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

MARCOS FELIPPE LOPES LEDUR

Dimensionamento de Portas Lógicas através de Programação Geométrica

Trabalho de Diplomação.

Prof. Dr. Marcelo de Oliveira Johann
Orientador

Porto Alegre, dezembro de 2009.

Agradecimentos

Agradeço à minha mãe, que, apesar de algumas dificuldades, nunca deixou faltar nada, compartilhou comigo muitas vitórias e sempre proferiu palavras sábias e acolhedoras nos momentos necessários. Obrigado, mãe!

Meus queridos irmãos, Graziela e Victor, muito obrigado pelo apoio e incentivo durante toda minha caminhada. Sem vocês, tudo seria muito mais difícil.

Meus sinceros agradecimentos ao meu orientador, Marcelo de Oliveira Johann, pela paciência e pelo incentivo à pesquisa.

Agradeço também aos professores Renato Perez Ribas e Ricardo Augusto da Luz Reis, por me proporcionarem oportunidades ímpares dentro e fora desta Universidade.

Também agradeço a todos os professores e funcionários que, durante minha trajetória na UFRGS, facilitaram o acesso ao conhecimento, peça fundamental na elaboração deste trabalho. Acrescento que muitos dos ensinamentos foram fora da sala de aula, e os levarei por toda a vida.

Aos meus caros colegas, sempre prestativos, dedicados e companheiros, meus agradecimentos e votos de sucesso em suas vidas!

Agradeço de modo especial a todos os meus ex-colegas, ex-professores e amigos que fiz na época de Colégio Militar, pois muito do que sou hoje eu devo a eles.

Finalmente, gostaria de agradecer a todos os meus amigos, pois sempre compreenderam minhas ausências, me deram forças nos momentos mais difíceis e compartilharam alegrias sempre que possível.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AIG	*And-inverter Graph*
BLIF	*Berkeley Logic Interchange Format*
CAD	*Computer-Aided Design*
CCS	*Composite Current Source*
CI	Circuito Integrado
DAG	*Directed Acyclic Graph*
ECSM	*Effective Current-Source Models*
EDA	*Eletronic Design Automation*
ISCAS	*International Symposium on Circuit and Systems*
LE	*Logical Effort*
MCNC	*Microelectronics Center of North Carolina*
NLDM	*Non-Linear Delay Models*
PG	Programa Geométrico
PGG	Programa Geométrico Generalizado
RTL	*Register Transfer Level*
SPICE	*Simulated Program with Integrated Circuits Emphasis*
ULA	Unidade Lógica e Aritmética
VLSI	*Very Large-Scale Integration*
YACC	*Yet Another Compiler-Compiler*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O presente trabalho consiste na aplicação dos conceitos de programação geométrica ao dimensionamento de portas lógicas, no contexto da concepção de circuitos integrados digitais. A programação geométrica é um modelo matemático que pode constituir um método de otimização simples e eficiente, onde a melhor solução global é sempre encontrada, caso o problema seja solucionável. Este aspecto é interessante, pois geralmente os métodos existentes utilizam heurísticas, ou trabalham sobre um conjunto reduzido de possibilidades. Como consequência, nem sempre a solução encontrada é ótima. Além disso, a programação geométrica consegue resolver maiores instâncias de problemas sem muitas restrições computacionais, e há atualmente um grande esforço no desenvolvimento de métodos numéricos para tornar a resolução dos problemas ainda mais rápida, prática e confiável.

Na parte inicial do trabalho, serão apresentados os conceitos referentes à síntese lógica de circuitos, programação geométrica e modelos de atraso em portas lógicas. Em um momento posterior, será realizada a correta modelagem do problema de dimensionamento, além da especificação de uma ferramenta para solucionar o problema. A seguir, o fluxo do projeto e todos os detalhes de implementação de cada módulo desta ferramenta serão apresentados. Alguns circuitos de teste servirão de entrada ao fluxo do projeto na próxima fase, e os resultados obtidos serão comentados e comparados com resultados de outras ferramentas existentes. Finalmente, será feita uma breve conclusão sobre os resultados obtidos, bem como uma análise da possibilidade de melhorias e trabalhos futuros.

Palavras-Chave: circuitos integrados, dimensionamento de portas lógicas, programação geométrica.

Logic Gate Sizing through Geometric Programming

ABSTRACT

The present work involves the application of geometric programming concepts to the logic gate sizing, in the context of digital integrated circuits design. Geometric programming is a mathematical model that can be a simple and efficient optimization method, where the best global solution is always found, if the problem is feasible. This is interesting because usually the existing methods use heuristics, or work on a limited set of possibilities. As a consequence, not always the solution found is optimal. In addition, geometric programming can solve larger instances of problems without a lot of computational restrictions, and there is currently a major effort in the development of numerical methods to solve the problems in even faster, more practical and more reliable ways.

At the beginning, the concepts concerning logic synthesis of circuits, geometric programming and delay models in logic gates will be presented. At a later moment, a correct modeling of the sizing problem will be held, besides the specification of a tool to solve the problem. Then, the design flow and all the implementation details of each module of this tool will be presented. Some test circuits will serve as input to the flow of the project in the next phase, and the results will be discussed and compared with results of other existing tools. Finally, there will be a brief conclusion on the results, as well as an analysis of possible improvements and future work.

Keywords: integrated circuits, gate sizing, geometric programming.

1 INTRODUÇÃO

A complexidade dos circuitos integrados digitais (CIs) tem sido aumentada exponencialmente desde os anos 60, sendo que o número de componentes em um único circuito dobra a cada 18 meses. Atualmente, alguns CIs possuem quase um bilhão de dispositivos, e uma quantidade de interconexões da mesma ordem. Tais circuitos dependem fortemente de potentes ferramentas de automação de processo (EDA) e concepção (CAD) de circuitos eletrônicos digitais, em vista da complexidade no desenvolvimento de tais dispositivos.

Existem diversas fases entre a especificação de um sistema e a obtenção do produto final. Dentre elas, a síntese é de grande importância. A síntese lógica é o processo de conversão de uma descrição básica de um circuito (por exemplo, nível de transferência entre registradores, ou *RTL – Register Transfer Level*) em uma representação otimizada ao nível de portas lógicas, utilizando-se uma biblioteca de células, e de acordo com algumas restrições. Já a síntese física, a próxima etapa, consiste em realizar o posicionamento e conexão das diversas células, sem afetar as restrições previamente especificadas e garantindo a funcionalidade do circuito. Neste trabalho, alguns aspectos da síntese lógica serão analisados e algumas ideias para melhorias do processo serão apresentadas.

1.1 Contexto

Na etapa de síntese lógica, além de mapear o circuito para uma biblioteca e uma tecnologia conhecida, o projetista deve inserir algumas restrições, como por exemplo, área, potência ou temporização. A ferramenta deve preservar estas restrições, e ao mesmo tempo, tentar otimizar o circuito para um determinado aspecto. Esta tarefa não é simples. Simulações elétricas são extremamente confiáveis, mas na prática não são utilizadas diretamente, pois a complexidade dos circuitos proíbe esta abordagem. Em vista disso, foram concebidos modelos que representam o comportamento de transistores, portas lógicas e blocos lógicos, para finalmente a otimização poder ser realizada.

Os algoritmos utilizados nas ferramentas de otimização dependem das restrições e do objetivo do projetista. Por exemplo, caso opte-se por construir um circuito com o menor atraso entre as entradas e as saídas, podem ser escolhidas as células da biblioteca que melhor satisfazem essa condição, mas deve ser levado em conta se alguma restrição de potência ou de área foi extrapolada. O algoritmo deve, portanto, levar todos estes aspectos em conta antes de fornecer uma resposta ao projetista.

Algoritmos para a otimização lógica de circuitos são uma grande fonte de estudos atualmente, e sempre há novidades nesse campo nas várias conferências onde são apresentados. O presente trabalho trata de uma solução para o problema de

dimensionamento de portas lógicas. Por exemplo, dentre os algoritmos propostos, há os que utilizam modelos matemáticos e físicos, análise de grafos, heurísticas, métodos gulosos e algoritmos genéticos (COUDERT e HADDAD, 1996).

Em Matemática, um *problema de otimização*, de um modo geral, consiste em escolher um elemento de um conjunto de alternativas, a partir de uma função-objetivo. O problema de dimensionamento de portas lógicas pode ser modelado como um problema de otimização, pois dentre as escolhas para o tamanho de cada porta lógica, há algumas que se aproximam do ótimo, sendo a função-objetivo a minimização do atraso ou da potência, por exemplo. O problema de dimensionamento pode ser resolvido utilizando-se *programação geométrica*, uma subclasse de resolução de problemas de otimização. Dentre outros métodos de resolução, tem-se, por exemplo, a programação linear, quadrática e convexa. Muitos problemas nesta área foram resolvidos com a utilização destes métodos.

1.2 Motivação

O que torna a programação geométrica interessante para a resolução de problemas em diversas áreas de conhecimento é a eficiência e a confiabilidade na obtenção de soluções para esses problemas práticos. Embora a modelagem não seja trivial, a programação geométrica é capaz de produzir soluções muito boas, em comparação com outros tipos de abordagem. Em problemas de otimização de circuitos digitais, por exemplo, diversas soluções são obtidas a partir de heurísticas, pois não é possível obter uma solução ótima, em virtude de não haver poder computacional para realizar a busca por esta solução. A abordagem da programação geométrica, no entanto, consegue obter uma solução próxima do ótimo. Ademais, quando temos um modelo corretamente formulado, instâncias em maior escala do mesmo problema poderão ser resolvidas, sem que haja grandes limitações computacionais.

1.3 Objetivos e Resultados Esperados

O trabalho aqui desenvolvido busca aplicar os conceitos de programação geométrica ao dimensionamento de portas lógicas. Para tanto, buscou-se realizar a modelagem do problema de otimização, e utilizar ferramentas existentes para resolver o problema e apresentar a solução.

Espera-se que, no decorrer do trabalho, os conceitos sejam bem compreendidos e aplicados ao desenvolvimento de uma ferramenta para realizar o dimensionamento de portas lógicas, a partir de circuitos e restrições, fornecidos como entrada. O resultado obtido deve ser avaliado e comparado com o resultado de ferramentas similares.

1.4 Organização do Trabalho

O trabalho está organizado da seguinte maneira: Na Seção 2, serão introduzidos os conceitos utilizados na concepção do projeto, como por exemplo, o dimensionamento de portas lógicas, modelos de atraso e programação geométrica. Na Seção 3, o projeto será especificado, onde se espera descrever as entradas e saídas e esboçar os módulos da ferramenta. Já na Seção 4, serão apresentados detalhes da implementação, em todos os estágios. Por fim, na Seção 5, o projeto será testado e avaliado, e será elaborada uma breve conclusão sobre os resultados obtidos e melhorias que podem ser feitas em trabalhos futuros.

