Iizuka não disponibilizou outros dados que pudessem ser comparados.

Tabela 6.1: Comparação com IIzuka

		Iizuka		CellGen	
Célula	# Trans.	# GAPs	Tempo (s)	# GAPs	Tempo (s)
INV	2	0	0,05	0	0,1
BUFFER	4	0	0,06	0	0,32
NAND2	4	0	0,04	0	0,35
NOR3	6	0	0,06	0	1,9
NAND4	8	0	0,04	0	2,3
NOR4	8	0	0,04	0	2,1
XOR2	10	1	0,12	1	4,49
OAI21	8	0	0,04	0	6,5
HAD1	14	2	18,35	2	11,9
FAD1	28	1	305,76	1	241,8



(a) Gerador de células (b) Iizuka

Figura 6.1: Comparação do leiaute de dois somadores

A Figura 6.1 mostra uma comparação com o leiaute de um somador completo gerado automaticamente pela nossa ferramenta e a de Iizuka. O somador produzido pelo nosso gerador apresentou melhora de 15,4% no comprimento horizontal das conexões. Isto é devido principalmente ao fato de Iizuka não ter utilizado nenhuma técnica para redução do comprimento das conexões na função de custo do seu posicionador de transistores.

6.1.2 Comparação com Células *Standard-Cell*

Bibliotecas de células *standard-cell* são leiautes de células normalmente feitas-à-mão por projetistas experientes, e que por esta razão, costumam ter boas características de atraso, consumo de potência, DFM e área.

A comparação com as células *standard-cells* foi feita utilizando o *netlist* SPICE extraído destas células como entrada para a nossa ferramenta de síntese. Todas as células produzidas como resultado da geração possuem um *netlist* equivalente (exceto pela aplicação eventual de folding) e com o mesmo dimensionamento de transistores das *standard-cells*. O passo da grade vertical utilizado no gerador foi de $1.4\mu m$ contra $1.3\mu m$ das *standard-cells*. Isto foi feito para que pinos de E/S vizinhos verticalmente (o que não era permitido nas *standard-cells*) pudessem ser posicionados sem violar nenhuma regra de desenho. A Figura 6.2 mostra alguns exemplos de leiautes obtidos pelo processo de geração automática da nossa ferramenta.

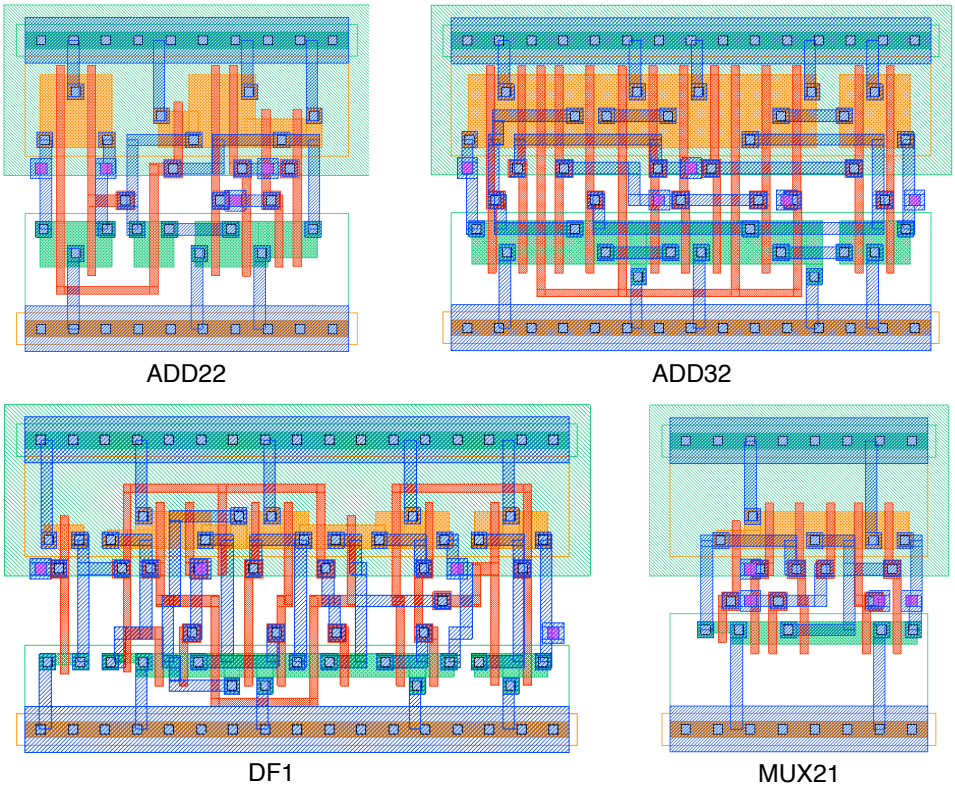


Figura 6.2: Leiautes de células geradas automaticamente

Os resultados obtidos em área entre as células *standard-cell* e as suas equivalentes, geradas automaticamente, são apresentados na Tabela 6.2.

O número de GAPs informado refere-se à quantidade de quebras de difusão das difusões P e N somadas. O número de transistores dos circuitos gerados pelo Cellgen difere em alguns casos do das *standard-cells* devido à aplicação de *folding*.

De uma forma geral, o número de GAPs encontrado nas células produzidas pelo gerador foi sempre mínimo, ou muito próximo disto. Algumas células apresentaram mais GAPs que as *standard-cells* devido ao fato de terem precisado realizar folding em alguns dos seus transistores para manter o seu dimensionamento especificado e atender os requisitos de altura da célula, ou para reduzir o comprimento total das conexões.

Analisando os resultados, percebe-se que as células que apresentaram folding foram as que obtiveram os piores resultados com acréscimo médio de área de 49,4% contra 7,5% das demais. A principal razão para isto é que as células *standard-cell* utilizam técnicas para roteamento do sinal de *gate* em duas dimensões para aumentar a largura dos transistores sem precisar aplicar folding como pode ser visto nos recortes da Figura 6.3. Esta técnica deverá ser experimentada nas próximas versões do gerador..

6.1.2.1 Simulação

Alguns dos leiautes testados foram extraídos e simulados para obtenção dos atrasos e consumo de potência. As extrações foram todas realizadas com a inclusão das capacitâncias parasitas utilizando a ferramenta Virtuoso da Cadence. Todas as células passaram por uma verificação de DRC antes de serem simuladas para garantir a conformidade do leiaute produzido com as regras de desenho da tecnologia.

