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

CRISTIANO LOPES DOS SANTOS

Verificação e Otimização de Atraso durante a Síntese Física de Circuitos Integrados CMOS

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

Prof. Dr. Ricardo Augusto da Luz Reis
Orientador

Prof. Dr. José Luis Almada Güntzel
Co-orientador

Porto Alegre, dezembro de 2005.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Santos, Cristiano Lopes dos

Verificação e Otimização de Atraso durante a Síntese Física de Circuitos Integrados CMOS / Cristiano Lopes dos Santos – Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

101 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2005. Orientador: Ricardo Augusto da Luz Reis; Co-orientador: José Luis Almada Güntzel.

1.Otimização de atraso. 2.Dimensionamento de transistores 3.Síntese física. I. Reis, Ricardo Augusto da Luz. II. Güntzel, José Luis Almada III. Título.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer a meus pais, cujos incentivos sempre foram minha maior motivação. Minha mãe Joice, por seu eterno amor e carinho. Meu pai, José Antônio, por me ensinar valores sobre a vida.

Da mesma forma, gostaria de agradecer a minha namorada Aline, pelo carinho, dedicação e companhia durante todo esse tempo, e pela compreensão e apoio nos muitos momentos de ausência. Somente eu sei o bem que ela me faz.

Gostaria também de agradecer a todos os colegas do grupo de microeletrônica da UFRGS, em especial àqueles que trabalharam diretamente comigo, Cristiano Lazzari, Daniel Ferrão e Gustavo Wilke. Os trabalhos em conjunto, debates e trocas de idéias foram fundamentais na construção do meu conhecimento. E agradeço também pelas comemorações boêmias, ocorridas após as maratonas de noites mal dormidas correndo atrás de prazos.

Ao meu orientador, Ricardo Reis, e ao meu co-orientador, Güntzel, meu agradecimento por toda ajuda, confiança e amizade, e pela oportunidade que tornou este trabalho possível.

Gostaria de agradecer também a todos os meus amigos de longa data, em especial aos que conviveram comigo nestes últimos 3 anos. A amizade de vocês tornou o caminho muito mais fácil e agradável.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ATPG	Automatic Test Pattern Generation
ASIC	Application Specific Integrated Circuit
CAD	Computer-Aided Design
CI	Circuito Integrado
CIF	CalTech Interchange Format
CMOS	Complementary Metal-Oxide Silicon
DAG	Direct Acyclic Graph
DRC	Design Rule Checking
DSM	Deep Sub-micron
ED	Elmore Delay
FOTC	Full Over the Cell
FTA	Functional Timing Analysis
ILP	Integer Linear Programming
IPO	In-Place Optimization
LP	Linear Programming
LVS	Layout versus Schematic
LUT	Look-Up Table
PEX	Parasitic Extraction
RTL	Register Transfer Level
SCCG	Static CMOS Complex Gate
SED	Scaled Elmore Delay
SPICE	Simulation Program with Integrated Circuits Emphasis
STA	Static Timing Analysis
TTA	Topological Timing Analysis
ULA	Unidade Lógica e Aritmética
VLSI	Very Large Scale Integration

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho propõe um método de otimização de atraso, através de dimensionamento de transistores, o qual faz parte de um fluxo automático de síntese física de circuitos combinacionais em tecnologia CMOS estática. Este fluxo de síntese física é independente de biblioteca de células, sendo capaz de realizar, sob demanda, a geração do leiaute a partir de um *netlist* de transistores. O método de otimização proposto faz com que este fluxo de síntese física seja capaz de realizar a geração do leiaute orientado pelas restrições de atraso, garantindo a operação do circuito na freqüência especificada pelo projetista.

Este trabalho inclui também uma pesquisa sobre os principais métodos de verificação e otimização de atraso, principalmente aqueles que podem ser aplicados quando a etapa de síntese física chega ao nível de transistores. Um método de análise de *timing* funcional é utilizado para identificar o atraso e o caminho críticos e, com isso, guiar o método de otimização proposto. Desta forma, não existe desperdício de esforço e desempenho para reduzir o atraso de caminhos que não contribuem efetivamente para determinar a freqüência do circuito.

O método proposto neste trabalho explora as possibilidades oferecidas por ser independente de biblioteca de células, mas impõe restrições aos circuitos otimizados para reduzir o impacto do dimensionamento nas etapas de geração de leiaute.

O desenvolvimento de um método incremental de seleção de caminhos críticos reduziu consideravelmente o tempo de processamento sem comprometer a qualidade dos resultados. Ainda, a realização de um método seletivo de dimensionamento de transistores, possibilitado pela adaptação de um modelo de atraso pino-a-pino, permitiu reduzir significativamente o acréscimo de área decorrente da otimização e aumentou a precisão das estimativas de atraso.

Palavras-Chave: Otimização de atraso, dimensionamento de transistores, análise de *timing*, síntese física orientada pelo atraso e geração de leiaute.

Timing Verification and Optimization in Physical Synthesis of CMOS Integrated Circuits

ABSTRACT

This work proposes a transistor sizing-based delay optimization method especially tailored for an automatic physical synthesis flow of static CMOS combinational circuits. Such physical synthesis flow is a library-free approach which is able to perform the layout generation using a transistor netlist level description of the circuit. The integration of the proposed optimization method to the automatic physical synthesis renders possible a timing-driven layout generation flow.

This work also includes a research of the major delay verification and optimization methods, mainly those that can be applied during the physical synthesis step at the transistor level. A functional timing analysis method is used to identify the critical delay and the critical paths and thus drive the proposed optimization method. Hence, there is no waste of effort to optimize paths which are not responsible for the delay of the circuit.

The optimization method proposed in this work explores the advantages provided by a library-free synthesis flow and imposes restrictions to the optimized circuits in order to minimize the impact of the transistor sizing in the layout generation steps.

The development of a method for incremental critical path selection reduces the CPU time consumed by the delay optimization step. A pin-to-pin gate delay model was adapted to perform a selective transistor sizing, resulting in a significantly reduction of the area overhead.

Keywords: Critical delay optimization, transistor sizing, timing analysis, timing-driven physical synthesis and layout generation.

1 INTRODUÇÃO

Nos dias atuais, com o avanço dos processos de fabricação de circuitos em tecnologia CMOS, os dispositivos eletrônicos que compõem os circuitos integrados (CIs) estão cada vez menores, o que permite uma maior capacidade de integração de diferentes circuitos eletrônicos em um único *chip.* Ao mesmo tempo, o tempo de projeto requerido pelo mercado de semicondutores está cada vez menor, expondo cada vez mais a necessidade de ferramentas de CAD (*Computer-Aided Design*) que automatizem as etapas de síntese e análise do circuito nos diferentes níveis e estágios de um fluxo de projeto. O padrão mais usado para automatizar a síntese de ASICs (*Application Specific Integrated Circuits*) é a utilização de células ou macro-blocos, estilo conhecido como *cell-based*. A composição de circuitos maiores e mais complexos é feita então pela associação destas células ou macro-blocos.

Em um fluxo de projeto de um circuito digital, a etapa de síntese física compreende um conjunto de transformações que tem por finalidade produzir o leiaute do circuito a partir de um *netlist* de portal lógicas, satisfazendo a uma série de restrições (COUDERT, 2002). Um fluxo de síntese física baseado em células é realizado, simplificadamente, pela seqüência de etapas conhecidas como planejamento topológico (*floorplanning*), posicionamento, geração do leiaute e roteamento. Durante o *floorplanning* é feita a alocação de área e o posicionamento dos sub-circuitos, o posicionamento dos pinos de entrada e saída (*pads*) e o planejamento das trilhas de alimentação. A etapa de posicionamento é responsável por reservar um espaço físico para cada célula ou macro-bloco dentro da área destinada a cada sub-circuito. Por fim, as etapas de geração e roteamento devem gerar o leiaute das células e macro-blocos e realizar as interconexões entre eles.

Cada uma destas etapas, além de alterar as características de atraso, área e consumo de potência do circuito, interfere na realização das etapas seguintes. Quando cada uma das etapas é acompanhada por um conjunto de ações que visam reduzir o atraso do circuito e permitir que as etapas seguintes também o façam, dizemos que o fluxo de síntese física é orientado pelo atraso (*timing-driven*). Para possibilitar esta característica ao fluxo, etapas de verificação e otimização de atraso, preferencialmente automatizadas por ferramentas de CAD, são cada vez mais necessárias ao longo de todo fluxo de projeto. Na Figura 1.1 é possível verificar a interação das etapas de verificação e otimização de atraso com diferentes etapas de um fluxo de projeto *cell-based*.

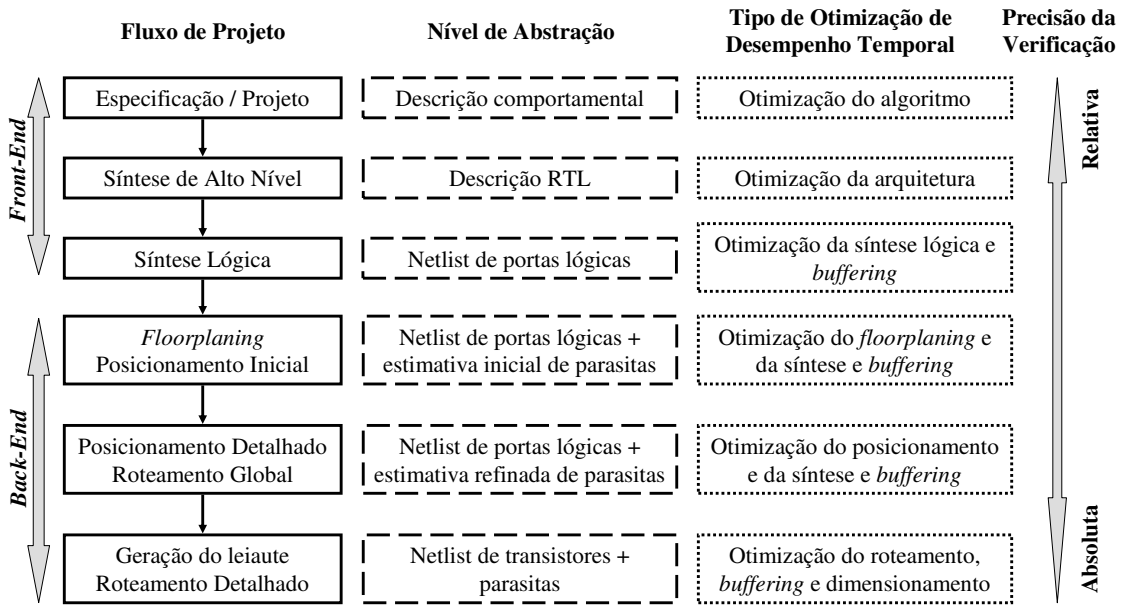

Figura 1.1: Verificação e otimização de atraso em um fluxo de projeto.

Porém, com a redução do tamanho dos dispositivos para escalas nanométricas nas novas tecnologias DSM (*Deep Sub-Micron*), estas etapas de verificação e otimização de atraso tornam-se especialmente mais complexas. Diversas questões elétricas que antes só eram consideradas em projeto de circuitos analógicos começaram a ser fundamentais mesmo em circuitos puramente digitais. Exemplos de tais questões são os efeitos dinâmicos ocasionados por capacitâncias de acoplamento entre conexões (*crosstalk*), ruído e integridade dos sinais, distribuição de temperatura e queda de tensão nas trilhas de alimentação (*IR drop*).

O estilo de síntese conhecido como *standard-cell*, que utiliza uma biblioteca de células padrão previamente projetada, é largamente empregado por permitir a concepção de circuitos complexos sem a necessidade de verificação do seu desempenho elétrico no nível de transistores. O fato de estas células serem pré-caracterizadas oferecia certa previsibilidade do desempenho do circuito, facilitando a detecção e otimização das violações de atraso desde etapas iniciais do fluxo de síntese. Apesar de cada porta lógica ser altamente otimizada, o número limitado de portas em cada biblioteca restringe a síntese lógica, resultando em circuitos pouco otimizados. É estimado que o uso deste tipo de síntese produza circuitos no mínimo 6 vezes mais lentos e com um consumo de potência no mínimo 10 vezes maior do que projetos similares implementados e otimizados manualmente (ROY, 2005).

Por outro lado, realizar a síntese do circuito independentemente de biblioteca, onde a geração do leiaute do circuito é feita sob demanda, oferece mais flexibilidade para as etapas de otimização. Mas, ao mesmo tempo, as etapas de verificação tornam-se mais difíceis, uma vez que as células ou macro-blocos precisam ser caracterizados também sob demanda. Porém, a importância do atraso decorrente das interconexões é, atualmente, tão determinante no desempenho de um circuito que já é possível constatar o surgimento de ferramentas especializadas em calcular o atraso de conexões mesmo em ferramentas comerciais de síntese física que são baseadas em biblioteca de células. E isto mostra que a falta de precisão na previsibilidade do desempenho de um circuito durante as etapas iniciais de síntese não é mais exclusividade de métodos que não são baseados em células pré-caracterizadas.

1.1 Síntese física *full custom* automática

O fluxo de síntese física FUCAS (***FU**ll **C**ustom **A**utomatic **S**ynthesis*), desenvolvido atualmente no grupo de microeletrônica da UFRGS (GME), tem como característica realizar a geração automática de circuitos CMOS estáticos independentemente de biblioteca de células. Ao invés de usar células pré-projetadas e pré-caracterizadas, o fluxo FUCAS permite que as equações lógicas que descrevem o comportamento do circuito sejam mapeadas para uma biblioteca virtual. O tamanho desta biblioteca é limitado apenas pela restrição, passada pelo projetista, de número máximo de transistores em série nas estruturas NMOS e PMOS de cada porta lógica. A Tabela 1.1 mostra o número de funções lógicas que é possível implementar de acordo com a restrição de número de transistores em série (DETJENS, 1987).

Tabela 1.1: Funções lógicas permitidas pelo número de transistores em série.

		Número de transistores PMOS em série				
		1	**2**	**3**	**4**	**5**
Número de transistores NMOS em série	**1**	1	2	3	4	5
	2	2	7	18	42	90
	3	3	18	87	396	1677
	4	4	42	396	3503	28435
	5	5	90	1677	28435	425803

A utilização de um gerador de leiaute (LAZZARI, 2003) capaz de gerar cada porta lógica sob demanda permite o uso de células com todas as combinações lógicas possíveis durante a etapa de síntese lógica. Esta característica traz uma grande liberdade de otimização em termos de área, consumo de potência e atraso do circuito (GOPALAKRISHNAN, 2001).

Como cada célula utilizada na síntese pode implementar qualquer função lógica, o circuito pode ser sintetizado com um número menor de células e, conseqüentemente, com um número menor de transistores. Além disso, como a geração de cada porta lógica é feita a partir do seu *netlist* de transistores, elas podem assumir qualquer possibilidade de ordenamento e tamanho dos transistores. Sendo assim, cada porta lógica pode ter um número muito grande de versões de atraso, área e consumo de potência.

Por exemplo, considerando a Tabela 1.1, podemos notar que é possível implementar até 3503 portas lógicas com uma limitação razoável de 4 transistores em série nas estruturas NMOS e PMOS de cada porta lógica. Considerando agora que cada porta lógica assuma apenas quatro diferentes configurações de tamanhos de seus transistores, é então possível atingir mais de 14 mil portas lógicas. Este valor está bem acima do que é oferecido por qualquer biblioteca de células em um método de síntese física baseado em *standard cells*.

O fluxo de síntese física FUCAS explora as vantagens do uso de portas complexas, onde possui total liberdade de otimização, por não estar limitado a uma biblioteca padrão. A possibilidade de geração de células complexas com qualquer combinação lógica, sem as restrições impostas por uma biblioteca de células, permite a síntese de circuitos otimizados em relação ao número de transistores utilizados. Aqui cabe lembrar que o consumo estático de um circuito implementado em tecnologia CMOS estática é diretamente proporcional ao número de transistores e que, em tecnologias DSM, o consumo estático tem contribuição importante no consumo total do circuito (KIM, 2003).

1.2 Organização desta dissertação

Neste trabalho serão abordados alguns dos principais métodos de verificação e otimização de atraso que podem ser aplicados durante a etapa síntese física, principalmente àquelas que não se baseiam em biblioteca de células e que, por isso, permitem a realização de otimizações de atraso no nível de transistores. Ainda, como objetivo principal deste trabalho, será proposto um método de otimização de atraso baseado em dimensionamento de transistores. A integração deste método com o fluxo de síntese física FUCAS tem por finalidade possibilitar a geração automática de circuitos integrados CMOS visando atingir as restrições de atraso impostas pelo projetista.

O capítulo 2 desta dissertação aborda os métodos de verificação de atraso, em especial o método conhecido como análise de *timing*, buscando mostrar características que podem influenciar o desempenho de algoritmos de otimização. O capítulo 3 traz um resumo sobre as técnicas de otimização de atraso que podem ser aplicadas quando o fluxo de projeto chega no nível de leiaute. O método de otimização de atraso conhecido como dimensionamento de transistores é abordado mais detalhadamente.

O capítulo 4 propõe a integração de uma ferramenta de verificação e otimização de atraso com o fluxo de síntese física FUCAS, de forma que os circuitos gerados respeitem as restrições de atraso. Serão mostradas características de algumas etapas do fluxo e a compreensão destas características será fundamental para a escolha do método de otimização a ser desenvolvido.

O capítulo 5 traz a proposta de um método de otimização através de dimensionamento de transistores. Diferentes metodologias e modelos de atraso foram comparados na busca de melhorias durante o desenvolvimento do método de otimização para que seu desempenho fosse aumentado. Comparações com uma ferramenta de análise de *timing* comercial e a geração automática dos circuitos são feitas para validar o método proposto. Por fim, são feitas algumas conclusões gerais no capítulo 6.

2 VERIFICAÇÃO DE ATRASO EM CIRCUITOS INTEGRADOS

Devido aos altos custos envolvidos no processo de fabricação de um circuito integrado (CI), o projetista deve certificar-se de que o circuito respeitará as especificações funcionais e elétricas antes de ser fabricado. Desta forma, etapas de verificação são indispensáveis durante todo fluxo de projeto de um circuito VLSI. A verificação de atraso tem por objetivo estimar o desempenho do circuito, predizendo a máxima freqüência de relógio (*clock*), no caso de circuitos seqüenciais, ou o atraso crítico no caso de circuitos combinacionais.

Como pode ser visto na Figura 1.1, em níveis mais altos do fluxo de projeto de CIs ainda não existem informações precisas sobre os elementos parasitas do circuito, e a verificação tem como principal objetivo antecipar a detecção de violações de atraso em tempo de realizar correções na arquitetura do CI. Quando a síntese do circuito chega no nível de leiaute, a extração dos capacitores, resistores e indutores provenientes da implementação física do circuito permite realizar uma estimativa acurada sobre o comportamento do circuito real após a sua fabricação.

As formas de verificação de atraso podem ser resumidas, basicamente, a simulação e análise de *timing*. A principal diferença entre elas é a necessidade ou não de padrões de estímulos (vetores) nas entradas do circuito, característica que é encontrada somente na simulação.

Simulação lógica *(logic-level simulation)* é um tipo de simulação geralmente usado para verificação da funcionalidade lógica do circuito, onde os nodos apresentam somente os valores lógicos "0" e "1". Com o uso de modelos de atraso para os componentes, portas ou blocos lógicos, e estimativas dos elementos parasitas do circuito, pode ser usada para verificação de atraso em níveis mais altos de abstração do projeto.

A **Simulação elétrica** *(circuit or electric-level simulation)* realiza uma análise detalhada e precisa das tensões e correntes nos nodos do circuito. SPICE (NAGEL, 1975) é o exemplo típico deste tipo de simulação *(Spectre* e *PSPICE* da *Cadence*®, *HSPICE* da *Synopsys*® e *Eldo* da *Mentor*® são exemplos de simuladores elétricos comerciais). Este tipo de simulação faz uso de modelos complexos com parâmetros tecnológicos fornecidos pelas *foundries* para representar de forma precisa o comportamento não-linear dos transistores (BSIM, 2005). Técnicas e métodos numéricos são utilizados para resolver as equações diferenciais não-lineares que

descrevem a corrente e a tensão de cada nodo do circuito em diferentes instantes de tempo. Embora muito precisos, os simuladores elétricos são demasiadamente lentos, tornando a verificação de atraso impraticável para circuitos com poucos milhares de transistores.

A **Simulação de *timing*** *(timing simulation)* faz uso de modelos de transistores menos precisos, ou mesmo *look-up tables* (LUTs), e técnicas de resolução mais simples para permitir a simulação de circuitos ligeiramente maiores que os usados em simulações elétricas (WOLF, 1994). Este tipo de simulação visa apenas informações de atraso, ou consumo de potência, de circuitos digitais. Simuladores elétricos simplificados conhecidos como *fast* SPICE (*MachTA* da *Mentor*®, *HSim* e *NanoSim* da *Synopsys*® e *UltraSim* da *Cadence*®) e simuladores no nível de chaves são exemplos de simuladores de *timing*. **Simulação no nível de chaves** *(switch-level simulation)* é um tipo de simulação muito usado em circuitos implementados com transistores CMOS. Cada transistor é modelado como uma chave, e a precisão da simulação depende do modelo usado para representar o comportamento do transistor. O abrir e fechar das chaves muda a conexão, e com isso os valores de corrente e/ou tensão, entre os nodos do circuito. É mais precisa que simulação no nível de portas lógicas e permite a análise de efeitos como acoplamento de carga, devido ao comportamento bidirecional dos transistores. *TV* (JOUPPI, 1987) e *Crystal* (OUSTERHOUT, 1985) são exemplos de simuladores no nível de chaves.

A **Análise de *timing*** *(timing analysis)* difere da simulação porque é independente de estímulos nas entradas, ou seja, não necessita de vetores (HITCHCOCK, 1982) (MCGEER, 1991) (CHEN, 1993). Neste método de verificação, cada bloco combinacional é representado por um grafo com valores de atraso em seus vértices e arestas de acordo com o atraso dos componentes do circuito (portas lógicas ou estágios *RC* e conexões). Então é feita uma procura através do grafo para identificar o maior atraso entre as entradas e saídas primárias do circuito. Em alguns métodos também é possível identificar qual é o caminho crítico, ou seja, qual é o caminho entre a entrada e a saída do circuito que é responsável pelo atraso crítico. *PrimeTime* e *PathMill* da *Synopsys*®, *SST Velocity* da *Mentor*® e *Pearl* da *Cadence*® são exemplos de ferramentas comerciais de análise de *timing*.

Em um circuito combinacional, a propagação de uma transição nos caminhos existentes entre as entradas e saídas primárias deste circuito depende da aplicação de estímulos nas suas entradas. Sendo assim, usar simulação para determinar o atraso de um circuito exige (BUSHNELL, 2000):

- Um esforço considerável para determinar previamente quais vetores exercitarão os caminhos críticos do circuito ou;
- Que os 2^n x (2^n-1) possíveis **pares** de vetores de entrada sejam verificados, onde n é o número de entradas do circuito.

Estas condições são inviáveis devido a complexidade dos circuitos projetados atualmente, considerando o poder de processamento dos computadores atuais e o tempo de projeto cada vez menor requerido pelo mercado de semicondutores. Neste contexto, por serem independentes de vetores nas entradas, os métodos de análise de *timing* têm sido cada vez mais utilizados durante as várias etapas do fluxo de projeto de circuitos digitais de alta integração.

O restante deste capítulo aborda aspectos relacionados a métodos de análise de *timing*, que foi o método de verificação de atraso utilizado no desenvolvimento deste trabalho. A seção 2.1 introduz a abordagem topológica para análise de *timing* e o problemas dos caminhos falsos. Na seção 2.2 são abordados os métodos de análise de *timing* funcional e os modelos computacionais de atraso do circuito para este tipo de análise. Os diferentes critérios de sensibilização, que são usados para justificar a existência de um caminho em uma análise de *timing* funcional, são mostrados na seção 2.3. Os modelos utilizados para computar o atraso de portas lógicas e conexões são apresentados, respectivamente, nas seções 2.4 e 2.5.

2.1 Análise de *timing* topológica e caminhos falsos

Em um método de análise de *timing*, é comum o uso de um grafo acíclico direcionado (*Direct Acyclic Graph* – DAG) para representar o circuito, onde os vértices e arestas representam suas portas lógicas e conexões, respectivamente. O atraso de cada porta é anotado no vértice que a representa, da mesma forma como o atraso de cada conexão é anotado na aresta que a representa. Então é feita uma procura através deste grafo para identificar o atraso crítico do circuito.

A maioria das soluções encontradas na literatura e nas ferramentas de CAD comerciais não leva em consideração a funcionalidade lógica das células na hora de fazer a procura no grafo, resumindo a busca do atraso ou do caminho crítico a encontrar o caminho com maior soma de atrasos de vértices e arestas no grafo. Esta solução é conhecida como análise de *timing* **topológica** (**TTA** – *Topological Timing Analysis*) ou **estática**[1] (**STA** – *Static Timing Analysis*) e pode ser resolvida em um tempo linear, em relação ao tamanho do grafo, pelo algoritmo *topological sort* (CORMEN, 1990). Considerando que, no exemplo da Figura 2.1, todas as portas lógicas tenham atraso unitário e que todas as conexões tenham atraso nulo, uma TTA apontaria os caminhos P_1= (i_1, G_1, G_2, G_3, G_4, G_6, G_7, G_8, h) e P_2=(i_2, G_1, G_2, G_3, G_4, G_6, G_7, G_8, h) como caminhos críticos do circuito, ambos com atraso igual a 7.

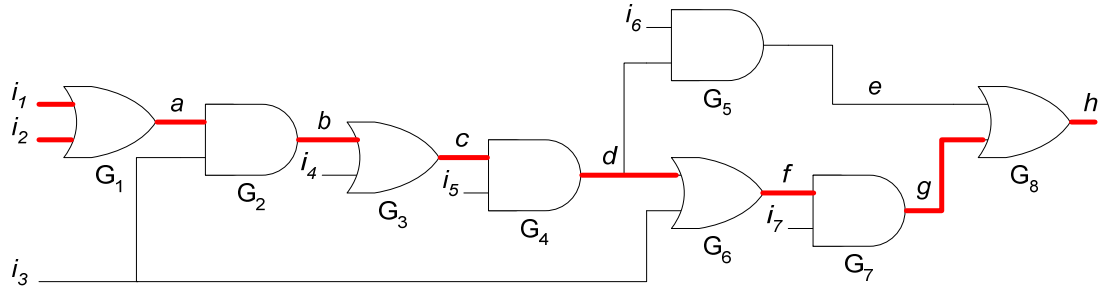

Figura 2.1: Circuito com caminho falso (HRAPCENKO, 1978).

O problema deste tipo de análise é que, se o circuito possuir caminhos falsos (HRAPCENKO, 1978), a estimativa de atraso gerada pode ser muito maior do que o atraso real do circuito. **Caminhos falsos**, ou caminhos não-sensibilizáveis, são aqueles pelos quais é impossível propagar uma transição, ou seja, não existe nenhum par de vetores capaz de exercitar estes caminhos. No circuito da Figura 2.1, enquanto i_3=0, nenhuma transição consegue ser propagada de *a* para *b*, e quando i_3=1, nenhuma transição pode ser propagada de *d* para *f*. Desta forma, nenhum par de vetores nas

[1] É preciso tomar cuidado para não confundir análise de *timing* estática (STA) com uma análise de *timing* funcional (FTA) que usa o critério estático de sensibilização (GÜNTEL, 2000).

entradas i_1-i_7 consegue exercitar os caminhos P_1 e P_2, os quais são ditos caminhos falsos. Conseqüentemente, o atraso real do circuito é menor do que 7.

2.2 Análise de *timing* funcional

Um método de análise de *timing* que considera a funcionalidade das portas lógicas, a fim de levar em conta a síndrome dos caminhos falsos, é chamado de análise de *timing* funcional (**FTA** – *Functional Timing Analysis*). Neste tipo de análise, o atraso crítico do circuito será o atraso do seu maior caminho **sensibilizável (**CHEN, 1993).

Güntzel (2000) mostrou que os métodos de FTA podem diferir sobre vários aspectos, tais como:

- Modelo computacional de atraso do circuito;
- Critérios de sensibilização utilizados para justificar a existência de um caminho;
- Algoritmo de FTA, ou seja, o método usado para computar o atraso do circuito e testar se os critérios de sensibilização são satisfeitos.

2.2.1 Modelos computacionais de atraso do circuito

Agora, sob o ponto de vista da FTA, é possível fazer uma outra definição para atraso crítico de um circuito: o atraso de propagação de um estímulo através de um circuito combinacional é o tempo necessário para que todos os sinais de saída estabilizem em seu valor final após o estímulo ser aplicado. Sendo assim, o atraso crítico do circuito será o maior atraso de propagação do circuito considerando todos os possíveis estímulos de entrada. Porém, o que será considerado como um estímulo válido vai depender do modelo computacional de atraso sobre o qual o circuito irá operar. Estes modelos diferem de acordo com a natureza dos estímulos, ou seja, se os estímulos serão **pares** de vetores ou **seqüência** de vetores.

Um par de vetores assume um vetor v_1 aplicado em $t=-\infty$ e um segundo vetor v_2 aplicado em $t=0$. Logo, podemos notar que todos os nodos do circuito são assumidos estáveis, em valores impostos pelo vetor v_1, quando o vetor v_2 é aplicado. O uso de pares de vetores para encontrar o atraso crítico do circuito é equivalente a assumir que o circuito opera em um modo completamente síncrono. Este tipo de operação é conhecido como **modo de transição** (*transition mode*), e o atraso crítico obtido por ele é referenciado como **atraso de transição** (*transition delay*) (DEVADAS, 1992). Apesar de ser restrito a circuitos síncronos capazes de garantir um modo totalmente síncrono de operação dos seus módulos combinacionais, o modo de transição tem como benefício ser capaz de fornecer o par de vetores responsável pelo atraso crítico, o que facilita a verificação deste atraso por simulação elétrica.

Problemas com a complexidade computacional do modo de transição motivaram o uso do modelo de **vetor único** (*single vector*), que é uma aproximação do atraso obtido por uma seqüência de vetores. Este modelo assume que os nodos do circuito estão em valor arbitrário, ou seja, "flutuando" antes de serem colocados em seu valor final pelo único vetor v. Esta característica do modelo, também chamado de **modo flutuante de operação** (*floating mode*) (CHEN, 1991), vem do fato de o circuito poder estar propagando vetores aplicados antes de v, o que deixaria os nodos do circuito flutuando. Devido à facilidade de implementação, ao menor custo computacional e à possibilidade

de usar algoritmos de ATPG já existentes, a maioria dos algoritmos de FTA adota o modelo de vetor único.

2.2.2 Algoritmos de FTA

Devido à semelhança entre as características dos problemas que tangem a análise de *timing* e a geração automática de vetores de teste (*Automatic Test Pattern Generation* – ATPG), muitos algoritmos básicos podem ser compartilhados entre eles. Desta forma, muitos algoritmos de análise de *timing* adaptam algoritmos de ATPG para testar a sensibilização dos caminhos. Existem também algoritmos de análise de *timing* que usam métodos formais, como **solvabilidade** (*satisfiability*), para resolver as funções de sensibilização. Estas funções são formadas pelo conjunto de condições necessárias para declarar um caminho sensibilizável sob um determinado critério de sensibilização.

Métodos tradicionais de FTA tentam sensibilizar um caminho de cada vez, começando pelo caminho topologicamente mais longo. Se as condições de sensibilização são satisfeitas, então o caminho é declarado como caminho crítico e seu atraso será o atraso crítico do circuito. Caso contrário, o próximo caminho topologicamente mais longo deve ser traçado e sua sensibilização deve ser testada. Este procedimento continua até que um caminho sensibilizável seja encontrado. Tais métodos são conhecidos como métodos de **sensibilização individual de caminhos** (*single path sensitization*) e são baseados em **enumeração de caminhos**.

Uma vez que em um circuito combinacional pode haver centenas de milhares de caminhos, uma alternativa mais eficiente, em termos de custo computacional, é tratar um conjunto de caminhos ao invés de um só caminho de cada vez. Para isso, os métodos devem ser baseados em uma estratégia de **enumeração de atrasos** ao invés de enumeração de caminhos. Os métodos capazes de tratar um conjunto de caminhos de cada vez são conhecidos como métodos de **sensibilização de múltiplos caminhos** (*concurrent or multiple path sensitization*). No entanto, este método de sensibilização de múltiplos caminhos não consegue identificar o caminho responsável pelo atraso crítico.

Existe também uma aproximação **mista**, que identifica um conjunto de potenciais caminhos críticos através de sensibilização de múltiplos caminhos e, então, aplica sensibilização individual de caminhos para identificar o caminho crítico.

Desta forma, é possível classificar os algoritmos de FTA quanto ao número de caminhos tratados ao mesmo tempo (GÜNTZEL, 2000):

- Sensibilização individual de caminhos,
- Sensibilização de múltiplos caminhos,
- Sensibilização mista.

Ou então pelos métodos para testar a sensibilização do caminho:

- Baseados em ATPG (*ATPG-based*)
- Baseados em solvabilidade (*SAT-based*)
- Outros (e.g., BDDs)

2.3 Critérios de sensibilização

O conjunto de condições usadas para decidir se um caminho é ou não sensibilizável é chamado de **critério de sensibilização**. Este conjunto de condições considera os valores lógicos necessários nas entradas pertencentes ao caminho e nas suas entradas laterais, e em alguns casos, o tempo no qual estes valores estabilizam.

Güntzel destaca quatro critérios de sensibilização como sendo os mais relevantes: **sensibilização estática** (BENKOSKI, 1991), **co-sensibilização estática** (DEVADAS, 1991), **viabilidade** (MCGEER, 1989) e **exato sob o modo flutuante** (CHEN, 1991). Os dois últimos são dependentes do tempo em que os sinais estabilizam. Todos estes critérios são válidos somente sobre o modo flutuante de operação. Porém, antes de mostrar a definição de cada um destes critérios, serão introduzidos alguns conceitos usados nestas definições.