2 BASES TEÓRICAS

Nesta Seção, são apresentados alguns conceitos presentes na literatura, os quais serão abordados na elaboração do trabalho. Dentre esses conceitos, os mais importantes são o dimensionamento de portas lógicas, modelos de atraso e programação geométrica. Além disso, serão citados alguns projetos de pesquisa semelhantes.

2.1 Dimensionamento de Portas Lógicas

O dimensionamento de portas lógicas consiste em escolher, para cada nodo de uma rede mapeada, uma implementação de porta lógica presente em uma biblioteca, de maneira que uma função de custo é otimizada, de acordo com algumas restrições. A escolha de certa porta lógica tem influências diretas na área total do circuito, no atraso e na dissipação de potência. A seguir, um circuito será apresentado, e serão feitas algumas considerações sobre como as funções de custo podem ser modeladas.

2.1.1 Topologia do Circuito

Considera-se um circuito combinacional composto por portas lógicas ligadas por fios de interconexão. Por simplicidade, é assumido que cada porta lógica possui uma saída, e uma ou mais entradas. Cada saída é conectada à entrada de outra porta lógica, ou é uma saída primária do circuito, enquanto que cada entrada é conectada à saída de outra porta lógica, ou é uma entrada primária do circuito. Também é assumido que o circuito é acíclico, ou seja, não há caminhos que levam a mesma porta lógica ao percorrer tal circuito. Pode-se, então, representar o circuito por um *grafo acíclico direto* (*DAG – Directed Acyclic Graph*), onde cada nodo representa uma porta lógica, e cada arco representa a interconexão, ou ainda, representá-lo por um *netlist.* Na Figura 2.1, temos um exemplo de um circuito com três entradas (*A*, *B* e *C*), uma saída (*D*), e oito portas lógicas.

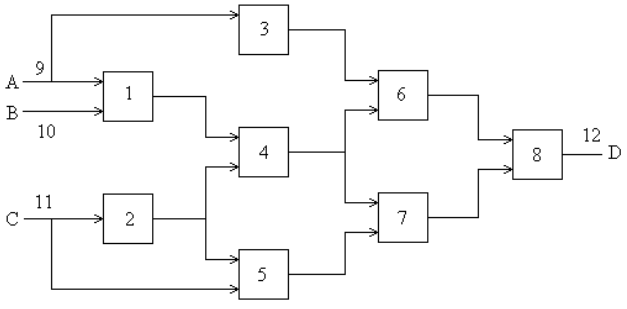


Figura 2.1: Circuito combinatório com oito portas lógicas

O DAG pode ser representado por uma estrutura de dados. Nesta estrutura, considera-se que as portas lógicas são indicadas com os índices 1…*n*, e as entradas e

saídas primárias são contadas a partir de $n + 1$. O *fan-in* da saída primária ou porta lógica, denominado FI(i), é o conjunto de predecessores de i no DAG, ou seja, portas lógicas ou entradas que ativam i. Já o *fan-out* da entrada primária ou porta lógica, denominado FO(i), é o conjunto de sucessores de i no DAG, ou seja, portas lógicas ou saídas primárias que são ativadas por i. Na Figura 2.1 temos, por exemplo

$$FI(1) = \{9,10\}, \qquad FI(5) = \{2,11\}, \qquad FO(4) = \{6,7\}, \qquad FO(8) = \emptyset.$$

2.1.2 Fator de Dimensionamento

Para cada porta lógica i, associamos um *fator de dimensionamento* x_i, que dimensiona os componentes de cada porta lógica. Por exemplo, se $x_5 = 1$ e $x_7 = 8$, temos que a porta lógica 5 possui o tamanho mínimo permitido, enquanto que a porta lógica 7 possui componentes[1] com o comprimento 8 vezes superior ao mínimo.

2.1.3 Atraso em uma Porta Lógica

Cada porta lógica possui um *atraso*, o tempo necessário para a mesma realizar a transição no valor da saída, ficando estável, a partir do momento em que a entrada é alterada e torna-se estável. Para essa transição, assumem-se várias definições, como por exemplo, o tempo entre a entrada estar a 50% entre o valor mínimo e máximo da tensão de alimentação, e o tempo para que ocorra o mesmo na saída da porta (*tempo de propagação*).

Dentre os grandes desafios da microeletrônica, está a obtenção de um modelo confiável de atraso para as portas lógicas, pois há muitas variáveis que definem este atraso. Por exemplo, podem ser citadas a tecnologia utilizada, o projeto da célula (acarretam alterações nas capacitâncias, resistências e indutâncias presentes), a temperatura de operação, a tensão de operação, a presença de ruído no sinal, e variantes na fabricação. Além disso, qualquer alteração na forma de onda da entrada ou mudança da capacitância de carga na saída modificam o valor do atraso. Na prática, isso implica que, após a fabricação do CI, o valor do atraso não é um valor fixo, pois depende de onde a porta lógica estiver no circuito.

A seguir serão apresentados alguns modelos para a obtenção do atraso em uma porta lógica:

- Simulações *SPICE* (NAGEL, 1973): Simulações elétricas em nível de transistor fornecem a maneira mais precisa de cálculo do atraso, mas são impraticáveis para um circuito complexo.
- *Logical Effort* (*LE*): *Esforço Lógico* é um método de obtenção de atraso em portas lógicas, desenvolvido por Ivan Sutherland e Robert Sproull em 1991. O atraso em uma porta lógica é baseado na expressão para um *atraso normalizado*, um valor adimensional. Para obter o atraso real, basta multiplicar o atraso normalizado por uma constante temporal, que é dependente da tecnologia empregada. A partir da expressão do atraso normalizado para um inversor, é possível estimar o atraso normalizado para outras portas lógicas, sendo esta a grande vantagem do método (SUTHERLAND, SPROULL e HARRIS, 1999).
- Rampas de tensão e capacitâncias de carga podem ser variadas, produzindo

[1] Os componentes citados podem ser, por exemplo, a área de difusão, ou apenas a largura de canal. Por isso, não é assumido que a área total aumentará na mesma proporção.

uma tabela bidimensional com o valor do atraso. O modelo NLDM (*Non-Linear Delay Models*) baseisa-se exatamente nessa aproximação, enquanto outros modelos, como CCS (*Composite Current Source*) e ECSM (*Effective Current-Source Models*) consideram formas de onda de corrente na capacitância de carga. Nesses dois casos, produzem-se tabelas com os tempos e subida e descida[2] para a saída, aproximando-se a forma de onda na carga por uma rampa, a qual servirá de entrada à próxima porta lógica. O formato de biblioteca *Liberty* suporta a construção de tabelas para todos esses modelos.

- *Elmore Delay Model*: Nesse modelo, o circuito é visto como uma rede de capacitâncias e resistências. O atraso entre a entrada e um nodo qualquer da rede é aproximado por um somatório entre o produto de algumas capacitâncias e resistências presentes entre os dois pontos (ELMORE, 1948).

2.1.3.1 Modelo RC: Elmore Delay e Logical Effort

A expressão para o valor do atraso pode ser obtida através da aproximação do circuito por um modelo RC simples conforme a Figura 2.2. Embora essa aproximação trata de um circuito onde o modelo de atraso Elmore possa ser utilizado, é importante ressaltar que as relações entre tensão e corrente em transistores não são lineares, tampouco os transistores são simples chaves (WESTE e HARRIS, 2003). Isso significa, na prática, que não existe uma resistência de valor fixo ao modelar uma porta lógica. Esse problema será contornado adiante, onde o método de *Logical Effort* fornecerá uma aproximação para essa resistência.

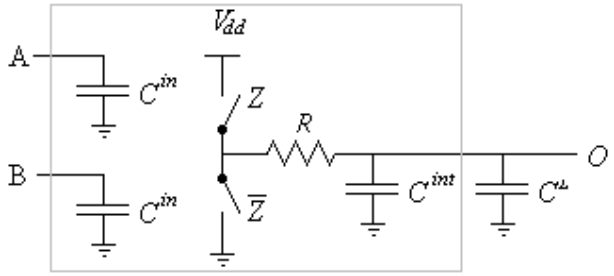


Figura 2.2: Modelo RC de uma porta lógica

Observa-se que a porta lógica possui uma *capacitância de entrada*, C^{in}, a qual representa a carga para a porta que aciona a presente porta lógica (ou representa a carga para uma entrada). Outra capacitância presente é a *capacitância intrínseca* C^{int}. Ambas as capacitâncias são proporcionais ao fator de dimensionamento x_i, ou seja

$$C_i^{in} = \bar{C}_i^{in} x_i \,, \qquad C_i^{int} = \bar{C}_i^{int} x_i \,,$$

onde $\bar{C}_i^{in}$ e $\bar{C}_i^{int}$ são as capacitâncias para a porta lógica i com dimensionamento unitário. A capacitância de carga C_i^L, em uma porta ou entrada primária i, é a soma das capacitâncias de entrada das portas ou saídas primárias que i ativa, ou seja

$$C_i^L = \sum_{j \in \mathrm{FO}(i)} C_j^{in}$$

[2] Tipicamente, tempo de subida e tempo de descida são os intervalos de tempo para uma alteração no sinal de saída, quando ela passa de 10% para 90% do valor da tensão nominal (tempo de subida), ou de 90% para 10% (tempo de descida).

A *resistência de ativação*, ou *resistência efetiva* R_i, em uma porta lógica *i*, é inversamente proporcional ao fator de dimensionamento,

$$R_i = \bar{R}_i / x_i \ ,$$

onde $\bar{R}_i$ é é a resistência de ativação na porta *i* com dimensionamento unitário.

Com esse modelo, podemos aproximar o atraso na porta *i*, D_i, como sendo

$$D_i = 0.69 R_i (C_i^{int} + C_i^L)$$

que é o tempo necessário para um circuito RC atingir o ponto médio entre as tensões que representam os níveis lógicos 0 e 1 (0 e V_{dd}, respectivamente), considerando-se o modelo de atraso Elmore.