As simulações dos leiautes extraídos foram feitas utilizando a ferramenta Spectre tam-

Tabela 6.2: Comparação de área com *standard-cell*

	Std-cell (Alt.=13μm)			Jungle Parrot (Alt.=12,6μm)				Ganho
Célula	#Tr.	GAPs	Largura	#Tr.	GAPs	Tempo	Largura	Área (%)
INV0	2	0	2,8μm	2	0	0,1s	2,8μm	3,08
NAND21	4	0	4,2μm	4	0	0,3s	4,2μm	3,08
AOI210	6	0	5,6μm	6	0	1,5s	7μm	-21,15
AOI312	8	0	8,4μm	15	1	10,7s	12,6μm	-45,38
NAND40	8	0	7μm	8	0	2,3s	7μm	3,08
OAI222	8	0	7μm	13	1	7,1s	12,6μm	-74,46
NOR33	9	0	9,8μm	15	2	7,6s	14μm	-38,46
OAI312	10	0	8,4μm	15	1	29,8s	12,6μm	-45,38
XOR20	10	1	9,8μm	10	1	4,5s	9,8μm	3,08
MUX21	12	0	8,4μm	12	0	7,3s	9,8μm	-13,08
ADD22	14	2	11,2μm	14	3	11,9s	14μm	-21,15
XNR30	20	4	15,4μm	20	3	13,8s	16,8μm	-5,73
DF1	26	7	21μm	26	4	103,4s	23,8μm	-9,85
MUX41	26	4	18,2μm	26	4	55,3s	21μm	-11,83
ADD31	28	4	21μm	28	2	132,9s	21μm	3,08
ADD32	28	4	21μm	28	2	241,8s	21μm	3,08
XOR41	30	9	21μm	30	4	52,7s	25,2μm	-16,31
JK1	34	11	26,6μm	34	6	229,2s	30,8μm	-12,23
TOTAL	257	39	226,8μm	280	30	808,8s	266μm	-13,68

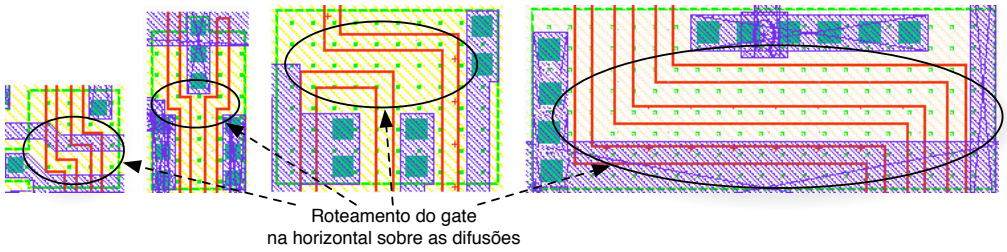


Figura 6.3: Técnica de roteamento do *gate* em 2D utilizada em *standard-cells*

Tabela 6.3: Comparação de atraso e potência com *standard-cells*

Célula	*Standard-cell*			Gerador			Ganho (%)		
	Atraso (ps)		Pot. (μW)	Atraso (ps)		Pot. (μW)	Atraso		Pot.
Saída	CO	S		CO	S		CO	S	
ADD22	237	349	308,1	220	336	292,7	7,2	3,7	5
ADD31	337	588	449,3	330	513	403,0	2,1	12,8	10,3
ADD32	283	467	731,4	286	443	682,1	-1	5,1	6,7
Saída	Q	QN		Q	QN		Q	QN	
DF1	749	852	542,6	669	763	507,5	10,7	10,4	6,5
Saída	Q			Q			Q		
NAND21	88		84,8	80		81,3	9,1		4,1
NAND40	286		123,5	254		101,3	11,2		18
XOR41	782		317,2	774		328,5	1		-3,6
TOTAL							7,2		6,3

bém da Cadence. Um inversor foi acoplado às saídas de todas as células para obtenção de resultados de atraso mais relevantes. Os dados completos de todas as simulações realizadas são apresentados na Tabela 6.3. O atraso de cada uma das saídas das células está indicado.

As simulações mostraram que o consumo de potência e atraso das células automaticamente geradas foram sempre muito próximos ou na maior parte das vezes melhores que as *standard-cells*. Mesmo quando comparando células com a mesma largura tais como ADD31, ADD32, NAND21 e NAND40, ainda assim foi possível obter resultados melhores.

O leiaute da célula *standard-cell* NAND40 foi analisado para tentar descobrir a razão da diferença no consumo de potência e atraso. Foi constatado que o espaçamento entre os sinais de *gate* dos transistores em série na rede N era duas vezes o mínimo da tecnologia e que, além disto, o espaçamento entre as difusões P e N era mais de duas vezes maior do que o espaçamento do leiaute gerado automaticamente. Estes dois fatores possuem um impacto negativo nas características elétricas da célula uma vez que aumentam a resistência das conexões e também suas capacitâncias associadas.

6.2 Compilador de Parte Operativa

Nesta seção é apresentado resultados obtidos com a geração automática de circuitos regulares e também de lógica aleatória pelo compilador de PO. Estes circuitos foram comparados com outras ferramentas e as conclusões destas comparações são apresentadas.

6.2.1 Geração de circuitos regulares

Dois circuitos de PO foram automaticamente produzidos para que pudessem ter seus leiautes comparados com outros métodos. As células utilizadas para geração destes circuitos foram obtidas na seção anterior durante a comparação com as *standard-cells* com exceção de uma porta AND que não existia nesta biblioteca.