Em uma grafo representando um circuito combinacional, um caminho P é uma seqüência alternada de nodos (n) e vértices (e). Cada vértice e_i, com $1 \leq i \leq n$, conecta a saída do nodo n_i com a entrada do nodo n_{i+1}. Todo nodo n_i, com $1 \leq i \leq n$, representa uma porta lógica. O nodo n_0 é uma entrada primária do circuito e o nodo n_{n+1} é uma saída primária do circuito. Um caminho completo pode ser representado por (n_0, n_1, n_2, ..., n_n, n_{n+1}) ou por (e_0, e_1, e_2, ..., e_n). Porém, para não causar confusão, o mais comum é representar o caminho por P=(n_0, e_0, n_1, e_1, n_2, e_2, ..., n_n, e_n, n_{n+1}).

A entrada e_i da porta n_{i+1} é chamada **entrada no caminho**. As demais entradas da porta são chamadas **entradas laterais** de n_{i+1}. O conjunto de todas entradas laterais ao longo de P são chamadas entradas laterais de P.

O **valor controlante** de uma porta lógica é o valor lógico que, aplicado em uma de suas entradas, determina sozinho o valor lógico da sua saída. O **valor controlado** de uma porta lógica é o valor lógico que resulta da aplicação de um valor controlante em uma de suas entradas. Considerando a Figura 2.2, o valor controlante de uma porta AND é 0 e seu valor controlado também é 0, enquanto que para uma porta OR o valor lógico 1 será o seu valor controlante e controlado.

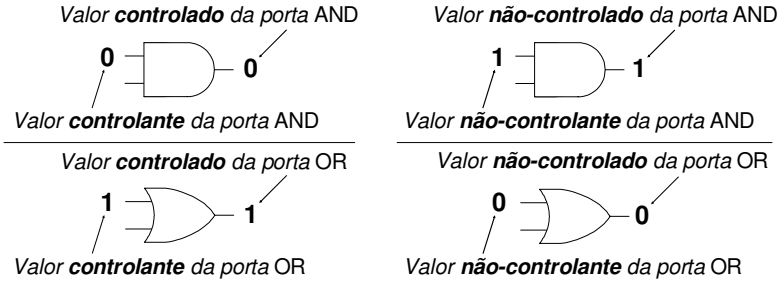

Figura 2.2: Valores (não)controlante e (não)controlado para portas AND e OR.

O valor **não-controlante** de uma porta lógica é o valor lógico que, aplicado em uma de suas entradas, não determina sozinho o valor lógico da sua saída. Valor **não-controlado** de uma porta é o valor lógico que resulta, na sua saída, da aplicação de um valor não-controlante em todas as suas entradas.

2.3.1 Critério de sensibilização estática

Definição: um caminho *P* é dito sensibilizável **estaticamente** se, e somente se, existe pelo menos um vetor de entrada *v* que é capaz de colocar todas as entradas laterais do caminho em valores não-controlantes (NC).

Este critério de sensibilização leva em consideração somente a funcionalidade lógica das portas e os valores lógicos nas entradas, ou seja, não considera nenhuma informação de atraso para tentar justificar o caminho. A Figura 2.3 ilustra a definição de caminho estaticamente sensibilizável, sendo e_i a entrada no caminho *P*.

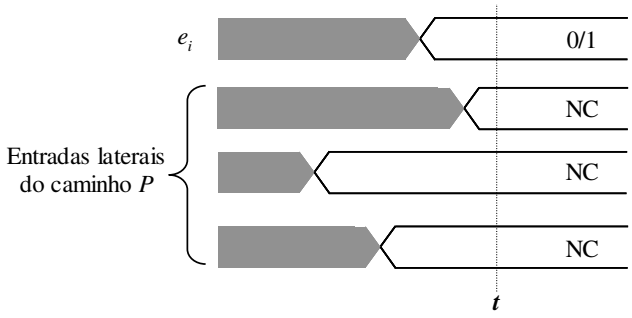

Figura 2.3: Condições para sensibilização estática.

2.3.2 Critério de co-sensibilização estática

Definição: um vetor *v* **co-sensibiliza estaticamente** um caminho *P* se, e somente se, cada porta lógica n_{i+1} com saída em valor controlado tem sua entrada no caminho e_i em um valor controlante (C).

Este critério também é independente dos valores de atraso, porém é menos restritivo que o critério estático. As Figura 2.4 (a) e (b) ilustram as condições para co-sensibilização estática.

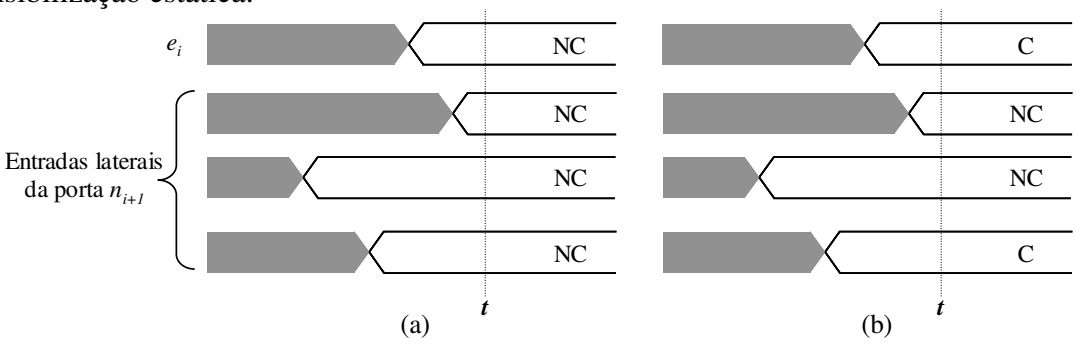

Figura 2.4: Condições para co-sensibilização estática.

2.3.3 Critério de viabilidade

Definição: um caminho *P* é dito **viável** se, e somente se, existe pelo menos um vetor de entrada *v* tal que as entradas laterais do caminho: a) possuam valores não-controlantes (NC) ou; b) estabilizem em um tempo **maior** que o tempo no qual a entrada no caminho e_i estabiliza.

Este critério considera, além dos valores lógicos, o tempo em que os sinais estabilizam para testar se um caminho é verdadeiro. A Figura 2.5 ilustra as condições para viabilidade de um caminho.

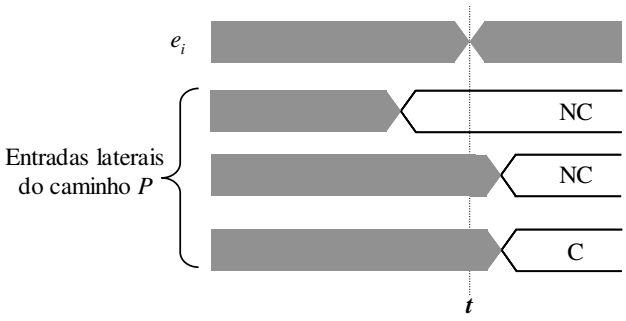

Figura 2.5: Condições para viabilidade do caminho.

2.3.4 Critério exato

Definição: um caminho P é dito **exatamente sensibilizável** sob o modo flutuante se, e somente se, existe pelo menos um vetor de entrada v tal que satisfaça as seguintes condições para cada porta lógica n_{i+1} pertencente a P:

(a) Se a saída de n_{i+1} está em um valor controlado, então sua entrada no caminho e_i precisa estar em um valor controlante (C) e, ainda, ter estabilizado neste valor em um tempo **menor** do que o tempo no qual as suas entradas laterais com valor controlante (C) estabilizaram;

(b) Se a saída de n_{i+1} está em um valor não-controlado, então a sua entrada no caminho e_i precisa ter estabilizado em um tempo **maior** do que o tempo no qual as suas entradas laterais estabilizaram.

Este critério também considera o tempo em que os sinais estabilizam para testar se um caminho é verdadeiro. As condições para um caminho ser sensibilizável pelo critério exato estão delineadas nas Figura 2.6 (a) e (b).

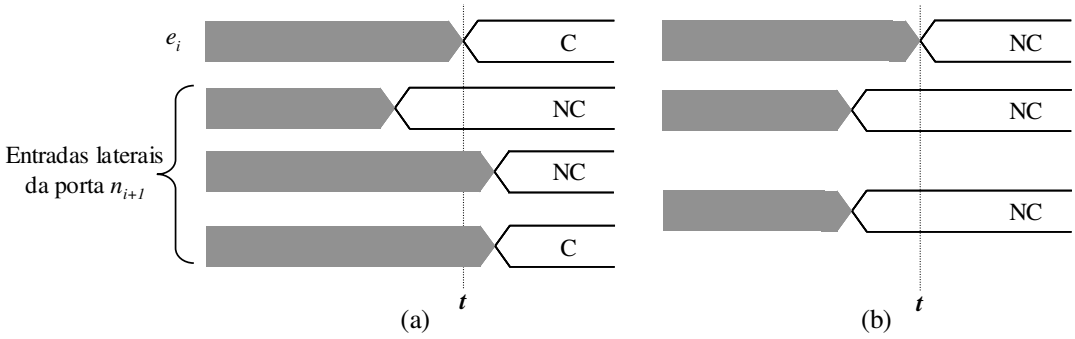

Figura 2.6: Condições para sensibilização exata sob o modo flutuante.

O comportamento destes quatro critérios de sensibilização em circuitos combinacionais apontam para as seguintes afirmações (GÜNTZEL, 2000):

- O critério **estático** é uma condição **suficiente** para declarar um caminho como sensibilizável, mas **não é necessária** para tal. Ou seja, ele pode **subestimar** o atraso do circuito por não conseguir sensibilizar um caminho com atraso maior, o qual pode ser responsável pelo atraso do circuito.
- O critério de **co-sensibilização estática** é uma condição **necessária** para declarar um caminho como sensibilizável, mas **não é suficiente** para tal. Isto significa que, utilizando este critério, o atraso do circuito pode ser

pessimista, pois seria possível sensibilizar um caminho que **não** seja responsável pelo atraso do circuito (caminho falso).

- **Viabilidade não implica** em co-sensibilização estática, ou seja, um caminho pode ser viável sem ser co-sensibilizável estaticamente, e assim, ser falso.
- O critério **exato** é uma condição **necessária e suficiente** para declarar um caminho sensibilizável sob o modo flutuante, sendo mais **fraca** que o critério estático e mais **forte** que a co-sensibilização estática.

A Figura 2.7 ilustra uma comparação entre os critérios de sensibilização, mostrando a relação do atraso obtida com a utilização de cada critério com o atraso real do circuito.

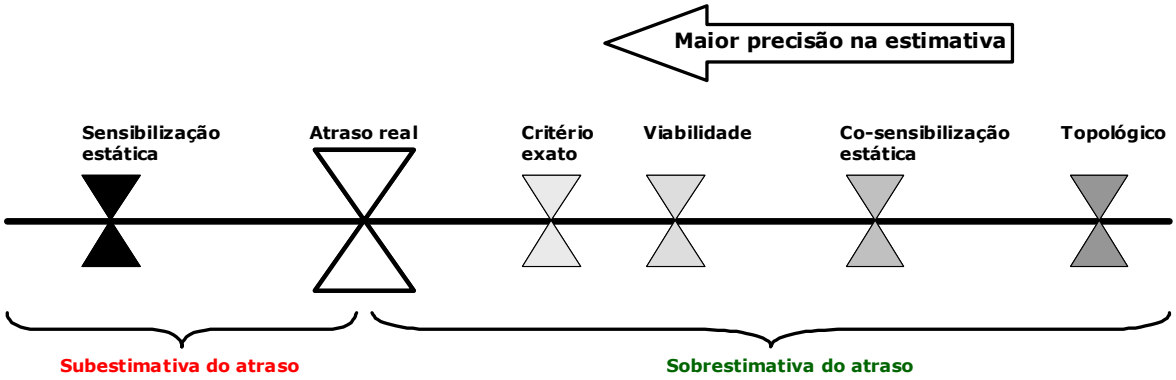

Figura 2.7: Comparação entre critérios de sensibilização.

2.4 Atraso de portas lógicas

O atraso de uma porta lógica é determinado, a grosso modo, pelo tempo necessário para que a mesma carregue ou descarregue sua capacitância de saída C_L através de seus transistores e, assim, transmita uma transição lógica da sua entrada até sua saída. Mais especificamente, é o tempo medido entre o ponto em que o sinal de entrada atinge 50% do valor lógico 1 e o ponto onde o sinal de saída atinge os mesmos 50%. A Figura 2.8 ilustra esta definição, sendo t_{dhl} o tempo de propagação de descida (*high-low*) e t_{dlh} o tempo de propagação de subida (*low-high*).

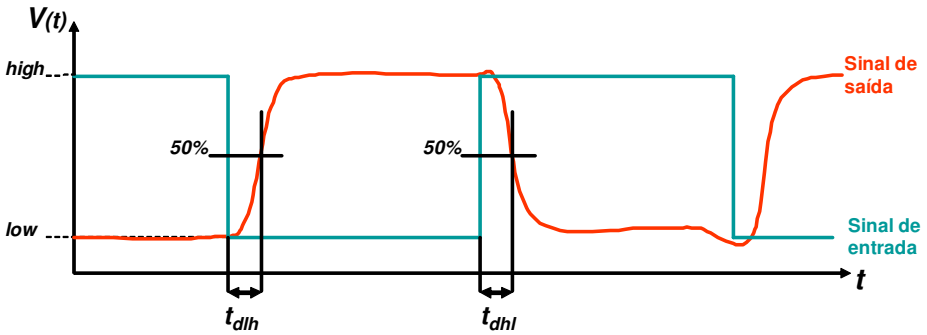

Figura 2.8: Tempos de propagação de subida e de descida.

2.4.1 Abordagem *switch-level*

Em uma análise no nível de chaves (*switch-level*), o circuito geralmente é dividido em estágios, onde cada estágio é uma rede de transistores conectados por seus nodos de fonte e dreno, indo desde a alimentação até o *gate* de um transistor. Como pode ser visto

na Figura 2.9, um estágio pode representar qualquer tipo de porta lógica e até mesmo transistores de passagem.

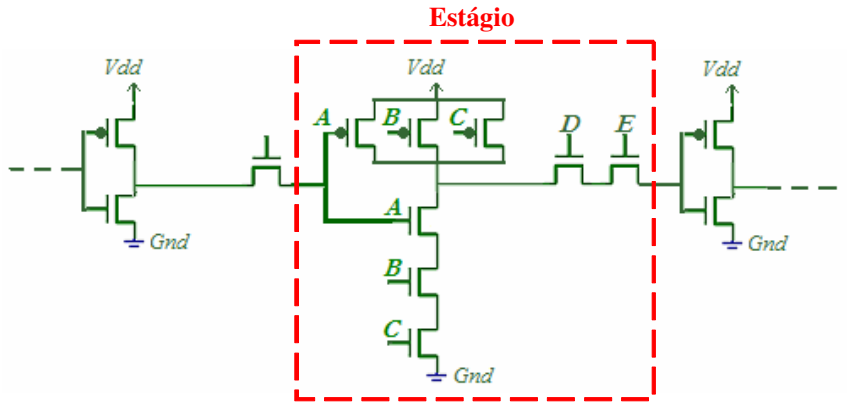

Figura 2.9: Ilustração de um estágio.

Cada transistor de um estágio é aproximado por uma resistência equivalente entre seus terminais de dreno e fonte, e a cada nodo é associada uma capacitância. Desta maneira, o estágio acaba sendo modelado como uma rede *RC*. A Figura 2.10 mostra como ficaria a representação *RC* do estágio mostrado na Figura 2.9.

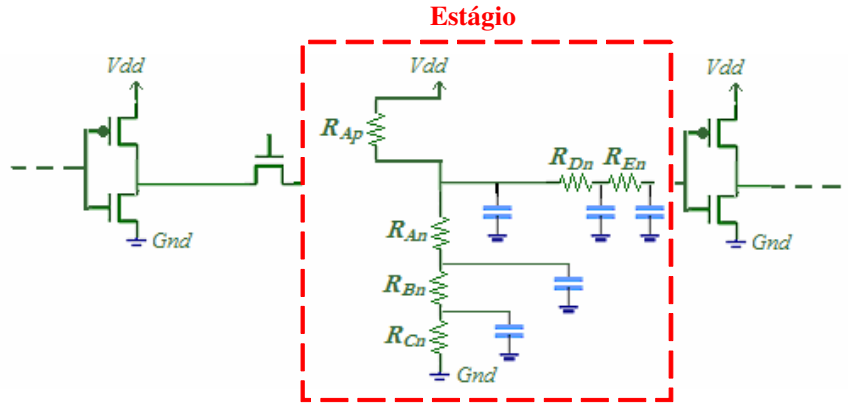

Figura 2.10: Representação *RC* de um estágio.

Neste tipo de análise, os métodos diferem basicamente na forma como aproximam o transistor por uma resistência e na maneira de computar o atraso da rede *RC*. Diferentes métodos para estimativa de atraso em redes *RC* já foram propostos na literatura e alguns deles podem ser vistos na seção 2.5.

Os modelos para aproximar um transistor por uma resistência efetiva têm sua precisão dependente de uma série de fatores que controlam a capacidade de condução de corrente deste transistor (UEBEL, 1995): ***i***) o tipo de transistor (P ou N); ***ii***) comprimento (*L*) e largura (*W*) do canal; ***iii***) tempo de transição do sinal de entrada; ***iv***) disposição do transistor na rede (paralelo, série ou de passagem); ***v***) capacitância drenada; ***vi***) efeito de corpo.

Uma conseqüência natural deste tipo de análise (*switch-level*) é que as interconexões, que também podem ser modeladas como redes *RC*, são implicitamente consideradas no cálculo do atraso sem que nenhum esforço adicional seja feito para isso.

2.4.2 Abordagem *gate-level*

Tipicamente, projetos de grandes circuitos digitais são sintetizados no nível de portas lógicas. Podemos então sentir a necessidade de possuir modelos de atraso que tenham boa precisão e que possam ser usados em diferentes níveis do projeto. Analisar o atraso de um circuito neste tipo de abordagem é computacionalmente mais simples, uma vez que cada porta é substituída por um modelo equivalente, eliminando-se seus nodos internos (SAPATNEKAR, 1993-b).

Em modelos de atraso mais simplistas, o atraso da célula geralmente é caracterizado de forma linear e é composto por duas partes: um atraso constante e inerente da tecnologia (*A*) e um atraso relacionado com a capacidade (*B*) da célula em drenar uma carga C_L.

$$t_{dhl(lh)} = A + B \,.C_L \tag{2-1}$$

Estas representações de atraso eram suficientemente precisas nas tecnologias micrônicas. Porém, nas novas tecnologias submicrônicas, fatores como capacitâncias de acoplamento, transição do sinal de entrada e saturação da velocidade dos portadores que provocam efeitos de segunda ordem precisam ser considerados para uma caracterização precisa do atraso das portas lógicas.

Equações empíricas são equações que modelam o atraso considerando a capacitância de carga C_L e o sinal de entrada τ_{in} de uma porta lógica. Coeficientes para cada tipo de porta são obtidos empiricamente, geralmente através de calibrações com simulação elétrica e regressão linear. A equação (2.1) é uma equação empírica, porém outras formulações conhecidas como *k-factor equations*, mais complexas e precisas, também já foram propostas. Em (CONG, 1996) é mostrada uma equação empírica que modela o atraso da porta sob a seguinte forma:

$$t_{dhl(lh)} = (k_1 + k_2 C_L)\tau_{in} + k_3 C_L^{\,3} + k_4 C_L + k_5 \tag{2-2}$$

Sapatnekar (1993-b) cita que métodos de interpolação de ***Look-up Tables*** (LUTs) são métodos bastante eficientes e rápidos para encontrar o atraso de uma porta lógica, sendo bastante usados em fluxos de síntese baseados em células padrão (*standard cells*). Na montagem desta tabela são considerados vários parâmetros que afetam o atraso de uma porta, como tamanho dos transistores, carga de saída, sinal de entrada, entre outros. Então, considerando a variação do valor de cada parâmetro de forma independente, são realizadas simulações elétricas para obter os valores de atraso a serem anotados na tabela. Uma interpolação, linear ou não, garante a possibilidade de obter valores de atraso para qualquer combinação de valores dos parâmetros.

Uma outra abordagem é a utilização de **macromodelos**. Nesta abordagem, também conhecida como formulação explícita, utiliza-se uma expressão analítica do atraso que inclui características da porta e parâmetros da tecnologia. Estes parâmetros podem ser obtidos também de forma analítica ou então empiricamente através de medições ou simulações elétricas. Geralmente é desenvolvido um modelo de atraso para um inversor e as demais portas lógicas são então mapeadas para um inversor "equivalente". O transistor NMOS (PMOS) do inversor equivalente é dimensionado para ter a mesma capacidade de condução de corrente que a estrutura de *pull-down* (*pull-up*) da porta modelada.

Weste (1993) apresenta um modelo analítico de inversor que leva em consideração a carga de saída da porta (C_L), uma constante de tecnologia para uma tensão de alimentação específica (A_N,A_P) e as dimensões dos transistores que compõem a porta lógica (β_N,β_P):

$$t_{dhl(lh)} = A_{N(P)} \frac{C_L}{\beta_{N(P)}} \tag{2-3}$$

$$A_N = \frac{1}{V_{DD}(1-v_{tn})}\left[\frac{2v_{tn}}{1-v_{tn}} + \ln\left(\frac{2(1-v_{tn})-V_O}{V_O}\right)\right] \tag{2-4}$$

$$A_P = \frac{1}{V_{DD}(1+v_{tp})}\left[\frac{-2v_{tp}}{1+v_{tp}} + \ln\left(\frac{2(1+v_{tp})-V_O}{V_O}\right)\right] \tag{2-5}$$

$$v_{tn} = \frac{V_{THn}}{V_{DD}} \tag{2-6}$$

$$v_{tp} = \frac{V_{THp}}{V_{DD}} \tag{2-7}$$

$$V_O = \frac{V_{out}}{V_{DD}} \tag{2-8}$$

$$\beta_{N(P)} = \frac{\mu\varepsilon}{t_{ox}}\left(\frac{W_{N(P)}}{L}\right) \tag{2-9}$$

onde μ, ε, t_{ox} e $V_{tn(p)}$ são parâmetros da tecnologia e V_{DD}, $W_{N(P)}$ e L são parâmetros de projeto. Também é mostrado que o coeficiente $A_{N(P)}$ pode ser extraído empiricamente através de simulação e que para estender o modelo para outras portas lógicas bastaria substituir $\beta_{N(P)}$ por um β_{eff}, onde:

$$\beta_{eff} = \frac{1}{\frac{1}{\beta_1} + \frac{1}{\beta_2} + \ldots + \frac{1}{\beta_j}} \tag{2-10}$$

para j transistores em série na estrutura *pull-down* (*pull-up*) da porta ou:

$$\beta_{eff} = \beta_1 + \beta_2 + \ldots + \beta_k \tag{2-11}$$

para k transistores em paralelo, considerando que todos os transistores da estrutura estão conduzindo. Para considerar o efeito do tempo de transição do sinal de entrada da porta, a equação de atraso é modificada para:

$$t_{dhl(lh)} = t_{dhls(lhs)} + \frac{\tau_{in}}{6}\left(1-2v_{tn(p)}\right) \tag{2-12}$$

onde $t_{dhls(lhs)}$ é o atraso calculado em (2.3) e τ_{in} é o tempo de transição do sinal de entrada da porta. A equação (2.12) só é válida para certas condições de τ_{in}.

Sakurai (SAKURAI, 1991) usa o seu conhecido modelo de transistor *alpha-power* para derivar o atraso de portas CMOS considerando alguns efeitos presentes em

tecnologias submicrônicas. Ele mostra duas equações que expressam o atraso do inversor, sendo uma para transições lentas e outra para transições rápidas do sinal de entrada. Um valor de tempo crítico de transição (τ_{in0}) é definido para realizar a classificação dos tempos de transição de entrada das portas do circuito.

Para transições rápidas do sinal de entrada ($\tau_{in} < \tau_{in0}$):

$$t_{dhl(lh)} = \tau_{in}\left\{\frac{1}{2} - \frac{1 - v_{tn(p)}}{n+1} + \frac{\left(v_V - v_{tn(p)}\right)^{n+1}}{(n+1)\left(1 - v_{tn(p)}\right)^{n}}\right\} + \frac{1}{2}\frac{C_L V_{DD}}{I_D} \tag{2-13}$$

Para transições lentas do sinal de entrada ($\tau_{in} > \tau_{in0}$):

$$t_{dhl(lh)} = \tau_{in}\left[v_{tn(p)} - \frac{1}{2} + \left\{\left(v_V - v_{tn(p)}\right)^{n+1} + \frac{(n+1)\left(1 - v_{tn(p)}\right)^{n}}{2\tau_{in} I_D / C_L V_{DD}}\right\}^{1/n+1}\right] \tag{2-14}$$

onde v_V é a razão entre a tensão de *threshold* lógico do inversor (V_{INV}) e V_{DD}. I_D é a corrente de dreno do transistor, dada pelo modelo *alpha-power,* e n é o índice de saturação de velocidade da tecnologia. Para que as equações sejam utilizadas para calcular o atraso de outras portas lógicas CMOS, os valores de n e I_D precisam ser recalculados considerando o número de transistores em série nas estruturas de *pull-up* e *pull-down* da porta. *Sakurai* mostra ainda que a razão entre o atraso de portas lógicas com n transistores em série em suas estruturas de *pull-up* e *pull-down* e o atraso de um inversor não é mais igual a n, e que esta razão vem diminuindo junto com a redução das dimensões do transistor em tecnologias DSM.

Daga et al. (1999) desenvolveram um modelo de inversor que considera efeitos de segunda ordem, provocados por fatores como corrente de curto-circuito, capacitância de acoplamento e saturação da velocidade dos portadores, presentes em tecnologias submicrônicas. O mapeamento para portas mais complexas usa uma técnica de redução de transistores conectados em série, considerando a posição da entrada que transiciona e efeitos não-lineares introduzidos pelo sinal de entrada.

Em (QIAN, 1994) é proposto um método para considerar o efeito das interconexões no atraso de um estágio lógico em uma abordagem *gate-level*. A análise do atraso da porta e da conexão é feita separadamente. Então, os atrasos resultantes são somados de forma apropriada para compor o atraso do estágio. Primeiramente é computado o atraso e a capacitância "efetiva" da conexão vista pela porta. Em seguida estes valores de capacitância são usados no cálculo do atraso da porta lógica. A seção 2.5 referencia o cálculo de capacitância efetiva na presença de elementos resistivos nas interconexões.

2.4.3 Modelo computacional de atraso para portas lógicas

Computacionalmente existem três formas de assinalar o atraso de uma porta lógica (GÜNTZEL, 2000). O mais simples é assumir apenas um atraso para cada porta (*single delay per gate*). Uma derivação deste é assumir um par de atrasos para cada porta (*pair of delays per gate*), sendo um atraso de subida e um atraso de descida. A terceira maneira é assumir um par de atrasos para cada entrada da porta, também conhecido como modelo pino-a-pino (*pin-to-pin*).

Em uma análise de pior caso para o atraso de uma porta lógica, consideram-se todos os caminhos de transistores que conectam a saída à alimentação da porta através das estruturas de *pull-down* e *pull-up*. Na hora de computar o atraso é escolhido então aquele caminho que garante o pior atraso de propagação. Além disso, considera-se a

pior de todas as possibilidades de chaveamento nas entradas da porta, o que recai sobre um único transistor chaveando. Logo, podemos classificar a precisão do modelo computacional em relação ao atraso real da porta da seguinte forma:

pin-to-pin **>** *pair of delays per gate* **>** *single delay per gate*

2.5 Atraso das interconexões

Com a redução das dimensões físicas dos componentes de um CI para valores nanométricos nas novas tecnologias DSM, a resistência por unidade de comprimento das linhas de metal está crescendo. Porém, devido aos efeitos de *fringing* e acoplamento entre linhas, a capacitância por unidade de comprimento destas linhas não está diminuindo. Além disso, com a crescente complexidade dos circuitos, o comprimento médio dos fios também não está sendo reduzido. Isto significa que ao mesmo tempo em que a velocidade de chaveamento das portas está cada vez maior, o tempo de propagação dos sinais pelas linhas de metal do circuito está crescendo, tornando fundamental a influência das interconexões de metal no desempenho de um circuito.

Os modelos utilizados para estimar o atraso de conexões envolvem desde o simplista modelo *RC* concentrado até sofisticados modelos que combinam momentos de alta ordem (*high order moment matching model*). Usado em tecnologias onde é possível ignorar a resistência da conexão, o modelo *RC* concentrado estima o atraso da porta por $0.69RC$, onde R é a resistência do *driver* e C é a soma das capacitâncias da conexão e das capacitâncias de carga (CONG, 1996).

Porém, em tecnologias onde a resistência da conexão não pode ser desprezada, as interconexões são geralmente modeladas como árvores *RC*, assim como mostrado na Figura 2.11. A grande maioria dos métodos atuais de estimativa de atraso de conexões considera este tipo de modelagem. Embora os efeitos indutivos das linhas de metal tenham se tornado cada vez mais importantes nas novas tecnologias, estes não serão apresentados aqui.

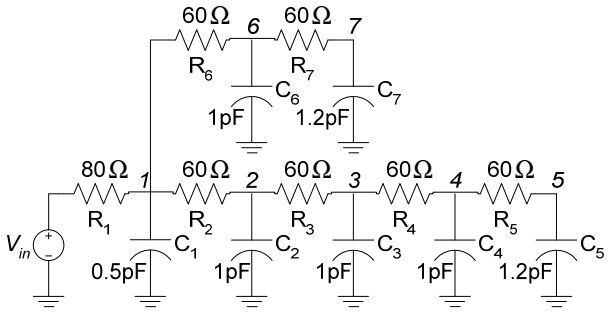

Figura 2.11: Exemplo de árvore *RC* (GUPTA, 1997).

Elmore (1948) aproximou o atraso de propagação de uma resposta monotônica para uma entrada do tipo degrau como metade da resposta ao impulso. *Rubenstein et al.* (1983) provaram que a resposta de uma rede *RC* é principalmente monotônica, e com isso descobriram um novo uso para a métrica *Elmore Delay* (ED), que é estimar o atraso de conexões. Como ED é uma aproximação do atraso real, eles propuseram limites superiores e inferiores para este atraso. Porém, *Gupta et al.* (1997) mostraram que ED por si mesmo já é um limite superior absoluto para o atraso real de uma rede *RC* e, que, mesmo para outros sinais de entradas, como uma rampa, ED continua sendo uma sobrestimativa do atraso da conexão.

Se considerarmos *h(t)* como a resposta ao impulso em um nodo da rede *RC* (ou *RLC*), a função de transferência *H(s)* do circuito, que é a transformada de Laplace de *h(t)*, pode ser representada por:

$$H(s) = \int_0^\infty h(t)e^{-st}dt = \sum_{i=0}^{\infty} \frac{(-1)^i}{i!} s^i \int_0^\infty t^i h(t)dt \tag{2-15}$$

O *i*-ésimo momento (m_i) da função de transferência *H(s)* é o coeficiente sem sinal da *i*-ésima potência de *s* na equação (2.15):

$$m_i = \frac{1}{i!} \int_0^\infty t^i h(t)dt \tag{2-16}$$

A proposta de *Elmore* foi analisar, para um circuito, a sua resposta ao impulso *h(t)*, que é a derivada da resposta ao degrau *g(t)*. Observando, na Figura 2.12, a resposta ao impulso como uma função distribuição, o atraso de 50% é a mediana (τ) desta função.

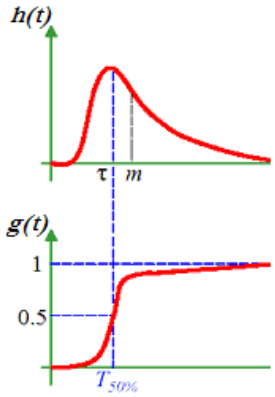

Figura 2.12: Resposta ao impulso e ao degrau.

Para obter a mediana de uma função é preciso resolver a equação (2.16), o que não é uma tarefa simples. Então, *Elmore* aproximou a mediana da função *h(t)* pela sua média (*m*). Como pode ser visto na Figura 2.12, a precisão desta aproximação é dependente da simetria da resposta ao impulso (GUPTA, 1997). Em uma distribuição simétrica, a média é exatamente igual à mediana.

$$\int_0^\tau h(t)dt = \int_\tau^\infty h(t)dt = 0.5 \tag{2-17}$$

$$m = \int_0^\infty t.h(t)dt \tag{2-18}$$

Podemos notar na equação (2.18) que *m* é a definição de primeiro momento da resposta ao impulso do circuito. Logo, é possível calcular indiretamente a média da função distribuição. Assim, a equação (2.19) descreve ED, ou primeiro momento de uma resposta ao impulso, em cada nodo *n* de uma árvore *RC*:

$$ED_n = \sum R_{kn} C_k \tag{2-19}$$

onde R_{kn} é a resistência do trecho que é comum ao caminho entre o *driver* e o nodo *k* e o caminho entre o *driver* e o nodo *n*. C_k é a capacitância entre o nodo *k* e o terra.

Mesmo tendo sido proposto há mais de 50 anos, ED ainda é uma das métricas mais usadas para computar rapidamente o atraso de conexões devido a ser (KASHYAP, 2000):

- Uma métrica expressa apenas em termos dos resistores e capacitores do circuito (ou em termos de parâmetros de projeto);
- Sempre uma sobrestimativa do atraso real em qualquer nodo do circuito e para qualquer tipo de sinal de entrada;
- Uma estimativa precisa do atraso real para os nodos mais afastados do *driver*.