É possível aproximar o valor de R_i com a obtenção do mesmo D_i a partir do modelo de esforço lógico. A expressão para o atraso de uma porta lógica, nesse modelo, é

$$D_i = d_i \tau$$

onde d_i é o atraso normalizado para a porta *i*, e τ é o valor do atraso de um inversor sem capacitâncias parasitas[3]. O atraso d_i possui dois componentes: o *atraso parasita* ou *intrínseco, p*, que é o atraso intrínseco da porta lógica e não depende da carga, e o *esforço da etapa*, *f*, que depende da carga. Assim, temos

$$d_i = f_i + p_i$$

Já o esforço da etapa é o produto entre o *esforço lógico*, g_i, que é a razão entre a capacitância de entrada da porta *i* e a capacitância de entrada de um inversor que consegue obter a mesma corrente de saída e o *esforço elétrico*, h_i, a razão entre a capacitância de entrada da carga pela capacitância de entrada da porta *i*. Dessa forma, a equação torna-se

$$d_i = g_i h_i + p_i \ .$$

Com essa fórmula, podemos calcular qualquer o atraso em qualquer porta, a partir dos dados de um inversor. Por definição, em um inversor que ativa apenas outro inversor semelhante, o esforço lógico, o esforço elétrico e o atraso parasita são iguais a um. Pela fórmula acima, o atraso normalizado de um inversor, nessas condições, é igual a 2 (dois).

A tabela 2.1 mostra o valor do esforço lógico para algumas portas lógicas. Nota-se que no caso das portas ou-exclusivas, o LE depende da entrada. Há ainda, na literatura, tabelas semelhantes para outras células, bem como fórmulas para atraso parasita de diversas portas. Com estes valores, obtém-se facilmente o atraso para qualquer porta lógica. O Apêndice A mostra o valor do esforço lógico e do atraso parasita para algumas portas lógicas.

A partir das fórmulas de atraso utilizando-se o modelo RC e o modelo de esforço lógico, obtém-se uma fórmula que aproxima a resistência intrínseca R_i:

$$R_i = \frac{\tau}{0.69} \frac{(g_i h_i + p_i)}{(C_i^{in} + C_i^L)}$$

[3] Em processos de 45nm, por exemplo, τ equivale a aproximadamente 5ps.

Tabela 2.1: Esforço Lógico para algumas células

Porta	*Número de Entradas*			
	2	*3*	*4*	*N*
NAND	4/3	5/3	6/3	(n+2)/3
NOR	5/3	7/3	9/3	(2n+1)/3
XOR, NOR	4,4	6,8,6	8, 16, 16, 8	---

Fonte: WESTE e HARRIS, 2003.

2.1.4 Atraso do Circuito e Área

Um *caminho* no circuito é uma sequência de portas lógicas conectadas, que ligam uma entrada primária a uma saída primária. O atraso do caminho é a soma dos atrasos em cada porta. O atraso do circuito, *D*, também chamado de 'pior caso', ou 'atraso do caminho mais longo', é o máximo atraso em qualquer caminho do circuito. No circuito da Figura 2.1, temos que a expressão para o atraso é

$$D = \max\{D_1+D_4+D_6+D_8,\ D_1+D_4+D_7+D_8,\ D_2+D_4+D_6+D_8,$$
$$D_2+D_4+D_7+D_8,\ D_2+D_5+D_7+D_8,\ D_3+D_6+D_8,\ D_5+D_7+D_8\}$$

A área do circuito pode ser aproximada, de uma maneira simplificada, pela soma das áreas de cada porta lógica, sendo que a área da porta *i* é diretamente proporcional ao fator de dimensionamento x_i. Dessa forma, obtemos a seguinte expressão

$$A = \sum_{i=1}^{n} x_i \bar{A}_i$$

onde $\bar{A}_i$ é a área do circuito com dimensionamento unitário.

2.2 Programação geométrica

Programação geométrica é um problema matemático de otimização (DUFFIN, PETERSON e ZENER, 1967).

Desde a sua formulação, a programação geométrica foi associada a diversos problemas de engenharia, tais como fluxo em redes não lineares, problemas de posicionamento ótimo e problemas de equilíbrio químico. Em 1985, a Programação geométrica, foi utilizada, pela primeira vez, para dimensionamento no nível de transistor e interconexão (FISHBURN, 1985). Desde então, vários problemas na área de projeto de circuitos integrados digitais foram modelados através de programação geométrica. Como exemplos, tem-se a aplicação em dimensionamento de porta em transistores, dimensionamento de interconexão, dimensionamento simultâneo de porta e interconexão, além de problemas gerais de dimensionamento, posicionamento, roteamento, projeto com mais de uma fonte de alimentação, maximização de *yield*, otimização de potência, dentre outros.

É importante salientar que a utilização crescente da programação geométrica em diversos campos de conhecimento ocorreu em vista da evolução na obtenção de suas soluções. Inicialmente, Duffin, Peterson e Zener utilizavam-se da abordagem analítica, sendo possível a obtenção de soluções apenas para pequenos problemas. Métodos numéricos para a obtenção de soluções foram concebidos na década de 70. Entretanto,

métodos computacionais mais robustos, como por exemplo o *método do ponto interior para programação geométrica*, foram desenvolvidos a partir de 1994 (NESTEROV, 1994). Em 2002, implementações de pacotes de software foram disponibilizadas, sendo que existem interfaces destes softwares disponíveis para *Mathworks MATLAB*. Outros softwares foram desenvolvidos para realizar a tarefa de *parser*, reconhecendo os problemas, modelando-os e automaticamente resolvendo os problemas de otimização.

Embora a elaboração de software para a resolução dos problemas de Programação geométrica seja trabalho para especialistas na área, espera-se um rápido progresso na rapidez da obtenção de soluções, bem como interfaces amigáveis para especificação e depuração.

A seguir, alguns conceitos e definições sobre programação geométrica serão apresentados.

2.2.1 Monômios e Posinômios

Sejam $x_1,\ldots, x_n$ a representação de n variáveis reais positivas, e x = $(x_1,\ldots, x_n)$ um vetor com componentes x_i. Uma função real $f(x)$, da forma

$$f(x) = cx_1^{a_1}x_2^{a_2}\cdots x_n^{a_n}$$

onde $c > 0$ e $a_i \in$, é chamada de *função monomial*, ou simplesmente *monômio*. A constante c é chamada de *coeficiente* do monômio, e as constantes $a_1,\ldots, a_n$ são os *expoentes* do monômio.

A soma de um ou mais monômios, ou seja, uma função da forma

$$f(x) = \sum_{k=1}^{K} c_k x_1^{a_{1k}}x_2^{a_{2k}}\cdots x_n^{a_{nk}}$$

onde $c_k > 0$, é chamada de *função posinomial*, ou apenas *posinômio* (com K termos, nas variáveis $x_1,\ldots, x_n$).

Todo monômio é um posinômio. Monômios e posinômios são fechados sobre adição, multiplicação, e multiplicação por escalar positivo. Posinômios podem ser divididos por monômios, sendo o resultado outro posinômio.

Como exemplos, temos que as funções $5x$, 7.21, e $2xyz^{-2}$ são monômios (por consequência, posinômios). As funções $1.5 + 2xy^2$, $4(x+2y)^3$, e $x+y+3/z$ são posinômios. Já as funções -2, $(2x+3y)^{1.5}$, $x-y$ e $3+\log(x)$ não são nem monômios, nem posinômios.

2.2.2 Programa Geométrico – Forma Padrão

Um *programa geométrico* (PG) é um problema de otimização da forma

$$\begin{aligned} \text{minimizar} \quad & f_0(x) \\ \text{sujeito a} \quad & f_i(x) \le 1, \quad i = 1,\ldots, m, \\ & g_i(x) = 1, \quad i = 1,\ldots, p, \end{aligned}$$

onde f_i são funções posinomiais, chamadas de *funções-objetivo*, g_i são monômios, também conhecidos como *restrições*, e x_i são as variáveis para as quais deseja-se a otimização. A forma descrita acima é a *forma padrão*, onde a função-objetivo é um posinômio minimizado, a restrição de igualdade é um monômio igual a um, e a restrição de desigualdade é um posinômio menor ou igual a um. Outras formas de representação de um PG serão apresentadas adiante.

2.2.3 Exemplo de um PG na Forma Padrão

A seguir, será apresentado um exemplo de um programa geométrico, que será modelado e apresentado na forma padrão. O exemplo baseia-se no trabalho de Boyd (2007).

Deseja-se construir uma caixa, sem tampa, com altura *a*, comprimento *b* e largura *c*. Tem-se que os limites na área são 2(*ab*+*ac*), nas paredes, e no fundo, o limite é *bc* . As razões *a/b* e *c/b* também possuem limites inferiores e superiores. Sendo estas as restrições, deseja-se maximizar o volume, ou seja, *abc*. A formulação do problema, desse modo, é a seguinte:

$$\begin{aligned} \text{maximizar} \quad & abc \\ \text{sujeito a} \quad & 2(ab+ac) \leq A_{parede}, \qquad bc \leq A_{fundo}, \\ & \alpha \leq a/b \leq \beta, \qquad \gamma \leq c/b \leq \delta \end{aligned}$$

A partir da correta descrição do problema, e com algumas manipulações matemáticas, podemos reescrever o mesmo problema para a forma padrão de um PG:

$$\begin{aligned} \text{minimizar} \quad & a^{-1}b^{-1}c^{-1} \\ \text{sujeito a} \quad & (2/A_{parede})ab + (2/A_{parede})ac \leq 1 \quad (1/A_{fundo})bc \leq 1 \\ & \alpha a^{-1}b \leq 1, \quad (1/\beta)ab^{-1} \leq 1, \quad \gamma bc^{-1} \leq 1, \quad (1/\delta)b^{-1}c \leq 1 \end{aligned}$$

2.2.4 Resolução de PGs

Como foi citado na introdução do trabalho, a principal motivação da modelagem em PG é a eficiência com que a otimização pode ser feita. Como um simples exemplo dos avanços pelo qual os métodos de resolução sofreram nos últimos anos, algoritmos de ponto interior podem solucionar um PG com milhares de variáveis e dezenas de milhares de restrições em poucos minutos (BOYD e KIM, 2005).

Outro aspecto é a confiabilidade da obtenção das soluções. A melhor solução (global) sempre é encontrada, e caso seja impossível de encontrar tal solução, é também indicado que o problema não é fazível.

Para resolver os problemas, várias abordagens podem ser feitas. O método do ponto interior, citado na subseção anterior, foi desenvolvido para a solução em *otimização convexa*, onde posinômios de um PG na forma padrão são transformados em uma sequência de equações lineares, e o princípio da convexidade é utilizado, tornando a resolução prática e simples do ponto de vista computacional.

É importante enfatizar que o problema de otimização convexo é manipulado totalmente por rotinas computacionais, sendo transparente para o usuário, bastando este apenas modelar corretamente o problema. Desta forma, não é abordagem do presente trabalho explicar os métodos numéricos para a resolução dos problemas, uma vez modelados. Considera-se, portanto, que a fase de resolução, bem como suas rotinas computacionais, são uma "caixa-preta", do ponto de vista do usuário.