O primeiro teste realizado foi a geração de um somador *ripple carry* de 4 *bits* de lar-

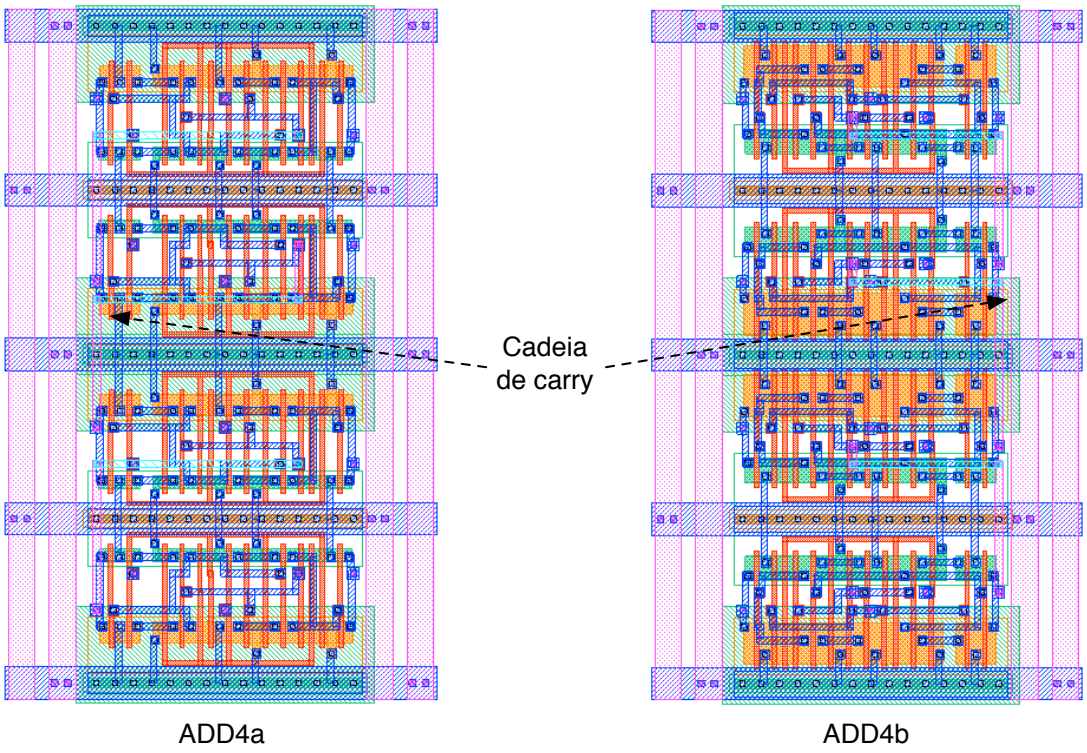


Figura 6.4: Leiaute de dois somadores Ripple-Carry de 4 bits usando células com diferentes potências

gura. Este somador apresenta uma estrutura altamente regular e o seu leiaute pode ser gerado com o mínimo de desperdício de área pelo gerador. Neste teste, dois somadores diferentes foram produzidos: o primeiro, circuito ADD4a, com células para baixo consumo de potência (ADD31), e segundo, ADD4b, com células de menor atraso (ADD32). A Figura 6.4 apresenta um recorte dos dois somadores com destaque para a cadeia de *carry*.

O teste seguinte foi a geração de um multiplicador *carry-save* de 4x4 *bits* chamado MUL4x4 cujo diagrama esquemático é mostrado na Figura 6.5. Multiplicadores são operadores comumente encontrados em projetos PO e por esta razão este circuito foi escolhido para teste. A estrutura do multiplicador *carry-save* permite projetar um leiaute com um posicionamento e o roteamento das suas redes internas com grande regularidade. A Figura 6.6 mostra o leiaute final do multiplicador produzido pelo compilador. Os espaços vazios são necessários para não comprometer a regularidade do circuito e também não aumentar o tamanho das conexões das células localizadas na banda superior do leiaute. Este desperdício tende a ser amortizado em multiplicadores de maior tamanho.

Os circuitos do somador e do multiplicador foram descritos em linguagem VHDL e sintetizados com a ferramenta RTL Compiler da Cadence. Os leiaute obtidos pela síntese automática e pela ferramenta Jungle Parrot foram extraídos e simulados com o auxílio das ferramentas Virtuoso e Spectre da Cadence. Os valores de área foram obtidos por medições das áreas úteis dos circuitos, desconsiderando-se as linhas de alimentação verticais. Os valores de atraso e potência foram obtidos através de simulação com vetores aleatórios dada a dificuldade em precisar o vetor exato que estimulasse o caminho crítico dos circuito. Os resultados obtidos são mostrados na Tabela 6.4.

Observando o resultado da síntese dos somadores ADD4a e ADD4b, foi possível per-

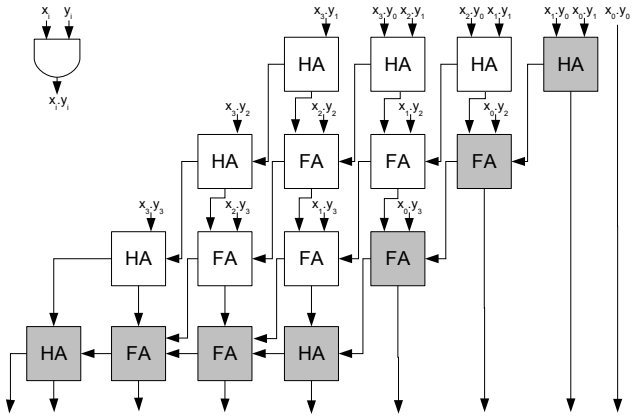


Figura 6.5: Diagrama esquemático de um multiplicador carry-save 4x4

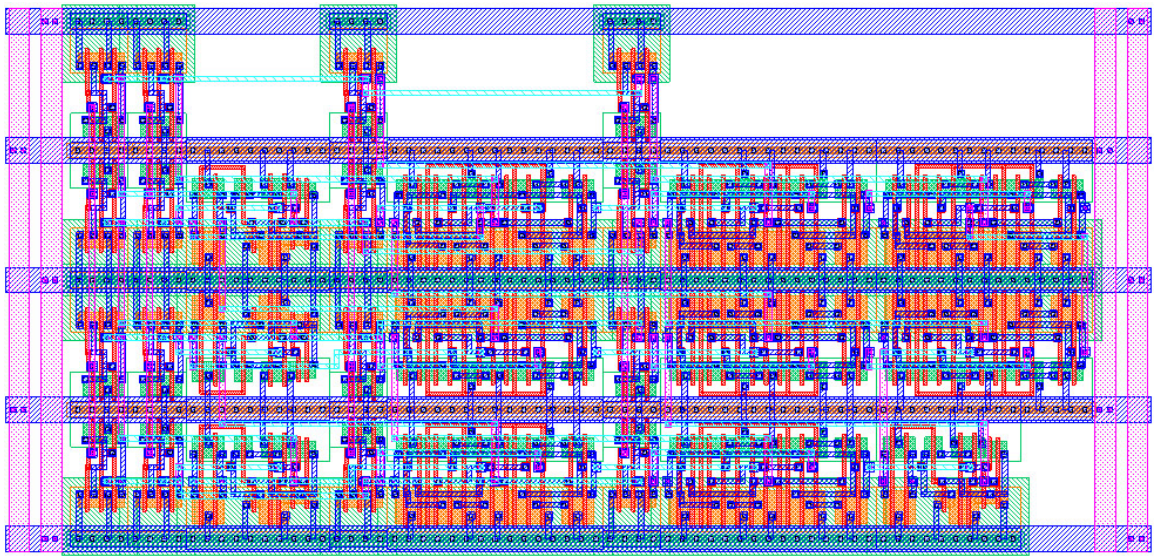


Figura 6.6: Leiaute de um multiplicador carry-save 4x4 gerado automaticamente pelo compilador de PO