O principal contraponto da métrica ED é sua imprecisão relativa, que pode variar para os diferentes nodos do circuito. *Pillegi* (1998) diz que se ED for usado como constante de tempo dominante, assumindo que o circuito tem um pólo dominante, então a tensão em um nodo deste circuito será igual a:

$$v(t) = V_{vdd}.(1 - e^{-p_d t}) \tag{2-20}$$

onde P_d =ED é o pólo dominante. Resolvendo (2.20) para encontrar o tempo em que $v(t)$=0.5V_{vdd} (quando o sinal cruza metade do valor da tensão de alimentação) encontramos o que *Pillegi* chama de *Scaled Elmore Delay* (SED):

$$SED = \frac{\ln(2)}{p_d} \approx 0{,}69ED \tag{2-21}$$

Porém, esta aproximação não resolve o problema de precisão relativa descrito anteriormente, pois ele apenas "desloca" o valor do atraso de todos os nós do circuito. E ainda pode subestimar o atraso para certos nodos, perdendo uma das vantagens do ED.

Especialmente por não considerar o efeito de **blindagem da resistência** (*resistance shielding*), o erro introduzido por ED é muito grande nos nodos mais próximos do *driver*. O efeito de blindagem da resistência atua mascarando a capacitância vista por um nodo. Considere o nodo 2 da Figura 2.11. Quanto maior forem as resistências entre este nodo e os nodos que estão além dele (nodos 3, 4 e 5), menor será o efeito das capacitâncias C_3, C_4 *e* C_5 sobre o atraso de propagação do sinal até o nodo 2.

Alguns trabalhos propõem o cálculo de uma capacitância "efetiva" para substituir os valores de capacitância da rede *RC*, considerando o efeito de blindagem da resistência. Em (KASHYAP, 2000) é proposto que, para cada nodo *n*, a porção da rede *RC* além de *n* seja simplificada para um modelo π (*CRC*), como na Figura 2.13:

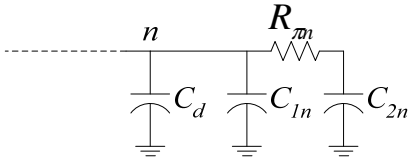

Figura 2.13: Simplificação da porção além de *n* da rede RC para um modelo π.

Então, o resistor $R_{\pi n}$ e o capacitor C_{2n} são substituídos por um capacitor "equivalente" (C_{eff}), que é usado em (2.19) para computar o atraso da rede.

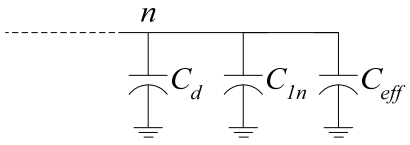

Figura 2.14: Capacitância efetiva (C_{eff}) vista do nodo *n*.

Contudo, para conseguir estimar o atraso de uma conexão com maior precisão, pode-se usar momentos de ordens mais altas. *AWE* (*Asymptotic Waveform Evaluation*) é um método generalizado para aproximar a resposta de um circuito por combinação de momentos (PILLAGE, 1990). O método *AWE* usa os primeiros $2q$ momentos para construir uma função transferência $\hat{H}(s)$ que aproxima a função de transferência real *H(s)*. Os q pólos e q resíduos utilizados para construir $\hat{H}(s)$ são determinados pelas condições iniciais de contorno e pelos primeiros $2q$ momentos de *H(s)*. A escolha do número de momentos (ordem de q) depende da precisão desejada. Os métodos chamados ***modelo de 2-pólos*** consideram o caso onde q=2 (segunda ordem) e aproximam o atraso usando apenas os três primeiros momentos, podendo ser mais facilmente usados para estimar o atraso de conexões (PILEGGI, 1998)(TUTUIANU, 1996).

2.6 Conclusão

Por ser computacionalmente mais barata que a maioria dos métodos de simulação e por não exigir que o projetista forneça vetores de teste, a análise de *timing* é atualmente o método mais utilizado para verificar o atraso de circuitos digitais. É também o método mais utilizado para selecionar o caminho crítico em etapas de otimização de atraso. A precisão das ferramentas de análise *timing* é totalmente dependente da qualidade dos modelos de atraso dos componentes do circuito e do método utilizado para fazer a procura do atraso e do caminho crítico através do grafo que representa o circuito.

Os métodos utilizados para procurar o atraso e o caminho crítico no grafo mostram um claro compromisso entre rapidez e precisão. TTA é o método que apresenta maior desempenho e também por isso é amplamente utilizado. Porém, a presença de caminhos falsos no circuito compromete a precisão deste método. Métodos de FTA, que visam tratar circuitos que contenham caminhos não-sensibilizáveis, diferem entre si pelos modelos, critérios e algoritmos usados na busca do atraso crítico. Ao contrário dos métodos de enumeração de atrasos, a enumeração de caminhos permite a identificação das portas lógicas pertencentes aos caminhos críticos, e esta habilidade é fundamental em etapas e métodos de otimização de atraso.

Os métodos de análise *switch-level* não precisam se preocupar com o modelo computacional de atraso da porta a ser escolhido, pois são inerentemente um modelo pino-a-pino. Porém, estes métodos geralmente usam elementos lineares (resistores) para aproximar os transistores do circuito, e estes transistores possuem um comportamento essencialmente não-linear. Desta forma, o erro na estimativa do atraso de cada estágio do circuito pode ser considerável.

Quando usamos LUTs para obter o atraso de portas lógicas, a precisão do valor do atraso é dependente do número de parâmetros e do número de valores que cada parâmetro assume, o que indesejadamente também faz crescer o tamanho da tabela a ser interpolada. E isto pode tornar o esforço para montar esta tabela inviável. Além disso, métodos baseados em LUTs e equações empíricas precisam de uma grande tempo de pré-caracterização de cada porta em cada vez que muda a tecnologia alvo da implementação física.

A dificuldade da utilização de macromodelos está em encontrar uma equação adequada e precisa, especialmente com a introdução de efeitos de segunda ordem nas novas tecnologias DSM. Porém, sua simplicidade e o fato de não necessitar pré-

caracterizar cada porta lógica, mas somente alguns parâmetros da tecnologia, tornam esta abordagem a melhor escolha a ser adotada quando os circuitos são gerados por métodos independentes de biblioteca de células.

O atraso das conexões de metal é, atualmente, crucial na determinação do atraso do circuito. Por levar em conta momentos de alta ordem, o método *AWE* consegue atingir uma precisão extremamente alta quando comparado a uma simulação elétrica. Entretanto, na prática não é possível representar os pólos e resíduos em termos dos parâmetros de projeto, tornando *AWE* um método difícil de ser usado em algumas etapas da síntese. O modelo de 2-pólos (ou de ordem mais alta) é usualmente muito mais complexo e mais preciso que ED, porém é muito mais rápido que uma simulação elétrica ou mesmo que *AWE*, mantendo um compromisso entre simplicidade e rapidez. A métrica ED tem sido a mais usada para a estimativa de atraso de conexões por ser extremamente rápida e expressa em termos de elementos *RC* que podem ser facilmente derivados dos parâmetros de projeto. Estas características fazem com que ED possa ser usada nas etapas de otimizações de atraso que alteram as conexões do circuito, tornando convergentes as otimizações feitas nas diferentes etapas da síntese.

3 TÉCNICAS DE OTIMIZAÇÃO DE ATRASO

Como pode ser visto na Figura 1.1, a identificação e otimização das violações de atraso podem ser realizadas durante diferentes etapas do fluxo de projeto de um CI. Quanto mais alto está o nível de abstração do projeto, maior a liberdade e a possibilidade de otimização do desempenho do circuito.

No nível arquitetural, um projeto eficiente geralmente requer a experiência do projetista sobre os algoritmos que irão implementar as funções desejadas, sobre as diferentes versões de operadores (somador, multiplicador, ULA) ou sobre quão rápido vai ser o acesso à memória. Definições sobre *pipeline*, número de estágios lógicos que cabem num período de *clock* e limitações de *fanin* e *fanout* ajudam na otimização do atraso durante a passagem do nível RTL para o nível lógico (WESTE, 1993).

A partir do momento em que temos um *netlist* de portas lógicas, ou de transistores, já é possível tomar decisões para otimizar o atraso no nível de leiaute.

3.1 Atraso de circuitos combinacionais

Um circuito digital síncrono consiste, tipicamente, de múltiplos estágios de lógica combinacional que ficam entre registradores controlados por um sinal de relógio (*clock*), conforme ilustrado na Figura 3.1.

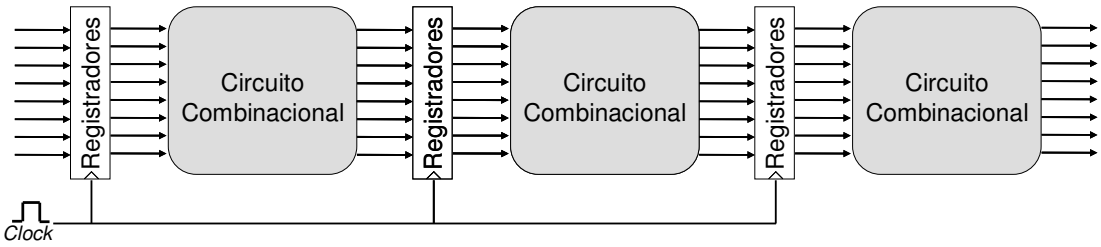

Figura 3.1: Circuito Seqüencial.

Um circuito seqüencial sem violações de atraso é aquele que tem todos os estágios combinacionais com seus sinais de saída estáveis, em seu valor final, disponíveis para o registrador seguinte antes da chegada da borda ativa do sinal de relógio. Ou seja, que cada estágio combinacional tenha um atraso crítico menor que o período de relógio. A Figura 3.2 ilustra a necessidade de que os sinais estejam estabilizados ao fim do período de relógio para que não haja erro no funcionamento do circuito.

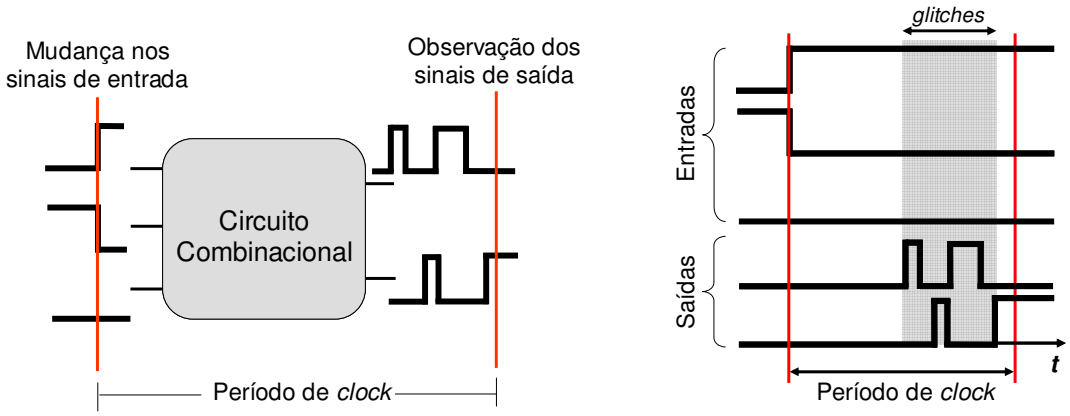

Figura 3.2: Atraso em estágios combinacionais.

Os métodos de otimização de atraso a serem mostrados nas seções seguintes abordarão somente otimização de circuitos combinacionais, uma vez que ao tratar com circuito seqüenciais podemos garantir que o CI irá operar na freqüência desejada se os seus estágios combinacionais tiverem sido devidamente otimizados.

3.2 Dimensionamento de transistores

Uma porta lógica torna-se lenta quando os seus transistores de *pull-up* e *pull-down* são muito pequenos para drenar a corrente necessária para carregar e descarregar sua capacitância de saída. Logo, esta porta pode ser acelerada pelo aumento do tamanho dos seus transistores ou então pela redução da capacitância de saída (HILL, 1989). Entenda-se aqui tamanho do transistor como sendo a largura (W) do seu canal, já que em circuitos digitais CMOS geralmente o comprimento (L) do canal é mantido no valor mínimo permitido pela tecnologia.

Em essência, o problema de dimensionar transistores para reduzir o atraso de um circuito recai sobre encontrar o tamanho de cada transistor de forma a minimizar a função custo e garantir que as restrições de atraso sejam atingidas. Como o aumento dos transistores ocasiona um acréscimo de área e, geralmente, de consumo, um compromisso entre atraso, área e consumo de potência deve ser mantido. Sendo assim, a função custo pode ser área do circuito, o consumo de potência ou mesmo o produto entre eles.

Assim como a maioria dos problemas encontrados em CAD para VLSI, especialmente em síntese física, dimensionamento de transistores é um **problema de otimização**. Segundo *Gerez* (1998), problemas de otimização exigem o conhecimento de todas as soluções possíveis, onde o espaço é o conjunto de todas as permutações de n objetos, tendo, desta forma, complexidade exponencial $O(n!)$. Neste caso, objeto seria cada configuração dos tamanhos de todos os transistores do circuito.

A busca exaustiva é o algoritmo mais genérico para solução exata de um determinado problema de otimização, consistindo simplesmente em uma varredura de todas as possibilidades, sendo que ao fim é escolhida a melhor entre elas. Porém, sua complexidade de tempo é exponencial e, apesar de poder ser aceitável para circuitos bem pequenos, o tempo de execução gasto por um algoritmo exponencial para tratar circuitos complexos pode ser de anos ou até séculos, dado o atual poder computacional. Desta forma, é possível encontrar uma variedade de trabalhos que propõem outros métodos para resolver o problema do dimensionamento de transistores.

3.2.1 Métodos Heurísticos

Algoritmos heurísticos possuem complexidade relativamente baixa, sendo possível encontrar uma solução rapidamente. As heurísticas são projetadas com base nas propriedades estruturais ou nas características das soluções dos problemas e são baseadas em um conhecimento prévio e específico dos mesmos. Quanto mais características do problema forem agregadas, melhor e mais eficiente será o algoritmo heurístico, fornecendo, em geral, soluções viáveis e de boa qualidade.

Algoritmos gulosos são um tipo de método heurístico aplicável a diversos problemas de otimização. Caracterizam-se por tomarem decisões precipitadas baseadas apenas em uma iteração do algoritmo. Um algoritmo guloso é incapaz de voltar atrás, desfazendo um passo. **Busca local** é um método guloso que faz uso da noção de vizinhança de uma solução possível. Esta vizinhança é um subconjunto de soluções possíveis diferentes em um movimento, onde movimento é uma transição de uma solução possível para uma outra solução possível. Por exemplo, no caso do dimensionamento de transistores, um movimento seria trocar o tamanho de um dos transistores de uma das portas lógicas do circuito. Movimentos mais complexos podem ser obtidos trocando-se o tamanho de mais de um transistor em cada iteração, onde, em um caso extremo, a vizinhança envolveria mudança de tamanho em todos os transistores do circuito e a busca local tornar-se-ia equivalente à busca exaustiva.

Algoritmos gulosos tendem a ficar presos em **mínimos locais**, pois não guardam informações de passos anteriores e não são capazes de olhar para frente (GEREZ, 1998). Em uma dada função *f(x)*, o mínimo global vai ser o ponto com menor valor de *f(x)* para todos os valores de *x* possíveis, enquanto mínimo local vai ser o ponto da função com menor valor de *f(x)* para os valores imediatamente próximos, à esquerda e à direita, de *x.* A Figura 3.3 ilustra as definições de mínimo local e mínimo global.

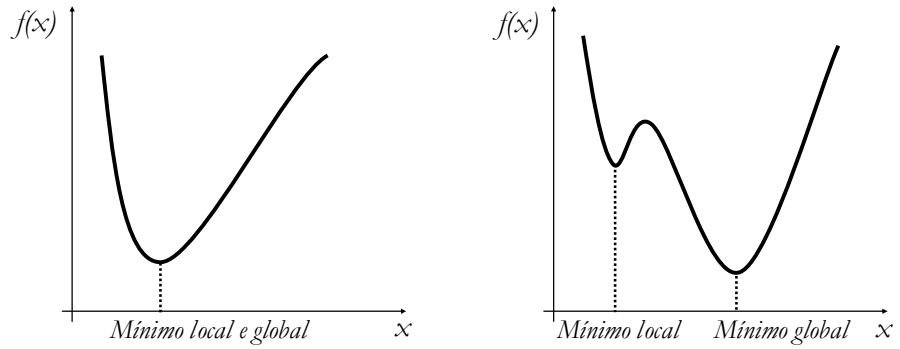

Figura 3.3: Mínimo Global e Mínimo Local

Métodos heurísticos não garantem uma solução ótima, porém, são muito mais eficientes que uma busca exaustiva. E em muitos casos, o algoritmo guloso tem uma convergência mais rápida para boas soluções por não perder tempo com soluções pouco promissoras.

TILOS (FISHBURN e DUNLOP, 1985) é um dos mais conhecidos métodos de dimensionamento de transistores. Foi o primeiro método a fazer uso do fato de que tanto a área quanto o atraso de circuitos combinacionais podem ser modelados como **posinômios** e são, portanto, **funções convexas** com relação ao tamanho dos transistores (ver subseção 3.2.3). *TILOS* utiliza uma abordagem *switch-level* para computar o atraso de cada porta através da métrica *Elmore Delay* (ED). O atraso do circuito é então identificado através de uma análise de *timing* topológica (TTA).

A área efetiva ocupada pelo leiaute de um circuito depende fortemente do método de geração utilizado. Desta forma, a métrica de área mais utilizada como função custo em problemas de dimensionamento é a soma das áreas de *gate* de cada transistor do circuito:

$$Área = \sum_{i=1}^{n} x_i \tag{3-1}$$

onde x_i é a área ($W.L$) do *gate* transistor e n é o número de transistores que compõem o circuito. Sendo assim, sempre que o termo área for usado como métrica da função custo de um método de dimensionamento, estaremos nos referindo a esta soma dada pela equação (3-1).

Começando com todos os transistores em seu tamanho mínimo (mínima área), a otimização de atraso é feita iterativamente através do seguinte algoritmo guloso:

- Busca do caminho crítico do circuito;
- Cálculo da sensibilidade de cada transistor sobre o caminho crítico;
- Aumento do tamanho do transistor com maior sensibilidade por um fator chamado *bumpsize*, que pode ser escolhido pelo usuário;
- Busca do caminho crítico do circuito;
- Novas iterações são realizadas até que não existam mais caminhos críticos (solução é dita encontrada) ou que o atraso do circuito aumente a cada nova iteração (solução não encontrada).

TILOS utiliza uma heurística conhecida como **sensibilidade.** Quando dimensionamos um transistor de uma porta lógica, esta porta fica mais rápida. Porém, a porta que está conectada ao *gate* do transistor dimensionado fica mais lenta. Esta diferença entre a aceleração da porta dimensionada e o aumento de atraso da porta no seu *fanin* é chamada **sensibilidade** do transistor, e mostra qual transistor vai acelerar mais o caminho crítico ao ser dimensionado.

Um dos motivos que levam métodos heurísticos a ficarem presos em mínimos locais é o fato deles assumirem apenas um caminho crítico dominante, sem considerar a interação deste caminho crítico com os demais caminhos do circuito. Como exemplo, considere a árvore de *fanout* da Figura 3.4, onde todos os caminhos da árvore são aproximadamente críticos. Considere também que a sensibilidade das portas G_1, G_2 e G_3 é igual a 1,5 e que a sensibilidade da porta G_0 é igual a 1.

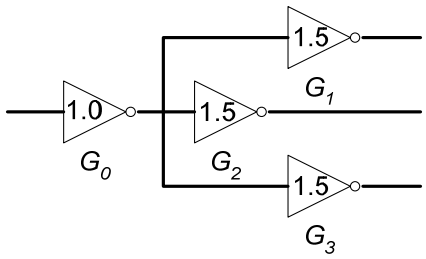

Figura 3.4: Exemplo de uma árvore de *fanout* (BAMJI, 1998).

Se o caminho crítico atual passa através de G_0-G_1, a heurística de sensibilidade tradicional irá escolher a porta G_1 para ser dimensionada de uma unidade. No entanto, os caminhos através de G_0-G_2 e G_0-G_3 também são críticos, e da mesma forma, as

portas G_2 e G_3 deverão ser dimensionadas por uma unidade cada nas iterações subseqüentes. Assim, o acréscimo total de área será de 3 unidades. Por outro lado, se a porta G_0 for escolhida para dimensionamento, então a mesma deverá ser dimensionada em 1,5 unidades para uma melhoria de atraso similar ao que é obtido pelo dimensionamento das portas G_1, G_2 e G_3. No entanto, neste caso a melhoria de atraso ocorre para todos os caminhos com um acréscimo de área total de apenas 1,5 unidades.

Bamji (1998) propôs uma melhora no cálculo da sensibilidade dos transistores para levar em conta o efeito do dimensionamento de um transistor em outros caminhos críticos do circuito. A utilização desta nova heurística em um algoritmo igual ao proposto em *TILOS* traz resultados melhores, resultando em menos acréscimo de área para as mesmas restrições de atraso.

POPS (AZEMARD, 2001) é uma ferramenta que utiliza um método heurístico de dimensionamento, estimando o atraso das portas através de macromodelos e fazendo uso de um método incremental para seleção do caminho crítico. O uso de um modelo de atraso mais preciso gera estimativas mais realistas sobre os atrasos dos caminhos do circuito e guia melhor o algoritmo de dimensionamento. A identificação do novo caminho crítico topológico do circuito é feita de forma incremental, a cada iteração, e acelera a etapa de otimização.

Para evitar cair em mínimos locais e achar a solução ótima para o problema de dimensionamento, surgiram métodos matemáticos capazes de reduzir o atraso do circuito com um acréscimo de área consideravelmente menor que o encontrado em métodos heurísticos. Porém, estes métodos são, em geral, bastante custosos em tempo de processamento e impõem algumas restrições quanto à modelagem do atraso e à maneira como os transistores são dimensionados.

3.2.2 Programação Linear e Não-linear

Assim como dimensionamento de transistores, muitos outros problemas de otimização podem ser reduzidos para programação linear (*Linear Programming* - LP) devido à existência de muitos resolvedores LP. Estes resolvedores são pacotes de *software* que encontram a solução exata e aceitam qualquer problema na forma LP. Como conseqüência da estrutura LP, o resolvedor não precisa de nenhum conhecimento sobre a origem do problema ou de alguma estrutura especial.

O único esforço necessário é conseguir traduzir o problema a ser otimizado para o formato LP sem perder nenhuma informação importante. O formato padrão de entrada LP é:

$$\textit{Minimizar } \boldsymbol{cx}$$

$$\textit{Sujeito a } \quad \boldsymbol{Ax = b}$$

$$\boldsymbol{x \geq 0}$$

onde x é o vetor de variáveis a serem resolvidas, A é a matriz de coeficientes conhecidos e c e b são vetores de coeficientes conhecidos. cx é a função custo e $Ax=b$ e $x \geq 0$ são as especificações. No caso do dimensionamento, a área do circuito seria a função custo (cx) a ser minimizada, Ax seria a função que modela o atraso do circuito e b seria o atraso requerido (BERKELAAR, 1990) (CHUANG, 1993). O vetor x representaria o tamanho dos transistores. Em um caso onde as variáveis só podem assumir valores inteiros, o que é o caso da otimização combinatória, o problema é então chamado de

ILP (*Integer Linear Programming*). Embora pareça uma pequena diferença, esta restrição aumenta ainda mais a complexidade do problema.

Em (CONN, 1998) existe um ambiente que integra um simulador elétrico com um resolvedor de programação não-linear para realizar o dimensionamento dos transistores. O simulador elétrico é usado para realizar uma avaliação precisa do atraso dos caminhos do circuito e para modelar os gradientes usados na otimização. Então, o pacote de programação não-linear *LANCELOT* (CONN, 1992) é usado para encontrar o tamanho dos transistores que satisfazem às restrições de atraso.

3.2.3 Programação Convexa (Geométrica)

Segundo *Sapatnekar* (1993-a), os únicos métodos que garantem achar a solução ótima para o problema do dimensionamento de forma eficiente são aqueles que conseguem explorar a propriedade da **convexidade** das funções que modelam este problema. A seguir serão mostradas algumas definições sobre funções convexas.

Convexidade é uma propriedade muito importante de uma função que garante que qualquer mínimo local será também o seu mínimo global. Uma dada função *f(x)* é dita convexa se, dados quaisquer dois pontos x_a e x_b, a linha ligando estes dois pontos está sobre ou acima da função. A Figura 3.5 ilustra o conceito de função convexa.

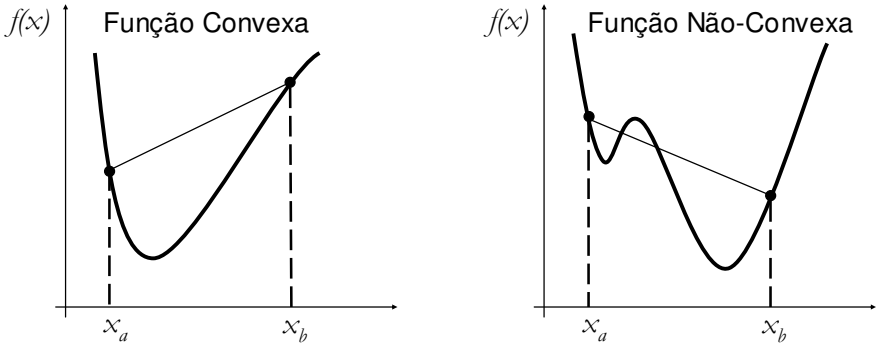

Figura 3.5: Convexidade de uma função

Já um posinômio é um polinômio onde todos os coeficientes são positivos e os expoentes são números reais. Ou seja, um posinômio é uma função *g* de uma variável positiva $\boldsymbol{x} = [x_1, x_2, ..., x_n] \in \mathbf{R}^n$ que tem a forma

$$g(x) = \sum_j \gamma_j \prod_{i=1}^{n} x_i^{\alpha_{ij}} \tag{3-2}$$

onde os expoentes $\alpha_{ij} \in \mathbf{R}$ e os coeficientes $\gamma_j > 0$.

Um posinômio tem uma característica muito útil que é poder ser mapeado para uma função convexa através de uma simples transformação de variável. Fazendo $\boldsymbol{x_i = exp(z_i)}$ para obter $\boldsymbol{G(z)}$, obtemos uma função convexa em $\boldsymbol{z}$. Sendo assim, um problema de otimização que é apresentado na forma:

Minimizar ***Área***

Sujeito a ***atraso crítico*** $\leq$ ***atraso requerido***

torna-se um problema de otimização convexa, caso a função ***Área*** seja uma função convexa e ***atraso crítico*** $\leq$ ***atraso requerido*** seja um conjunto convexo. Um conjunto é

dito convexo quando a linha ligando quaisquer dois pontos *x* e *y* também está inteiramente dentro do conjunto (ver Figura 3.6). Se *f(x)* é uma função convexa, então *f(x)* < *c* é um conjunto convexo. A interseção de dois ou mais conjuntos convexos também é um conjunto convexo.

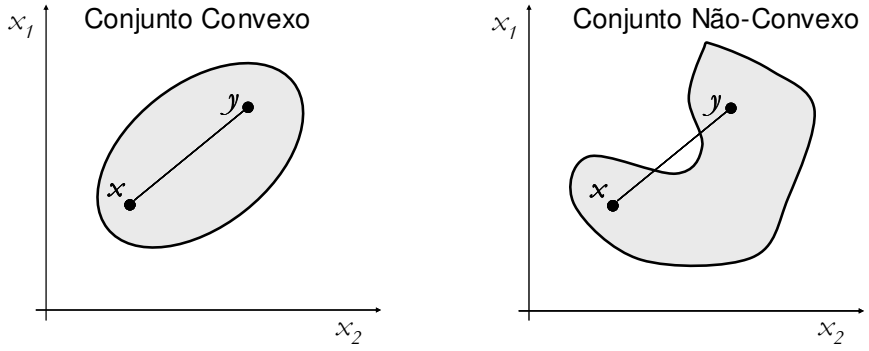

Figura 3.6: Conjunto convexo e conjunto não-convexo

Fishburn e *Dunlop* (1985) foram os primeiros a constatar que, para circuitos combinacionais, a área e o atraso do circuito podem ser modelados como **posinômios** e, portanto, são **funções convexas** com relação ao tamanho dos transistores. Porém, *TILOS* não tirou vantagem do fato do problema de dimensionamento ter sido modelado como um problema de otimização convexa.

O primeiro método que tirou vantagem deste fato e resolveu o problema do dimensionamento de forma exata, ou seja, que atingiu a restrição de atraso com o mínimo de acréscimo de área possível foi proposto por *Sapatnekar et al* (1993-a). Eles usaram a métrica ED para macromodelar cada porta do circuito e uma TTA para então identificar o atraso e o caminho crítico. A área do circuito foi modelada de acordo com a equação (3-1). O método de otimização convexa proposto encontra um polígono convexo (conjunto formado pela interseção entre as restrições de atraso) que contenha a solução ótima, e então segue iterativamente os seguintes passos:

- Encontra a configuração de tamanho dos transistores correspondente ao centro do polígono;
- Verificar se esta configuração de tamanho dos transistores satisfaz a restrição de atraso;
- Reduz o volume do polígono garantindo que a solução ótima está dentro dos seus limites;

O algoritmo pára quando o polígono já é suficientemente pequeno para garantir que a solução escolhida é a solução ótima.

Uma nova função para modelar a área total do circuito é proposta em (MAHESHWARI, 1995), uma vez que a função dada pela equação (3-1) apresenta algumas desvantagens por não fazer nenhuma consideração sobre qual será a área efetiva ocupada pelo leiaute do circuito. Por exemplo, podemos notar na Figura 3.7 que a área sombreada está sendo desperdiçada e que a porta *G* poderia ter seu tamanho aumentado sem nenhum prejuízo à área total do circuito. A nova função proposta também exibe propriedade convexa, e descreve a área total ocupada pelo leiaute em uma geração baseada em bandas (*linear matrix*). Neste estilo de leiaute, a área total é dada pelo produto entre a largura da banda mais comprida e a soma da altura de cada banda. E a altura de cada banda é determinada pelo tamanho da sua célula mais alta.

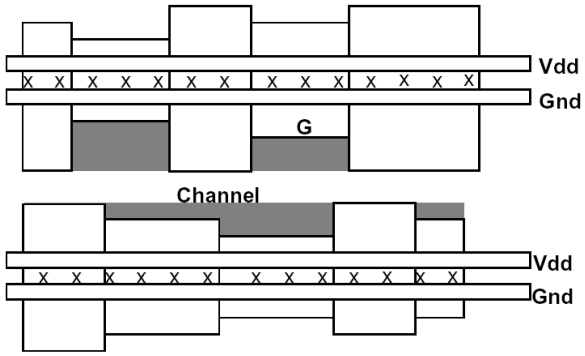

Figura 3.7: Geração de leiaute baseado em bandas (MAHESHWARI, 1995).

A maioria dos métodos que pretende explorar de alguma forma a propriedade convexa das funções que modelam o problema de dimensionamento usa a métrica ED para estimar o atraso das portas. Porém, como foi discutido no capítulo 2, esta métrica é bastante imprecisa, especialmente nas novas tecnologias DSM. Em (KETKAR, 2000-a) é proposto um macromodelo empírico capaz de estimar de forma mais precisa o atraso de portas lógicas, mas que mantém a propriedade convexa pelo uso de posinômios generalizados. Estes modelos são usados em (KETKAR, 2000-b) para modelar o problema de dimensionamento de forma convexa, para então resolvê-lo de forma exata e eficiente através de resolvedores de programação geométrica.

Um compêndio sobre a aplicação de programação geométrica para dimensionamento de transistores e para outros problemas de otimização característicos de ferramentas de EDA pode ser encontrado em (BOYD, 2005).

3.2.4 Outros métodos de dimensionamento

Simulated Annealing é um método de otimização de propósito geral que faz analogia ao processo de cristalização dos metais e também pode ser aplicado a dimensionamento de transistores (DUTTA, 1994). Neste método existe um *loop* externo, no qual a temperatura é gradualmente reduzida a partir de temperatura inicial, e um *loop* interno no qual a solução intermediária é aleatoriamente perturbada por movimentos (troca do tamanho de um transistor). Estes movimentos são aceitos ou rejeitados de acordo com uma probabilidade que é calculada em função da temperatura e da qualidade da nova solução. Assim como no processo de cristalização dos metais, a temperatura começa alta, fazendo com que a probabilidade de aceitar resultados piores também seja alta, dando um comportamento praticamente aleatório ao algoritmo. Em temperaturas baixas, o método aceita somente modificações julgadas como boas, fazendo com que o algoritmo comporte-se praticamente como um algoritmo guloso (HENTSCHKE, 2002).

Alguns métodos propõem duas etapas para realizar a otimização. Inicialmente é realizada uma rápida etapa heurística para achar uma solução inicial e então uma etapa mais custosa de pós-processamento utilizando algum método matemático é aplicada para refinar esta solução inicial.

Em (CHEN, 1998) e (SUNDARARAJAN, 2002) é usado um artifício elegante para acelerar o algoritmo de otimização que garante encontrar a solução ótima para o problema de dimensionamento. São feitas restrições de atraso para cada porta do circuito ao invés de modelar as restrições de atraso através dos caminhos críticos. Com isso, ao invés de um número **exponencialmente** proporcional, só é preciso considerar um número de restrições de atraso **linearmente** proporcional ao número de portas do

circuito. *Chen* (1998) resolve o problema pela técnica de *Langrangian Relaxation*. O método proposto em (SUNDARARAJAN, 2002) atua em duas etapas, onde, partindo de uma solução inicial dada por algum algoritmo heurístico como *TILOS*, são realizados iterativamente os dois seguintes passos:

- É calculado um orçamento ótimo de atraso para cada porta do circuito, baseado no tamanho dos seus transistores;
- É feito o dimensionamento dos transistores da porta de forma a encontrar o atraso dado pelo orçamento anterior.