2.2.5 Programação Geométrica Generalizada

Na subseção 1.1.1, vimos que algumas expressões não podem ser consideradas posinômios, como por exemplo $f(x)^{0.5} \leq 1$. Entretanto, podemos inserir uma nova variável e uma nova restrição a esse PG:

$$f(x) \leq t \qquad t^{0.5} \leq 1$$

Aqui, a variável t foi inserida. Ela é o *limite superior* da função posinomial $f(x)$. Dessa maneira, temos duas restrições que são compostas de posinômios compatíveis com um programa geométrico.

Podemos estender essa ideia para outro tipo de operação: máximo entre dois posinômios. Seja uma expressão da forma $\max\{f_1(x), f_2(x)\}$. Podemos limitar as duas funções da seguinte forma, obtendo dois posinômios:

$$f_1(x) \leq t \qquad f_2(x) \leq t$$

Desta forma, algumas expressões não podem ser considerados posinômios, porém podem ser transformados recursivamente em posinômios. Essas expressões são denominadas *posinômios generalizados*, dentro do contexto de *programação geométrica generalizada*.

Formalmente, uma função f, composta de variáveis positivas $x_1,\ldots, x_n$ é um posinômio generalizado se ele pode ser formado por posinômios utilizando-se operações de adição, multiplicação, potência positiva (fracional) e máximo, além de outras operações, como divisão por monômios e composição de posinômios.

Um *programa geométrico generalizado* (PGG) é um problema de otimização da forma

$$\begin{aligned} \text{minimizar} \quad & f_0(x) \\ \text{sujeito a} \quad & f_i(x) \leq 1, \quad i = 1,\ldots, m, \\ & g_i(x) = 1, \quad i = 1,\ldots, p, \end{aligned}$$

onde f_i são posinômios generalizados e g_i são monômios. Como qualquer posinômio generalizado é também um posinômio, conclui-se que qualquer PG é também um PGG.

2.2.6 Exemplo de um PGG: Problema de Posicionamento

O exemplo apresentado a seguir, também baseado no trabalho de Boyd, mostra de uma forma clara a aplicação de programação geométrica generalizada para a resolução de problemas práticos.

Em um *problema de posicionamento*, consideramos a presença de alguns retângulos que devem ser posicionados de maneira que eles não se sobreponham, com o objetivo de minimizar o retângulo-envelope, o qual engloba todos os retângulos. O problema de posicionamento considera o aspecto geométrico em ferramentas de EDA. Alguns exemplos de aplicação do problema de posicionamento são minimizar a área do circuito, limitar as áreas de contato entre os diversos blocos, onde se utilizam fios de ligação (*bond wire*s), introdução de blocos de propriedade intelectual, onde conhece-se previamente a área a ser utilizada, e otimização para a etapa de definição das interligações (*routing).*

Classicamente, a restrição de que os blocos não se sobreponham é um complicado problema de otimização combinatória. Entretanto, caso o posicionamento relativo seja informado, o problema de posicionamento pode ser formulado como um PGG.

Como um exemplo, na Figura 2.3, temos três blocos, A, B e C, que devem ser posicionados com as seguintes restrições:

- A está à esquerda de B.

- A e B estão acima de C.

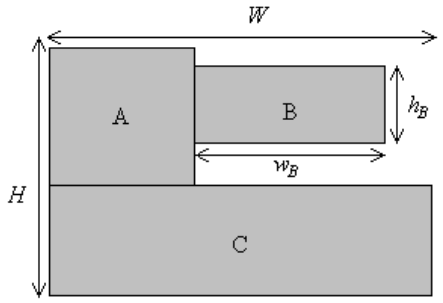


Figura 2.3: Exemplo de um problema de posicionamento

O comprimento de A e B, é $w_A + w_B$. Dessa forma, o comprimento do retângulo-envelope é $W = \max\{w_A+w_B, w_C\}$. Da mesma maneira, a altura do retângulo-envelope é $H = \max\{h_A,h_B\} + h_C$. Logo, a área pode ser representada pela seguinte expressão:

$$WH = \max\{w_A+w_B, w_C\}(\max\{h_A,h_B\} + h_C)$$

O problema, na forma de um PGG, pode ser reescrito da seguinte forma:

$$\begin{aligned} \text{minimizar} \quad & \max\{w_A+w_B, w_C\}(\max\{h_A,h_B\} + h_C) \\ \text{sujeito a} \quad & h_A w_A = a, \quad h_B w_B = b, \quad h_C w_C = c, \\ & 1/\alpha_{\max} \le h_A/w_A \le \alpha_{\max}, \quad 1/\alpha_{\max} \le h_B/w_B \le \alpha_{\max}, \\ & 1/\alpha_{\max} \le h_C/w_C \le \alpha_{\max}, \end{aligned}$$

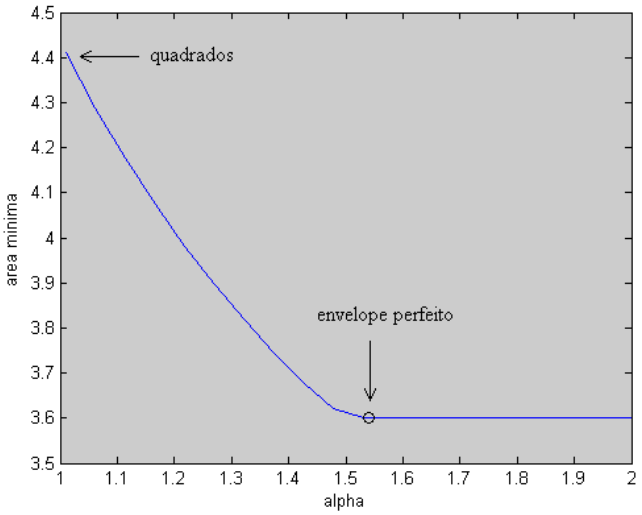
Nesse PGG, as variáveis são w_A..., w_C e h_A..., h_C. Os parâmetros a, b e c são as áreas de cada retângulo, e $\alpha_{\max}$ é o limite na relação entre altura e comprimento.

Como um exemplo numérico, pode-se definir os parâmetros como $a = 1$, $b = 0.8$, $c = 1.8$. Após inserirmos os parâmetros, variáveis e funções em um arquivo de edição do MATLAB, a função para a resolução de PGGs, do *toolbox* GGPLAB foi chamada. Variou-se o parâmetro $\alpha_{\max}$ entre 1 e 2. A Figura 2 mostra os resultados. À medida que $\alpha_{\max}$ é relaxado, a área mínima para o retângulo-envelope diminui. Tem-se um resultado ótimo quando $\alpha_{\max} = 1.53$. A partir desse ponto, a área vai ser sempre a mesma, não importando quanto o parâmetro $\alpha_{\max}$ for mudado.

Em outra variável, como mostra a Figura 2.4, obtêm-se os resultados ótimos para as variáveis w_A..., w_C e h_A..., h_C.

2.3 Trabalhos Relacionados

Há cada vez mais trabalhos envolvendo programação geométrica na concepção de circuitos integrados, graças à facilidade que as ferramentas vêm proporcionando no meio acadêmico.



	1	2
1	'ha'	1.0811
2	'hb'	1.0811
3	'hc'	1.0811
4	'wa'	0.9249
5	'wb'	0.7400
6	'wc'	1.6649

Figura 2.4: Resultados para o problema de posicionamento

Entre os trabalhos que envolvem dimensionamento e programação geométrica, temos:

- Che, Hseih E Pedram (2000) propõem um algoritmo para dimensionamento de portas lógicas e posicionamento, de maneira simultânea. Basicamente, são realizadas operações em alguns caminhos críticos, com o objetivo de diminuir a carga das células nesses caminhos. As principais operações, a cada iteração do algoritmo proposto, são a redução no tamanho das células e a obtenção de um correto posicionamento. Cada uma das operações é introduzida em um programa que utiliza PGG, sendo esse aspecto o responsável por eficientes técnicas de resolução. Para a validação, é feita uma comparação com algoritmos que realizam as duas operações de maneira separada, utilizando-se circuitos de referência (*benchmarks*).

- Pattanaik, Banerjee, e Bahinipati (2003) propõem um método para o dimensionamento de um inversor utilizando programação geométrica e um modelo modificado de tensão-corrente. Como resultados, o método obteve o limite absoluto de desempenho, dados os parâmetros da tecnologia, a frequência e capacitância de carga. A precisão do método foi confirmada através de simulações em SPICE.

- Chu e Wong (2001) realizam o dimensionamento no nível de porta lógica. De acordo com modelos já conhecidos, é definido um tipo especial de programa geométrico, denominado *programa geométrico unário*. Um algoritmo guloso é proposto para solucionar tais programas geométricos. Segundo os autores, quando aplicados a circuitos VLSI,

  > **"é provado que o tempo de execução do algoritmo guloso é linear em relação ao número de componentes do circuito [...] Na prática, a aproximação por um programa geométrico unário para dimensionamento é centenas de vezes mais rápida que outras abordagens".**

Por último, cabe a citação de um artigo que mostra uma abordagem do dimensionamento de portas lógicas sob outro ângulo: Coudert e Haddad, (1996)

comparam alguns métodos inovadores para dimensionamento de portas lógicas. No entanto, o artigo refuta a hipótese que a programação geométrica – assim como outros métodos semelhantes – seja a maneira mais eficiente de resolver o problema de dimensionamento, afirmando que

> **"os modelos de atraso e potência não são mais realísticos, ou são muito simplificados para uma técnica de otimização; [...] alguns métodos assumem que a função-objetivo e a região factível são convexas, o que não sustenta modelos precisos de atraso e potência; além de algumas abordagens mencionadas acima são muito intensas computacionalmente para serem aplicadas a circuitos com mais de 1000 nós.".**

3 ESPECIFICAÇÃO E DESCRIÇÃO DO PROJETO

3.1 Descrição do Projeto

Deseja-se conceber um programa que realize o dimensionamento de células (instâncias de portas lógicas) de um circuito, mapeado para uma biblioteca, com o objetivo de minimizar o atraso e respeitando-se restrições de área. Internamente, e transparente ao usuário, estão a síntese lógica do circuito, modelos de aproximação para o atraso, e a utilização de programação geométrica para a resolução do problema de dimensionamento.

O projeto foi composto de várias etapas. Cada módulo será detalhado a seguir. O fluxo do projeto, juntamente com todas as etapas, é apresentado, de maneira simplificada, na Figura 3.1.