Tabela 6.4: Comparação de área, atraso e potência de circuitos de parte operativa

	Síntese Std-cell			Jungle Parrot			Ganho (%)		
Circuito	# Pot. (mW)	Área (μm^2)	Atraso (ps)	# Pot. (mW)	Área (μm^2)	Atraso (ps)	# Pot.	Área	Atraso
ADD4a	0,925	1092	2055	0,863	1058,4	2053	6,7	3,1	0,1
ADD4b	1,603	1092	1588	1,508	1058,4	1598	1,1	3,1	-0,6
MUL4x4	6,448	6716	2174	3,973	5070,2	1896	38,4	24,5	12,8

Tabela 6.5: Comparação em área com Parrot Punch de circuitos de lógica aleatória

	# Células	# Trans.	Punch (μm)	Compilador (μm)	Ganho (%)
c17	6	24	408,7	376,7	7,8
c432	190	804	19617,3	19095,0	2,7
c499	364	1556	48739,0	48820,9	-0.2
c880	555	1802	56476,4	67939,1	-20,3

ceber que as mesma células ADD31 e ADD32 foram utilizadas nos dois projetos. Por esta razão, os ganhos individuais obtidos em algumas das células geradas automaticamente se refletiram diretamente nos resultados dos somadores analisados. Com isto, foi possível obter como resultado uma redução de 3,1% na área total ocupada e uma diminuição do consumo de potência entre 1-7%. O Atraso pouco se modificou devido ao tempo de propagação do sinal CO muito próximos em ambos projetos.

A comparação do circuito multiplicador MUL4x4 mostrou que mesmo quando otimizado para área, a implementação *standard-cell* necessitou de 24,5% mais área que o compilador. Uma das razões para isto foi o uso da célula AND2 pelo compilador que possui área 20% menor do que o conjunto NAND2 e Inversor encontrado em diversos pontos do projeto para *standard-cell* devido a inexistência desta célula na biblioteca. Além disto, foi constatada uma redução de 38,4% no consumo de potência e 12,8% no atraso deste circuito na implementação do compilador de PO Jungle Parrot. A razão para este ganho provavelmente pode ser explicada pela redução no número de transistores entre as duas implementações: 376 para a síntese *standard-cell* contra 634 do compilador.

Apesar dos ótimos resultados obtidos nestes testes, mais circuitos precisam ainda ser gerados e comparados para das mais evidência à vantagem da utilização do compilador de Parte Operativa em relação ao fluxo ASIC para esta classe de circuitos.

6.2.2 Geração de circuitos de lógica aleatória

Algumas vezes, circuitos de lógica aleatória são necessários em projetos de parte operativa para implementar operadores com pouca regularidade ou que não tenham ainda seu gerador implementado.

Para testar a habilidade do compilador em gerar circuitos de lógica aleatória, uma comparação com a ferramenta Parrot Punch (LAZZARI, 2003) desenvolvida na nossa universidade foi realizada. Os circuitos utilizados para teste implementam lógicas combinacionais simples e foram retirados do conjunto de *benchmarks* do ISCAS85. Os mesmos circuitos, com o mesmo dimensionamento de transistores, foram utilizados em ambos os testes. A ferramenta de síntese de células lógicas desenvolvida neste trabalho foi responsável pela geração da biblioteca de células lógicas utilizada pelo compilador para produção dos circuitos. Os resultados obtidos são apresentados na Tabela 6.5.

Com a análise dos dados é possível perceber que os circuitos menores obtiveram melhor resultado do que os com maior quantidade de células. A razão para isto é que as células utilizadas na biblioteca possuíam em geral menor altura e largura do que as produzidas pela ferramenta Punch fazendo com que os maiores ganhos fossem obtidos nos circuitos menores onde o posicionamento e roteamento não são tão críticos. Nos circuitos com maior quantidade de células, espaços em brancos tiveram de ser inseridos e células de maior altura (para comportar um maior número de trilhas de roteamento horizontais)