O algoritmo pára quando os orçamentos calculados no primeiro passo são exatamente atingidos pelos tamanhos de transistores calculados no segundo passo.

Coudert (1996) e *Sapatnekar* (1993-b) fizeram um estudo comparativo entre outros métodos que podem ser aplicados a dimensionamento, como algoritmos genéticos e MFD (*Method of Feasible Directions*). Estes métodos de propósito gerais são relativamente custosos em termos de processamento e muitos não garantem a solução ótima.

3.3 Inserção de *Buffers*

Buffers são portas sem função lógica, mas que são amplamente usadas por poderem ser dimensionadas e posicionadas de forma a diminuir atrasos de propagação de outras portas e conexões do circuito, drenar altas cargas capacitivas ou regenerar sinais lógicos degradados. Tipicamente, a inserção de *buffers* em um circuito visa atender a um ou mais dos seguintes objetivos (COUDERT, 2002):

A) Drenagem de cargas capacitivas – Permite drenar a corrente de carga e descarga de uma alta carga capacitiva em um tempo aceitável. Na Figura 3.8 podemos ver um caso onde um *buffer B* foi dimensionado e posicionado para atacar um *fanout*, com portas de diferentes tamanhos pertencentes a caminhos críticos, em um tempo menor do que a porta *G* conseguiria atacar. Para ser vantajoso, a soma entre o atraso do *buffer B* e o atraso da porta *G* deve ser menor que o atraso da porta *G* sem o *buffer*.

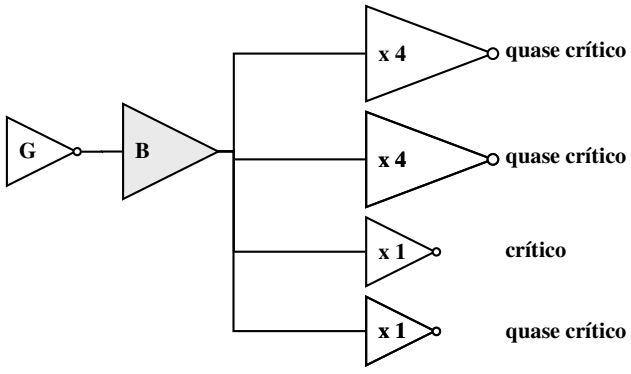

Figura 3.8: *Buffer B* drenando uma alta carga capacitiva.

B) Isolamento de cargas capacitivas – Faz com que uma carga capacitiva, que não pertença a um caminho crítico, não interfira no atraso de uma porta lógica que está sobre um caminho crítico. No exemplo da Figura 3.9 foi colocado um *buffer B* com tamanho mínimo (*x1*) para isolar uma alta carga capacitiva (duas portas com tamanho *x4*) que não pertence a nenhum caminho crítico. Desta forma, a porta *G* que antes

atacava uma carga *x10* agora ataca uma carga *x3*, reduzindo o atraso dos caminhos que são críticos.

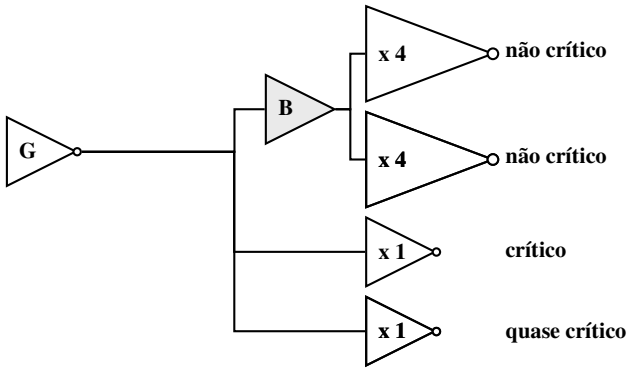

Figura 3.9: *Buffer B* isolando uma alta carga capacitiva.

C) <u>Redução de atraso de conexões</u> – Como o atraso das conexões é proporcional ao quadrado do seu comprimento (ℓ^2), a quebra de uma conexão muito longa em conexões menores, através da inserção de *buffers* nesta conexão, pode ser vantajosa. Basta que a redução de atraso pela quebra da conexão seja maior que o acréscimo de atraso dos *buffers* introduzidos. Além disso, a inserção de *buffers* também ajuda a regenerar um sinal que tenha sido degradado por uma queda de tensão em uma conexão resistiva.

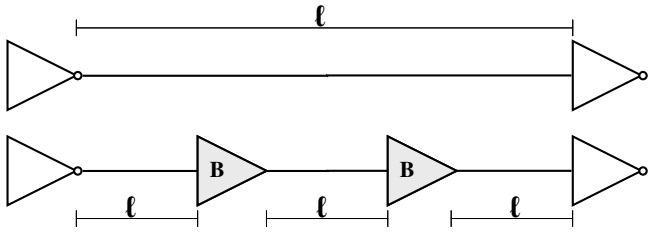

Figura 3.10: Inserção de *buffers* para reduzir o tamanho de uma conexão.

Dada uma rede de conexões, o problema de inserção de *buffers* consiste em achar os locais para inserir os *buffers* nesta rede de forma a minimizar o seu atraso ou atingir uma determinada restrição de *timing*. O método proposto por *Ginneken* (1990) é um clássico, pois foi o primeiro a garantir a solução ótima para um conjunto de possíveis locais para inserção dos *buffers* em uma dada topologia de conexão. Usando a métrica ED para modelar o atraso da árvore de conexão, o método usa programação dinâmica para encontrar as posições ótimas para inserir *buffers*, no conjunto de possíveis posições, de forma a minimizar o atraso nesta árvore.

A maioria dos outros métodos de inserção de *buffers* propostos apresenta formas de reduzir o tempo de processamento, reduzir o número de *buffers* inseridos ou considerar outros aspectos (como modelos de atraso mais precisos, ruído, *crosstalking* e o posicionamento físico das células) na etapa de inserção de *buffers*. Porém, *Jiang* (1998) *Chen* (2000) e *Alpert* (2004) mostraram a vantagem de encontrar um compromisso entre os métodos de inserção de *buffers* e dimensionamento de transistores durante a etapa de otimização de um circuito, possibilitando a obtenção de resultados melhores do que os obtidos pela utilização isolada destes dois métodos.

Coudert (2002) e *Jiang* (1998) citam que, quando *buffers* são inseridos nas árvores de conexão do circuito, não é possível encontrar uma solução ótima para o problema de dimensionamento de transistores. Desta forma, *Jiang* propõe um método heurístico de inserção de *buffers* e dimensionamento que, a cada iteração do algoritmo, escolhe uma

das seguintes três opções para reduzir o tempo gasto por uma porta para carregar ou descarregar uma carga capacitiva:

- Dimensionar os transistores da porta lógica em questão;
- Inserir um *buffer* para isolar cargas capacitivas que não pertencem a caminhos críticos (tipo B);
- Inserir um *buffer* para drenar a carga capacitiva (tipo A).

Chen formula o problema matematicamente como um problema de otimização não-linear e resolvido pelo pacote *LANCELOT* (CONN, 1992). O algoritmo apresentado por *Alpert* é uma extensão do método proposto por *Ginneken* de forma a considerar o dimensionamento dos transistores do *driver* de cada árvore de conexão.

3.4 Ordenamento de transistores

Em uma porta CMOS estática (Figura 3.11), a ordem com que os sinais chegam nos seus pinos de entrada altera a ordem com que as capacitâncias internas da porta (capacitâncias de difusão) são carregadas ou descarregadas. Sendo assim, a disposição ou ordenamento dos transistores em cada estrutura, PMOS ou NMOS, de uma porta lógica altera as características de *timing* e de consumo de potência desta porta.

Ordenamento de transistores é a técnica de arranjar os transistores de uma porta lógica, ou de um circuito, de forma a reduzir seu atraso ou consumo de potência sem alterar seu comportamento lógico.

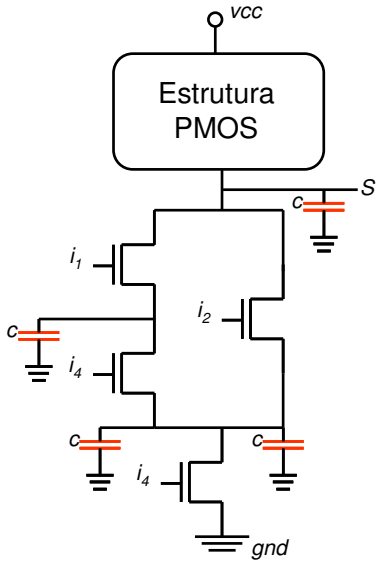

Figura 3.11: Estrutura NMOS de uma porta lógica CMOS estática.

Intuitivamente podemos pensar que, para reduzir o atraso de uma porta, o transistor sobre o caminho crítico deveria sempre ser posicionado o mais próximo da saída. Porém, *Carlson* e *Chen* (1993) mostraram que o melhor ordenamento dos transistores não pode ser tão facilmente previsto e também é dependente do tempo de transição do sinal de entrada. Por exemplo, a Figura 3.12 mostra dois circuitos com lógica equivalente mas com diferentes ordenamentos de transistores. V_{in} é o sinal de entrada com maior tempo de chegada, ou seja, o sinal de entrada que faz a saída da porta transicionar.

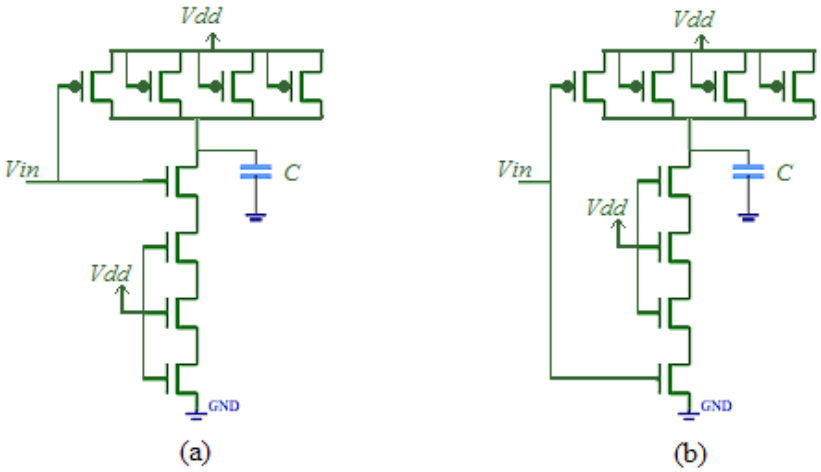

Figura 3.12: Portas lógicas NAND de 4 entradas com diferentes ordenamentos de transistores. (a) Ordenamento *top-critical* e (b) ordenamento *bottom-critical*.

A razão entre o valores dos atrasos de propagação de descida (t_{dhl}) e de propagação de subida (t_{dlh}) dos circuitos mostrados na Figura 3.12 (a) e na Figura 3.12 (b) varia de acordo com o tempo de transição do sinal V_{in}. De acordo com os resultados de simulação elétrica para uma tecnologia 0.35μm (modelo de transistor BSIM3v.3), mostrados na Tabela 3.1, esta razão é igual 0,76 quando V_{in} tem um tempo de subida de 0.1ns. Porém, quando V_{in} tem um tempo de subida de 1ns, esta razão passa a ser igual a 1,17.

Tabela 3.1: Simulação elétrica de uma porta lógica NAND de 4 entradas (W_P/W_N= 2 e C= 20 *fF*) com diferentes ordenamentos de transistores (Figura 3.12 (a) e (b)).

Ordenamento de transistores	**V_{in} = 0.1 *ns***		**V_{in} = 1.0 *ns***	
	t_{dhl} (ns)	***t_{dlh} (ns)***	***t_{dhl} (ns)***	***t_{dlh} (ns)***
Top-critical	217.4	97.0	339.4	193.6
Bottom-critical	287.3	130.7	290.8	263.6

Carlson e *Chen* então propuseram escolher, para cada porta lógica de um circuito, a melhor de duas possibilidades de ordenamento de transistores: ordenamento *top-critical* e ordenamento *bottom-critical*. Após classificar os sinais nas entradas da porta de acordo com os tempos de chegada, o ordenamento *top-critical* (*bottom-critical*) posiciona o transistor conectado na entrada com maior tempo de chegada mais perto da saída (alimentação) da porta. Foi usado um simulador elétrico para determinar o atraso e o tempo de transição da porta para as duas diferentes possibilidades de ordenamentos de transistores. Sendo assim, mesmo sendo rápido e com complexidade linear em relação ao número de portas do circuito, o método mostrava-se lento para circuitos relativamente grandes devido a estas duas simulações que precisam ser feitas para cada porta.

Prasad e *Roy* (1996) disseram que, se as características de *timing* para cada pino de entrada da porta são conhecidas previamente, como acontece com as células de uma biblioteca, basta conectar o sinal com maior tempo de chegada ao pino com menor atraso de propagação. No entanto, apesar da facilidade de conhecer prévia e precisamente os atrasos pino-a-pino de cada porta, a estrutura rígida do *netlist* de transistores das células de uma biblioteca limita o número de permutações possíveis entre os pinos de entrada. Sendo assim, quando aplicada em uma síntese baseada em

s*tandard cells*, esta técnica é chamada de assinalamento de pinos ao invés de ordenamento de transistores.

Hashimoto et al. (1999) mostraram que o atraso de uma porta pode mudar em função dos diferentes ordenamentos de transistores possíveis, mesmo em transições ocorridas em arranjos paralelos de transistores. Na Tabela 3.1 podemos notar que os atrasos de propagação de subida (t_{dlh}) são diferentes nos dois tipos de ordenamento. Além disso, é mostrado que um pino de entrada com menor atraso de subida (descida) em um ordenamento de transistores pode não ser o pino com menor atraso de descida (subida) neste mesmo ordenamento. Novamente na Tabela 3.1 podemos ver que, para V_{in} igual a 1*n*s, enquanto o atraso de descida (t_{dhl}) da porta é maior no ordenamento *top-critical*, o atraso de subida (t_{dlh}) da porta é maior no ordenamento *bottom-critical*. Sendo assim, para encontrarmos o melhor ordenamento de transistores devem ser considerados ambos atrasos de descida e subida para as duas estruturas de transistores (NMOS e PMOS).

3.5 Outros métodos de otimização de atraso

Gate cloning ou replicação de portas é um método que insere cópias de uma porta lógica em paralelo com a porta original para aumentar sua capacidade de drenar corrente. Teoricamente, inserir *n* cópias de uma porta traria o mesmo ganho que multiplicar *n* vezes o tamanho de todos os transistores da porta original. Porém, *Jacob et al.* (2001) fizeram as seguintes considerações sobre a técnica de *gate cloning*:

- Apesar de trazer um pouco mais de liberdade ao posicionamento, irão existir conexões adicionais entre a porta original e a sua cópia;
- A variabilidade do processo de fabricação pode introduzir um comportamento ligeiramente diferente entre a porta original e a sua cópia;
- A influência das capacitâncias de acoplamento e do ruído de chaveamento das conexões em torno da cópia e da porta original pode ser diferente.

Jiang et al. (1998) fazem comentários sobre a utilização de *gate cloning* para substituir a inserção de *buffers* (*buffers* tipo A na seção 3.3), no caso de ser desejável drenar uma capacitância alta. Esta substituição teria como vantagem não aumentar a profundidade lógica do caminho crítico, o que aumenta o atraso do mesmo. Além disso, tomando como exemplo a duplicação de um inversor, esta substituição traz uma economia de dois transistores, uma vez que duplicar o inversor tem um custo de dois transistores e inserir um buffer tem um custo de quatro transistores. Porém *Jiang et al.* falam que esta substituição no método proposto não trouxe nenhum ganho prático nos circuitos otimizados. Ele atribui isso ao fato de que estas situações de vantagem não ocorreram em número suficiente nos circuitos utilizados.

Assim como inserção de *buffers*, técnicas como dimensionamento das conexões e otimização da topologia de interconexão ajudam a reduzir o atraso resultante das linhas de metal. *Cong* (1996) e *Chu* (2001) mostram um resumo sobre estas duas técnicas de otimização do atraso de interconexões. Quando a resistência da conexão puder ser ignorada, otimizar a topologia da interconexão recai sobre minimizar o comprimento total dos fios. Quando esta resistência torna-se considerável, então é preciso controlar o comprimento das linhas de metal que vão do *driver* até os nodos com atraso crítico. O dimensionamento das linhas de metal consiste em alterar a largura de alguns trechos da conexão de forma a reduzir a sua resistência. Estes dois métodos apresentam um ganho

relativamente interessante durante as etapas de roteamento e geração das linhas de conexão em um fluxo de síntese automático.

3.6 Conclusões

Métodos de otimização de atraso no nível de transistores são técnicas usadas, de forma efetiva, para atingir as restrições de atraso em circuitos seqüenciais e combinacionais. Estes métodos são geralmente usados quando a etapa de síntese física do fluxo de projeto chega no nível de leiaute. Tendo em vista que o trabalho desenvolvido nesta dissertação será inserido no fluxo de síntese física FUCAS, procurou-se tirar algumas conclusões sobre quais técnicas de otimização de atraso podem oferecer maiores vantagens em um tipo de síntese independente de biblioteca de células.

Inserção de *buffers* e replicação de portas são métodos bastante eficientes de otimização, e podem ser usados separadamente ou combinados com dimensionamento. Porém, a utilização destes métodos é praticamente independente do estilo de síntese, e, com isso, a implementação dos mesmos não traz benefícios específicos ao tipo de síntese adotado.

Em um método de síntese baseado em biblioteca de células, realizar o ordenamento de transistores recai sobre um problema de assinalamento de pinos, uma vez que o leiaute de cada célula é previamente desenhado e não pode ser mudado. Sendo assim, um estilo de síntese que oferece a possibilidade de geração do leiaute a partir de um *netlist* de transistores, como é o caso do fluxo FUCAS, oferece mais liberdade de otimização para estas técnicas de ordenamento de transistores. Mas, visto que o ganho resultante deste tipo de otimização é pequeno, mesmo sendo obtido a um custo de área quase nulo, sua implementação é pouco atrativa.

A eficiência dos métodos de dimensionamento de transistores depende das possibilidades de tamanho que cada transistor ou porta lógica pode assumir. Desta forma, a liberdade de otimização no nível de transistores oferecida pela síntese FUCAS acarreta em uma grande liberdade para este método de otimização de atraso, tornando bastante atrativa a sua implementação.

Fazer uma classificação sistemática dos métodos de dimensionamento é uma tarefa difícil, pois eles podem diferir sobre vários aspectos, tais como:

- Algoritmo de otimização;
- Método usado para identificar o atraso e o caminho críticos;
- Modelos de atraso das portas e conexões do circuito;
- Dimensionamento contínuo ou discreto;
- Dimensionamento de transistores ou dimensionamento de portas lógicas.

A decisão sobre a escolha de cada um destes aspectos procura manter um compromisso entre acréscimo de área e tempo de processamento, e alguns comentários sobre estes aspectos são necessários.

Os métodos matemáticos de dimensionamento conseguem atingir as restrições de atraso com um acréscimo de área quase ótimo, mas consomem um tempo de processamento muito maior que os métodos heurísticos. Devido à necessidade de achar

uma função que seja capaz de modelar o atraso do caminho crítico, ou para acelerar a etapa de identificação dos caminhos críticos e compensar o grande tempo de processamento requerido, a grande maioria dos algoritmos matemáticos utiliza abordagens topológicas de análise de *timing* (TTA). No entanto, TTA é fortemente afetada pelo fenômeno dos caminhos falsos, podendo incluir caminhos não-sensibilizáveis na lista de caminhos críticos e gerar sobrestimativas do atraso do circuito. E o dimensionamento de portas pertencentes a caminhos falsos aumenta desnecessariamente o acréscimo de área e causa desperdício no tempo de processamento, uma vez que a aceleração destas portas não é convertida em redução do atraso do circuito.

O uso de modelos de atraso muito simplistas é necessário para possibilitar o "enquadramento" do problema de dimensionamento em uma técnica de otimização matemática, ou simplesmente para garantir a convexidade da função que modela o atraso do circuito. A utilização destes modelos menos precisos pode gerar estimativas de atraso inadequadas para os caminhos críticos do circuito, confundindo e baixando consideravelmente o desempenho do algoritmo de otimização. Todavia, modelos de atraso mais precisos tornam o problema de dimensionamento não-convexo ou não se encaixam no tipo de modelagem exigido por alguns métodos matemáticos (COUDERT, 1996).

Alguns métodos só garantem achar boas soluções se o tamanho de cada transistor puder assumir valores contínuos. Estas soluções são aceitáveis quando o leiaute do circuito for desenhado a mão (*full-custom*) a partir do *netlist* otimizado. Contudo, em um fluxo de síntese automático, o dimensionamento é um problema essencialmente discreto, mesmo quando as equações lógicas são mapeadas em transistores e o leiaute é gerado seguindo o estilo *linear matrix*. Métodos de arredondamento, utilizados para aproximar soluções contínuas para valores discretos, também podem falhar em encontrar uma solução razoável (CHAN, 1990).

4 GERAÇÃO AUTOMÁTICA DE LEIAUTE ORIENTADA PELO ATRASO

A liberdade de otimização no nível de leiaute, utilizando dimensionamento de transistores, depende do estilo de síntese física que estamos utilizando (WESTE, 1993). O método baseado em *standard cells* é atualmente o mais utilizado para o projeto e síntese de ASICs. Este método faz uso de uma biblioteca de células para implementar a funcionalidade do circuito, onde as portas lógicas já estão previamente desenhadas e caracterizadas. O fato de cada porta ser previamente caracterizada oferece certa predição do comportamento do circuito em diferentes etapas da síntese física. Cada porta da biblioteca também pode apresentar diferentes versões com relação ao seu tamanho, atraso e consumo. Sendo assim, a otimização do atraso e a área final do circuito tornam-se dependentes do tamanho da biblioteca de células, ou seja, dependem de quantas versões de cada porta lógica foram projetadas e pré-caracterizadas.

Em um método de síntese *gate-array* existe uma matriz de transistores pré-difundidos no silício, apenas esperando a etapa de metalização que irá definir a funcionalidade e personalizar a aplicação do circuito[2]. Desta forma, não é possível modificar o tamanho dos transistores a fim de aumentar o desempenho de uma porta lógica, mas apenas ligar transistores em paralelo, o que limita as possibilidades de otimização.

Já projetos que utilizam a abordagem *full-custom* oferecem liberdade total ao projetista, pois os transistores são desenhados individualmente e cada porta lógica pode ser projetada para atingir as requisições necessárias de atraso e consumo. A grande desvantagem da abordagem *full-custom* é o tempo de projeto gasto e a escassez de ferramentas para automatizar e acelerar a geração do leiaute. Segundo *Hill* (1989), dimensionamento de transistores é mais eficiente quando usado com ferramentas de geração *custom* de leiautes a partir de um *netlist* de transistores

O fluxo de síntese física FUCAS tem como característica realizar a geração automática de circuitos CMOS estáticos independentemente de biblioteca de células. Ao invés de usar células pré-projetadas e pré-caracterizadas, o fluxo FUCAS explora as vantagens do uso ilimitado de portas complexas. Além disso, a presença de um gerador automático de leiaute permite que cada porta lógica possa assumir diferentes versões através de diversas combinações de tamanho dos seus transistores.

---

[2] Atualmente existem os ASICs estruturados (*Structured* ASICs), que também têm a sua funcionalidade personalizada através das camadas de metal (ZAHIRI, 2003).

A Figura 4.1 mostra quais são as etapas realizadas durante a geração automática de leiaute no fluxo de síntese física FUCAS, em tecnologia CMOS estática, pela ferramenta *Punch* (LAZZARI, 2004).

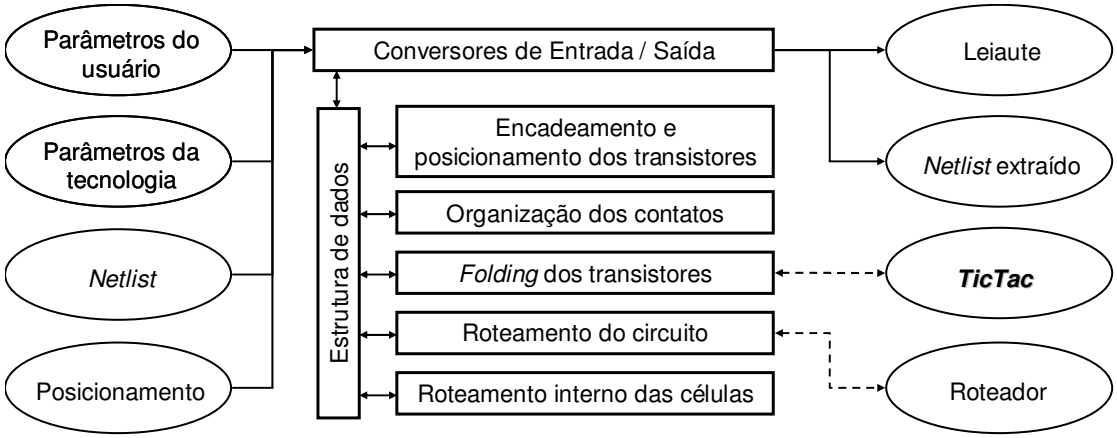

Figura 4.1: Etapas da geração automática de leiaute da ferramenta *Punch*.

O leiaute do circuito é gerado seguindo o estilo *linear matrix*, onde os transistores de um mesmo tipo (NMOS ou PMOS) são dispostos em bandas. Como podemos ver na Figura 4.2, estas bandas alternam-se verticalmente duas a duas, de modo que transistores de um mesmo tipo possam compartilhar a mesma linha de alimentação e a mesma região de poço (*well*).

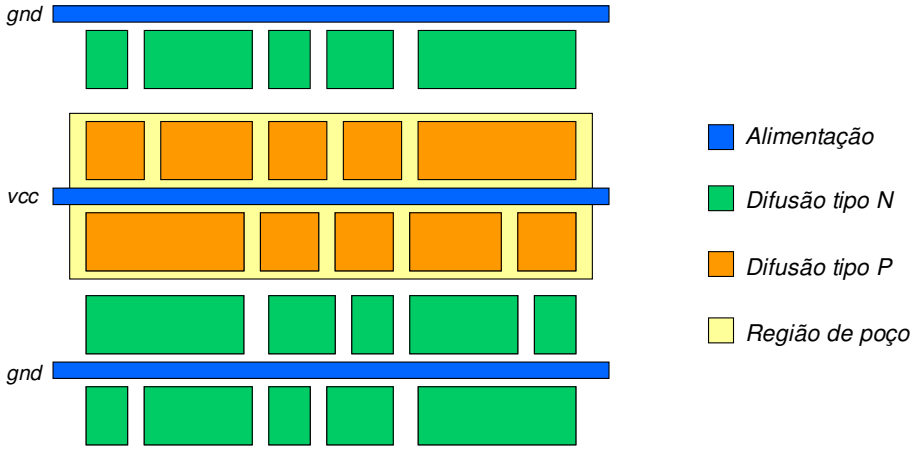

Figura 4.2: Estilo de leiaute *linear matrix*.

Uma vez que o posicionamento de cada porta lógica é conhecido, o posicionamento de cada transistor desta porta lógica, na respectiva banda, é feito de forma a minimizar o congestionamento de conexões sobre os transistores. Isto consiste em encontrar o caminho de *Euler* para cada porta, possibilitando a conexão entre os transistores tipo P e tipo N que compartilham o mesmo sinal de *gate* através de uma simples camada de polissilício (LAZZARI, 2003).

Uma característica importante deste gerador é que o leiaute só é gerado após o roteamento do circuito. A ferramenta faz uma estimativa do tamanho de cada célula e indica possíveis locais para inserção dos contatos dos pinos de entrada e saída da célula. Com isso, o roteador tem mais flexibilidade na hora de implementar as linhas de conexão. Uma vez que o roteamento é feito completamente sobre as células (*Full Over*

the Cell[3] *– FOTC*), o roteador indica onde espaços devem ser inseridos no leiaute, se estes forem necessários, para garantir a roteabilidade completa do circuito. E quando possível, estes espaços inseridos pelo roteador são usados para inserção de contatos que servem para polarizar o substrato (*body-ties*).

Por fim, é feito o roteamento interno das células, ou seja, as conexões de fonte e dreno dos transistores e a ligação de polissilício entre os *gates* dos transistores que compartilham o mesmo sinal de entrada. A Figura 4.3 mostra um exemplo de leiaute gerado pela ferramenta *Punch*. A Figura 4.4 mostra o mesmo exemplo de leiaute, porém com as camadas de roteamento entre diferentes portas lógicas.

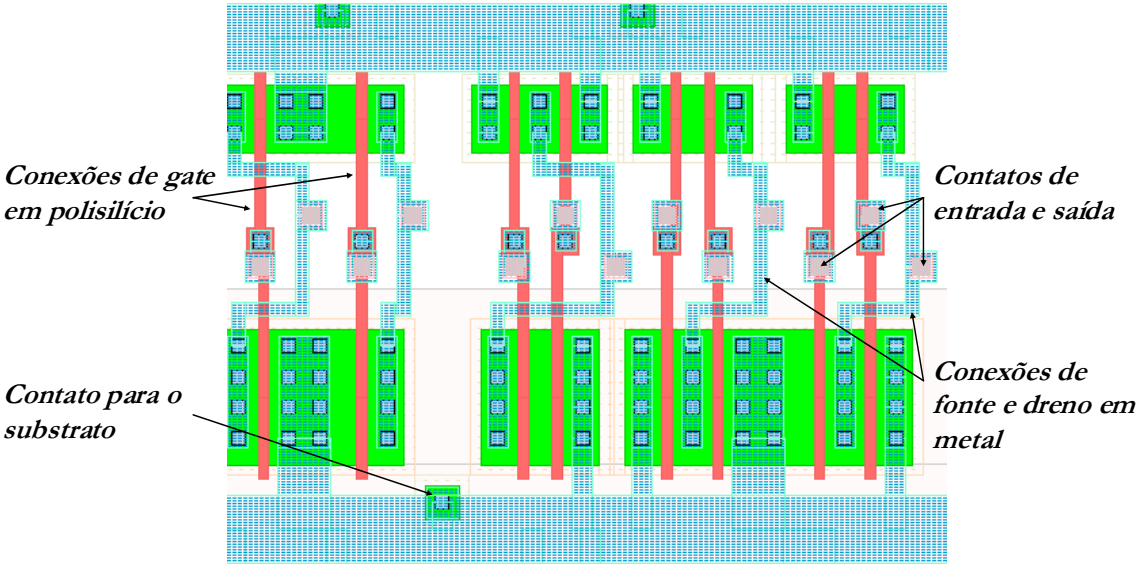

Figura 4.3: Exemplo de leiaute gerado pela ferramenta *Punch*.

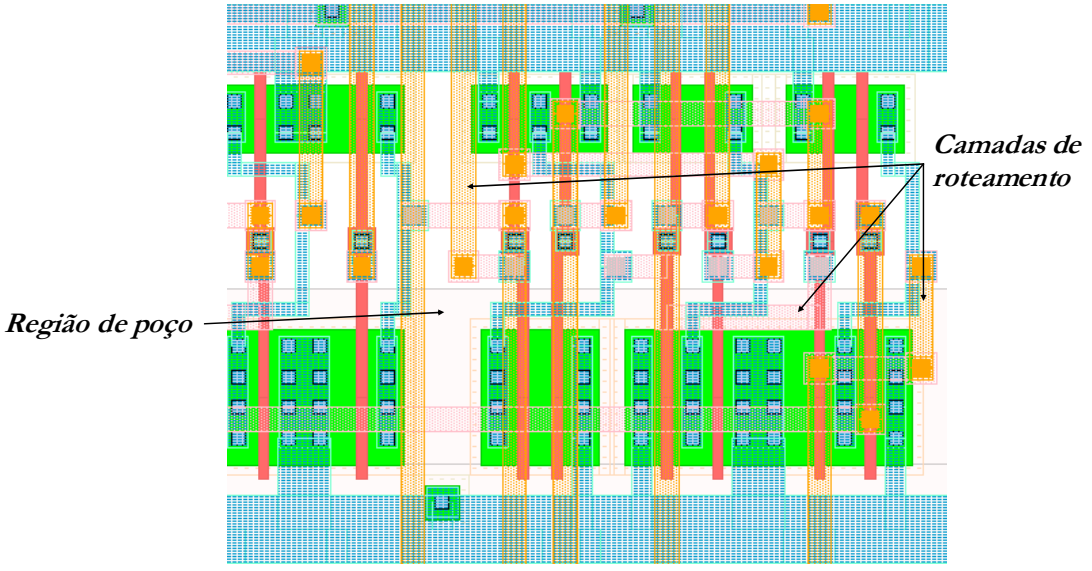

Figura 4.4: Roteamento sobre as células (FOTC).

[3] Existe também a metodologia OTC (*Over the Cell*), onde apenas parte do roteamento é feito sobre as células.

A integração de uma ferramenta de verificação e otimização de atraso (*TicTac*) dentro do fluxo FUCAS permitirá que os circuitos sejam gerados visando atingir as especificações de atraso desejadas pelo projetista. A Figura 4.5 mostra como será o fluxo de síntese física FUCAS orientado pelas restrições de atraso.