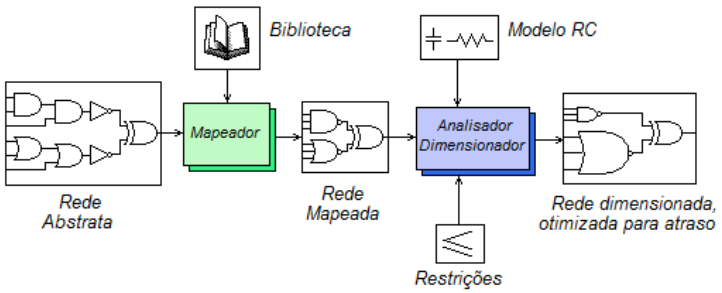


Figura 3.1: Fluxo do projeto

3.2 Especificações do Projeto

3.2.1 Entradas e Saídas

O usuário deverá entrar no fluxo do projeto com os seguintes dados:

- uma descrição funcional do circuito;
- uma biblioteca de células;
- a tecnologia empregada;
- restrições para a área total do circuito;

A biblioteca deve ter implementações “unitárias” de cada célula, ou seja, o seu menor tamanho possível, como já foi discutido anteriormente. Já como saída, espera-se que o programa informe o dimensionamento ideal para cada célula. O valor para cada dimensionamento é um número real. O usuário decide, então, se utiliza instâncias mais próximas dos valores ótimos obtidos, ou se redefine a biblioteca, visando a utilização de

implementações que se aproximem desses valores.

3.2.2 Mapeamento Tecnológico

O mapeamento tecnológico (baseado em biblioteca) consiste em transformar uma rede, descrita no nível lógico e independente de tecnologia, em outra rede, onde são escolhidas células de uma biblioteca para a implementação dessa rede. Há diversos algoritmos que realizam essa escolha, como por exemplo, mapeamento por árvore (KEUTZER, 1987) e mapeamento por DAG (KUKIMOTO et al., 1998).

Dependendo do circuito ao qual se deseja aplicar o dimensionamento, essa etapa é importante. Neste trabalho serão utilizados os circuitos de teste ISCAS ’85 (BRGLEZ e FUJIWARA, 1985), os quais não apresentam nenhum tipo de otimização em sua versão estrutural. O mapeamento tecnológico, nesses circuitos, pode diminuir consideravelmente o número de células, a área, o atraso e interconexões.

Para o mapeamento tecnológico, foi escolhido o programa ABC (BERKELEY LOGIC SYNTHESIS AND VERIFICATION GROUP, 2009), que utiliza *and-inverter graphs* (AIGs), DAGs e algoritmos para síntese sequencial e verificação funcional. Além de um mapeamento tecnológico, o programa realiza diversas otimizações lógicas.

A saída do mapeamento tecnológico é uma descrição otimizada do circuito, já mapeada para a biblioteca de interesse. Isso significa que as portas lógicas que compõem o circuito final não serão alteradas. O que vai mudar é justamente o tamanho de cada célula, atribuição da fase de dimensionamento.

3.2.3 Analisador

Para a aplicação do algoritmo de dimensionamento, é preciso ler o circuito, construir algumas estruturas de dados e colocá-lo no formato correto. Compete ao analisador (*parser*, em inglês) realizar essas tarefas.

O analisador possui, como entrada, o arquivo mapeado e a biblioteca (juntamente com os modelos de atraso para cada célula). Como saída, espera-se um arquivo compatível com um programa que realize otimização convexa (programação geométrica). Esse arquivo representa a modelagem do problema de otimização, ou seja, a obtenção da correta função-objetivo e restrições, os quais serão detalhados adiante.

3.2.4 GGPLAB

A última parte no fluxo do projeto é a utilização de programação geométrica para o dimensionamento das células. Para tanto, o arquivo obtido a partir do analisador deve possuir um formato compatível com GGPLAB, um conjunto de ferramentas para especificação e solução de problemas envolvendo programação geométrica e programação geométrica generalizada.

A partir da modelagem do problema, as rotinas de resolução serão chamadas, e o dimensionamento ideal para cada célula do circuito é informado, finalizando o fluxo da síntese lógica.

3.3 Metodologia a ser Utilizada na Implementação

Para a construção do analisador, um programa simples pode ser feito, onde arquivos são lidos, e passados a estruturas de dados internas. Após a manipulação dessas estruturas, o programa deve ser capaz de fornecer novos arquivos, que serão passados às próximas fases.

3.4 Metodologia a ser Aplicada no Teste

Na etapa de teste dos programas, vários circuitos *benchmarks* poderão ser fornecidos como entrada, além de uma ou mais bibliotecas. A biblioteca escolhida deve conter, no mínimo, a equação lógica que representa cada célula, para que o programa de mapeamento consiga realizar as otimizações necessárias. Além disso, é necessário que as células dos circuitos de teste tenham exatamente o mesmo nome das células da biblioteca. É desejável, como explicado anteriormente, que exista apenas uma implementação de cada porta lógica na biblioteca.

O modelo RC dos circuitos de teste deve ser o mais aproximado possível da tecnologia, para que o atraso calculado seja confiável e comparável a outros resultados na fase de validação.

Após o dimensionamento ter sido realizado, algumas informações serão apresentadas. A principal é o tamanho de cada célula, um valor real obtido a partir da resolução do programa geométrico. Além disso, a área total do circuito e o atraso máximo, para a solução ideal, completam as variáveis de saída.

O tempo de execução do algoritmo é outro interesse na etapa de teste. Além de fornecer uma expectativa do custo computacional, desde os mais simples até os maiores circuitos, é um paralelo interessante para a fase de validação, pois é um fator determinante do sucesso do algoritmo, caso este tempo de execução seja igual ou inferior ao de outros dimensionadores existentes.

3.5 Validação do Projeto

A validação do projeto consiste em comparar o resultado obtido nos testes com programas semelhantes. As saídas do teste devem, portanto, ter o mesmo grau de abstração que resultados obtidos de outros softwares.

Algumas comparações podem ser feitas, como, por exemplo, o resultado para o atraso e a área, além do tempo de execução. Com isso, espera-se saber se o programa concebido pode dar resultados melhores que as soluções já existentes, ou se o mesmo deve ser modificado e melhorado.

4 IMPLEMENTAÇÃO DO PROJETO

Nesta Seção, todas as fases do fluxo do projeto serão detalhadas, ao mesmo tempo em que os conceitos apresentados até agora serão aplicados.

4.1 Mapeamento do Circuito de Entrada

Embora não faça parte da implementação, o mapeamento fornece a descrição a nível estrutural do circuito, onde o formato da saída do mapeamento precisa ser compatível com a próxima etapa.

A ferramenta ABC exige apenas alguns passos para o mapeamento do circuito. O comando **read_library** lê uma biblioteca de células no formato GENLIB. Um exemplo de biblioteca nesse formato está na Figura 4.1.

Após a entrada da biblioteca desejada, a leitura do circuito de entrada é feita, com o comando **read.** Atualmente, os formatos suportados são *.aig, .baf, .bench, .blif, .eqn, .pla,* e *.verilog*.

Utilizando-se o comando **map**, o circuito é mapeado para a biblioteca. A ferramenta realiza algumas manipulações lógicas, como a utilização de AIGs e balanceamento otimizado para atraso e, a seguir, realiza o mapeamento propriamente dito. Todas essas operações são transparentes.

O circuito mapeado está, agora, presente na memória. Para obtermos a descrição de tal circuito, devemos escolher qual o formato para a saída. No presente trabalho, optou-se pelo formato *Berkeley Logic Interchange Format* (*BLIF*). Na Figura 4.2, temos um circuito neste formato. Dessa maneira, o comando a ser executado é **write_blif**, seguido do nome do arquivo de saída. Após a execução do comando, completa-se o ciclo de mapeamento tecnológico.

4.2 Analisador

O analisador foi concebido utilizando-se a linguagem C/C++. Ele divide-se em dois módulos: a leitura de um circuito de entrada e a escrita de um arquivo compatível com o MATLAB. As duas funções serão detalhadas, bem como alguns métodos, funções e estruturas de dados.

4.2.1 Leitura do Circuito de Entrada e Biblioteca

Inicialmente, o analisador abre um arquivo *.blif* contendo a descrição do circuito de entrada. De modo a verificar se o arquivo está correto, é feita a etapa de *análise*, propriamente dita. Com a ajuda da interface de gerador de análise sintática YACC (JOHNSON, 1975), e do reconhecedor léxico LEX (LESK, 1975) foram definidas regras para a correta leitura do circuito. Caso haja algum erro no arquivo de entrada, o

programa encerra a execução.

```
GATE buf1    1     O=a;                PIN * INV 1 999 0.9 0.3 0.9 0.3
GATE inv1    1     O=!a;               PIN * INV 1 999 0.9 0.3 0.9 0.3
GATE nand2   2     O=!(a*b);           PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE nand3   3     O=!(a*b*c);         PIN * INV 1 999 1.1 0.3 1.1 0.3
GATE nand4   4     O=!(a*b*c*d);       PIN * INV 1 999 1.4 0.4 1.4 0.4
GATE nor2    2     O=!(a+b);           PIN * INV 1 999 1.4 0.5 1.4 0.5
GATE nor3    3     O=!(a+b+c);         PIN * INV 1 999 2.4 0.7 2.4 0.7
GATE nor4    4     O=!(a+b+c+d);       PIN * INV 1 999 3.8 1.0 3.8 1.0
GATE and2    3     O=a*b;              PIN * NONINV 1 999 1.9 0.3 1.9 0.3
GATE or2     3     O=a+b;              PIN * NONINV 1 999 2.4 0.3 2.4 0.3
GATE xor     5     O=a*!b+!a*b;        PIN * UNKNOWN 2 999 1.9 0.5 1.9 0.5
GATE xnor    5     O=!(!a*b+a*!b);     PIN * UNKNOWN 2 999 2.1 0.5 2.1 0.5
GATE aoi21   3     O=!(a*b+c);         PIN * INV 1 999 1.6 0.4 1.6 0.4
GATE aoi22   4     O=!(a*b+c*d);       PIN * INV 1 999 2.0 0.4 2.0 0.4
GATE oai21   3     O=!((a+b)*c);       PIN * INV 1 999 1.6 0.4 1.6 0.4
GATE oai22   4     O=!((a+b)*(c+d));   PIN * INV 1 999 2.0 0.4 2.0 0.4
GATE zero    0     O=CONST0;
GATE one     0     O=CONST1;
```

Figura 4.1: Exemplo de biblioteca no formato GENLIB

```
# Benchmark "c17" written by ABC on Fri Oct 20 01:27:39 2009
.model c17
.inputs 1 2 3 6 7
.outputs 22 23
.gate nand2 a=3     b=1    O=n7
.gate nand2 a=6     b=3    O=n8
.gate nand2 a=n8    b=2    O=n9
.gate nand2 a=n9    b=n7   O=22
.gate nand2 a=n8    b=7    O=n11
.gate nand2 a=n11   b=n9   O=23
.end
```

Figura 4.2: Exemplo de circuito no formato BLIF

Na descrição do circuito, cada pino de célula, seja entrada ou saída, é representado por uma sequência de caracteres. Caso a mesma sequência se repita em outra célula, significa que ambas estão ligadas. Finalmente, se a sequência de caracteres for encontrada no conjunto de entradas e saídas (primárias), os pinos correspondentes da entrada e saída do circuito serão ligados aos pinos correspondentes nas células.