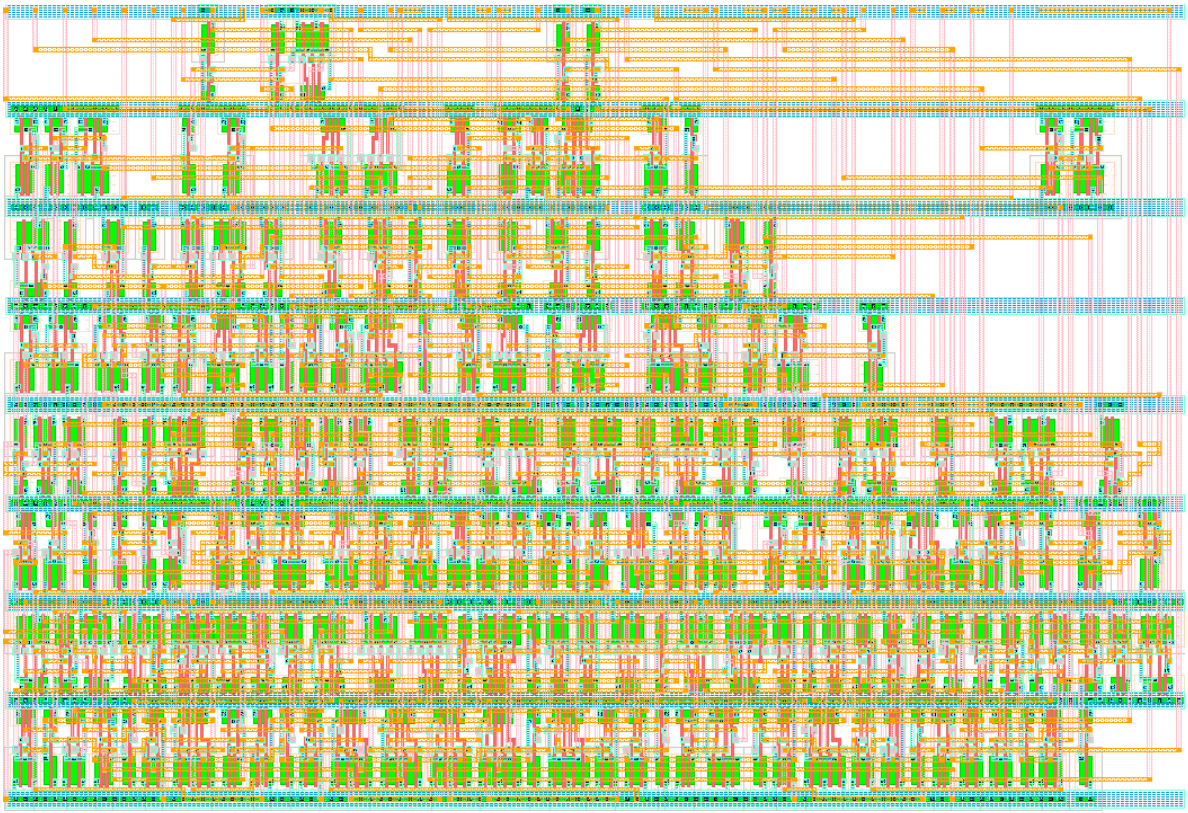


Figura 6.7: Leiaute do circuito c432 gerado automaticamente pelo compilador de PO

utilizadas para que o roteamento pudesse ser completado. Punch obteve vantagem nestes casos porque sua integração com o roteador Worm permite a inserção de trilhas extras somente nos locais congestionados e não em todas as bandas como foi necessário na nossa abordagem. O uso de um posicionador de células que leve em conta os pontos de congestinamento deve ajudar o compilador a produzir resultados melhores, com menor desperdício de espaço.

A Figura 6.7 mostra o resultado da geração do circuito c432 onde foi necessário a inclusão de 33,3% de espaços em branco e também o aumento da altura das células do mínimo de 9 para 10 trilhas para que o roteamento pudesse ser completado.

7 CONCLUSÃO

Neste trabalho foi apresentado um método para automatização do processo de geração de leiaute de circuitos de parte operativa. Duas ferramentas foram desenvolvidas para atingir este objetivo: uma para síntese de células CMOS e outra para montagem de circuitos de parte operativa com o uso de uma biblioteca de células. Esta metodologia tem o objetivo de criar um fluxo de síntese que permita independência tecnológica e suporte otimizações para redução de área, atraso e potência. As otimizações podem ser feitas por ferramentas de síntese lógica e/ou de análise de atraso com a modificação do *netlist* das células antes do processo de geração do leiaute. Apesar do suporte à estas ferramentas ter sido provido no fluxo, sua utilização não foi tratada neste trabalho.

Outra característica da metodologia desenvolvida é a separação da etapa de geração de parte operativa do processo de síntese de células lógicas. Isto permite que as células geradas automaticamente pela ferramenta de síntese possam ser verificadas e modificadas (para adição de alguma melhoria) antes de serem incorporadas a biblioteca de células e utilizadas pelo compilador. Também devido a esta independência entre as duas ferramentas, bibliotecas *standard-cell* e/ou desenvolvidas por outros métodos/ferramentas também podem ser utilizadas no processo de geração de PO.

Para obtenção da melhor estratégia possível para o desenvolvimento deste trabalho, um amplo estudo bibliográfico foi realizado nos Capítulos 2 e 3. Esta pesquisa possibilitou o desenvolvimento deste fluxo, de forma modular, e com suporte a diferentes técnicas de otimização.

O Capítulo 4 apresentou as estratégias adotadas para geração automática do leiaute de células lógicas. A ferramenta de síntese desenvolvida apresenta como principais características:

- Geração de **qualquer tipo de célula**, de diferentes famílias lógicas;
- **Dimensionamento contínuo** dos transistores e fiel ao especificado no *netlist* inicial;
- Aplicação automática de **folding** para atender aos requisitos de altura da célula;
- Posicionamento de transistores com objetivo **simultâneo** de minimização de área, GAPs e comprimento interno das conexões;
- Estilo de leiaute com pinos, altura, largura e *ties* alinhados à grade de roteamento;

Esta ferramenta teve como objetivo permitir o rápido desenvolvimento de uma biblioteca inteira de células CMOS com o menor desperdício de área possível e o mais fiel possível à especificação original do circuito.

O compilador de parte operativa foi apresentado no Capítulo 5. A ferramenta utiliza como linguagem de entrada um arquivo de descrição de parte operativa relativamente simples, mas que dá grande liberdade ao projetista em escolher as células que serão usadas para geração de cada um dos blocos funcionais e também especificar a forma como estes blocos se conectam.

Os resultados obtidos mostraram que as células automaticamente geradas tiveram um acréscimo de área médio de apenas 13,7% quando comparadas às células de uma biblioteca *standard-cell*. Comparações feitas através de simulações mostraram que a nossa metodologia obteve uma redução média de 7,2% no atraso e 6,3% no consumo de potência das células.

O compilador de parte operativa permitiu o desenvolvimento de três circuitos aritméticos que foram comparados com sua implementação utilizando um fluxo de posicionamento e roteamento automáticos baseado em *standard-cells*. Os resultados mostraram que os circuitos produzidos pela síntese *standard-cell* necessitaram todos maior quantidade de área e potência do que os circuitos equivalentes produzidos pelo compilador, mesmo quando utilizando as mesmas células e com o mesmo dimensionamento de transistores. No pior caso houve apenas um aumento de 0,6% no atraso de um dos circuitos.

7.1 Trabalhos Futuros

Ao longo dos Capítulos, várias possibilidades de melhorias foram comentadas nos algoritmos utilizados neste trabalho. O resumo destas melhorias e mais algumas outras são listadas a seguir:

- Melhorar o algoritmo de roteamento de sinal do gerador de células que sofre de um problema comum em MSTs.