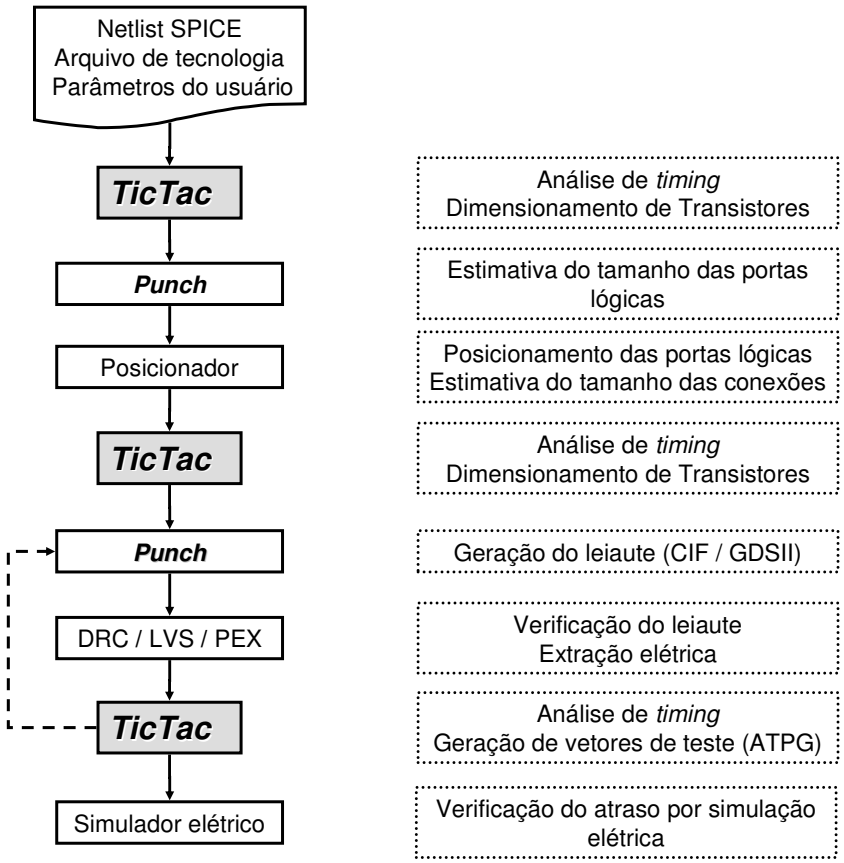

Figura 4.5: Fluxo de síntese física FUCAS orientado a *timing*.

Para realizar a síntese física do circuito através do fluxo descrito na Figura 4.5, o projetista precisa fornecer 3 arquivos de entrada. Um arquivo com a descrição do circuito em formato SPICE contendo a dimensão inicial dos transistores, um arquivo com as regras e parâmetros da tecnologia de fabricação e um arquivo com parâmetros e preferências do usuário e com as restrições de atraso. A partir destes arquivos, a ferramenta de análise e otimização de atraso ***TicTac*** irá gerar um relatório de atrasos. Se as restrições de atraso estiverem sendo violadas, a ferramenta realizará o dimensionamento dos transistores do circuito para tentar atingir estas restrições.

Em seguida é feita uma estimativa da área que será ocupada pelo leiaute de cada porta lógica. Estas estimativas são usadas pela ferramenta de posicionamento de células, que poderá fornecer, após o término do posicionamento, uma estimativa do tamanho de cada conexão do circuito. Tais estimativas podem ser traduzidas em valores de capacitâncias e resistências para que a ferramenta *TicTac* refine as avaliações de atraso e, caso necessário, redimensione o tamanho dos transistores.

A próxima etapa é a geração do leiaute e o roteamento do circuito, realizada pela ferramenta *Punch*. Apesar de o leiaute ter sido gerado baseado nas regras da tecnologia alvo, é desejável que etapas de verificação sejam realizadas após a geração. A partir do arquivo de leiaute (CIF ou GDSII) gerado pela ferramenta *Punch*, podem então ser

feitas etapas de verificação das regras de projeto (DRC), verificação da equivalência entre o leiaute e a descrição SPICE inicial (LVS) e extração elétrica (PEX).

A ferramenta *TicTac* pode ser novamente realimentada com o *netlist* SPICE extraído, que contém agora os elementos RC parasitas resultantes da implementação física, para refazer as estimativas de atraso. Se mais uma vez as restrições de atraso forem violadas, será necessário refinar o tamanho dos transistores através de uma IPO (*In-Place Optimization*). A seta tracejada da Figura 4.5 indica que apenas a geração do leiaute precisa ser refeita para acomodar os novos tamanhos de transistores. Por fim, a ferramenta *TicTac* é capaz de gerar vetores para *delay test* dos caminhos críticos do circuito, em formato SPICE, para serem utilizados em um simulador elétrico.

4.1 Geração dos transistores dimensionados

Para que as especificações de atraso sejam atingidas com o menor acréscimo de área possível, cada porta lógica deve oferecer diferentes possibilidades de tamanho de seus transistores (BASTIAN, 2004-b). Porém, em uma geração de leiaute baseada em bandas, a altura de cada banda é determinada pela sua célula mais alta, como podemos notar pela Figura 4.2. E, se olharmos a Figura 4.3, podemos perceber que a altura da banda de difusão de cada célula é determinada pelo tamanho (W) do seu maior transistor. Com isso, quando a geração do leiaute é feita no estilo *linear matrix*, transistores de diferentes tamanhos podem levar a um desperdício de área.

Para evitar este desperdício, o gerador de leiaute emprega uma técnica de dobra (*folding*) para permitir que transistores de tamanhos diferentes sejam gerados na mesma banda, sem alterar a sua altura. Transistores grandes são "quebrados" em transistores menores (com tamanho igual à altura da banda de difusão) que são conectados em paralelo. *Bastian* (2004-a)(2004-b) desenvolveu um método de *folding* que, considerando o posicionamento inicial dos transistores dentro da célula, encontra o arranjo ótimo dos transistores quebrados para o tamanho inicial do transistor e da banda. O método mantém o posicionamento inicial dos transistores e coloca os transistores quebrados lado a lado. Com isso, o compartilhamento da difusão é otimizado e não existe a necessidade de conexões de metal entre os segmentos do mesmo transistor.

Porém, a aplicação deste método de *folding* pode alterar o caminho de *Euler* original, e, com isso, alterar o número de aberturas na difusão (*gaps*). Observando o grafo que representa a estrutura PMOS de uma porta lógica, na Figura 4.6(b), vemos que o caminho de *Euler* da porta lógica da Figura 4.6(a) é *vcc-**a**-n_1-**d**-s-[gap]-vcc-**b**-n_1-**c**-vcc*. Agora, considerando que cada transistor da porta em questão tenha seu tamanho dobrado, a técnica de *folding* fará com que obtenhamos a situação ilustrada na Figura 4.6(c). O caminho de *Euler* da porta dimensionada é *vcc-**a**-n_1-**a'**-vcc-**b**-n_1-**b'**-vcc-[gap]-n_1-**d**-s-**d'**-n_1-**c**-vcc-**c'**-n_1-[gap]*, e podemos notar que um novo *gap* foi inserido.

Desta forma, para minimizar o número de novas aberturas na difusão, o método oferece a possibilidade de, durante a busca do caminho de *Euler*, inserir um transistor onde deveria ser inserido um *gap*. Estes transistores inseridos podem ser conectados na mesma entrada do transistor original ou serem polarizados de forma a operar na região

de corte[4]. Com a inserção destes transistores, o caminho de *Euler* resultante seria *vcc-**a**-n_1-**a'**-vcc-**a''**-n_1-**b**-vcc-**b'**-n_1-**d**-s-**d'**-n_1-**c**-vcc-**c'**-n_1-**c''**-vcc* e estaria livre de *gaps*.

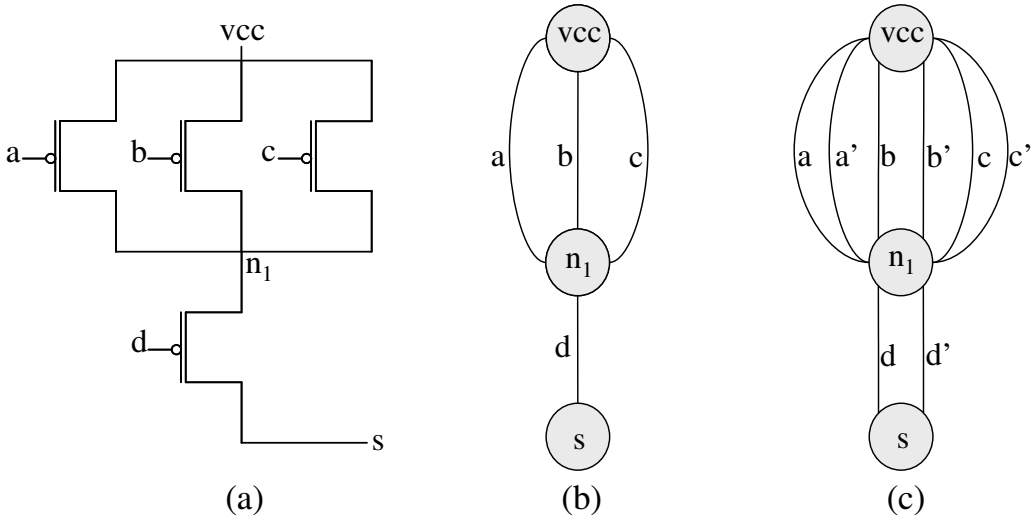

Figura 4.6: Representações da estrutura PMOS de uma porta lógica.

A Figura 4.7 mostra um exemplo de leiaute onde foi aplicada a técnica de *folding*. Podemos notar que a existência de transistores com tamanhos diferentes na mesma banda de difusão não fez com que esta tivesse sua altura alterada.

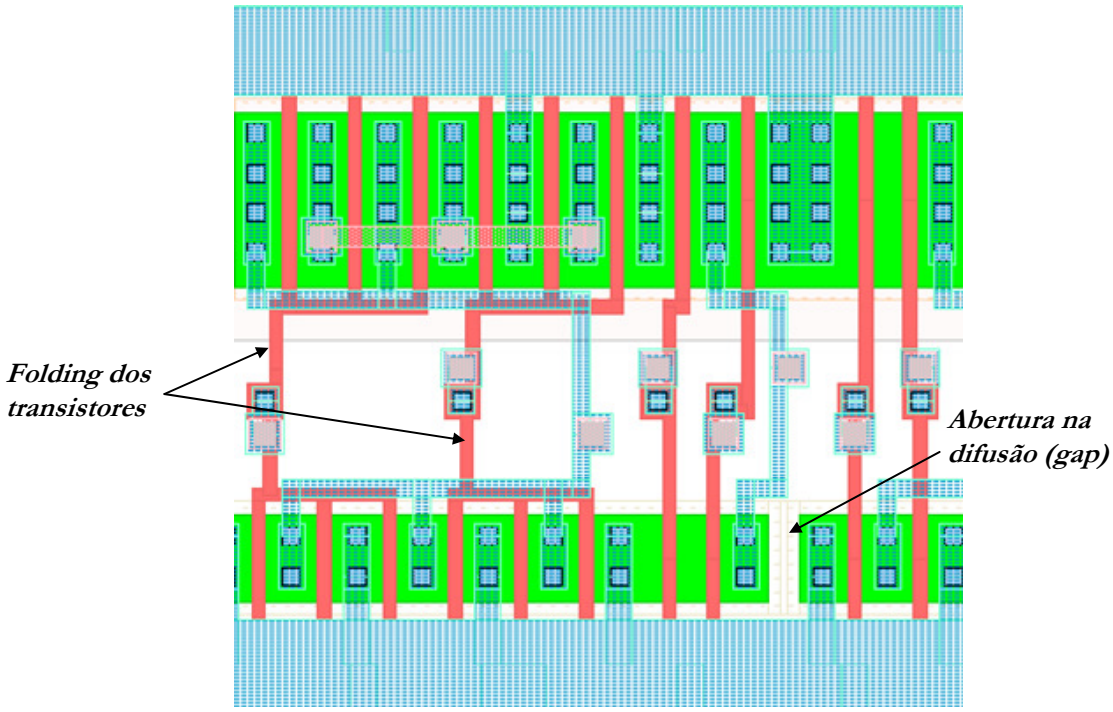

Figura 4.7: *Folding* de transistores.

[4] Esta técnica de colocar transistores para isolar regiões adjacentes de difusão, conhecida como *gate isolation* (SAKASHITA, 1985), reduz o comprimento da banda e aumenta a regularidade do leiaute.

4.2 Identificação do atraso e do caminho críticos

Por ser computacionalmente mais barata que uma simulação elétrica, ser independente de vetores de entrada e conseguir evitar a síndrome dos caminhos falsos, a etapa de identificação de atraso e caminhos críticos do circuito é feita pela ferramenta de análise de *timing* funcional *TicTac* (FERRÃO, 2003). Embora métodos de FTA que utilizam uma abordagem de enumeração de atrasos tendam a ser computacionalmente mais eficientes na identificação do atraso crítico, os métodos de otimização de atraso necessitam saber qual é o caminho responsável pelo atraso do circuito, sendo necessário utilizar uma abordagem de enumeração de caminhos. A Figura 4.8 resume o método de FTA utilizado.

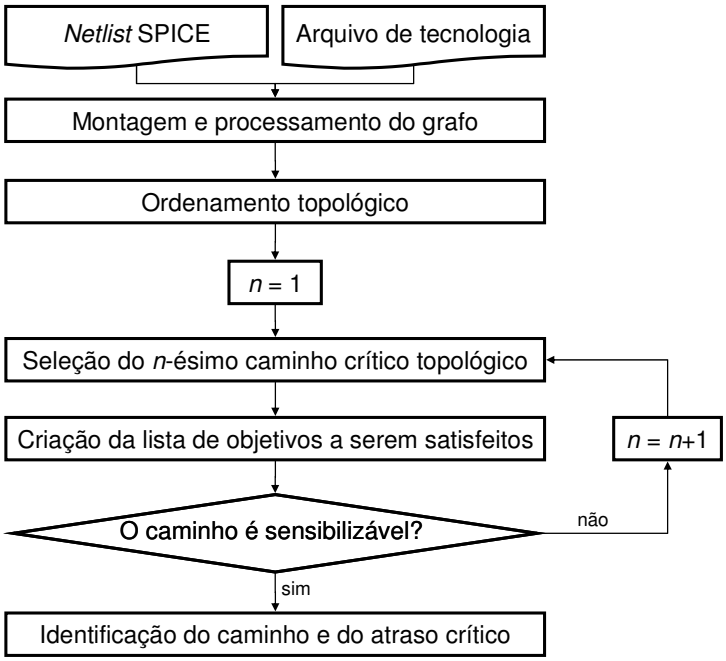

Figura 4.8: Método de FTA para identificação do caminho e do atraso críticos.

A partir de um *netlist* SPICE é montado e processado o DAG que representa o circuito. São então calculados os atrasos das portas e conexões do circuito por seus respectivos modelos de atraso, que fazem uso da topologia do circuito e das características elétricas da tecnologia. O algoritmo *topological sort* (CORMEN, 1990) é então usado para listar os caminhos do circuito em ordem não-crescente de atraso. Esta lista serve para definir a ordem em que os caminhos terão sua sensibilização testada, começando pelo caminho crítico topológico (n=1).

A sensibilização do caminho selecionado vai sendo testada ao mesmo tempo em que o caminho vai sendo estendido. Cada porta que vai sendo adicionada tem sua sensibilização testada. Caso a porta adicionada durante a extensão torne o trecho de caminho não-sensibilizável, o processo de sensibilização do caminho selecionado pára, e qualquer outro caminho que compartilhe o mesmo trecho não-sensibilizável não será adicionado na lista de caminhos a serem testados. Este procedimento vai podando o espaço de busca e, com isso, reduzindo o tempo de processamento.

As condições de sensibilização ao longo do caminho, ou seja, os valores necessários nas entradas laterais do caminho, definidas pelo critério de sensibilização adotado, são transformadas em uma lista de objetivos. Na ferramenta *TicTac* é possível utilizar três dos quatro critérios apresentados na seção 2.3: sensibilização estática, co-sensibilização

estática e exato sob o modo flutuante. Para tentar satisfazer esta lista de objetivos, é usado um algoritmo de ATPG baseado no algoritmo PODEM (GOEL, 1981), que foi modificado para levar em conta informações de atraso durante o teste de sensibilização.

Quando um caminho completo (caminho entre uma entrada e uma saída primária do circuito) consegue ser sensibilizado, este é declarado como caminho crítico e seu atraso será o atraso crítico do circuito.

4.2.1 Tratamento de portas complexas

Geradores de blocos e circuitos combinacionais, como a ferramenta *Punch* que é utilizada no fluxo FUCAS, são exemplos que ilustram a necessidade de ferramentas de FTA capazes de tratar circuitos que contenham portas complexas[5]. Devido a esta necessidade, *Güntzel* (2000) desenvolveu um método de FTA capaz de tratar qualquer circuito contendo SCCGs. Para isso, cada porta, simples ou complexa, deve ter sua função lógica identificada. Porém, como o fluxo de síntese física FUCAS não é baseado em uma biblioteca de células com suas funções lógicas previamente identificadas, cada porta lógica precisa ter sua função determinada sob demanda.

Em uma tecnologia CMOS, portas lógicas estáticas possuem duas redes de transistores distintas. Uma rede de transistores PMOS que conecta a saída da porta à fonte de alimentação e uma outra rede de transistores NMOS que conecta a saída ao terra. Estas duas estruturas possuem o mesmo número de transistores, e estes estão arranjados apenas em associações série/paralelo. Como a estrutura NMOS é dual a estrutura PMOS com relação à associação dos seus transistores, a análise destes arranjos série/paralelo em apenas uma das estruturas, durante a fase de processamento do grafo, é suficiente para identificar a função lógica da porta.

4.2.2 Modelos de atraso para portas lógicas

É utilizado um macromodelo proposto primeiramente por *Daga et al.* (1999) para, através de uma abordagem *gate-level*, computar o atraso das portas lógicas a partir do *netlist* de transistores do circuito. O uso deste macromodelo faz com que qualquer circuito que contenha qualquer tipo de porta CMOS estática possa ser analisado, sendo possível tratar qualquer circuito gerado pelo fluxo de síntese FUCAS.

Este modelo de atraso usa um inversor como primitiva, mapeando todas as demais portas do circuito para inversores "equivalentes". Para modelar o atraso do inversor, foram identificadas três diferentes regiões de operação durante a carga e descarga da sua capacitância de saída: região de *overshoot*, região de curto-circuito e região de drenagem (MAURINE, 2002). A partir da análise destas regiões de operação, o modelo de atraso do inversor foi desenvolvido em duas partes. A primeira descreve o atraso para entradas tipo degrau (*step*), e a segunda parte estende o modelo para entradas reais (tipo rampa).

---

[5] No contexto desta dissertação, portas simples são portas lógicas que implementam as funções mais básicas da álgebra Booleana. Particularmente, o conjunto de portas simples CMOS se restringe às portas lógicas NAND, NOR e inversor. Portas complexas são portas lógicas capazes de implementar funções com maior complexidade, que só seriam possíveis com uma associação de portas simples. Portas complexas CMOS estáticas, ou SCCGs, correspondem ao subconjunto de portas complexas implementas em tecnologia CMOS.

Os atrasos de descida ($t_{hl(s)}$) e subida ($t_{lh(s)}$) do inversor para uma entrada do tipo degrau são dados por:

$$t_{hl(s)} = \tau_{st} \times \frac{C_L}{2C_n} = \tau_{st} \times \frac{1+k}{2} \times \frac{C_L}{C_{IN}} \tag{4-1}$$

$$t_{lh(s)} = \tau_{st} \times R_\mu \times \frac{C_L}{2C_p} = \tau_{st} \times R_\mu \times \frac{1+k}{2k} \times \frac{C_L}{C_{IN}} \tag{4-2}$$

C_L é capacitância de saída do inversor, C_n e C_p são as capacitâncias de *gate* dos transistores NMOS e PMOS e C_{IN} é capacitância de entrada do inversor (C_n+C_p). A configuração interna do inversor (W_p/W_n) é dada por k. R_μ representa a diferença de velocidade dos portadores dos transistores PMOS e NMOS. O parâmetro τ_{st} funciona como indicador da velocidade máxima da tecnologia. Este parâmetro é dependente da velocidade máxima dos portadores e do valor da tensão de alimentação (AUVERGNE, 2000). Apesar de terem sido desenvolvidos analiticamente, os parâmetros R_μ e τ_{st} são obtidos por simulação elétrica durante a calibração do modelo para a tecnologia escolhida.

Para estender o modelo para portas mais complexas, é usado um fator de correção que indica a redução da corrente máxima da porta com relação à corrente máxima do inversor. Esta redução acontece pela presença de associações de transistores em série nas estruturas PMOS e NMOS da porta. Durante o processamento do grafo, cada porta lógica é identificada pela maior cadeia de transistores em série contida nas suas estruturas. Sendo assim, o fator de correção indica quão maior é a capacidade máxima de corrente do inversor equivalente com relação à cadeia de transistores em série, onde todos os transistores possuem a mesma largura de canal. Como o modelo computacional de atraso da porta é *par de atraso* por porta, os atrasos de subida e descida devem representar os piores casos, ou seja, pressupor que o caminho de transistores entre a alimentação e a saída da porta é o caminho com a maior cadeia de transistores em série.

Para considerar sinais de entrada reais, é preciso considerar a influência da capacitância de acoplamento entre a entrada e a saída do inversor. Para entradas com transição rápida, os atrasos de descida ($t_{hl(s)}$) e subida ($t_{lh(s)}$) da porta lógica são dados por:

$$t_{hl} = v_{tn} \times \frac{\tau_{in}}{2} + \left(1 + 2\frac{C_M}{C_M + C_L}\right) \times t_{hl(s)} \times K_n \tag{4-3}$$

$$t_{lh} = v_{tp} \times \frac{\tau_{in}}{2} + \left(1 + 2\frac{C_M}{C_M + C_L}\right) \times t_{lh(s)} \times K_p \tag{4-4}$$

τ_{in} é duração do sinal de entrada e $v_{tn(p)}$ é igual a razão entre a tensão de *threshold* do transistor N(P) e *vdd* ($V_{THn(p)}$/*vdd*). $K_{n(p)}$ é o fator usado para estender o modelo para portas mais complexas. C_M é a capacitância de acoplamento entre a entrada e a saída do inversor, e é dada por *Bisdounis* (1998):

$$C_M = \frac{1}{2} \times W \times L \times C_{ox} \tag{4-5}$$

onde C_{ox} é a capacitância de óxido por unidade de área característica da tecnologia em questão.

Conforme vai aumentando o tempo de transição do sinal de entrada, aumenta-se também a complexidade para modelar o atraso do inversor. Logo, para garantir uma equação válida em um conjunto amplo de sinais de entrada, foi introduzido um termo de correção para tratar os casos onde a transição do sinal de entrada for lenta. Este fator de correção considera a configuração interna do inversor, a tensão de alimentação e a duração do sinal de entrada. Com isso, chegamos às equações finais que modelam o atraso das portas lógicas:

$$t_{hl} = v_{tn} \times \frac{\tau_{in}}{2} + \left(1 + 2\frac{C_M}{C_M + C_L}\right) \times t_{hl(s)} \times K_n \times \left(1 - \frac{\alpha_n \times (1 - v_{tn})}{1 + (\beta_n \times k)} \left(\frac{\tau_{in}}{t_{hl(s)}}\right)^{\gamma_n}\right) \qquad (4\text{-}6)$$

$$t_{lh} = v_{tp} \times \frac{\tau_{in}}{2} + \left(1 + 2\frac{C_M}{C_M + C_L}\right) \times t_{lh(s)} \times K_p \times \left(1 - \frac{\alpha_p \times (1 - v_{tp})}{1 + \frac{\beta_p}{k}} \left(\frac{\tau_{in}}{t_{lh(s)}}\right)^{\gamma_p}\right) \qquad (4\text{-}7)$$

onde $\alpha_{n(p)}$, $\beta_{n(p)}$ e $\gamma_{n(p)}$ são coeficientes pseudo-empíricos que, assim como o fator de redução $K_{n(p)}$, são característicos da tecnologia e são obtidos a partir de simulação elétrica durante a calibração do modelo.

4.2.3 Modelo de atraso para conexões

Para tornar a estimativa do atraso robusta nas tecnologias onde o atraso das interconexões é determinante no desempenho do circuito, existe a possibilidade de considerar o atraso das conexões durante a identificação do atraso e do caminho críticos. Uma vez que o atraso das portas lógicas é modelado em uma abordagem *gate-level*, existe a necessidade de separar o cálculo do atraso das conexões e das portas lógicas, analisá-las individualmente e então compor de forma adequada o atraso do estágio lógico (QIAN, 1994).

Primeiramente é computado o atraso e a capacitância "efetiva" da conexão vista pela porta, através do método proposto por *Kashyap* (2000). Então estes novos valores de capacitância são usados no cálculo do atraso da porta lógica através do modelo citado na subseção 4.2.2. O método de cálculo de atraso das conexões adotado é tão rápido quanto a conhecida métrica ED, porém com uma precisão consideravelmente maior (FONSECA, 2005).

4.2.4 Estimativa das capacitâncias parasitas e *back-annotation*

Considerar os efeitos de etapas posteriores durante a realização de uma etapa do fluxo de síntese ajuda na convergência para os resultados esperados. Basicamente, existem duas maneiras de considerar estes efeitos: usar estimativas dos resultados ou realizar *back-annotation*. Durante as diferentes etapas da síntese física vai sendo refinado o detalhamento da implementação do circuito, e com isso vão surgindo elementos *RC* parasitas que alteram o desempenho elétrico do mesmo.

Geralmente, procura-se garantir o pior caso quando do uso de estimativas. Isso faz com que sejam evitadas voltas a etapas anteriores do fluxo, com o objetivo de obter o leiaute final do circuito em um tempo menor. Porém, o caráter pessimista das estimativas faz com que possa haver um acréscimo desnecessário de área, ou até mesmo que as restrições de desempenho sejam declaradas inatingíveis. A realização de *back-*

annotation garante precisão às etapas de verificação, mas necessita que etapas de síntese sejam refeitas caso as restrições de atraso não sejam atendidas. E, se estas etapas de síntese não forem capazes de refazer a síntese de forma incremental, pode ocorrer de não haver convergência durante o fluxo.

Para cálculo da capacitância de saída (C_L) de cada porta lógica do circuito, são consideradas as capacitâncias ativas (C_{at}), capacitâncias de difusão (C_{dif}) e capacitância das conexões provenientes da etapa de roteamento (C_{rot}):

$$C_L = C_{at} + C_{dif} + C_{rot} \tag{4-8}$$

A capacitância ativa (C_{at}) é formada pela capacitância de *gate* dos transistores que estão conectados na saída da porta. Esta capacitância de *gate* de cada transistor é, por simplificação, considerada linear e calculada desconsiderando sobreposição entre o óxido e as regiões de fonte e dreno do transistor:

$$C_{at} = W \times L \times C_{ox} \tag{4-9}$$

A ferramenta de análise de *timing TicTac* é capaz de ler um arquivo contendo informações sobre elementos parasitas associados a nodos do circuito. Com isso, é possível que informações sobre as capacitâncias de difusão e capacitâncias de conexões, estimadas em etapas iniciais da síntese ou extraídas de uma implementação física mais detalhada, sejam passadas para a ferramenta. Se as capacitâncias de difusão não forem passadas através deste arquivo, o cálculo destas capacitâncias de difusão será realizado pela própria ferramenta. Neste caso, esta capacitância também é considerada linear, e o cálculo da mesma considera as contribuições das zonas de depleção formadas sob a área de base e pelas áreas laterais da região de difusão:

$$C_{dif} = \left(A_{D(S)} \times CJ \times \left(1 + \frac{VJ}{PB} \right)^{-MJ} \right) + \left(P_{D(S)} \times CJSW \times \left(1 + \frac{VJ}{PB} \right)^{-MJSW} \right) \tag{4-10}$$

onde $A_{D(S)}$ é a área e $P_{D(S)}$ é o perímetro do dreno (fonte) dos transistores que estão conectados na saída da porta. *CJ* e *CJSW* são, respectivamente, as capacitâncias de junção por unidade de área e por unidade de perímetro e, assim como *VJ*, *PB*, *MJ*, e *MJSW*, são parâmetros da tecnologia.

4.3 Geração de vetores de teste

Durante a fabricação podem ocorrer problemas de manufatura que façam com que aumentem os atrasos de transistores e conexões do CI. Neste caso, um teste de manufatura com vetores específicos para detecção de falhas de atraso deve ser realizado após a fabricação. É importante notar que este teste de atraso é somado ao teste que verifica falhas de colagem. Outra situação comum é que, ainda durante o projeto do CI, pode ser desejável verificar o atraso do circuito através de simulação elétrica, uma vez que, por mais precisos que sejam, os modelos de atraso geralmente fornecem uma sobrestimativa do atraso do componente que modelam.

Nestes dois casos acima, uma seleção criteriosa dos vetores a serem aplicados ocasiona uma redução drástica de custos. Seja redução de custo computacional, no caso de uma simulação elétrica, seja no custo do CI, visto que a etapa de teste de manufatura tem seu custo diretamente vinculado ao número de vetores aplicados. A ferramenta *TicTac* utiliza um método de geração automática de vetores de teste (ATPG).

O método de ATPG desenvolvido pode ser dividido em 2 etapas (FERRÃO, 2004):

1) identificação dos "*k*" caminhos críticos sensibilizáveis a serem testados, onde *k* é um valor passado pelo usuário;

2) geração dos vetores de teste para estes caminhos.

A etapa de identificação dos caminhos críticos é feita por um método de FTA que utiliza o critério de sensibilização *exato*, e termina quando o número de caminhos identificados é igual a *k* ou quando não é possível identificar nenhum novo caminho sensibilizável. A definição do número *k* de caminhos a serem testados pode ser feita por meio de um número absoluto ou por um valor percentual. Este valor percentual indica a faixa de atraso que será considerada na definição dos caminhos a serem testados.

A geração de vetores de teste é baseada no modelo de falha conhecido como PDF (*Path Delay Fault*). Ao contrário do modelo de falhas GDF (*Gate Delay Fault*), que considera que os defeitos afetam apenas uma porta ou conexão, o modelo PDF é capaz de identificar falhas decorrentes da soma de pequenas falhas nos transistores e conexões de um caminho.

Os vetores de teste a serem gerados podem ser classificados em dois tipos, de acordo com as condições sob as quais garantem detectar a falha no caminho a ser testado (BUSHNELL, 2000):

- **Teste *não-robusto*** – só garante identificar uma falha se o caminho testado for o único a falhar dentre todos os caminhos que compartilham pelo menos uma porta ou conexão com o caminho testado.
- **Teste *robusto*** – garante identificar uma falha no caminho testado, independente da existência de outros caminhos com falhas que possam afetar portas e conexões do caminho testado.

Um método comum para geração de vetores de teste é a geração do par de vetores (V_1,V_2) através do uso da *álgebra de cinco valores* (BUSHNELL, 2000). No entanto, este método requer uma tabela-verdade, baseada nesta álgebra, para cada porta lógica do circuito. Porém isto não é desejado quando esta tabela não está pré-definida para cada porta, como é o caso de circuitos mapeados para uma biblioteca virtual.

O método desenvolvido desassocia os vetores do mesmo par e realiza a geração de cada vetor de forma independente, satisfazendo as condições necessárias para cada vetor. As diferentes condições necessárias para geração de teste robusto/não-robusto são traduzidas em um critério de sensibilização apropriado. Com isso, não é necessário o uso da álgebra de cinco valores e nem existe a necessidade de uma tabela-verdade para cada porta lógica.

Os valores necessários nas entradas laterais do caminho para a geração de um par de vetores para teste robusto são:

1) Para o vetor V_2:
   - Todas as entradas laterais precisam estar em valores não-controlantes.
2) Para o vetor V_1:
   - Se a entrada sobre o caminho está em um valor não-controlante (durante V_1), então todas as entradas laterais do caminho precisam estar em valor não-controlante.

- Se a entrada sobre o caminho está em um valor controlante (durante V_1), então as entradas laterais do caminho podem estar em qualquer valor lógico.

Podemos notar que as condições para V_2 e V_1 acima são iguais às condições necessárias para sensibilização de um caminho nos critérios de sensibilização estática e co-sensibilização estática, respectivamente.

Para geração do par de vetores para teste não-robusto, as condições necessárias para V_2 são as mesmas do teste robusto, e não existem condições necessárias nas entradas laterais do caminho para geração de V_1:

1) Para o vetor V_2:
   - Todas as entradas laterais precisam estar em valores não-controlantes.
2) Para o vetor V_1:
   - Um vetor complementar a V_2 é suficiente.

Se a entrada sobre o caminho está em um valor controlante (durante V_1), então as entradas laterais do caminho podem estar em qualquer valor lógico

No método de ATPG desenvolvido, o caminho a ser testado é sensibilizado para gerar V_2, sendo logo após sensibilizado novamente para gerar V_1. Desta forma, a geração do par de vetores de teste é feita pela aplicação do critério de sensibilização adequado a cada tipo de teste, onde a geração de teste robusto tem preferência sobre a geração de teste não-robusto:

O critério de sensibilização exato, utilizado durante a etapa de seleção do caminho a ser testado, leva em consideração informações do atraso de portas e conexões. Já os critérios de sensibilização estática e co-sensibilização estática, usados para geração dos vetores de teste, levam em conta apenas informações lógicas. Esta diferença entre os critérios pode fazer com que alguns caminhos sensibilizados sob o critério de sensibilização exato não sejam sensibilizáveis pelo critério estático, e, com isso, não tenham nem mesmo um par de vetores de teste não-robusto gerado, sendo declarados caminhos não-testáveis. No entanto, como todos os caminhos selecionados para teste são capazes de propagar uma transição até as saídas primárias do circuito, é altamente desejável gerar testes para todos estes caminhos.

Uma possibilidade oferecida pelo método de ATPG desenvolvido, ilustrado na Figura 4.9, é a utilização do vetor gerado pela sensibilização com o critério exato durante a identificação do caminho. Desta forma temos um par de vetores, chamado de teste *flutuante*, onde V_2 será o vetor resultante da etapa de sensibilização com critério exato e V_1 será o vetor complementar de V_2. Porém, esta opção só será utilizada quando não for possível sensibilizar o caminho com o critério de sensibilização estática, visto que a sensibilização estática do caminho garante no mínimo um par de vetores para teste não-robusto.