Caso análise tenha sido feita com sucesso, o próximo passo é agregar os dados obtidos a certas estruturas de dados, dentro do código do analisador. A primeira estrutura é um vetor contendo todas as células do circuito; cada célula é representada por uma estrutura onde são armazenados o tipo de célula, as entradas e a saída. Já a segunda contém, respectivamente, as entradas e saídas primárias do circuito.

A seguir, a biblioteca deve ser lida. Como os dados obtidos, por enquanto, são apenas o nome, a área ocupada e a capacitância de entrada (respectivamente, segunda, terceira e oitava colunas do arquivo GENLIB – ver Figura 4.1), não é necessária a utilização de LEX/YACC.

4.2.2 Preparação para Escrita do Arquivo MATLAB

Feita a leitura da biblioteca e do circuito de entrada, com os dados filtrados e salvos em algumas estruturas de dados, o próximo passo é desenvolver funções para produzir o

arquivo que será usado como entrada na ferramenta MATLAB. Cabe ressaltar que o motivo da existência das funções aqui descritas ficará mais claro na subseção 4.3, onde o algoritmo de dimensionamento será detalhado.

Inicialmente, deseja-se obter o conjunto *fan-in* para cada célula do circuito, entrada primária e saída primária. Para realizar esta tarefa, todas as células do circuito são percorridas, e a saída delas é comparada com a célula na qual deseja-se obter o conjunto *fan-in*. Além disso, percorre-se o conjunto de entradas primárias, pois elas são, também, entradas de algumas células.

O conjunto *fan-in* para as entradas primárias é nulo, enquanto que o mesmo conjunto para as saídas é unitário, e para cada célula do circuito, igual ao seu número de entradas. Dessa forma, tem-se uma maneira equivalente de descrever o circuito, mas muito mais útil do ponto de vista do percorrimento entre os diversos caminhos, como será explicitado adiante.

A estrutura de dados que contém os conjuntos *fan-in* é um vetor, e foi descrita como Para efeitos de índice nesse vetor, primeiramente foram inseridas as células, seguidas pelas entradas primárias, e após, pelas saídas primárias.

Finalmente, devemos obter, ainda nessa fase, a restrição para a área total do circuito, como uma opção do usuário. Por simplicidade, optou-se pelo cálculo da área máxima a partir de uma constante, em detrimento da inserção direta pelo usuário de um valor pronto. Essa constante, definida por '*averageFactor*', indica a área máxima da seguinte forma: a partir da soma das áreas das células do circuito, com implementação unitária, a restrição de área é obtida multiplicando-se essa soma por '*averageFactor'*. Por exemplo, consideremos um circuito com área de 200 unidades, concebido totalmente por células de implementação unitária; se a ideia do usuário é obter, em média, células de implementação quatro vezes maior que a unitária, ele insere o fator 4, sendo a área total igual ou inferior a 800. Dessa forma, o usuário pode não conhecer a área total do circuito com implementação unitária, apenas indicando um valor médio de implementação de cada célula. A desvantagem dessa abordagem é justamente criar uma falsa expectativa em relação ao tamanho das células, pois o dimensionamento pode gerar valores bem afastados da média, dependendo do circuito envolvido.

4.3 Dimensionador

Após o analisador realizar as suas tarefas, o arquivo compatível com a ferramenta MATLAB pode ser criado. O próprio analisador encarrega-se de escrever o novo arquivo, pois já possui todos os dados que precisa. O arquivo a ser escrito tem o nome '*circuito.m*', onde *circuito* é o mesmo nome do circuito que é informado ao analisador como entrada.

Ao invés de apresentar o código do analisador, é mais interessante observar um arquivo de saída. O Apêndice B mostra o circuito obtido, **c17**, com seis células *nand2*. Nota-se a divisão em três partes, que serão exploradas a seguir.

4.3.1 Dados do Circuito

Todas as informações relativas à entrada estão nesta parte do arquivo. Primeiramente, a biblioteca é passada ao MATLAB, na forma de estrutura, com os seguintes dados: nome da célula, capacitância de entrada, capacitância intrínseca, resistência de ativação e área da célula. Após, o número de células e a área máxima (já calculada pelo analisador) são inseridas.

Após, as células do circuito são instanciadas, na variável `gates`. Dois conjuntos, `primary_inputs` e `primary_outputs`, indicam quais são os índices, dentro da tabela de *fan_in*, para as entradas primárias e as saídas primárias, respectivamente.

O próximo passo é a construção da tabela de *fan_in*, sendo esta uma simples cópia da estrutura de dados do analisador. Finalmente, são definidas as capacitâncias de entrada das saídas primárias, e capacitâncias de carga das entradas primárias.

4.3.2 Dados Derivados

Todos os dados para o início dos procedimentos matemáticos estão presentes. Entretanto, alguns outros dados necessitam ser obtidos a partir dos dados do circuito, para possibilitar a modelagem e a resolução do programa geométrico relacionado.

O conjunto de *fan-out* para cada célula, entrada e saída primárias, será utilizado adiante, e é facilmente obtido a partir do conjunto `FI`. Por fim, são inseridos variáveis do tipo vetor com a extração do modelo RC para cada célula do circuito, levando em consideração a implementação unitária (`Cin_norm`, `Cint_norm` e `Rdrv_norm`), além de outro vetor com a especificação de área de cada célula com implementação unitária (`A_norm`). A área total do circuito é proporcional ao fator de dimensionamento, `x`, o qual será otimizado na etapa de solução do problema de otimização geométrico. A expressão para a área total é `area = A_norm'*x`.

Os vetores `Cin`, `Cint` e `R` são vetores, onde o índice representa uma célula do circuito. Pelo modelo de aproximação do circuito, é indicado que as capacitâncias de entradas são diretamente proporcionais à capacitância da implementação unitária, e a resistência é inversamente proporcional à resistência normalizada. Em todos os casos, o fator a ser utilizado é o fator de dimensionamento `x`.

Para a obtenção da capacitância de carga, é construído um novo vetor, `Cload`. Caso a célula esteja ligada a uma saída primária, o valor de `Cload` será igual à capacitância de carga escolhida para cada saída. Do contrário, verifica-se no conjunto de *fan-out* da célula a capacitância de entrada da célula ligada à saída da atual célula, sendo esse o valor para a capacitância de carga.

Por fim, utiliza-se a fórmula para o atraso, apresentada na Seção 2; O vetor `D` representa a aplicação da fórmula em cada célula do circuito. O comando é

```
» D = 0.69*ones(m,1).*R.*(Cint+Cload)
```

4.3.3 Utilização do GGLPAB

Antes de apresentar os métodos de modelo do problema e posterior resolução, cabe uma breve explicação das funções e estruturas encontradas no GGPLAB.

As ferramentas de especificação e solução de programas geométricos e programas geométricos generalizados, presentes no GGPLAB, podem ser utilizadas no software MATLAB a partir da versão 6.1. Antes de executar o arquivo, deve-se indicar o caminho para a ferramenta, com o comando

```
» addpath /path/ggplab ,
```

onde */path/ggplab* é o local de descompactação da ferramenta.

As variáveis do tipo *gpvar* são objetos do MATLAB, e representam as variáveis de um programa geométrico. Já os monômios podem ser criados através das operações de

multiplicação, divisão e potenciação, a partir de constantes, variáveis *gpvar*, ou outros monômios. Posinômios podem ser construídos a partir da soma e multiplicação de constantes, monômios e outros posinômios. A divisão não é permitida entre posinômios, mas pode ser feita ente um posinômio e um monômio. Por fim, posinômios generalizados podem ser formados por todas as operações citadas para os posinômios, além de potência fracionária positiva, e máximo (utilizando-se a função `max`).

As funções de restrição podem ser elaboradas utilizando-se a forma `f<=g`, onde `f` é um posinômio generalizado, e `g` é um monômio, ou ainda, `f1==f2`, onde ambos são monômios.

Após a correta modelagem do problema, a função para a obtenção pode ser chamada, com a seguinte sintaxe:

```
» [objvalue,solution,status]=gpsolve(obj,constr_array,type)
```

Aqui, `obj` é a função objetivo, `constr_array` é um vetor com todas as restrições, e `type` indica se o problema é do tipo de minimização da função objetivo (`min`), ou maximização (`max`). Esse parâmetro é opcional. Os valores de retorno são o valor ideal da função objetivo (`objvalue`), um vetor de variáveis *gpvar* e os valores ótimos para cada variável (`solution`), e o status da resolução do problema: '`Solved`', caso a otimização tenha sido realizada com sucesso, '`Infeasible`', caso o problema não seja solucionável, ou '`Failed`', se a otimização não pôde ser realizada.

O comando MATLAB `assign` toma as variáveis *gpvars* como entrada, e possui como saída os valores ótimos (geralmente do tipo *double*). A sintaxe, para o exemplo acima, é `assign(solution)`, onde solution agora possui esses valores ótimos.

4.3.4 Modelagem do PG de Dimensionamento e Solução

Com todos os dados já inseridos, a próxima etapa é a resolução do programa geométrico, utilizando o contexto do GGPLAB. As duas variáveis do tipo *gpvar* são

- `x`: dimensionamento para cada célula do circuito, portanto, um vetor com número de entradas igual ao número de células do circuito.
- `T`: representa a restrição para o atraso, em um ponto do circuito. Essa restrição indica que o atraso, em uma célula *i*, é o limite superior entre as restrições de atraso para as entradas da célula *i*, mais o valor do atraso na própria célula *i*. Caso a célula esteja ligada a uma entrada primária do circuito, a restrição é apenas em relação ao atraso de *i*. Desta maneira, se considerarmos o circuito como um DAG, `T` indicará o atraso máximo para qualquer caminho no DAG até a célula *i*, obtidos de maneira recursiva.