- Estudar uma maneira de implementar roteamento do sinal de *gate* em duas dimensões sobre a difusão durante a compactação. Esta é uma técnica muito utilizada em *standard-cell* para aumentar a largura do transistor sem precisar realizar folding que penaliza muito mais a área da célula. Esta técnica deverá ser utilizada como uma opção definida pelo usuário uma vez que leiautes para tecnologias abaixo de 180nm já evitam este tipo de abordagem por causar problemas na manufatura da célula;

- Permitir o posicionamento das interfaces da célula em outras regiões além das trilhas internas de roteamento;

- Inserir heurísticas para encontrar um posicionamento das portas de entrada/saída da célula que minimize o impacto na largura da célula e que também não limite a aproximação dos sinais de *gate* dos transistores sobre a difusão;

- Adicionar técnicas de convergência no algoritmo de geração de células para que encontre uma solução sem necessitar de intervenção. Atualmente o usuário é requerido para modificar a posição e o número das trilhas sempre que o algoritmo de roteamento falha.

- Criar uma interface gráfica que permita a visualização do leiaute e a intervenção do usuário em estágios intermediários do processo de geração de células como, por exemplo, após o posicionamento e/ou roteamento dos transistores;

- Expandir o número de geradores de módulos dentro do gerador de parte operativa.
- Implementar suporte ao espelhamento vertical das células para reduzir o comprimento total das redes de interconexão.

REFERÊNCIAS

AMMAR, L. B.; GREINER, A. A high density datapath compiler mixing random logic with optimized blocks. In: EUROPEAN CONFERENCE ON DESIGN AUTOMATION WITH THE EUROPEAN EVENT IN ASIC DESIGN, 1993, Paris, France. **Proceedings...** Los Alamitos: IEEE Computer Society, 1993. v.4, p.194–198.

BASARAN, B. et al. GeneSys: a leaf-cell layout synthesis system for ghz vlsi designs. In: INTERNATIONAL CONFERENCE ON VLSI DESIGN - VLSI FOR THE INFORMATION APPLIANCE, 12., 1999, Goa, India. **Proceedings...** Washington: IEEE Computer Society, 1999. p.448.

BATTEN, C. **Spongepaint**. Datapath Generator Tool. Disponível em: <http://cag.lcs.mit.edu/ cbatten>. Acesso em: 15 nov. 2007.

BORAH, M.; OWENS, R. M.; IRWIN, M. J. Transistor sizing for minimizing power consumption of CMOS circuits under delay constraint. In: INTERNATIONAL SYMPOSIUM ON LOW POWER DESIGN, ISLPED, 1995, Dana Point, CA, USA. **Proceedings...** New York: ACM Press, 1995. p.167–172.

BOYER, D. G. Symbolic layout compaction review. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION, DAC, 25., 1988, Atlantic City, New Jersey, United States. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 1988. p.383–389.

CARRO, L. **Gerador Parametrizável de Partes Operativas CMOS**. 1989. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

CHAN, T. et al. Challenges of CAD Development for Datapath Design. **Intel Technology Journal**, [S.l.], n.Q1, p.17, 1999.

CHOWDHARY, A. et al. A general approach for regularity extraction in datapath circuits. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1998, San Jose, California, United States. **Proceedings...** New York: ACM Press, 1998. p.332–339.

DUECK, G.; SCHEUER, T. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. **J. Comput. Phys.**, San Diego, CA, USA, v.90, n.1, p.161–175, 1990.

FLACH, G.; HENTSCHKE, R.; REIS, R. Algorithms for improvement of RotDL router. In: SOUTH SYMPOSIUM ON MICROELECTRONICS, SIM, 2004, Ijuí: Unijuí. **Proceedings...** [S.l.: s.n.], 2004.

GAJSKI, D. D. **Principles of Digital Design**. [S.l.]: Prentice Hall, 1997.

GUO, P.-N.; CHENG, C.-K.; YOSHIMURA, T. An O-tree representation of non-slicing floorplan and its applications. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION, DAC, 36., 1999, New Orleans, Louisiana, United States. **Proceedings...** New York: ACM Press, 1999. p.268–273.

GUPTA, A.; HAYES, J. P. Width minimization of two-dimensional CMOS cells using integer programming. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1996, San Jose, California, United States. **Proceedings...** Washington: IEEE Computer Society, 1996. p.660–667.

GUPTA, A.; HAYES, J. P. CLIP: an optimizing layout generator for two-dimensional cmos cells. In: ANNUAL CONFERENCE ON DESIGN AUTOMATION, DAC, 34., 1997, Anaheim, California, United States. **Proceedings...** New York: ACM Press, 1997. p.452–455.

GUPTA, A.; HAYES, J. P. Optimal 2-D cell layout with integrated transistor folding. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1998, San Jose, California, United States. **Proceedings...** New York: ACM Press, 1998. p.128–135.

GUPTA, A.; HAYES, J. P. CLIP: integer-programming-based optimal layout synthesis of 2d cmos cells. **ACM Transactions on Design Automation of Electronic Systems, TODAES**, [S.l.], v.5, n.3, p.510–547, 2000.

GUPTA, A.; THE, S.-C.; HAYES, J. P. XPRESS: a cell layout generator with integrated transistor folding. In: EUROPEAN CONFERENCE ON DESIGN AND TEST, EDTC, 1996. **Proceedings...** Washington: IEEE Computer Society, 1996. p.393.

GUPTA, P. et al. Selective gate-length biasing for cost-effective runtime leakage control. In: ANNUAL CONFERENCE ON DESIGN AUTOMATION, DAC, 41., 2004, San Diego, CA, USA. **Proceedings...** New York: ACM Press, 2004. p.327–330.

GURUSWAMY, M. et al. CELLERITY: a fully automatic layout synthesis system for standard cell libraries. In: DESIGN AUTOMATION CONFERENCE, DAC, 34., 1997, Anaheim, California, United States. **Proceedings...** New York: ACM, 1997. p.327–332.

HENTSCHKE, R. **Algoritmos para o Posicionamento de Células em Circuitos VLSI**. 2002. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

HENTSCHKE, R. **Algorithms for Wire Length Improvement of VLSI Circuits With Concern to Critical Paths**. 2007. Tese (Doutorado em Ciência da Computação) — PPGC, UFRGS, Porto Alegre. Não homologada.

HENTSCHKE, R. et al. Quadratic placement for 3d circuits using z-cell shifting, 3d iterative refinement and simulated annealing. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 19., 2006, Ouro Preto, MG, Brazil. **Proceedings...** New York: ACM Press, 2006. p.220–225.