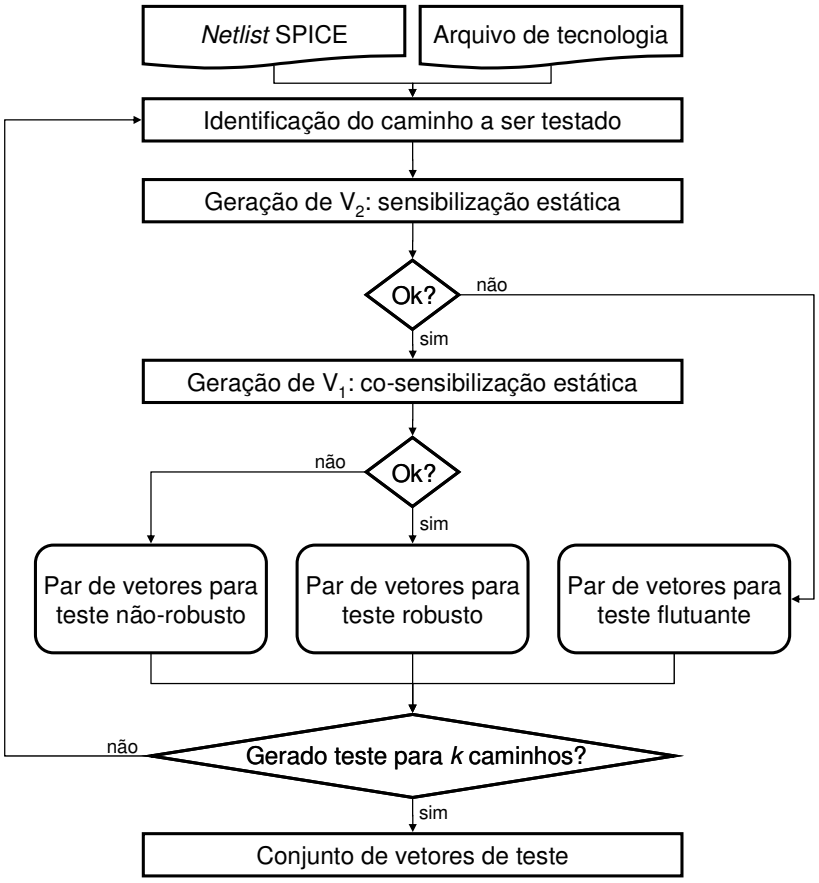

Figura 4.9: Geração automática de vetores para teste de atraso.

O método de ATPG, fica, então, da seguinte forma:

1) Identificação do caminho a ser testado;
2) Sensibilização do caminho selecionado com critério de sensibilização estática;
3) Sensibilização do caminho selecionado (com inversão da polaridade) pelo critério de co-sensibilização estática;
4) Se 2) e 3) forem realizados com sucesso, então um par de vetores para teste robusto é gerado;
5) Se apenas 2) foi realizado com sucesso, então é gerado um par de vetores para teste não-robusto;
6) Se 2) e 3) falharem, então o vetor gerado em 1) é usado para gerar um par de vetores de teste.

Para avaliar a cobertura de falhas do método de ATPG proposto, foram usados alguns dos circuitos combinacionais que estão descritos na Tabela A-6.1 do apêndice. Foram gerados vetores de teste para todos os caminhos sensibilizáveis com atraso até 30% inferior ao atraso do caminho crítico do circuito (k=30%). A Tabela 4.1 resume os resultados obtidos.

A segunda e a terceira colunas mostram, respectivamente, o número de caminhos falsos e o número de caminhos selecionados para teste dentre os caminhos que possuíam atraso até 30% inferior ao atraso do caminho crítico. A quarta e a quinta

coluna contêm o número de pares de vetores para teste robusto e teste não-robusto que foram gerados, enquanto a sexta coluna mostra quantos caminhos foram declarados não-testáveis sob o critério de sensibilização estática.

As três últimas colunas mostram a cobertura de falhas para cada um dos tipos de teste. Falhas simples englobam os casos onde apenas um caminho pode falhar por vez, e, assim sendo, tanto os vetores para teste não-robusto quanto os vetores para teste robusto foram usados para calcular a cobertura de falhas. Falhas múltiplas assumem que mais de um caminho pode falhar ao mesmo tempo. Com isso, apenas os vetores para teste robusto foram considerados no cálculo da cobertura. A última coluna mostra os vetores gerados para teste flutuante. Podemos notar que a soma dos valores apresentados na sétima e na última coluna sempre é igual a 100%, indicando que nenhum caminho que fora selecionado durante a etapa de identificação dos caminhos críticos é deixado sem vetores para teste de atraso.

Tabela 4.1: Análise da cobertura de falhas.

Circuito	Cam. falsos	Cam. selec.	Teste robusto	Teste não-robusto	Caminhos não-testáveis	Cobertura de falhas simples	Cobertura de falhas múltiplas	Cobertura do teste flutuante
9sym	125	85	84	1	0	100%	98%	0%
alu2	66130	239	42	114	83	65%	17%	35%†
bw	55	151	136	0	15	90%	90%	10%
c17	4	8	8	0	0	100%	100%	0%
csa2	26	18	8	10	0	100%	44%	0%
csa4	134	61	28	26	7	88%	46%	12%
csa8	1831	123	62	44	17	86%	50%	14%
f50	27	7	0	2	5	28%	0%	72%†

† Casos onde apenas vetores para teste flutuante puderam ser gerados para o caminho crítico do circuito.

Os vetores gerados para o teste flutuante têm algumas limitações. Falhas de manufatura que modifiquem o atraso dos componentes do CI ou imprecisões do modelo de atraso podem fazer com que o caminho sensibilizado com critério exato torne-se falso e algum outro caminho declarado como falso torne-se verdadeiro. E isso pode fazer com que os vetores para teste flutuante não consigam detectar a falha de atraso.

Porém, mesmo com estas limitações, o método de ATPG melhora a cobertura de falhas dos vetores gerados, pois garante a geração de vetores de teste mesmo para caminhos onde a sensibilização estática falha e não é possível gerar vetores de para teste não-robusto. Em alguns casos, como no circuito *alu2*, os vetores para teste flutuante são os únicos disponíveis para o caminho crítico do circuito.

4.4 Conclusão

A presença de um gerador automático de leiaute no fluxo FUCAS permitiu a integração de um método de otimização de atraso, capaz de atuar no nível de transistores, e conduzir a geração dos circuitos pelas restrições de atraso.

Para acomodar os transistores dimensionados no estilo de geração de leiaute empregado, utiliza-se uma técnica de *folding*, onde os transistores são quebrados em transistores menores e conectados em paralelo. Isto torna o problema de dimensionamento de transistores um problema essencialmente discreto, ou seja, os transistores do circuito só podem assumir valores múltiplos do seu tamanho inicial, que é igual à altura da banda de difusão na qual ele está inserido.

A ferramenta de FTA *TicTac*, capaz de tratar circuitos contendo SCCGs, é utilizada para evitar a síndrome dos caminhos falsos durante a identificação do atraso e do caminho críticos. A complexidade em caracterizar o atraso dos dispositivos do circuito em tecnologias DSM tem aumentando consideravelmente. Atentando para esta dificuldade, procurou-se o uso de um modelo de atraso capaz de considerar efeitos de segunda ordem presentes nestas novas tecnologias. Ainda, quando escolhida alguma tecnologia onde o atraso das conexões pode ser decisivo, existe a opção do uso de um método para estimar atraso de conexões totalmente integrado com método de cálculo de atraso das portas do circuito.

A etapa de ATPG para teste de atraso possibilita que os atrasos do circuito sejam verificados também por simulação elétrica, ou que falhas de manufatura sejam detectadas após a fabricação. O método de ATPG proposto buscou aproveitar as características já existentes na ferramenta *TicTac*. Para evitar a necessidade de criar previamente uma tabela para cada porta do circuito, não foi utilizado o tradicional método baseado na álgebra de cinco valores. Ao invés disso, os vetores de um mesmo par foram desassociados e gerados independentemente, satisfazendo as condições necessárias para geração de teste robusto/não-robusto de cada vetor. Ainda, o método oferece a possibilidade de geração de vetores para os caminhos declarados não-testáveis pelo método tradicional de geração de teste robusto e não-robusto. Estes vetores, chamados de vetores flutuantes, apesar de limitados, aumentam a cobertura de falhas do circuito e, em alguns casos, são o único teste disponível para os caminhos com atraso mais crítico.

5 PROPOSTA DE UM MÉTODO DE DIMENSIONAMENTO DE TRANSISTORES

Anteriormente foram mostrados diferentes métodos de otimização de atraso que podem ser utilizados quando o fluxo de projeto chega ao nível de transistores. Dimensionamento de transistores é a técnica de otimização de atraso que mais consegue tirar proveito do fato de, como mostrado no capítulo anterior, o fluxo de síntese física adotado ser independente de biblioteca de células e realizar a geração de leiaute sob demanda a partir de um *netlist* de transistores. A fim de explorar da melhor forma as características deste fluxo de síntese física, o método de dimensionamento irá considerar algumas das suas principais características:

1. O uso de um método de FTA na identificação do atraso e dos caminhos críticos evita a síndrome dos caminhos falsos, fazendo com que haja uma redução no acréscimo de área provocado pelo dimensionamento de transistores. Isso acontece porque não será necessário otimizar portas lógicas que não contribuem efetivamente para o atraso do circuito. Porém, métodos de FTA fazem com que não seja possível modelar o atraso de um circuito como uma função matemática, descartando, assim, o uso de algoritmos de dimensionamento que requerem esta característica.
2. Se observarmos a Figura 2.7, podemos ver que o critério de sensibilização exato sob o modo flutuante é o critério que consegue maior precisão na identificação do atraso crítico. Entretanto, por levar em consideração o atraso das portas lógicas, este critério faz com que haja necessidade de identificar o caminho e o atraso críticos, iterativamente, a cada vez que uma porta do circuito tiver seu atraso alterado.
3. Qualquer transistor que seja aumentado será "quebrado" pela técnica de *folding* para ser acomodado adequadamente nas bandas de difusão durante a geração do leiaute. Isso faz com que os transistores precisem ser dimensionados de forma discreta, assumindo tamanhos que sejam múltiplos inteiros do seu tamanho inicial.

Estas três características acima, principalmente, impedem que alguns algoritmos de dimensionamento de transistores sejam adotados e fazem com que um algoritmo heurístico de dimensionamento seja a melhor escolha a ser implementada. As seções seguintes mostram o desenvolvimento das diferentes etapas deste método heurístico de dimensionamento, assim como algumas comparações que ajudaram a aprimorar e validar o método proposto.

5.1 Método heurístico de dimensionamento

A Figura 5.1 ilustra o método de dimensionamento de transistores implementado (SANTOS, 2003).

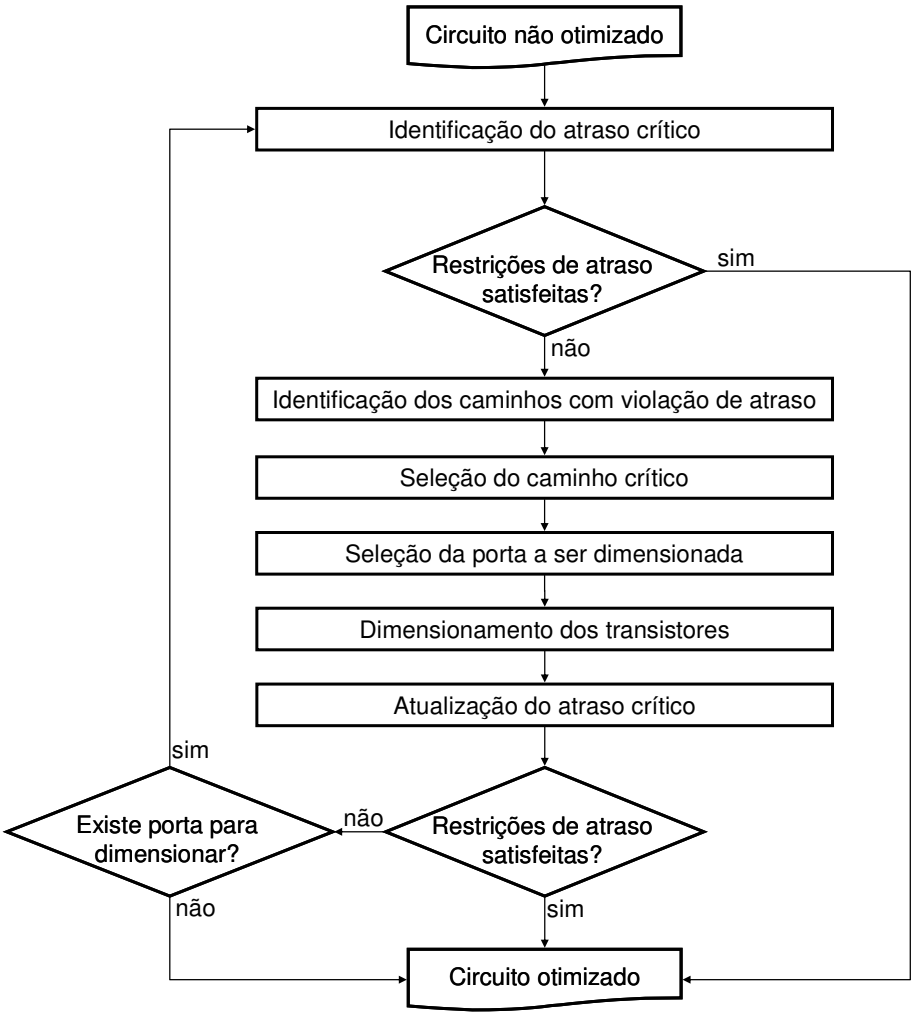

Figura 5.1: Método de dimensionamento de transistores.

O método inicia a partir de um *netlist* de transistores de um circuito não otimizado, identificando o atraso do circuito. Se este atraso for maior que a restrição requerida pelo projetista, haverá a identificação do conjunto de caminhos que violam esta restrição. Este conjunto de caminhos será usado para identificar as portas lógicas que mais influenciam no conjunto de caminhos com atraso superior ao atraso requerido.

O método então identifica o caminho crítico do circuito, assim como as portas lógicas que pertencem a este caminho. Apenas estas portas pertencentes ao caminho crítico poderão se candidatar ao dimensionamento. O método usa heurísticas para identificar, dentre as candidatas, a porta que será dimensionada. Após o dimensionamento dos transistores desta porta, as restrições de atraso são novamente verificadas. O método pára quando o caminho crítico sensibilizável possui atraso inferior ao atraso requerido pelo projetista, ou então quando não existem mais portas candidatas ao dimensionamento.

5.1.1 Identificação do conjunto de caminhos críticos

O conjunto de caminhos que violam o atraso especificado pelo projetista abrange todos os caminhos cujos atrasos estão entre o atraso do caminho crítico sensibilizável e a restrição de atraso (ver Figura 5.2). Apesar de o caminho crítico ser identificado por um método de FTA, os demais caminhos deste conjunto são identificados através de um método de TTA. Isto acontece por dois motivos. Um deles é o custo computacional que haveria em sensibilizar todos os caminhos do conjunto. O outro é devido ao fato de a ferramenta usar o critério de sensibilização exato, o que faz com que todos os caminhos do conjunto possam se tornar sensibilizáveis na medida em que as portas vão sendo dimensionadas.

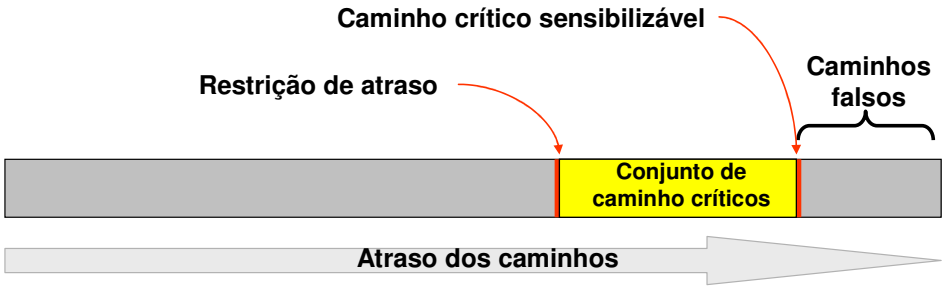

Figura 5.2: Identificação do conjunto de caminhos críticos.

5.1.2 Seleção da porta a ser dimensionada

Cada porta lógica é marcada de forma a indicar quantas vezes ela aparece no conjunto de caminhos críticos previamente identificados. Isto serve para indicar quantos caminhos críticos são afetados quando uma porta é modificada. Como dito anteriormente, somente portas que pertençam ao caminho crítico poderão ser candidatas ao dimensionamento de transistores. Além disso, para ser candidata, a porta precisa ter:

- Atraso superior ao atraso médio requerido por porta;
- Fator de carga (FC) maior que 2.

A restrição de atraso é distribuída entre as portas que pertencem ao caminho crítico, de forma a identificar qual é o atraso médio requerido de cada porta para que a restrição seja atingida. Já fator de carga é a razão entre a capacitância de saída da porta (C_L) e a capacitância do transistor PMOS ou NMOS do inversor equivalente:

$$FC = C_L/C_{N(P)} \tag{5-1}$$

O fator de carga é diretamente proporcional ao atraso da porta, como podemos notar pelas equações (4-1) e (4-2). A exclusão de portas com fator de carga inferior a dois ocorre para evitar que transistores sejam dimensionados demasiadamente.

O número de marcas e o fator de carga de cada porta são usados como critérios para escolher, dentre as portas candidatas, qual será a porta selecionada para ser dimensionada. Podemos ver que ao usar o critério do número de marcas, estamos fugindo do problema de usar o critério da *sensibilidade*, mostrado na Figura 3.4. Neste caso, a porta G_0 estaria com 3 marcas e seria a porta selecionada para dimensionamento.

5.1.3 Dimensionamento dos transistores

Considere a estrutura PMOS de uma porta lógica mostrada na Figura 5.3. Como visto na subseção 4.2.2, o modelo computacional de atraso da porta é par de atraso por

porta e, para garantir o pior caso de atraso, o fator de correção de corrente máxima da porta é identificado considerando a maior cadeia de transistores em série em cada estrutura. Sendo assim, as cadeias de transistores *a-b* e *c-d* são igualmente críticas se todos os transistores tiverem o mesmo tamanho. Se apenas uma destas cadeias tiver seus transistores aumentados, o atraso da porta não sofrerá alteração, uma vez que a cadeia que não foi dimensionada manterá seu atraso. Para garantir que a porta selecionada terá seu atraso reduzido, todos os seus transistores da mesma estrutura serão igualmente dimensionados a cada iteração.

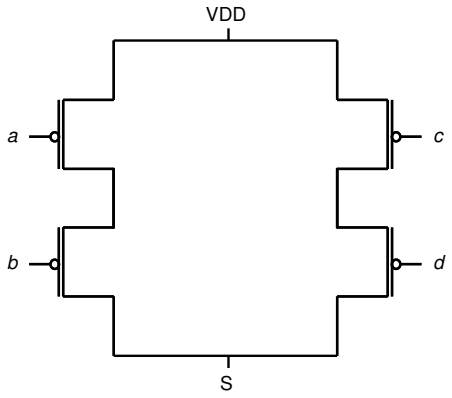

Figura 5.3: Dimensionamento dos transistores em um modelo par de atrasos.

Ainda, o dimensionamento dos transistores precisa ser discreto, pois mesmo que uma variação contínua de tamanho conseguisse atingir as restrições de atraso com menos acréscimo de área, o tamanho dos transistores precisaria ser arredondado para valores discretos na hora da geração do leiaute. Porém, não haveria garantia de que, depois de corrigido o tamanho dos transistores, as estimativas de atraso continuariam adequadas.

5.1.4 Resultados

O método de dimensionamento de transistores ilustrado na Figura 5.1 foi implementado na ferramenta *TicTac*. Para avaliar o método proposto, foram usados alguns circuitos puramente combinacionais implementados em tecnologia 0.35μm. A Tabela 5.1 mostra os resultados obtidos pelo método de otimização quando foram feitas diferentes restrições de atraso para cada circuito.

Todos os circuitos partem de uma descrição inicial onde seus transistores possuem canal com largura (W) de 1μm e comprimento (L) igual ao mínimo permitido pela tecnologia. A primeira coluna da tabela traz informações sobre o circuito, como atraso topológico inicial (T_u) e atraso funcional inicial (F_u). Os sufixos *se* e *sccg* indicam, respectivamente, se o circuito utiliza apenas portas simples de 2 entradas (Tabela A-6.1) ou se utiliza portas complexas (Tabela A-6.2). A segunda coluna mostra as restrições de atraso que foram impostas ao método de otimização nos experimentos realizados.

A terceira coluna mostra qual foi a redução de atraso obtida pelo método. Podemos notar que algumas vezes o método consegue atingir valores ligeiramente melhores que os valores que foram impostos. Isso acontece porque o método aplica dimensionamento discreto e, com isso, a mudança nos valores de atraso também é discreta. A quarta e a quinta coluna mostram, respectivamente, o acréscimo de área resultante da otimização e o número de portas que foram modificadas durante o dimensionamento. Aqui cabe lembrar que o valor de área considerado é igual ao calculado pela equação (3-1).

A sexta coluna da tabela mostra qual é o atraso crítico topológico resultante após a otimização do circuito. Podemos ver que, em alguns casos, o atraso crítico topológico do circuito sofre uma redução menor do que a redução do atraso crítico funcional. Isto acontece devido à presença de caminhos falsos com atraso superior ao atraso crítico funcional, os quais não são considerados pelo método de otimização uma vez que este é guiado por um método de FTA.

A última coluna da tabela mostra o acréscimo de consumo de potência decorrente da otimização de atraso. O consumo de potência dos circuitos originais e otimizados foi estimado através de simulação elétrica. Foram simulados, em todos os casos, 1000 vetores aleatórios em uma freqüência de 100 MHz.

Tabela 5.1: Resultados do método de otimização.

Circuito	Restrição de atraso (*ns* / %)	Redução de atraso (%)	Acréscimo de área (%)	Portas modificadas	Atraso topológico (*ns*)	Acréscimo de consumo (%)
9sym_se	1.70 / 12.0	12.5	1.5	14	1.69	0.9
T_u = 1.93 *ns*	1.50 / 22.3	22.5	9.4	52	14.9	9.8
F_u = 1.93 *ns*	1.30 / 32.5	32.6	28.5	87	1.32	39.7
alu2_se	3.50 / 10.5	11.7	0.7	13	4.13	0.0
T_u = 4.59 *ns*	3.10 / 20.7	21.0	3.5	42	3.85	3.2
F_u = 3.91 *ns*	2.90 / 25.8	26.4	7.3	73	3.87	6.5
bw_se	1.40 / 14.1	14.1	1.2	8	1.40	1.0
T_u = 1.63 *ns*	1.20 / 26.5	26.6	6.4	30	1.30	8.7
F_u = 1.63 *ns*	1.00 / 38.5	40.4	26.0	79	0.97	26.7
csa2_se	0.65 / 11.0	11.9	9.4	3	0.73	13.3
T_u = 0.84 *ns*	0.60 / 18.0	18.2	30.2	9	0.70	33.0
F_u = 0.73 *ns*	0.50 / 31.5	32.1	134.9	21	0.60	149.8
csa4_se	1.00 / 13.5	13.5	15.1	9	1.33	15.1
T_u = 1.43 *ns*	0.90 / 22.5	23.3	46.2	24	1.22	46.8
F_u = 1.16 *ns*	0.80 / 30.0	30.0	160.0	27	1.18	161.3
csa8_se	1.50 / 11.0	11.7	4.3	5	2.36	4.6
T_u = 2.60 *ns*	1.30 / 23.0	23.1	19.9	20	2.36	18.5
F_u = 1.69 *ns*	1.10 / 35.0	35.2	76.2	40	2.33	74.5
9sym_sccg	1.46 / 12.0	17.4	1.0	3	2.04	4.0
T_u = 2.38 *ns*	1.29 / 22.3	22.5	1.9	5	2.04	5.8
F_u = 1.66 *ns*	1.12 / 32.5	33.5	4.0	10	2.03	11.5
alu2_sccg	1.14 / 10.5	11.35	1.5	4	4.66	2.3
T_u = 4.65 *ns*	1.00 / 20.7	20.9	4.0	9	4.55	4.9
F_u = 1.27 *ns*	0.93 / 26.0	27.0	5.0	12	4.56	6.3
bw_sccg	1.40 / 32.7	34.6	7.5	7	3.19	8.7
T_u = 3.55 *ns*	1.20 / 42.3	42.6	9.9	11	3.22	12.2
F_u = 2.08 *ns*	1.00 / 52.0	52.0	19.5	22	2.80	23.8
t481_sccg	2.09 / 10.0	11.5	0.2	3	2.24	1.5
T_u = 2.73 *ns*	1.90 / 18.0	18.4	1.0	11	2.24	3.8
F_u = 2.32 *ns*	1.74 / 25.0	25.4	4.6	38	2.17	11.5

Podemos notar que, para pequenas restrições de atraso, o método consegue otimizar o circuito com uma penalidade pequena em área e consumo de potência. Mas, na medida em que as restrições aumentam, os acréscimos de área e consumo aumentam de forma quase exponencial. Em resumo, isso acontece por causa de dois fatores. Se imaginarmos a distribuição dos atrasos dos caminhos do circuito em um gráfico de atraso versus o número de caminhos, poderemos notar que, conforme o atraso vai sendo reduzido, o número de caminhos que precisam ser otimizados cresce quase que exponencialmente também. E quanto maior o número de portas modificadas no circuito,

maior a chance de caminhos declarados falsos se tornarem verdadeiros e, desta forma, também precisarem ser otimizados.

Como pode ser verificado através da quarta e da última coluna da Tabela 5.1, o comportamento do acréscimo de consumo de potência dos circuitos tem um comportamento muito parecido com o acréscimo de área dos mesmos. Isto parece evidente quando lembramos que a métrica de área usada na avaliação dos resultados representa a soma das áreas de *gate* dos transistores do circuito, que é diretamente proporcional a capacitância ativa que é carregada e descarregada durante o chaveamento do circuito. Isto mostra que, mesmo que o método de otimização de atraso não seja guiado por nenhuma técnica que vise minimizar o acréscimo de consumo, a redução do acréscimo de área do circuito implicará também na redução do acréscimo de consumo do mesmo.

O fato de utilizar portas complexas na implementação dos circuitos ocasionou uma alteração nos valores de atraso topológico e funcional de cada circuito, além de uma redução considerável do número de portas lógicas e transistores (Tabela A-6.2). Sendo assim, procurou-se balizar as restrições de atraso de forma que fosse possível fazer comparações entre as versões dos circuitos com portas simples e com portas complexas. Para os circuitos em que a versão com portas complexas possui atraso funcional inferior, foram feitas restrições em valores percentuais idênticos aos pedidos ao seu equivalente com portas simples. No caso do circuito *bw*, cujo atraso funcional da versão com portas complexas ficou bem acima, foram feitas restrições de atraso, em valores absolutos, idênticas às feitas para a versão do circuito *bw* com portas simples.

Podemos notar que todas estas restrições foram atingidas, resultando em circuitos mais rápidos e menores do que aqueles que foram implementados somente com portas simples. Isto mostra que o uso de portas complexas, além de trazer um óbvio ganho em área, não compromete o desempenho de velocidade dos circuitos. A Figura 5.4 e a Figura 5.5 mostram o comportamento do acréscimo de área resultante frente a diferentes restrições de atraso para as duas versões dos circuitos *bw*, *9sym* e *alu2*.

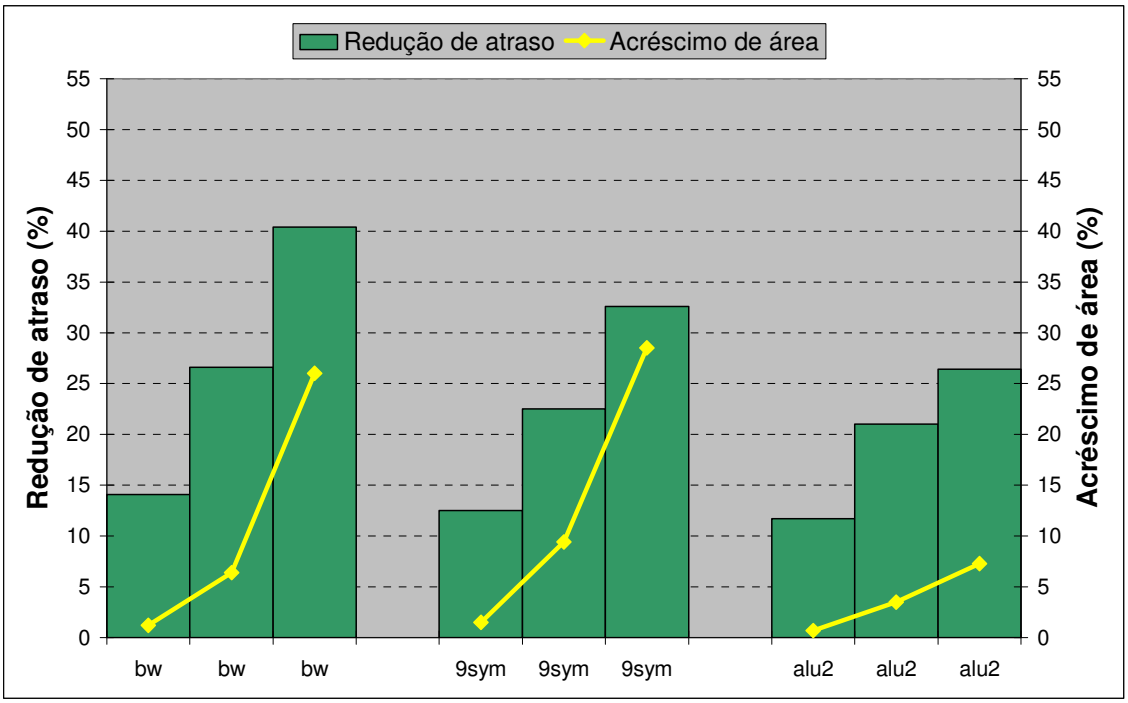

Figura 5.4: Redução de atraso vs. acréscimo de área para circuitos com portas simples.

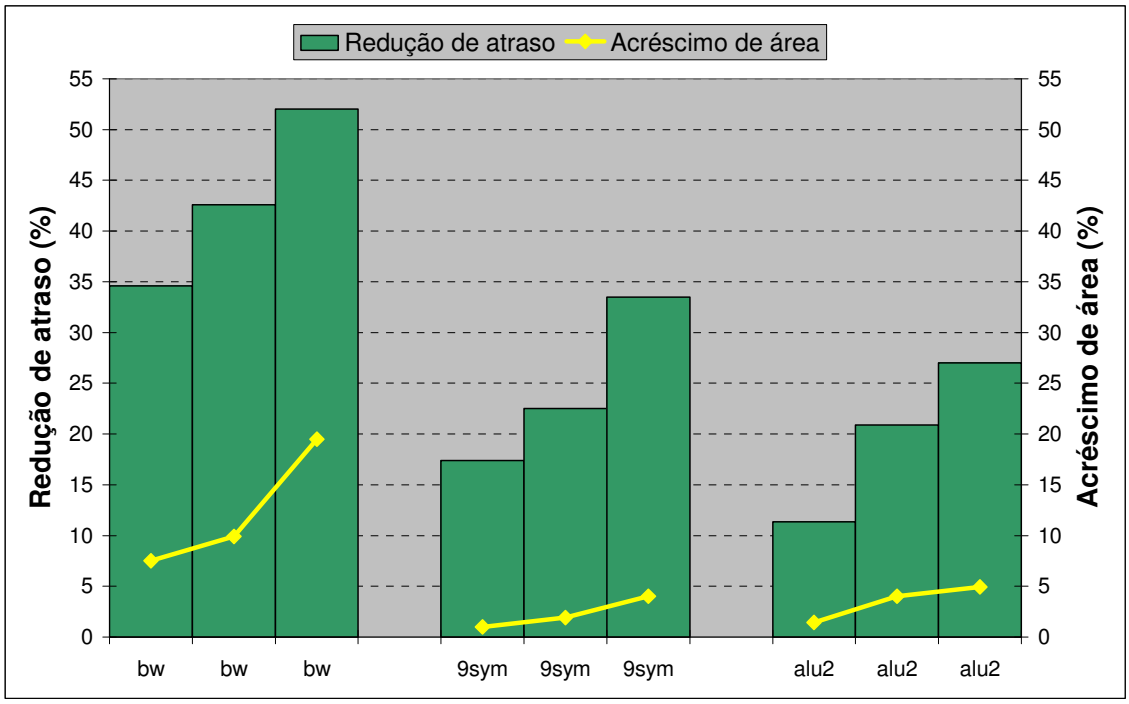

Figura 5.5: Redução de atraso vs. acréscimo de área para circuitos com SCCGs.

5.2 Método incremental de identificação do caminho crítico

Os resultados mostrados na Tabela 5.1 evidenciam a importância do uso de um método de FTA para identificação do caminho crítico. Porém, este método é muito custoso em termos de CPU, o que, de certa forma, limita o tamanho dos circuitos que podem ser otimizados. A etapa de sensibilização dos caminhos é a que consome mais tempo de CPU. O fato de esta etapa precisar ser repetida a cada iteração, uma vez que a mudança no atraso das portas pode mudar também a sensibilização dos caminhos, torna o método muito mais lento que outros que utilizam um método de TTA para identificar o caminho crítico.