Existem, no total, três restrições: `timing_constraint`, um vetor de posinômios construído a partir de `T`. Todas as restrições de `T` são inseridas nesse vetor. Desse modo, o programa irá, de maneira recursiva, obter o atraso em todos os caminhos do circuito para realizar as otimizações em atraso e área. As duas outras restrições são em relação à área: o limite mínimo `x_constr` indica que cada implementação será, no mínimo, igual à implementação unitária; por fim, `area <= Amax` indica a área máxima do circuito. Todas essas restrições são inseridas na variável `constr`.

A função-objetivo é um posinômio generalizado do tipo 'máximo', ou seja, o limite

superior entre as restrições de atraso para toda as portas conectadas à saída, e o comando para a construção da função é `D = max(T(output_gates))`.

A seguir, o problema de PG é formulado e solucionado, sob os seguintes comandos:

```
» [obj_value, solution, status] = gpsolve(D, constr);
» assign(solution);
```

Com isso, finaliza-se a etapa de dimensionamento. A saída, no MATLAB, é o atraso ótimo, juntamente com o dimensionamento ótimo para cada célula. Finaliza-se, então, o fluxo do projeto de dimensionamento.

Como uma opção a parte, o usuário pode visualizar a curva de atraso versus área máxima, variando-se esse último parâmetro. Uma vez que as variáveis que representam as restrições de atraso no circuito já foram avaliadas, o programa geométrico pode variar o parâmetro de área, fazendo uma chamada para a função de solução `gpsolve`, tantas vezes quanto forem desejadas.

Escolheu-se variar o parâmetro de área 20 vezes, sendo esse o número de pontos da curva de atraso versus área máxima e o número de vezes que a função de solução será chamada. O limite mínimo para a área total do circuito é igual à metade do valor original, e o limite máximo é o dobro desse valor.

Finalmente, o gráfico pode ser apresentado. A forma da curva vai indicar ao usuário se a escolha da área máxima foi bem feita. Além disso, obtém-se uma possibilidade de flexibilização, caso o usuário possa alterar a área, em detrimento de mudanças no atraso, ou vice-versa.

5 TESTE E AVALIAÇÃO DO PROJETO

5.1 Avaliação do Projeto

O projeto foi avaliado para diversos circuitos de teste. Como foi afirmado anteriornente, na subeção 3.2, optou-se pelos *benchmarks* ISCAS ’85, pois representam circuitos que poderiam fazer parte de diversos projetos VLSI. Além disso, os circuitos apresentam variedade tanto no tamanho quanto na composição das portas lógicas.

A tabela 5.1 mostra todos os circuitos analisados, além de uma breve descrição do funcionamento de cada um. Já a Figura 5.1 apresenta o circuito **c17**, no formato de entrada *.bench*, escolhido para a entrada do mapeador. Neste formato, as entradas e saídas primárias são listadas; após, cada linha representa a saída de uma porta lógica como função das entradas desta porta.

Para o mapeamento de tais circuitos, utilizou-se a biblioteca *mcnc.genlib* (YANG, 1991). A biblioteca foi apresentada na Figura 4.1. Ao todo, há 18 células na biblioteca.

Na fase de mapeamento tecnológico, no ambiente ABC, todos os onze circuitos foram lidos e mapeados para a mesma biblioteca. A Figura 4.2 é um exemplo de resultado da saída do mapeador, para o circuito **c17**.

Tabela 5.1: Descrição dos circuitos de teste ISCAS ‘85

Circuito	*Entradas*	*Saídas*	*Portas*	*Descrição*
c17	3	2	6	Somador completo (1 bit)
c432	36	7	160	Controlador de interrupções com 27 canais
c499	41	32	202	Corretor de erro (32 bits) - portas XOR
c880	60	26	383	ULA 8 bits
c1355	41	32	546	Corretor de erro (32 bits) - portas NAND
c1908	33	25	880	Detector e corretor de erro 16 bits
c2670	233	140	1193	ULA (12 bits) e controlador
c3540	50	22	1669	ULA (8 bits), lógica binária/BCD, operações lógicas
c5315	178	123	2406	ULA (9 bits), operações lógicas, paridade
c6288	32	32	2406	Multiplicador (16 bits x 16 bits)
c7552	207	108	3512	Somador e Comparador (34 bits)

Fonte: YANG, 1991.

```
# c17: 5 inputs 2 outputs 0 inverter 6 gates (6 NANDs)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(6)
INPUT(7)
OUTPUT(22)
OUTPUT(23)
10 = NAND(1,3)
11 = NAND(3,6)
16 = NAND(2,11)
19 = NAND(11,7)
22 = NAND(10,16)
23 = NAND(16,19)
```

Figura 5.1: Exemplo de circuito no formato BENCH

Para a estimativa dos valores de capacitância e resistência das células da biblioteca, foi necessária a escolha de uma tecnologia. A fase de validação do projeto irá utilizar uma biblioteca onde as células foram projetadas em processos de 45nm. Foi escolhida uma biblioteca de código aberto de 45nm, elaborada pela Nangate Inc., desenvolvedora de softwares para EDA. Em vista disto, optou-se pela utilização da mesma tecnologia, onde o valor de τ é aproximadamente 5ps. Desta forma, foi possível calcular a resistência intrínseca, a partir da fórmula deduzida em 2.1.3.1.

Tabela 5.2: Resultados de modelagem para uma biblioteca em 45nm

Porta	$\bar{R}^{int}(k\Omega)$	$\bar{C}^{in}(fF)$	$\bar{C}^{int}(fF)$	$\bar{A}$ (nm^2)
inv1	21.73	0.46	0.44	0.532
buf1	51.42	0.48	0.45	0.798
nand2	26.22	0.52	0.53	0.798
nand3	28.98	0.59	0.61	1.064
nand4	30.05	0.66	0.69	1.330
nor2	29.25	0.57	0.52	0.798
nor3	37.07	0.68	0.61	1.064
nor4	43.26	0.72	0.62	1.330
xor2	46.26	1.12	1.01	1.596
xnor2	37.57	1.17	0.99	1.596
aoi21	41.59	0.59	0.49	1.064
aoi22	47.13	0.66	0.57	1.330
oai21	48.03	0.57	0.45	1.064
oai22	49.58	0.60	0.54	1.330
and2	51.56	0.55	0.51	1.064
or2	58.45	0.63	0.56	1.064

A tabela 5.2 mostra os valores calculados para todas as células da biblioteca, bem como a área da implementação. Os dados de área e capacitâncias foram obtidos direto da biblioteca, e a resistência foi estimada.

A seguir, basta chamar o programa *yacc-parser*, para analisar todos os circuitos mapeados. A forma de chamar o programa é a seguinte:

```
» ./yacc-parser circuito.blif
```

A saída, *circuito.m*, é um arquivo que servirá de entrada ao fluxo no MATLAB. Um exemplo de saída do analisador é apresentado no Apêndice B, para o circuito **c17**. De posse dos arquivos necessários, basta chamar o programa MATLAB, para que o mesmo execute todas as rotinas de otimização.

5.2 Resultados Obtidos

Para a obtenção dos resultados, utilizou-se um *average factor* igual a três, ou seja, a restrição de área total do circuito é igual a três vezes a área do circuito com implementações de força iguais a um. Após a solução do programa geométrico generalizado de dimensionamento, obtêm-se o atraso do caminho crítico, o atraso em cada célula e a área de cada célula, representando, essa última, uma aproximação para a implementação ideal.

Um exemplo de saída do fluxo está na Figura 5.2. O vetor x apresenta a área ideal para cada uma das seis portas NAND, e o atraso calculado para esse dimensionamento (em torno de 58ps).

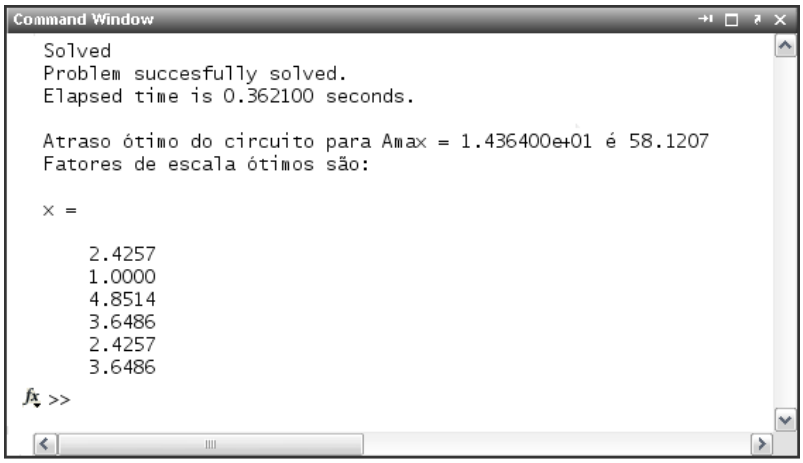


Figura 5.2: Saída para o circuito c17

Como opção de saída no dimensionamento do circuito c17, obteve-se a curva de atraso versus área, quando a área é variada entre a metade e o dobro da área obtida na solução. A Figura 5.3 mostra a curva. Nota-se que o atraso pode variar de 45ps até mais de 80ps, dentro desta janela de área.

Na tabela 5.3, são mostrados o número de células do circuito (após o mapeamento tecnológico feito pelo programa ABC), a área total, o atraso ideal para essa área, além do tempo de execução das rotinas de otimização que utilizam programação geométrica. A título de comparação, a penúltima coluna apresenta o atraso sem dimensionamento, onde a restrição da área total permanece a mesma, e todas as células possuem o mesmo tamanho, que é igual ao *average factor.*

Nota-se que na tabela 5.3 há apenas cinco circuitos (c17, c499, c1355, c1908 e c6288) que foram devidamente solucionados. Isso ocorreu em vista de um erro na chamada da resolução do GGPLAB para os outros circuitos, impedindo a resolução do problema. Sugere-se, a partir da Figura 5.4, que exista um erro na rotina do GGPLAB que realiza a extração de desigualdades posinomiais. Talvez seja possível a existência de algum erro no *parser* que realiza a modelagem, mas os arquivos foram depurados, e não havia erros aparentes. Apenas uma análise minuciosa das rotinas do GGPLAB indicaria se o erro aí está presente.