HSIEH, Y.-C. et al. LiB: a cell layout generator. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION, DAC, 27., 1990, Orlando, Florida, United States. **Proceedings...** New York: ACM Press, 1990. p.474–479.

HU, J. **A Datapath Compiler with Technology Portability**. 2000. Dissertação (Mestrado em Ciência da Computação) — Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada.

IIZUKA, T.; IKEDA, M.; ASADA, K. High speed layout synthesis for minimum-width CMOS logic cells via Boolean satisfiability. In: CONFERENCE ON ASIA SOUTH PACIFIC DESIGN AUTOMATION, ASP-DAC, 2004, Yokohama, Japan. **Proceedings...** Piscataway: IEEE Press, 2004. p.149–154.

IIZUKA, T.; IKEDA, M.; ASADA, K. Exact minimum-width transistor placement without dual constraint for CMOS cells. In: ACM GREAT LAKES SYMPOSIUM ON VLSI, GLSVSLI, 15., 2005, Chicago, Illinois, USA. **Proceedings...** New York: ACM Press, 2005. p.74–77.

IIZUKA, T.; IKEDA, M.; ASADA, K. Exact Minimum-Width Transistor Placement for Dual and Non-dual CMOS Cells. **IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences**, [S.l.], v.E88-A, n.12, p.3485–3491, 2005.

ITOOLS. Disponível em: <http://www.internetcad.com/>. Acesso em: 22 mar. 2007.

JAMIER, R. **Génération Automatique de Parties Opératives de Circuits VLSI de Type Microprocesseur**. 1986. Tese (Doutorado em Ciência da Computação) — IÍnstitut National Polytechnique de Grenoble, Grenoble, France.

KAHNG, A. B.; ROBINS, G. A new class of Steiner tree heuristics with good performance: the iterated 1-steiner approach. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1990, Santa Clara, CA, USA. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 1990. p.428–431.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by Simulated Annealing. **Science**, [S.l.], v.220, n.4598, p.671–680, May 1983.

LAZZARI, C. **Automatic Layout Generation of Static CMOS Circuits Targeting Delay and Power Reduction**. 2003. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

LEE, C. Y. An Algorithm for Path Connections and Its Applications. **IRE Transactions on Electronic Computers**, New York, v.EC-10, p.346–365, Sept. 1961.

LEFEBVRE, M.; MARPLE, D.; SECHEN, C. The future of custom cell generation in physical synthesis. In: DESIGN AUTOMATION, DAC, 34., 1997, Anaheim, California, United States. **Proceedings...** New York: ACM Press, 1997. p.446–451.

LPSOLVE. Disponível em: <http://sourceforge.net/projects/lpsolve>. Acesso em: 17 jul. 2007.

MARPLE, D.; SMULDERS, M.; HEGEN, H. An efficient compactor for 45° layout. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION, DAC, 25., 1988, Atlantic City, New Jersey, United States. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 1988. p.396–402.

MATSUMOTO, N. et al. Datapath generator based on gate-level symbolic layout. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION, DAC, 27., 1990, Orlando, Florida, United States. **Proceedings...** New York : ACM Press, 1990. p.388–393.

MAZIASZ, R. L.; GURUSWAMY, M.; RAMAN, S. **US Patent 6209123**: methods of placing transistors in a circuit layout and semiconductor device with automatically placed transistors. 2001.

MAZIASZ, R. L.; HAYES, J. P. Exact width and height minimization of CMOS cells. In: ACM/IEEE DESIGN AUTOMATION, DAC, 28., 1991, San Francisco, California, United States. **Proceedings...** New York: ACM Press, 1991. p.487–493.

MCMURCHIE, L.; EBELING, C. PathFinder: a negotiation-based performance-driven router for fpgas. In: ACM INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE GATE ARRAYS, FPGA, 3., 1995, Monterey, California, United States. **Proceedings...** New York: ACM Press, 1995. p.111–117.

MEINHARDT, C. **Geração de Leiautes Regulares Baseados em Matrizes de Células**. 2006. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

ONG, C.-L.; LI, J.-T.; LO, C.-Y. GENAC: an automatic cell synthesis tool. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION, DAC, 26., 1989, Las Vegas, Nevada, United States. **Proceedings...** New York: ACM Press, 1989. p.239–244.

PATTERSON, D. A.; HENNESSY, J. L. **Computer organization and design**: the hardware/software interface. 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

POIRIER, C. J. Excellerator: custom cmos leaf cell layout generator. **IEEE Transactions in Compututer-Aided Design**, [S.l.], v.8, n.7, p.744–755, July 1989.

PROGENESYS. Disponível em: <http://www.prolificinc.com/progenesis.html>. Acesso em: 12 ago. 2007.

RABAEY, J. M. **Digital integrated circuits**: a design perspective. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

REIS, A.; ROBERT, M.; REIS, R. Topological parameters for library free technology mapping. In: BRAZILIAN SYMPOSIUM ON INTEGRATED CIRCUIT DESIGN, SBCCI, 11., 1998, Armação de Buzios, RJ, Brazil. **Proceedings...** Los Alamitos: CA: IEEE Computer Society, 1998. p.213–216.

REIS, R. A New Standard Cell CAD Methodology. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE, 1987, Portland, OR, USA. **Proceedings...** New York: IEEE, 1987. p.385–388.

REKHI, S.; TROTTER, J. D.; LINDER, D. H. Automatic layout synthesis of leaf cells. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION, DAC, 32., 1995, San Francisco, California, United States. **Proceedings...** New York: ACM, 1995. p.267–272.

RIEPE, M. A.; SAKALLAH, K. A. Transistor placement for noncomplementary digital VLSI cell synthesis. **ACM Trans. Des. Autom. Electron. Syst.**, New York, NY, USA, v.8, n.1, p.81–107, 2003.

SANTOS, C. et al. A Transistor Sizing Method Applied to an Automatic Layout Generation Tool. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 16., 2003, São Paulo. **Proceedings...** Washington: IEEE Computer Society, 2003. p.303–307.

SANTOS, C. L. dos et al. Incremental Timing Optimization for Automatic Layout Generation. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005, Kobe, Japão. **Proceedings...** USA: IEEE, 2005. v.4, p.3567–3571.

SANTOS, G.; JOHANN, M.; REIS, R. Channel Based Routing in Channel-less Circuits. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2006. **Proceedings...** [S.l.]: IEEE, 2006.