A solução encontrada para acelerar o processo de identificação do caminho crítico foi realizar um método incremental, de forma que não fosse necessário repetir todo o processo de sensibilização a cada iteração (SANTOS, 2005-a). Como visto na seção 2.3, o critério de sensibilização estática não utiliza informações de atraso na etapa de sensibilização. Isto não a torna mais rápida, e ainda pode fazer com que seja subestimado o atraso crítico do circuito. Porém, o uso deste critério faz com que as alterações de atraso nas portas do circuito não influenciem na sensibilização dos caminhos críticos (CHENG, 1991). Uma vez que um caminho seja declarado falso ou verdadeiro, o dimensionamento dos transistores das portas do circuito não irá alterar a sua condição de sensibilização.

O método de dimensionamento será então dividido em duas etapas. A primeira etapa será responsável por obter um laço rápido de otimização, utilizando um método incremental de seleção do caminho crítico baseado no critério de sensibilização estática. A segunda etapa, que utiliza o método proposto anteriormente, é aplicada então para tratar caminhos que tornaram-se sensibilizáveis e que não puderam ser identificados quando do uso do critério de sensibilização estática.

5.2.1 Etapa incremental

A Figura 5.6 mostra como é feito a etapa incremental do método de otimização.

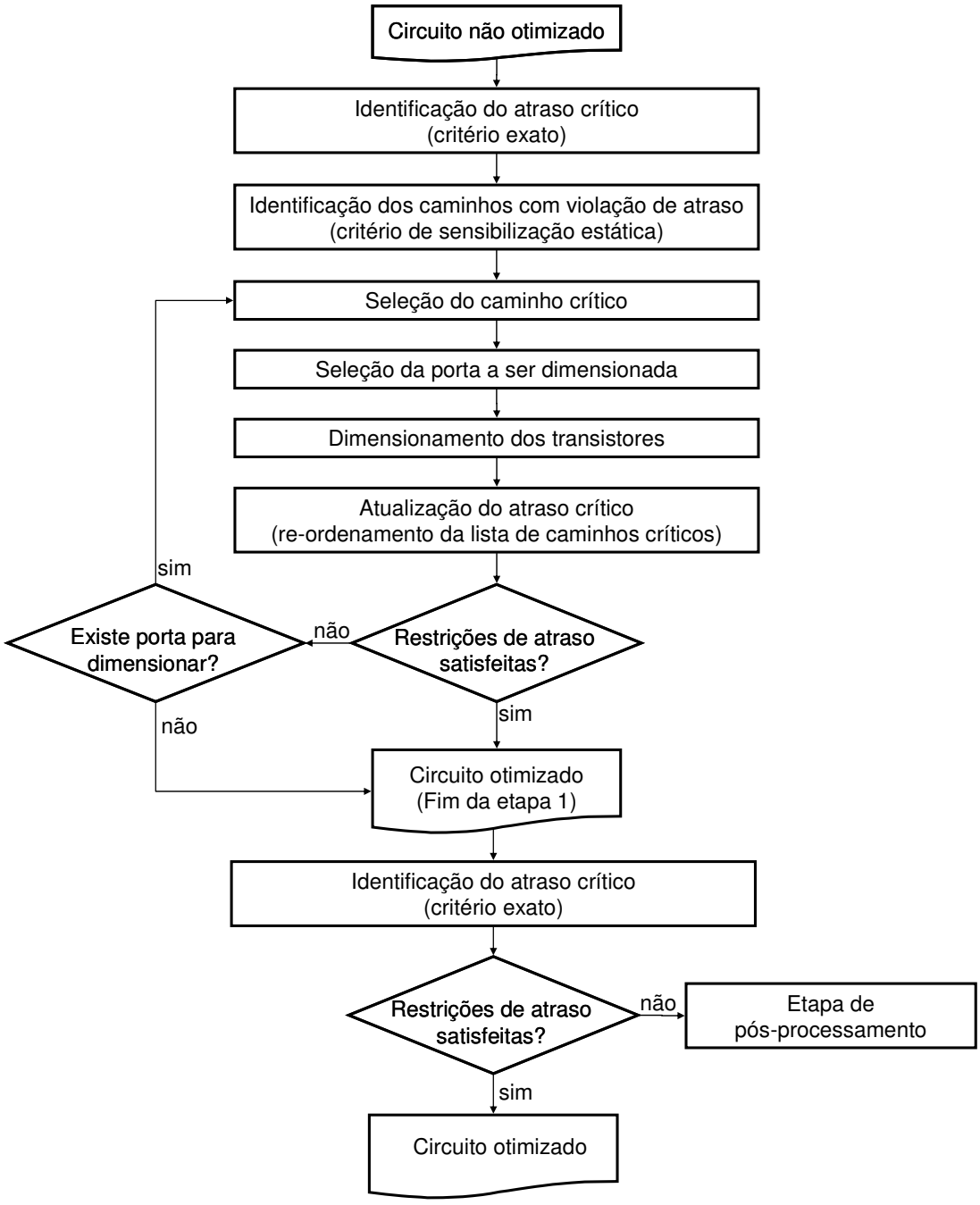

Figura 5.6: Método incremental de identificação do caminho crítico.

A primeira iteração desta etapa incremental é responsável por verificar se a restrição de atraso é atingida. Caso seja necessário otimizar o circuito, é montada uma lista de caminhos críticos segundo o critério de sensibilização estática. Esta lista compreende todos os caminhos sensibilizáveis por este critério cujos atrasos estão entre o atraso do caminho crítico, identificado pelo uso do critério exato, e a restrição de atraso requerida. As próximas iterações desta etapa realizam o dimensionamento dos transistores até que todos os caminhos da lista tenham sido otimizados. Note que, após a otimização de cada porta lógica, basta que seja feito um re-ordenamento da lista de caminhos críticos

considerando todas as portas que tiveram seu atraso afetado. Dessa forma, nenhum processo custoso de sensibilização é realizado a cada iteração, resultando em uma etapa de otimização rápida e simples de computar.

5.2.2 Etapa de pós-processamento

Ao fim da etapa incremental, uma segunda etapa é realizada para certificação de que nenhum caminho sensibilizável, agora usando o critério exato, permanece com atraso incompatível com as restrições. Esta etapa de pós-processamento consiste no método proposto na seção 5.1, e é feita para identificar e otimizar caminhos que se tornaram sensibilizáveis durante a etapa incremental e que não puderam ser identificados quando do uso do critério de sensibilização estática.

Este processo em duas etapas tem como objetivo garantir um compromisso entre a precisão do critério exato e a simplicidade oferecida pelo uso do critério de sensibilização estática. A etapa incremental inicial reduz significativamente o número de caminhos submetidos à etapa mais custosa, acelerando o processo de otimização.

5.2.3 Resultados

Para proporcionar comparações mais interessantes, três versões da etapa de seleção do caminho crítico foram aplicadas a circuitos combinacionais: uma versão que utiliza somente o critério de sensibilização exato; uma versão que não considera a sensibilização dos caminhos (topológica) e a versão incremental descrita nas duas subseções anteriores. A Tabela 5.2 mostra os resultados obtidos desta comparação.

Tabela 5.2: Comparação entre os métodos de seleção do caminho crítico.

Circuito	Restrição de atraso (*ns* / %)	Não-Incremental			Incremental		Δ CPU (%)
		Δ área Topol. (%)	Δ área Exata (%)	CPU Exata	Δ área Incr. (%)	CPU Incr.	
9sym	1.70 / 10.5	1.4	1.6	39.5 *s*	3.0	11.5 *s*	240
T_u = 1.90 *ns*	1.50 / 21.1	8.4	7.8	4.6 *m*	9.5	26.2 *s*	950
F_u = 1.90 *ns*	1.30 / 31.6	24.5	23.5	32.0 *m*	24.3	1.0 *m*	3050
alu2	3.50 / 10.0	9.0	0.7	16.6 *m*	2.4	5.3 *m*	215
T_u = 4.57 *ns*	3.30 / 15.7	17.9	2.3	68.3 *m*	5.6	7.4 *m*	820
F_u = 3.89 *ns*	3.10 / 20.3	39.2	8.0	430.0 *m*	8.3	13.1 *m*	3170
bw	1.33 / 14.7	1.34	1.1	12.1 *s*	1.7	6.2 *s*	95
T_u = 1.56 *ns*	1.15 / 26.3	6.3	5.6	2.2 *m*	7.4	21.0 *s*	520
F_u = 1.56 *ns*	0.95 / 39.1	26.8	20.0	12.8 *m*	32.4	35.3 *s*	2085
c17	0.43 / 11.6	10.5	10.5	0.1 *s*	10.5	0.1 *s*	0
T_u = 0.48 *ns*	0.38 / 20.8	15.6	13.2	0.2 *s*	63.0	0.2 *s*	0
F_u = 0.48 *ns*	0.33 / 31.3	68.4	65.8	0.4 *s*	71.0	0.3 *s*	33
csa2	0.60 / 14.3	62.3	11.3	0.3 *s*	32.0	0.3 *s*	0
T_u = 0.77 *ns*	0.54 / 22.9	95.3	47.2	2.0 *s*	66.0	0.5 *s*	300
F_u = 0.70 *ns*	0.49 / 30.0	⊥	101.9	7.9 *s*	121.7	1.0 *s*	690
csa4	1.00 / 10.7	147.2	9.4	4.7 *s*	68.4	2.4 *s*	95
T_u = 1.36 *ns*	0.95 / 15.2	⊥	53.3	37.4 *s*	113.6	3.6 *s*	940
F_u = 1.12 *ns*	0.89 / 20.5	⊥	116.5	2.1 *m*	134.4	4.2 *s*	2900
csa8	1.50 / 9.1	⊥	2.8	1.4 *m*	2.3	35.8 *s*	140
T_u = 2.53 *ns*	1.35 / 18.2	⊥	10.8	4.0 *m*	18.4	35.4 *s*	575
F_u = 1.65 *ns*	1.19 / 27.9	⊥	75.9	35 *m*	89.6	1.4 *m*	2430
f50	0.33 / 13.1	123.8	9.5	0.1 *s*	14.3	0.1 *s*	0
T_u = 0.44 *ns*	0.30 / 21.0	⊥	14.3	0.2 *s*	14.3	0.1 *s*	100
F_u = 0.38 *ns*	0.25 / 34.2	⊥	28.6	0.4 *s*	9.5	0.2 *s*	100

⊥: Restrição de atraso não atingida.

A primeira coluna traz informações sobre os circuitos a serem otimizados, como atraso topológico (T_u) e atraso funcional (F_u). A segunda coluna indica as diferentes requisições de atraso para cada circuito. Os resultados de acréscimo de área (Δ *área*) dos métodos que utilizaram as versões exata e topológica da etapa de seleção de caminhos são mostrados na coluna rotulada como "*Não-incremental*". Esta coluna também traz o tempo de processamento necessário para otimizar os circuitos quando a versão exata foi utilizada. Como esperado, podemos notar que o custo em área da abordagem topológica pode ser significantemente maior que o da abordagem exata. E em alguns casos, a restrição de atraso não pode ser atingida quando do uso da versão topológica. Isso acontece especialmente em circuitos com atraso topológico superior ao atraso funcional, ou seja, circuitos que possuem caminhos falsos com atraso superior ao atraso do caminho crítico sensibilizável.

A coluna rotulada como "*Incremental*" traz os resultados de acréscimo de área e de tempo de processamento quando foi utilizada a versão incremental da etapa de seleção do caminho crítico. A última coluna da Tabela 5.2 mostra o ganho, em porcentagem, do uso do método incremental com relação à utilização do método que utiliza somente o critério de sensibilização exato.

A Figura 5.7 e a Figura 5.8 mostram os tempos de CPU acumulados para o processo de otimização dos circuitos *alu2* e *9sym* com relação ao número de iterações. A linha que representa o tempo de CPU da abordagem exata cresce muito rapidamente, uma vez que em cada iteração é necessário realizar uma nova etapa de sensibilização. Por outro lado, a abordagem incremental somente realiza a sensibilização dos caminhos críticos no passo inicial e, após esta primeira iteração, seu tempo de CPU acumulado cresce lentamente, assim como ocorre com a abordagem topológica.

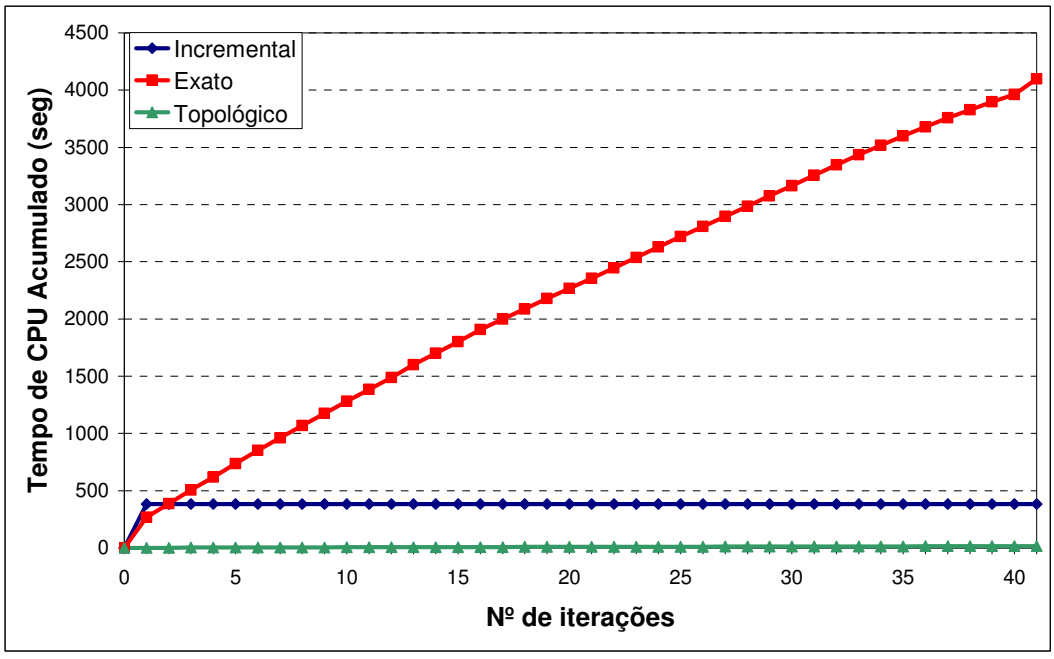

Figura 5.7: Tempo de CPU acumulado para otimização (3.3 ns) do circuito *alu2*.

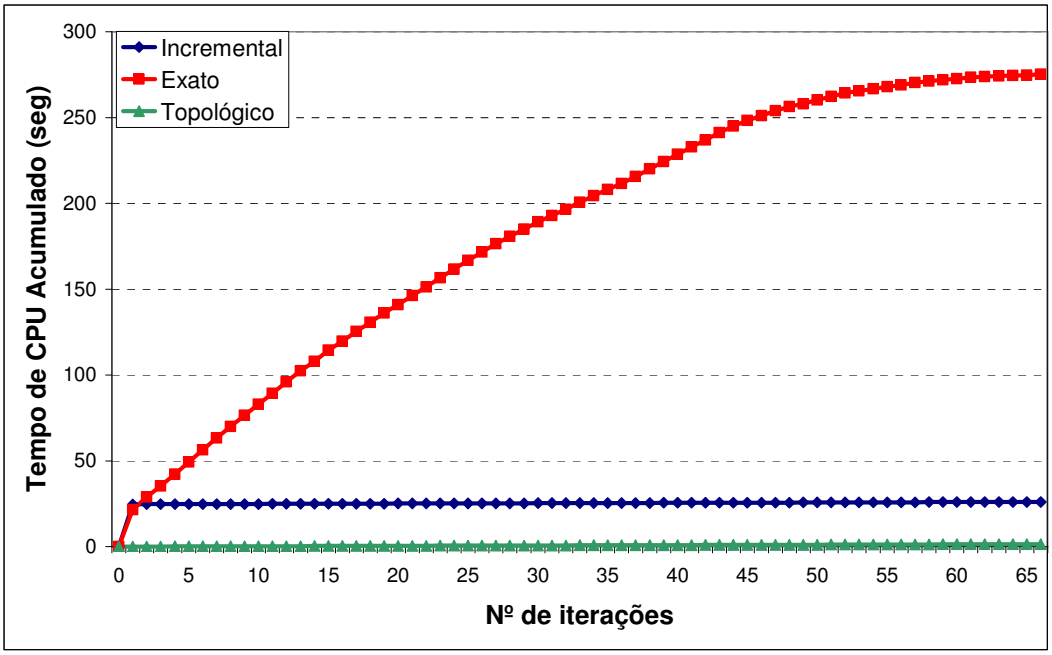

Figura 5.8: Tempo de CPU acumulado para otimização (1.5 ns) do circuito *9sym*.

É possível notar que em alguns casos o acréscimo de área da versão incremental é maior que o acréscimo resultante da versão exata. Mas, na maioria dos casos, este acréscimo de área é muito parecido entre estas duas versões, e fica bem abaixo do acréscimo de área obtido pela versão topológica. A Figura 5.9 e a Figura 5.10 ilustram esta situação, mostrando, respectivamente, os resultados de redução de atraso versus acréscimo de área para os circuitos *alu2* e *9sym*.

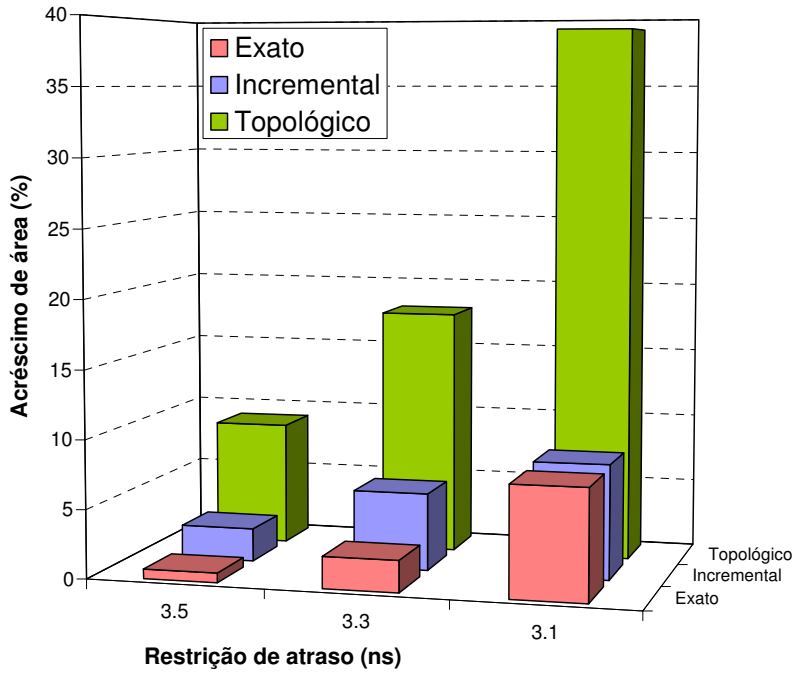

Figura 5.9: Restrições de atraso versus acréscimo de área para o circuito *alu2*.

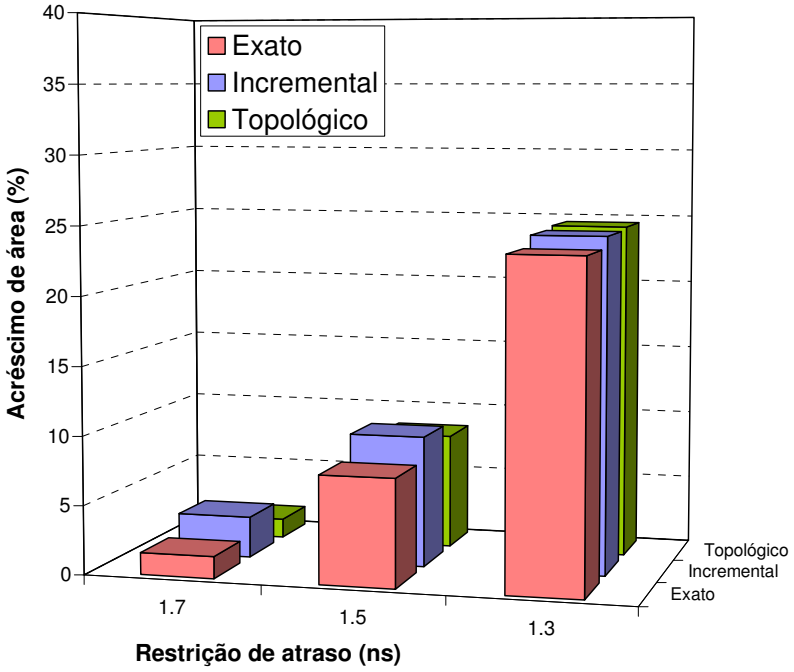

Figura 5.10: Restrições de atraso versus acréscimo de área para o circuito *9sym*.

Sendo assim, é possível verificar que o método incremental proposto permite realizar otimização de atraso utilizando uma abordagem funcional para seleção do caminho crítico em um tempo parecido com o tempo gasto por uma abordagem topológica. Ao mesmo tempo, o acréscimo de área decorrente deste método incremental está perto daquele que resulta ao utilizarmos somente o critério de sensibilização exato para identificar o caminho crítico.

5.3 Dimensionamento seletivo de transistores

Considere o trecho de um caminho crítico que passa pelo pino *a* de uma porta *NAND* de três entradas, como mostrado na Figura 5.11. Considere também que, para otimizar este caminho, queremos reduzir o atraso de subida da porta *NAND*. O uso de um modelo par de atrasos por porta faz com que todos os transistores PMOS da porta *NAND* precisem ser dimensionados. No entanto, como só interessa reduzir o atraso do caminho crítico, bastaria que o transistor conectado ao pino *a* da porta *NAND* fosse aumentado.

Este exemplo mostra que o uso de um modelo par de atrasos leva a um acréscimo de área desnecessário porque os transistores conectados aos pinos *b* e *c* também precisam ser dimensionados para que a redução de atraso seja detectada. Ainda, o aumento destes dois transistores pode fazer com que os caminhos que passam por estes pinos tenham seus atrasos aumentados e precisem ser otimizados em iterações futuras.

No exemplo abaixo, para que seja possível detectar a redução de atraso do caminho crítico apenas com o dimensionamento do transistor conectado ao pino *a*, é necessário ter a informação de um par de atrasos para cada pino da porta. E esta característica é encontrada no modelo computacional de atraso conhecido como modelo pino-a-pino.

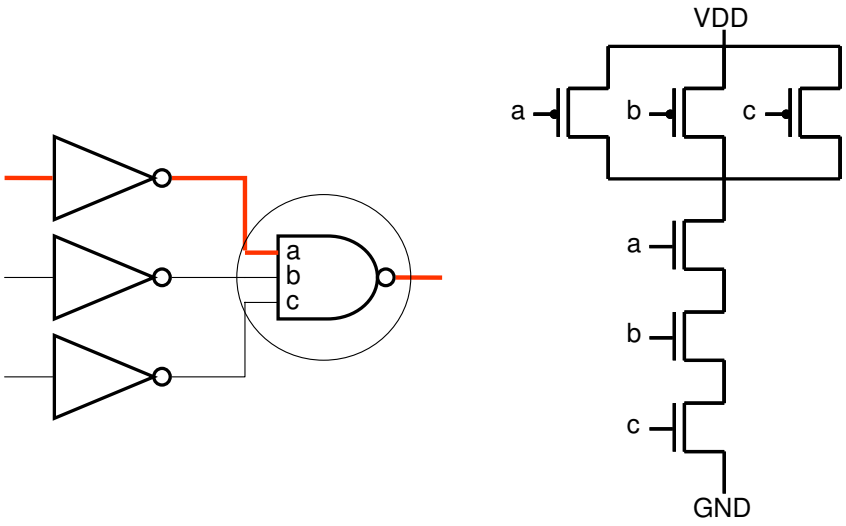

Figura 5.11: Dimensionamento dos transistores em um modelo pino-a-pino.

5.3.1 Modelo de atraso pino-a-pino

É sabido que modelos de atraso pino-a-pino são mais precisos, especialmente quando portas complexas são consideradas, uma vez que nestas portas existem diferentes caminhos de transistores entre a alimentação e a sua saída. E, em um método de dimensionamento de transistores, a precisão na identificação do atraso e do caminho críticos faz com que portas lógicas não sejam dimensionadas desnecessariamente, reduzindo o acréscimo de área.

Além de modificar a estrutura de dados para que o método de verificação e otimização de atraso pudesse suportar um par de atrasos para cada pino da porta, algumas modificações no modelo mostrado na subseção 4.2.2 também precisaram ser realizadas. Verificando a Figura 5.12, podemos notar que agora uma porta pode ter transistores com tamanhos diferentes em uma mesma estrutura, conforme seus diferentes pinos vão sendo otimizados.

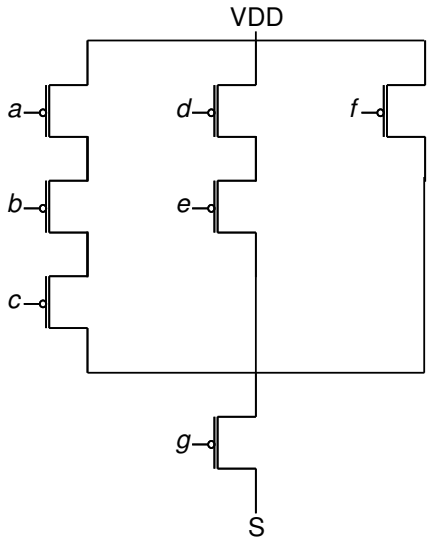

Figura 5.12: Estrutura PMOS de uma porta lógica.

Considere que queiramos identificar o atraso de subida do pino *g*. Suponha que os transistores *a*, *b*, *c* e *g* têm tamanho igual a 2, enquanto que os transistores *d* e *e* têm tamanho igual a 1 e o transistor *f* tem tamanho 3. De acordo com o macromodelo utilizado para computar o atraso da porta, o inversor equivalente de cada cadeia de transistores em série que passa pelo transistor *g* precisa refletir o pior caso de condução de corrente desta cadeia. Para isso, é assumido que o tamanho do transistor do inversor equivalente é igual ao tamanho do menor transistor de cada cadeia de transistores em série. Desta forma, o inversor equivalente da cadeia dos transistores *a-b-c-g* assumiria tamanho 2, dos transistores *d-e-g* tamanho 1 e dos transistores *f-g* tamanho 2.

Temos agora, passando pelo pino *g*, cadeias com diferentes números de transistores em série e com inversores equivalentes de tamanhos diferentes. Para decidir qual destas cadeias de transistores em série representa o pior caso de atraso para o pino *g*, é utilizada a razão entre o tamanho do inversor equivalente e o fator de redução que é dado para o número de transistores contido na cadeia (ver subseção 4.2.2). Considere que os fatores de redução são 4, 3 e 2 para cadeias com 4, 3 e 2 transistores em série, respectivamente. Assim, as cadeias de transistores *a-b-c-g, d-e-g* e *f-g* teriam, respectivamente, razão 0,5, 0,33 e 0,5.

Com isso é possível identificar, para cada pino, a cadeia de transistores em série que apresenta menor capacidade de condução de corrente entre a alimentação e a saída da porta. No exemplo acima, a cadeia de transistores com menor capacidade de corrente para o pino *g* é a cadeia dos transistores *d-e-g*.

5.3.2 Dimensionamento seletivo dos transistores

Com o uso deste modelo pino-a-pino, somente é necessário dimensionar os transistores da pior cadeia de transistores em série que passa pelo pino de entrada da porta que está sobre o caminho crítico, e não mais todos os transistores de uma mesma estrutura (SANTOS, 2005-b). Caso os transistores desta cadeia possuam tamanhos diferentes, somente os transistores com tamanho igual ao tamanho do transistor equivalente precisarão ser dimensionados para que o atraso do pino em questão seja reduzido. Por exemplo, se quiséssemos reduzir o tempo de subida do pino *g* da porta mostrada na Figura 5.12, bastaria dimensionar os transistores *d* e *e*.

5.3.3 Resultados

A Tabela 5.3 e a Tabela 5.4 trazem os resultados de uma comparação entre duas versões do método de otimização. A primeira versão utiliza o modelo par de atrasos e realiza, a cada iteração, o dimensionamento de todos os transistores de uma mesma estrutura da porta selecionada. A outra versão utiliza o modelo de atraso pino-a-pino e realiza o dimensionamento seletivo de transistores.

As mesmas restrições de atraso, mostradas na segunda coluna, foram feitas para cada uma das versões e correspondem à máxima redução de atraso atingida quando do uso do modelo par de atrasos. As duas próximas colunas, rotuladas como *Modelo par de atrasos* e *Modelo pino-a-pino*, mostram o atraso inicial do circuito e o acréscimo de área resultante da otimização. A última coluna mostra o ganho obtido pelo uso de um modelo de atraso pino-a-pino em conjunto com um método de dimensionamento seletivo de transistores, e é igual a razão entre o acréscimo de área resultante da versão par de atrasos e o acréscimo de área resultante da versão pino-a-pino. Os resultados

foram gerados considerando uma tecnologia 0.35μm e cada circuito tem, originalmente, todos os seus transistores do mesmo tamanho (1μm).

A Tabela 5.3 traz os resultados da comparação feita em circuitos que foram mapeados utilizando portas complexas (Tabela A-6.2). Circuitos com portas complexas tendem a tirar mais proveito de um modelo de atraso pino-a-pino e de um método de dimensionamento seletivo de transistores, pois cada porta possui diferentes caminhos de transistores entre a alimentação e a sua saída. Os resultados de ganho mostram que, realizando dimensionamento seletivo de transistores, foi possível atingir as mesmas restrições de atraso com um acréscimo muito menor em área. Ainda, as estimativas de atraso inicial de cada circuito mostram maior precisão do modelo de atraso pino-a-pino.

Tabela 5.3: Resultados para os circuitos mapeados para portas complexas.

Circuito	Restrição de atraso (ns)	Modelo par de atrasos		Modelo pino-a-pino		Ganho em área
		Atraso inicial (ns)	Acréscimo de área (%)	Atraso inicial (ns)	Acréscimo de área (%)	
9sym	1.45	2.53	40.5	2.33	12.5	3.24
alu2	3.05	4.98	143.4	3.74	13.2	10.86
bw	1.60	3.69	132.9	3.30	50.7	2.62
C1355	2.20	3.15	115.4	2.72	30.5	3.78
C1908	3.05	5.07	95.1	4.48	24.1	3.95
C2670	2.35	3.54	55.2	3.31	12.1	4.56
C432	3.50	7.86	102.4	6.36	24.5	4.18
C499	2.30	3.24	102.6	2.86	5.5	18.65
C880	3.00	5.50	56.5	4.86	12.9	4.38
t481	1.65	2.80	59.0	2.64	9.9	5.96

A Tabela 5.4 mostra os resultados obtidos em circuitos que foram mapeados utilizando apenas portas simples de duas entradas (Tabela A-6.1). O uso destes circuitos representa uma situação oposta ao uso dos circuitos mostrados anteriormente, uma vez que existem no máximo dois caminhos de transistores entre a alimentação e a saída de cada porta. Podemos notar que agora as estimativas de atraso inicial dos circuitos são muito parecidas, diferindo apenas pela maior habilidade do modelo pino-a-pino em identificar o tempo de transição das entradas de cada porta (τ_{in}). O ganho em área obtido pelo dimensionamento seletivo, embora menor agora, ainda é considerável e justifica o uso do método proposto em circuitos com qualquer tipo de mapeamento.

Tabela 5.4: Resultados para os circuitos que possuem somente portas simples.

Circuito	Restrição de atraso (ns)	Modelo par de atrasos		Modelo pino-a-pino		Ganho em área
		Atraso inicial (ns)	Acréscimo de área (%)	Atraso inicial (ns)	Acréscimo de área (%)	
9sym	1.25	1.94	67.2	1.93	36.4	1.85
alu2	3.00	4.61	29.3	4.59	19.5	1.50
bw	0.85	1.63	88.0	1.60	61.7	1.43
C1355	2.90	3.72	31.9	3.67	22.3	1.43
C1908	3.30	4.55	22.7	4.42	14.7	1.54
C2670	2.20	3.46	26.9	3.41	16.0	1.68
C432	2.75	3.81	83.4	3.76	50.2	1.66
C499	2.30	3.23	61.4	3.19	42.2	1.45
C880	2.50	3.58	26.2	3.53	6.8	3.85
t481	1.65	2.48	35.9	2.48	25.7	1.40

A Figura 5.13 mostra o comportamento da redução de atraso frente o acréscimo de área para os circuitos da Tabela 5.3.