SAWICKI, S.; HENTSCHKE, R.; FLACH, G.; JOHANN, M.; REIS, R. Studying the influence of I/O pads placement on wirelength and 3D-Vias of VLSI 3D integrated circuits. In: SOUTH SYMPOSIUM ON MICROELECTRONICS, SIM, 2007, Porto Alegre, RS, Brazil. **Proceedings...** Porto Alegre: SBC, 2007. p.89–92.

SERDAR, T.; SECHEN, C. Automatic datapath tile placement and routing. In: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, DATE, 2001, Munich, Germany. **Proceedings...** Piscataway: IEEE Press, 2001. p.552–559.

SHIGEHIRO, Y. et al. Optimal layout recycling based on graph theoretic linear programming approach. In: IFIP TC10/WG 10.5 INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION, VLSI, 1994. **Proceedings...** Amsterdam: North-Holland Publishing, 1994. p.25–34.

SUSIN, A. A. **Etude des Parties Operatives a Elements Modulaires pour Processeurs Monolithiques**. 1981. Tese (Doutorado em Ciência da Computação) — Université Grenoble, Grenoble, France.

TALIERCIO, M.; FOLETTO, G.; LICCIARDI, L. A procedural datapath compiler for VLSI full custom applications. In: IEEE CONFERENCE ON CUSTOM INTEGRATED CIRCUITS, 1991, San Diego, CA, USA. **Proceedings...** [S.l.: s.n.], 1991. p.22.5/1–22.5/4.

UEHARA, T.; VANCLEEMPUT, W. M. Optimal layout of CMOS functional arrays. **IEEE Transactions on Computers**, New York, v.30, n.5, p.305–312, 1981.

VAHIA, D.; CIESIELSKI, M. Transistor level placement for full custom datapath cell design. In: INTERNATIONAL SYMPOSIUM ON PHYSICAL DESIGN, ISPD, 1999, Monterey, California, United States. **Proceedings...** New York: ACM Press, 1999. p.158–163.

WILKE, G. et al. Finding the Critical Delay of Combinational Blocks by Floating Vector Simulation and Path Tracing. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 15., 2002, Porto Alegre. **Proceedings...** Washington: IEEE Computer Society, 2002. p.277–282.

ZIESEMER, A. et al. Cell Size Estimative in an Automatic Layout Generation Flow. In: SOUTH SYMPOSIUM ON MICROELECTRONICS, SIM, 21., 2006, Porto Alegre. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 2006. p.257–260.

APÊNDICE A REGRAS DE PROJETO USADAS PELA FERRAMENTA CELLGEN

```
W2P1 Minimum GATE length
W2DF Minimum GATE width
S1DFDF Minimum DIFF spacing to DIFF
S1CTCT Minimum CONT spacing
S1P1P1 Minimum Gate spacing
S2P1P1 Minimum POLY1 spacing to POLY1
S1M1M1 Minimum MET1 spacing to MET1
S1M2M2 Minimum MET2 spacing to MET2
S1M3M3 Minimum MET3 spacing to MET3
S1M4M4 Minimum MET4 spacing to MET4
S1M5M5 Minimum MET5 spacing to MET5
S1DFP1 Minimum POLY1 spacing to DIFF
S1DFP2 Minimum POLY2 spacing to DIFF
S1P1P2 Minimum POLY1 spacing to POLY2
S1CTP1 Minimum DIFFCON spacing of GATE
S1CTDP Minimum NDIFFCON spacing of PDIFF
S1CTDN Minimum PDIFFCON spacing of NDIFF
S1CTDF Minimum POLY1CON spacing of DIFF
S1CTP2 Minimum POLY1CON spacing of POLY2
S1M2M1 Minimum MET2 spacing to MET1 over CPOLY
S1M1M2 Minimum MET1 spacing to MET2 over CPOLY
S2M2M2 Minimum MET2 spacing to WIDE_MET2
S1DNWN Minimum NDIFF spacing to NTUB
W1M1 Minimum MET1 width
W1M2 Minimum MET2 width
W1M3 Minimum MET3 width
W1M4 Minimum MET4 width
W1M5 Minimum MET5 width
W1M6 Minimum MET6 width
W1M7 Minimum MET7 width
W1M8 Minimum MET8 width
W1M9 Minimum MET9 width
W1M10 Minimum MET10 width
W2CT Fixed CONT size
W2VI Fixed VIA size
```

```
W2V2 Fixed VIA2 size
W2V3 Fixed VIA3 size
W2V4 Fixed VIA4 size
W2V5 Fixed VIA5 size
W2V6 Fixed VIA6 size
W2V7 Fixed VIA7 size
W2V8 Fixed VIA8 size
W2V9 Fixed VIA9 size
W2V10 Fixed VIA10 size
S1VIVI Minimum VIA spacing to VIA
S1V2V2 Minimum VIA2 spacing to VIA2
S1IPIP PPLUS spacing to PPLUS
S1ININ NPLUS spacing to NPLUS
R1P1 Maximum ratio of POLY area to touched GATE area
R1M1 Minimum ratio of MET1 area to die area
R2M1 Maximum ratio of MET1 area to connected GATE and CPOLY area
R2M2 Maximum ratio of MET2 area to connected GATE and CPOLY area
E1P1DF Minimum POLY1 extension of GATE
E1DFP1 Minimum DIFF extension of GATE
E1DNP1 Minimum NDIFF extension of GATE when butted to PDIFF
E1DDP1 Minimum PDIFF extension of GATE when butted to NDIFF
E1INDF Minimum NPLUS extension of DIFF
E1IPDF Minimum PPLUS extension of DIFF
E1M1CT Minimum MET2 enclosure of CONT
E1DFCT Minimum DIFF enclosure of CONT
E1P1CT Minimum POLY1 enclosure of CONT
E1P2CT Minimum POLY2 enclosure of CONT
E1M1VI Minimum MET1 enclosure of VIA
E2M1VI Minimum WIDE_MET1 enclosure of VIA
E1M2VI Minimum MET2 enclosure of VIA
E1WNDP Minimum NTUB enclosure of PDIFF
E1M2V2 Minimum MET2 enclosure of VIA2
E1M3V2 Minimum MET3 enclosure of VIA2
E1M3V3 Minimum MET3 enclosure of VIA3
E1M4V3 Minimum MET4 enclosure of VIA3
E1M4V4 Minimum MET4 enclosure of VIA4
E1M5V4 Minimum MET5 enclosure of VIA4
```