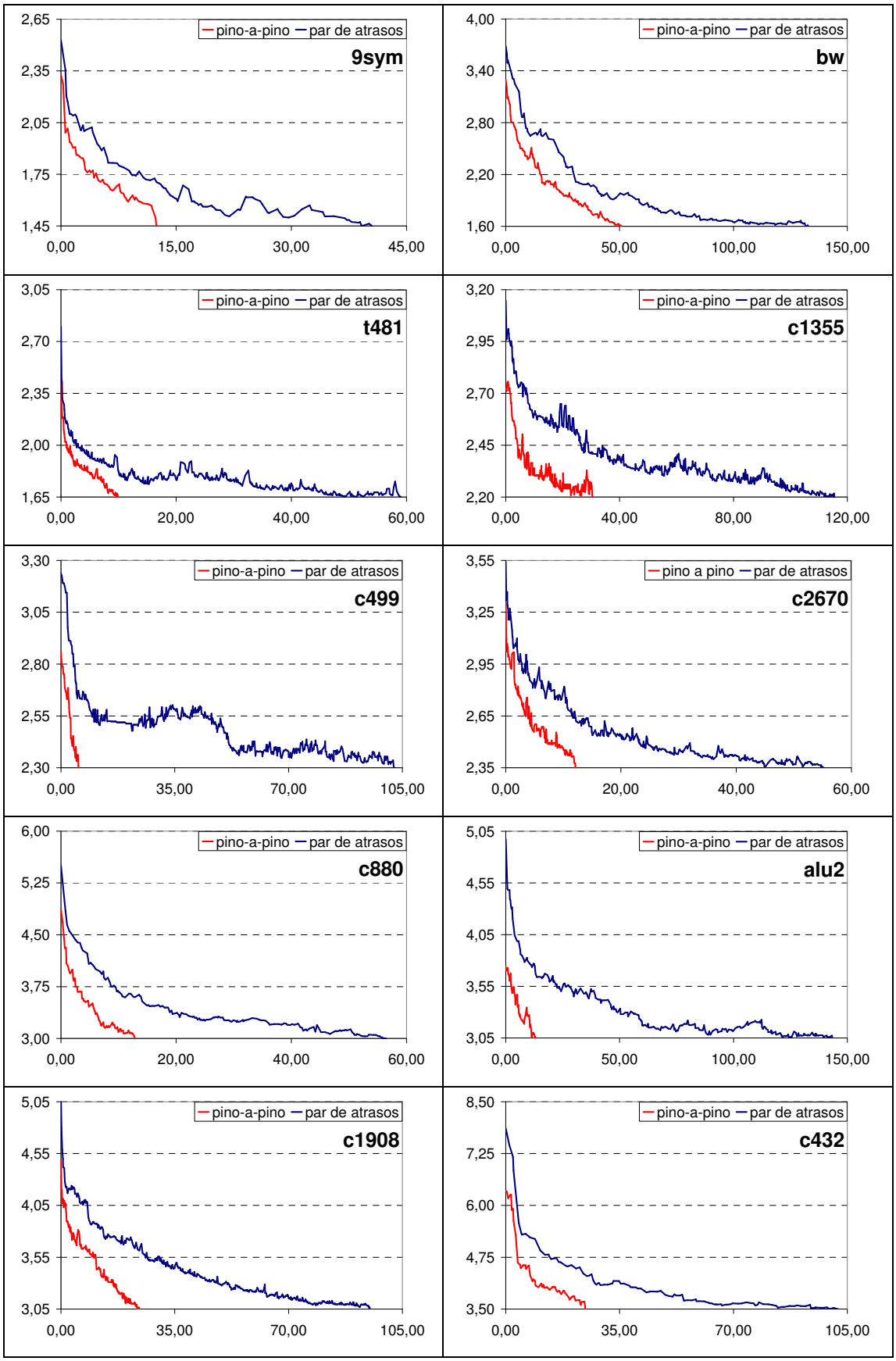

Figura 5.13: Atraso (*ns*) versus acréscimo de área (%) para os circuitos da Tabela 5.3.

Em cada gráfico temos, para os modelos de atraso pino-a-pino e par de atrasos, os valores de atraso (em *n*s) no eixo vertical e os valores de acréscimo de área (em %) no eixo horizontal. O limite inferior do eixo vertical de cada gráfico é igual à restrição de atraso imposta a cada circuito.

Podemos notar que, à medida que a linha que representa o modelo par de atrasos vai se aproximando da restrição imposta, reduzir o atraso do circuito vai se tornando cada vez mais difícil e resulta em acréscimos maiores de área. Isso acontece pelo maior número de caminhos que precisam ser otimizados para uma efetiva redução do atraso do circuito, lembrando que as restrições correspondem à máxima redução obtida pelo uso do modelo par de atrasos.

A linha que representa o modelo pino-a-pino mostra um comportamento menos exponencial, pois foi possível dimensionar um número menor de transistores, a cada iteração, para as mesmas reduções de atraso de cada porta. Ainda, o número de caminhos que precisaram ser otimizados para atingir estas restrições foi menor. A diferença entre os pontos de partida de cada linha, a partir do eixo vertical, demonstra a maior precisão do modelo pino-a-pino. Esta maior precisão nas estimativas faz com que seja necessária uma menor redução de atraso para que as restrições sejam atendidas.

5.4 Validação dos resultados

Para ajudar a validar o método de otimização proposto, os circuitos da Tabela 5.3 serão analisados por uma ferramenta de análise de *timing* comercial. A ferramenta *PathMill* da *Synopsys*® é capaz de realizar TTA a partir de uma descrição do circuito no nível de transistores. Além de fazer uma análise *switch-level*, a ferramenta *PathMill* também utiliza um modelo de atraso pino-a-pino, conseguindo analisar os mesmos circuitos analisados pela ferramenta *TicTac*.

Os circuitos originais e otimizados serão analisados para verificar, através das estimativas da ferramenta *PathMill*, qual foi a redução de atraso efetivamente obtida. As mesmas condições de operação que foram impostas para geração dos resultados da Tabela 5.3 também foram feitas a ferramenta *PathMill*:

- Entradas primárias sendo excitadas por um sinal com tempo de transição (τ_{in}) igual a 0.1*n*s;
- Saídas primárias drenando uma carga (C_{out}) de 8*p*F.

A Tabela 5.5 mostra uma comparação entre os resultados de redução de atraso obtidos pela ferramenta *TicTac* e os valores de redução de atraso verificados pela ferramenta *PathMill*. Os valores de redução de atraso obtidos pela ferramenta *TicTac*, mostrados na segunda coluna da Tabela 5.5, foram extraídos da Tabela 5.3 referente ao resultados do modelo de atraso pino-a-pino. Aqui cabe salientar que, para permitir esta comparação, todos os resultados da subseção 5.3.3 foram gerados através do uso de análise de *timing* topológica (TTA).

Podemos notar que os valores de redução de atraso verificados com o uso da ferramenta comercial *PathMill* são muito parecidos dos valores obtidos pela ferramenta *TicTac*.

Tabela 5.5: Comparação de redução de atraso entre as ferramentas *TicTac* e *PathMill*.

Circuito	Redução de atraso *TicTac* (%)	Redução de atraso *PathMill* (%)
9sym	38.2	41.1
alu2	18.9	18.1
bw	51.5	42.7
C1355	19.3	22.5
C1908	32.1	33.3
C2670	29.1	22.5
C432	45.6	46.2
C499	20.0	20.2
C880	38.7	38.3
t481	37.5	34.3

A ferramenta de geração automática de leiaute *Punch* (LAZZARI, 2004) foi usada para avaliar o impacto do dimensionamento dos transistores na área ocupada pelo leiaute dos circuitos. Foram gerados os leiautes dos circuitos originais e otimizados, mostrados na Tabela 5.3, com o uso do modelo de atraso pino-a-pino e dimensionamento seletivo de transistores. A Tabela 5.6 mostra a comparação entre os resultados de acréscimo de área estimados pela ferramenta *TicTac* e os resultados de acréscimo de área verificados pela geração dos leiautes.

Tabela 5.6: Comparação de acréscimo de área entre as ferramentas *TicTac* e *Punch*.

Circuito	Acréscimo de área *TicTac* (%)	Acréscimo de área *Punch* (%)
9sym	12.5	14.7
alu2	13.2	14.8
bw	50.7	61.6
C1355	30.5	35.3
C1908	24.1	27.8
C2670	12.1	8.5
C432	24.5	25.6
C499	5.5	17.8
C880	12.9	4.2
t481	9.9	10.9

Como dito anteriormente, a métrica de acréscimo de área estimada pela ferramenta *TicTac*, e que foi mostrada ao longo deste capítulo, se refere ao acréscimo da área de *gate* dos transistores do circuito, dada pela equação (3-1). Esta métrica não considera aspectos da geração do leiaute como espaço para roteamento, quebra das bandas de difusão e posicionamento das células. Mesmo assim, é possível verificar que os resultados de acréscimo de área estimados pela ferramenta *TicTac* mostram uma boa correlação com o acréscimo de área efetivo obtido após a geração do leiaute.

5.5 Conclusões

As características apresentadas pelo fluxo de síntese física FUCAS levaram a escolha de um método heurístico de dimensionamento de transistores. Os resultados mostraram que restrições de atraso mais brandas puderam ser atingidas com um pequeno acréscimo de área e que mesmo restrições mais severas foram atendidas, em sua maioria, com um acréscimo de área aceitável. Aqui cabe salientar que o

desempenho do método de otimização atraso é bastante dependente da topologia do circuito.

Os resultados também mostraram a importância da utilização de um método de FTA para identificação do atraso e do caminho críticos, reduzindo drasticamente o número de caminhos e portas lógicas a serem otimizados. Para circuitos com presença elevada de caminhos falsos com atraso superior ao atraso do caminho crítico funcional, houve uma redução efetiva do atraso do circuito com um acréscimo de área bastante inferior ao acréscimo decorrente da utilização de um método de TTA. Em alguns casos, nem seria possível atingir as restrições de atraso se não fossem desconsiderados os caminhos falsos do circuito. O fato de o método realizar dimensionamento discreto permite que os transistores alterados sejam "enquadrados" pela técnica de *folding*, e assim seja possível realizar a geração do leiaute do circuito com menor desperdício de área.

A proposta de um método incremental de seleção do caminho crítico reduziu o tempo de processamento utilizado pelo método de dimensionamento, e, com isso, aumentou o conjunto de circuitos que podem ser otimizados em um tempo razoável. Os resultados obtidos mostram que a aceleração obtida não comprometeu o desempenho do método de dimensionamento, trazendo um acréscimo de área ligeiramente maior do que o obtido pelo método não-incremental. Reduções ainda maiores no tempo de processamento podem ser obtidas se o método de FTA fosse capaz de tratar os circuitos hierarquicamente, permitindo que apenas a porção do circuito que foi alterada fosse re-analisada a cada iteração.

A utilização de um método de dimensionamento seletivo de transistores, associado a um modelo de atraso pino-a-pino, possibilitou que as restrições de atraso fossem atingidas com um acréscimo de área consideravelmente menor, especialmente em circuitos contendo portas complexas. Além disso, o uso de um modelo de atraso pino-a-pino aumentou a precisão das estimativas de atraso.

Comparações feitas entre os valores de redução de atraso obtidos pelas ferramentas *TicTac* e *PathMill* mostraram que as restrições de atraso foram efetivamente atingidas pelo método proposto. A avaliação do impacto do dimensionamento dos transistores na área do leiaute dos circuitos mostrou que o acréscimo de área reportado pela ferramenta *TicTac* é um bom indicativo do impacto da otimização de atraso na área final do circuito.

6 CONCLUSÕES GERAIS

Visando propor um método de otimização de atraso de circuitos combinacionais CMOS, este trabalho expôs e analisou vários aspectos das metodologias de verificação de atraso e também características do fluxo de projeto que podem ser determinantes na composição da freqüência de operação do circuito.

O método de otimização de atraso proposto, baseado em dimensionamento de transistores, foi implementado na ferramenta *TicTac* e integrado ao fluxo FUCAS para permitir que os circuitos fossem gerados visando atingir as especificações de atraso impostas durante o projeto. Este fluxo de síntese física, por sua vez, tira vantagens do uso de portas complexas, onde possui total liberdade de otimização por não estar limitado a uma biblioteca padrão. Restrições foram impostas aos circuitos otimizados de forma a reduzir o impacto do dimensionamento nas etapas de geração de leiaute.

O capítulo 2 mostrou as principais formas de verificação de atraso, com ênfase especial para a análise de *timing*. A diversidade de modelos para caracterizar o atraso de portas lógicas e conexões mostrou um compromisso entre precisão e simplicidade na obtenção dos valores de atraso, que precisa ser realizada sob demanda. Os diferentes métodos e critérios de sensibilização, utilizados para levar em conta a funcionalidade dos circuitos na hora de estimar o seu atraso, mostram o esforço realizado para que as estimativas de atraso não sejam afetadas pela síndrome dos caminhos falsos.

A análise dos métodos de otimização de atraso mais comumente aplicados quando o projeto chega ao nível de leiaute, feita no capítulo 3, junto com a análise das características do fluxo de síntese FUCAS, feita no capítulo 4, foi determinante na definição do método de otimização implementado. Os resultados apresentados no capítulo 5 mostraram que o método de dimensionamento proposto consegue reduzir efetivamente o atraso de circuitos combinacionais e, por considerar características de geração de leiaute, não compromete a qualidade dos circuitos gerados.

O uso de um método de FTA reduziu drasticamente o número de caminhos e portas lógicas que precisaram ser otimizados. Melhorias feitas no método de seleção do caminho crítico reduziram o tempo de processamento gasto para a otimização dos circuitos. A adaptação de um modelo de atraso pino-a-pino e a realização de dimensionamento seletivo dos transistores aumentaram a precisão e o desempenho do método de verificação e otimização de atraso.

A utilização do método de dimensionamento de transistores proposto em diferentes etapas do luxo de síntese física FUCAS trouxe reduções significativas de atraso em um

tempo aceitável e com pequeno acréscimo de área. Mesmo que a área ocupada pelo leiaute do circuito não seja o principal objetivo a ser perseguido, é importante notar que a redução do acréscimo de área acarreta em redução do acréscimo de consumo de potência. Além disso, a redução da área do circuito geralmente resulta em redução do comprimento médio das conexões, as quais têm importância fundamental no atraso e consumo dos circuitos atuais.

Por fim, a integração de um método de verificação de atraso em diferentes etapas do fluxo FUCAS, acompanhado por um conjunto de ações que visam reduzir o atraso do circuito nestas etapas, foi fundamental para aumentar a qualidade e a competitividade deste fluxo de síntese física ainda em desenvolvimento.

REFERÊNCIAS

ALPERT, C. J. et al. Simultaneous driver sizing and buffer insertion using a delay penalty estimation technique. **IEEE Transactions on Computer-Aided Design**, New York, v. 23, n. 1, p. 136-141, 2004.

AUVERGNE, D.; DAGA, J. M. Signal Transition Time Effect on CMOS Delay Evaluation. **IEEE Transactions on Circuits and Systems**, New York, v. 47, n. 9, p. 1362-1369, Sept. 2000.

AZEMARD, N.; AUVERGNE, D. POPS : A tool for delay/power performance optimization. **Journal of Systems Architecture**, Amsterdan, n. 47, p. 375-382, 2001.

BAMJI, C.; BORAH, M. An Improved Cost Heuristic for Transistor Sizing. In: INTERNATIONAL CONFERENCE ON VLSI DESIGN, 11, 1998, India. **Proceedings…** [S.l.: s.n.], 1998.

BASTIAN, F.; LAZZARI, C.; SANTOS, C.; WILKE, G.; GÜNTZEL J. L.; REIS, R. A New Parametrizable Folding Algorithm Applied to an Automatic Layout Generation Tool. In: SIMPÓSIO SUL DE MICROELETRÔNICA, 19, 2004. **Proceedings…** [S.l.: s.n.], 2004. p. 71-74.

BASTIAN, F. B.; LAZZARI, C.; GÜNTZEL, J. L.; REIS, R. A New Transistor Folding Algorithm Applied to an Automatic Full-Custom Layout Generation Tool. In: INTEGRATED CIRCUIT AND SYSTEM DESIGN, POWER AND TIMING MODELING, OPTIMIZATION AND SIMULATION, PATMOS, 14, 2004, Greece. **Proceedings…** [S.l.]: Springer, 2004. p. 732-741.

BENKOSKI, J.; STEWART, R. TATOO: An industrial Timing Analyzer with False Path Elimination and Test Pattern Generation. In: CONFERENCE ON EUROPEAN DESIGN AUTOMATION, 1991, Amsterdam, Netherlands. **Proceedings…** Los Alamitos: IEEE Computer Society, 1991. p.256-260.

BERKELAAR, M. R.; JESS, J. A. Gate Sizing in MOS Digital Circuits with Linear Programming. In: CONFERENCE ON EUROPEAN DESIGN AUTOMATION, 1990, Glasgow, Scotland. **Proceedings…** Los Alamitos: IEEE Computer Society, 1990. p. 217-221.

BISDOUNIS, L. et al. Switching response modeling of the CMOS inverter for sub-micron devices. In: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, 1998, France. **Proceedings…** Washington: IEEE Computer Society, 1998. p. 729-737.

BOYD, S. et al. Geometric Programming and its Applications to EDA Problems. In: DESIGN, AUTOMATION AND TEST IN EUROPE, DATE, 2005. **Tutorial.** Piscataway: IEEE Computer Society, 2005.

BSIM - Official BSIM Site. BSIM3 and BSIM4 Release. Disponível em: http://www-device.eecs.berkeley.edu/~bsim/. Acesso em: junho 2005.

BUSHNELL, M. L.; AGRAWAL, V. D. **Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits**. Boston: Kluwer Academic Publishers, 2000.

CARLSON, B. S.; CHEN, C. Y. R. Performance enhancement of CMOS VLSI circuits by transistor reordering. In: DESIGN AUTOMATION CONFERENCE, DAC, 30, 1993. **Proceedings…** New York: ACM, 1993. p. 361-366.

CHAN, P. K. Algorithms for Library-specific Sizing of Combinational Logic. In: DESIGN AUTOMATION CONFERENCE, DAC, 27, 1990, Orlando, USA. **Proceedings…** New York: ACM, 1990. p. 353-356.

CHEN, H.-C.; DU, D. Path Sensitization in Critical Path Problem. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1991, Santa Clara, USA. **Digest of Technical Papers.** Los Alamitos: IEEE Computer Society Press, 1991. p. 208-211.

CHENG, J.; SALEH, R. Incremental techniques for the identification of statically sensitizable critical paths. In: IEEE DESIGN AUTOMATION CONFERENCE, DAC, 28, 1991, San Francisco, USA. **Proceedings…** New York: ACM, 1991. p. 541-546.

CHEN, H.-C.; DU, D. Path Sensitization in Critical Path Problem. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, New York, v. 12, n. 2, p. 196-207, 1993.

CHEN, C.; CHU, C. C.; WONG, D. F. Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, San Jose, California, 1998. **Proceedings…** [S.l.: s.n.], 1998. p. 617-624.

CHEN, W; HSIEH, C. T; PEDRAM, M. Simultaneous gate sizing and fanout optimization. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 2000, San Jose, USA. **Proceedings…** Piscataway: IEEE Press, 2000. p. 374-378.

CHUANG, W.; SAPATNEKAR, S. S.; HAJJ, I. N. Delay and Area Optimization for Discrete Gate Sizes under Double-Sided Timing Constraints. In: IEEE CUSTOM INTEGRATED CIRCUITS CONFERENCE, San Diego, California, 1993. **Proceedings…** [S.l.: s.n.], 1993. p. 941-944.

CHU, C.; WONG, D. F. Closed form solutions to simultaneous buffer insertion/sizing and wire sizing. **ACM Transactions on Design Automation of Electronic Systems (TODAES)**, New York, v. 6, n. 3, p. 343-371, 2001.

CONN, A. R.; GOULD, N. I. M.; TOINT, P. **LANCELOT:** A Fortran Package for Large-Scale Nonlinear Optimization. [S.l.]: Springer-Verlag, 1992.

CONG, J.et al. Performance Optimization of VLSI Interconnect Layout. **Integration, the VLSI Journal**, Amsterdam, v. 21, n. 1&2, p. 1-94, 1996.

CONN, A. R. et al. JiffyTune: circuit optimization using time-domain sensitivities. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, New York, v. 17, n. 12, p. 1292-1309, 1998.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. **Introduction to Algorithms.** Cambridge: MIT, 1990.

COUDERT, O.; HADDAD, R.; MANNE, S. New algorithms for gate sizing: a comparative study. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, DAC, 33, 1996, Las Vegas, USA. **Proceedings…** New York: ACM, 1996. p. 734-739.

COUDERT, O. Timing and Design Closure in Physical Design Flows. INTERNATIONAL SYMPOSIUM ON QUALITY ELECTRONIC DESIGN, ISQED, 2002. **Proceedings…** [S.l.: s.n.], 2002. p.511–516.

DAGA, J. M.; AUVERGNE, D. Comprehensive Delay Macro Modeling for Submicrometer CMOS Logics. **IEEE Journal of Solid-State Circuits**, New York, v. 34, n. 1, p. 42-55, 1999.

DEVADAS, S.; KEUTZER, K.; MALIK, S. Delay Computation in Combinational Logic Circuits: Theory and Algorithms. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1991. **Proceedings…** [S.l.: s.n.], 1991. p.176-179.

DEVADAS, S. et al. Certified Timing Verification and the Transition Delay of a Logic Circuit. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, DAC, 29, 1992, Anaheim, USA. **Proceedings…** Los Alamitos: IEEE Computer Society, 1992. p.549-555.

DETJENS, E. et al. Technology mapping in MIS. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1987. **Proceedings…** [S.l.: s.n.], 1987. p.116–119.

DUTTA, S; NAG, S; ROY, K. ASAP: A Transistor Sizing Tool for Speed Area and Power Optimization of Static CMOS Circuits. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 1994. **Proceedings…** [S.l.: s.n.], 1994. p. 61-64.

ELMORE, W. C. The Transient Response of Damped Linear Network with Particular Regard to Wideband Amplifiers. **Journal of Applied Physics**, Woodbury, v. 19, n. 1, p. 55-63, 1948.

FERRÃO, D.; WILKE, G.; REIS, R.; GÜNTZEL, J. Improving Critical Path Identification in Functional Timing Analysis. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 16, 2003. **Proceedings…** Los Alamitos: IEEE Computer Society, 2003. p.297-302.

FERRÃO, D.; SANTOS, C. L.; WILKE, G.; GÜNTZEL, J. L.; REIS, R.; LUBASZEWSKI, M. Path Delay Fault Test Generation Using Exact Floating Mode Sensitization. In: IEEE LATIN-AMERICAN TEST WORKSHOP, LATW, 5, 2004, Cartagena, Colombia. **Digest of Papers…** [S.l.: s.n.], 2004. p. 174-177.

FISHBURN, J.; DUNLOP, A. TILOS: A posynomial programming approach to transistor sizing. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1985. **Proceedings…** [S.l.: s.n.], 1985. p. 326-328.

FONSECA, R.A.; SANTOS, C. L.; FERRÃO, D.; REIS, R. Signal Delay in RC Interconnect. In: SIMPÓSIO SUL DE MICROELETRÔNICA, 20, 2005, Santa Cruz do Sul, Brasil. **Proceedings…** [S.l.: s.n.], 2005. p. 101-104.

GEREZ, S. H. **Algorithms for VLSI Design Automation**. [S.l.]: John Wiley and Sons, 1998.

GINNEKEN, L. P. P. P. Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 1990. **Proceedings…** [S.l.: s.n.], 1990. p. 865-868.

GOEL, P. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. **IEEE Transactions on Computers**, Los Alamitos, v. C-30, n. 3, p. 215-222, 1981.

GOPALAKRISHNAN, P.; RUTENBAR, R. A. Direct transistor-level layout for digital blocks. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 2001, San Jose, USA. **Proceedings…** Piscataway: IEEE Press, 2001. p. 577-584.

GÜNTZEL, J. L. **Functional Timing Analysis of VLSI Circuits Containing Complex Gates**. 2000. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

GUPTA, R.; TUTUIANU, B.; PILEGGI, L. T. The Elmore Delay as a Bound for RC Trees with Generalized Input Signals. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, New York, v. 16, n. 1, p. 95-104, 1997.

HASHIMOTO, M.; ONODERA, H.; TAMARU, K. A Power and Delay Optimization Method Using Input Reordering in Cell-Based CMOS Circuits. **IEICE Trans. Fundamentals**, [S.l.], v. E82-A, n. 1, p. 159-166, 1999.

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

HILL, D. et al. **Algorithms and Techniques for VLSI Layout Synthesis.** Boston: Kluwer Academic Publishers, 1989.

HITCHCOCK, R. B. Timing Verification and the Timing Analysis Program. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 1982. **Proceedings…** [S.l.: s.n.], 1982. p. 594-604.

HRAPCENKO, V. Depth and Delay in a Network. **Soviet Math. Dokl**.,[S.l.], v.19, n.4, p.1006-1009, 1978.

JACOBS, E. T. A. F. **Power Dissipation and Timing in CMOS Circuits**. 2001. Tese (Doutorado em Ciência da Computação) — Technische Universiteit Eindhoven, Eindhoven.

JIANG, Y; SAPATNEKAR, S. S; BAMJI, C; KIM, J. Interleaving buffer insertion and transistor sizing into a single optimization. **IEEE Transactions on Very Large Scale Integrated Systems**, New York, v. 6, n. 4, p. 625-633, 1998.

JOUPPI, N. P. Timing Analysis and Performance Improvement of MOS VLSI Designs. **IEEE Transactions on Computer-Aided Design**, Los Alamitos, California, v.CAD-6, n. 4, p.650-665, 1987.

KASHYAP, C. V.; ALPERT, C. J.; DEVGAN, A. An “Effective” Capacitance Based Delay Metric for RC Interconnect. In: IEEE/ACM INTERNATIONAL CONFERENCE

ON COMPUTER-AIDED DESIGN, ICCAD, 2000, San Jose, USA. **Proceedings…** Piscataway: IEEE Press, 2000. p. 229-235.

KETKAR, M.; KASMASETTY, K.; SAPATNEKAR, S.S. Convex Delay Models for Transistor Sizing. In: DESIGN AUTOMATION CONFERENCE, DAC, 37, 2000, Los Angeles, USA. **Proceedings…** New York: ACM, 2000. p. 655-660.

KETKAR, M.; SAPATNEKAR, S. S.; PATRA, P. Convexity-Based Optimization for Power-Delay Tradeoff using Transistor Sizing. In: IEEE/ACM INTERNATIONAL WORKSHOP ON TIMING ISSUES IN THE SPECIFICATION AND SYNTHESIS OF DIGITAL SYSTEMS, 2000. **Proceedings…** [S.l.: s.n.], 2000. p. 52-57.

KIM, N. et al. Leakage Current: Moore's Law Meets Static Power. **IEEE Computer, Special Issue on Power and Temperature-Aware Computing**, [S.l.], v. 36, n. 12, p.68-75, Dec. 2003.

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.

LAZZARI, C.; GÜNTZEL, J. L.; REIS, R. Automatic Full-Custom Layout Generation of Static CMOS Circuits Targeting Delay and Power Reduction. In: IFIP STUDENT FORUM, 2004, Toulouse, França. **Proceedings…** [S.l.: s.n.], 2004. p. 177-186.

MAHESHWARI, N.; SAPATNEKAR, S. S. Gate Size Optimization for Row-based Layouts. In: MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS, 38, 1995. **Proceedings…** [S.l.: s.n.], 1995.

MAURINE, P. et al. Transition Time Modeling in Deep Submicron CMOS. **IEEE Transactions on CAD of Integrated Circuits and Systems**, New York, v. 21, n. 11, p. 1352-1363, Nov. 2002.

MCGEER, P.; BRAYTON, R. Efficient Algorithms for Computing the Longest Viable Path in a Combinational Circuit. In: DESIGN AUTOMATION CONFERENCE, 1989. **Proceedings…** [S.l.: s.n.], 1989. p.561-567.

MCGEER, P.; BRAYTON, R. **Integrating Functional and Temporal Domains in Logic Design**: **The False Path Problem and its Implications**. Norwell, MA: Kluwer Academic Publishers, 1991. 212p.

NAGEL, W. **SPICE2, A Computer Program to Simulate Semiconductor Circuits**. Berkeley, California: University of California, Department of Electrical Engineering and Computer Sciences, 1975. 63p. (UCB/ERL M75/520).

OUSTERHOUT, J. K. A Switch-Level Timing Verifier for Digital MOS VLSI. **IEEE Transactions on Computer-Aided Design**, Los Alamitos, California, v. CAD-4, n. 3, p.336-349, 1985.

PILLAGE, L. T.; ROHRER, R. A. Asymptotic waveform evaluation for timing analysis. **IEEE Transactions on Computer-Aided Design**, Los Alamitos, California, v. 9, n. 4, p. 352–366, 1990.

PILEGGI, L. T. Timing metrics for physical design of deep submicron technologies. In: INTERNATIONAL SYMPOSIUM ON PHYSICAL DESIGN, 1998, Monterey, California, United States. **Proceedings…** [S.l.: s.n.], 1998. p.28-33.

PRASAD, S. C.; ROY, K. Transistor reordering for power minimization under delay constraint. **ACM Transactions on Design Automation of Electronic Systems (TODAES)**, New York, v. 1, n. 2, p. 280-300, 1996.

QIAN, J.; PULLELA, S.; PILLAGE, L. Modeling the "Effective" Capacitance for the RC Interconnect of CMOS Gates. **IEEE Transactions on Computer-Aided Design**, New York, v. 13, n. 12, p. 1526-1535, 1994.

ROY, R.; BHATTACHARYA, D.; BOPPANA, V. Transistor-Level Optimization of Digital Designs with Flex Cells. **IEEE Computer**, [S.l.] v. 38, n. 2, p. 53-61, 2005.

RUBENSTEIN, J.; PENFIELD, P.; HOROWITZ, M. A. Signal Delay in RC Tree Networks. **IEEE Transactions on Computer-Aided Design**, New York, p. 202-211, 1983.

SAKASHITA, K. et al. A 10K-Gate CMOS Gate Array Based on a Gate Isolation Structure. **IEEE Journal of Solid-State Circuits**, New York, v. sc-20, n. 1, p. 413-417, 1985.

SAKURAI, T.; NEWTON, A. R. A simple mosfet model for circuit analysis. **IEEE Transactions on Electron Devices**, New York, v. 38, p. 887-894, Apr. 1991.

SANTOS, C.; WILKE, G.; LAZZARI, C.; GÜNTZEL, J. L.; REIS, R. A Transistor Sizing Method Applied to an Automatic Layout Generation Tool. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 16, 2003. **Proceedings…** Los Alamitos: IEEE Computer Society, 2003. p. 303–307.

SANTOS, C. L.; FERRÃO, D.; GÜNTZEL, J. L.; REIS, R. Incremental Timing Optimization for Automatic Layout Generation. In: IEEE INTERNATIONAL CONFERENCE ON CIRCUIT AND SYSTEMS, ISCAS, 2005, Kobe, Japan. **Proceedings…** New York: IEEE Press, 2005. p. 3567-3570.

SANTOS, C.; FERRÃO, D.; WILKE, G.; LAZZARI, C.; GÜNTZEL, J. L.; REIS, R. Effects of Using a Pin-to-pin Delay Model on a Library-Free Transistor/Gate Sizing Scheme. In: IEEE INTERNATIONAL MIDWEST SYMPOSIUM ON CIRCUIT AND SYSTEMS, MWSCAS, 48, 2005, Cincinnati, USA. **Proceedings…** [S.l.: s.n.], 2005.

SAPATNEKAR, S.; RAO, V. B.; VAIDYA, P. M.; KANG, S. M. An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization. **IEEE Transactions on Computer-Aided Design**, New York, v. 12, p. 1621-1634, 1993.

SAPATNEKAR, S. S.; KANG, S. M. **Design Automation for Timing-driven Layout Synthesis**. Boston: Kluwer Academic Publishers, 1993.

SENTOVICH, E. M. et al. **SIS:** A system for sequential circuit synthesis. Berkeley: Electronics Res. Lab., Univ. California at Berkeley, 1992. (Tech. Rep. UCB/ERL M92/41).

SUNDARARAJAN, V.; SAPATNEKAR, S. S.; PARHI, K. K. Fast and Exact Transistor Sizing Based on Iterative Relaxation. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, New York, v. 21, n. 5, p. 568–581, 2002.

TUTUIANU, B.; DARTU, F.; PILEGGI, L. An explicit RC-circuit delay approximation based on the first three moments of the impulse response. In: IEEE/ACM DESIGN AUTOMATION CONFERENCE, DAC, 33, 1996, Las Vegas, USA. **Proceedings…** New York: ACM, 1996. p. 611-616.

UEBEL, L. F. **Verificação de *Timing* em circuitos VLSI CMOS Digitais**. 1995. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

WESTE, N. H. E.; ESHRAGHIAN, K. **Principles of CMOS VLSI Design:** A Systems Perspective. 2nd ed. [S.l.]: Addison-Wesley, 1993.

WOLF, W. H. Analysis and Synthesis Algorithms. In: WOLF, W. H. **Modern VLSI Design: A Systems Approach**. New Jersey: Prentice-Hall, 1994.

ZAHIRI, B. Structured ASICs: Opportunities and Challenges. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, 2003. **Proceedings…** [S.l.: s.n.], 2003. p. 404.

APÊNDICE DESCRIÇÃO DOS CIRCUITOS UTILIZADOS

A Tabela A-6.1 e a Tabela A-6.2 mostram a descrição dos circuitos combinacionais utilizados na geração dos resultados desta dissertação. A maioria destes circuitos pertence aos conjuntos de *benchmarks* ISCAS'85 e MCNC. Todos os circuitos foram mapeados pela ferramenta SIS (SENTOVICH, 1992). Para os circuitos da Tabela A-6.1, foi permitido apenas o uso de portas CMOS estáticas simples de 2 entradas (INV, NOR e NAND). Alguns destes circuitos foram remapeados, permitindo agora o uso de portas complexas (com limitação de quatro transistores em série em cada estrutura da porta), e estão descritos na Tabela A-6.2.

Tabela A-6.1: Circuitos implementados com portas simples de 2 entradas.

Circuito	Entradas	Saídas	Portas Lógicas	Transistores
9sym	9	1	331	1140
alu2	10	6	704	2406
bw	5	28	279	994
C1355	41	32	709	2454
c17	5	2	13	38
C1908	33	25	686	2266
C2670	157	63	1128	3662
C432	36	7	297	1018
C499	41	32	638	2104
C880	60	26	512	1700
csa2	5	3	31	106
csa4	9	5	62	212
csa8	17	9	124	414
f50	4	1	11	42
t481	16	1	2201	7598

Tabela A-6.2: Circuitos implementados com portas complexas.

Circuito	Entradas	Saídas	Portas Lógicas	Transistores
9sym	9	1	96	698
alu2	10	6	130	818
bw	5	28	75	416
C1355	41	32	296	1376
C1908	33	25	283	1314
C2670	157	63	502	2410
C432	36	7	100	546
C499	41	32	293	1346
C880	60	26	187	1032
t481	16	1	406	2068