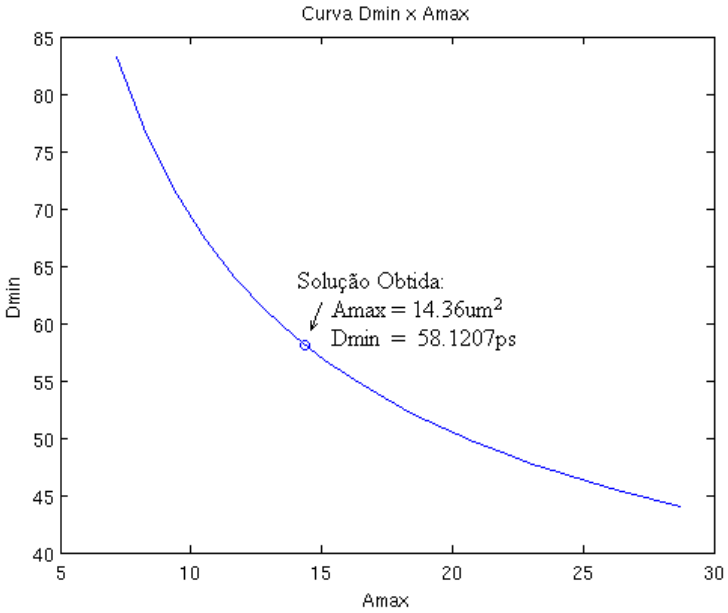


Figura 5.3: Atraso mínimo em função da área máxima para o circuito c17

Tabela 5.3: Resultados do dimensionamento para os circuitos de teste

Circuito	*Células*	*Área Máx. (nm²)*	*Atraso Obtido-A1(ps)*	*Atraso sem Dimens.-A2 (ps)*	*(A2-A1)/A2 %*	*Tempo de Exec.(s)*
c17	6	14.364	58.12	68.15	**14,7**	0.36
c499	218	844.284	634.40	872.28	**27,2**	42.47
c1355	218	844.284	634.40	872.28	**27,2**	41.61
c1908	269	915.306	626.41	1017.51	**38,4**	60.10
c6288	1524	5012.272	2669.57	3717.95	**28,2**	2243.83

5.3 Validação do Projeto

Para validar os resultados obtidos, a tabela 5.3 poderia fornecer uma ideia razoável sobre a eficiência da programação geométrica, pois nota-se uma boa melhoria no valor do atraso quando comparamos o circuito dimensionado e não dimensionado, para uma mesma área. Pode-se ir mais além, e comparar os resultados com aqueles obtidos através de uma ferramenta comercial. Entretanto, a comparação pode não ser válida. As seguintes considerações devem ser levadas em conta antes que qualquer conclusão precipitada seja feita:

- Os modelos RC e de esforço lógico para o atraso de portas lógicas são muito simplificados, e não levam em consideração importantes aspectos de simulação, como por exemplo, a forma de onda na entrada O método de obtenção do atraso poderia ser mais preciso caso fossem utilizados os modelos mais completos, como CCS, NLDM e ECSM.
- Na biblioteca *open-cell*, as implementações de força maior não aumentam a área total da célula. No modelo proposto nesse trabalho, a área da célula é proporcional à força da célula. Nesse sentido, qualquer comparação em relação à área total seria negligente.
- Não foi considerada a área das interconexões, o que aparentemente faria o modelo proposto ganhar em diversas comparações de área.

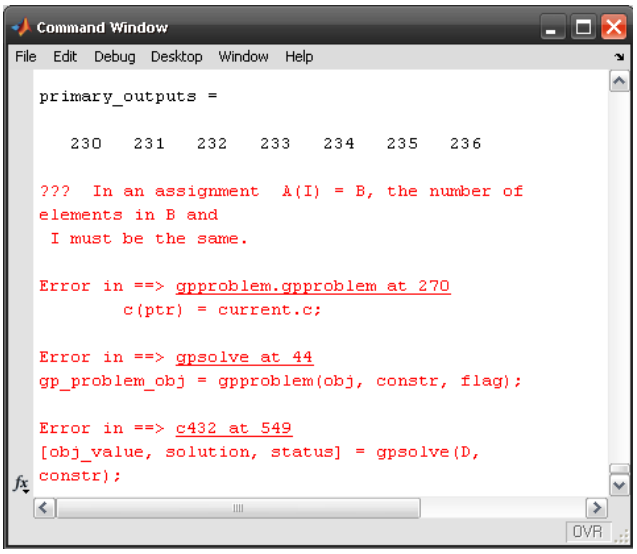


Figura 5.4: Tentativa de resolução do circuito c432

Em vista destes aspectos, uma comparação circuito a circuito com outra ferramenta não seria plausível. O que pode ser feito é a observação de como comportam-se as saídas de um produto existente, desde o mapeamento tecnológico à análise estática de tempo. A comparação, portanto, seria no sentido de verificar os pontos onde o projeto pode ser melhorado, e observar onde estão os acertos.

Encounter RTL Compiler, da Cadence Inc., produtora de softwares EDA, é uma ferramenta que realiza toda a síntese lógica, a partir de uma descrição estrutural de um circuito e uma biblioteca. Utilizando-se o programa, foi feita a síntese de todos os circuitos de teste, juntamente com a biblioteca *open-cell* da Nangate Inc., implementação *corner typical*. Não foram introduzidas restrições. Após, com dois comandos para análise de tempo e área, obtém-se alguns Na tabela 5.4, está o resultado da síntese.

Nota-se que o número de células obtidas no Encounter é menor em vista da biblioteca utilizada no projeto ser a *mcnc,genlib*, modificada para a tecnologia 45nm.

A otimização para o atraso ocorre baseada no caminho crítico. É uma desvantagem em comparação ao método proposto, pois a programação geométrica consegue otimizar

todas as células do circuito, pois percorre-se o circuito recursivamente por todos os caminhos. Entretanto, a programação geométrica mostrou-se mais lenta, justamente por causa da otimização global, e não apenas em um determinado caminho.

Tabela 5.4: Resultados obtidos no Cadence Encounter RTL Compiler

Circuito	*Número de Células*	*Área das Células (nm²)*	*Área das Conexões (nm²)*	*Atraso (ps)*	*Tamanho do Caminho Crítico*	*Tempo de Execução (s)*
c17	6	5	2	101	4	0.24
c432	102	111	48	1858	24	4.79
c499	170	229	67	1120	14	4.22
c880	196	274	82	1251	22	11.55
c1355	170	229	67	1107	15	7.73
c1908	212	251	72	1331	20	10.93
c2670	277	331	141	1277	20	9.67
c3540	531	585	283	2148	30	21.11
c5315	559	769	309	1801	21	18.09
c6288	1022	1705	371	5486	69	27.44
c7552	776	983	340	2640	29	22.68

5.4 Análise Final e Perspectiva de Trabalhos Futuros

Ao que o trabalho propôs-se, apesar das dificuldades encontradas, um bom resultado foi obtido. Comparando-se dois circuitos de mesma área, um dimensionado e o outro não dimensionado, as reduções no atraso variaram entre 14% e 38%. Com esse resultado, pode concluir-se que a programação geométrica mostrou-se bastante eficiente para resolver o problema de dimensionamento.

Talvez o tempo de execução não tenha sido um ponto a favor da ferramenta proposta. Para o circuito **c6288**, por exemplo, o tempo de execução, em comparação com os resultados do Encounter, foi muito desfavorável. A resolução de um problema que envolva muitas células deve, a princípio, demorar muito tempo. Para tentar contornar este problema, o circuito pode ser dividido em circuitos menores, e o dimensionamento realizado nesses novos circuitos.

Ainda em relação ao tempo de execução, deve ser ressaltado que a resolução de outros algoritmos é mais simples, do ponto de vista da execução computacional. A obtenção de caminhos críticos (utilizada por grande parte destes algoritmos) não leva em consideração todas as células do circuito ao mesmo tempo, e não é garantido que a solução é a melhor possível. Embora a *solução ótima* não seja uma exigência na síntese lógica, à medida que certas limitações computacionais desaparecerem, os algoritmos baseados em programação geométrica poderão ganhar mais espaço nesta área.

Um questionamento que surgiu durante o trabalho foi a possibilidade de tentar impor

como restrição o *menor atraso*. Tal restrição exige que o atraso em qualquer caminho do circuito tenha um valor mínimo. Com isso, caminhos que são muito rápidos e não são críticos poderiam ser otimizados para área. Verificou-se que isso não é possível, pois de acordo com a forma de um PPGG, a função *mínimo* está restrita a monômios, e a expressão para o atraso, como foi visto, é posinomial.

Para futuros trabalhos, esperam-se várias melhorias, as quais são esboçadas abaixo.

Em relação ao cálculo do atraso nas portas lógicas, espera-se a utilização de modelos mais realísticos para o atraso, como os já citados CCS, ECSM e NLDM, considerados padrões da indústria atual. Desta forma, uma maior fidelidade para os resultados seria obtida. Para utilizar os modelos, bastaria analisar uma biblioteca caracterizada (por exemplo, uma biblioteca no modelo Liberty) para obter o atraso na célula (ou seja, buscar em uma tabela o valor do atraso). Essa é uma tarefa simples, que a própria rotina de resolução do PG poderia implementar.

Outro modelo que pode ser mais bem utilizado e estudado é o de esforço lógico. O cálculo de atraso na célula poderia ser estendido para o cálculo do atraso nos caminhos do circuito. Atualmente, algoritmos que utilizam esforço lógico calculam o atraso no(s) caminho(s) crítico(s), para após começar a rotina de otimização. Caso fosse utilizada programação geométrica, todos os caminhos, e não apenas os caminhos críticos, seriam pelo menos considerados. Como na abordagem acima, a modelagem do problema não seria difícil,bastando a rotina implementar as fórmulas de cálculo do esforço lógico e percorrimento no grafo que representa o circuito.

Ao que tange a programação geométrica, outras restrições poderiam ser incluídas, como por exemplo a potência. A potência em circuitos digitais constitui-se de duas componentes: a *potência dinâmica*, que depende das capacitâncias, da tensão de alimentação e da freqüência de transição de chaveamento (pode ser obtida através de uma simulação com o vetor de entrada que for desejado), e a *potência estática*, esta dependendo da corrente de fuga. À primeira vista, o problema de potência parece ser mais complicado, mas basta uma correta modelagem das equações para que estas sejam parte das restrições de um PG. Além da potência, outra restrição – capacitância máxima de carga (limitando-se a soma das capacitâncias de entrada do conjunto *fanout*) – poderia ser incluída.

Finalmente, poderiam ser aproveitadas as estruturas de dados, onde se percorre recursivamente o circuito, para a concepção de um método onde o mapeamento tecnológico pudesse ser realizado juntamente com o dimensionamento. A partir da divisão do circuito em várias áreas de cobertura, as melhores células dentro dos caminhos poderiam ser escolhidas, mapeando cada área, de modo a diminuir o atraso. Este problema é considerado NP-difícil, portanto são utilizadas heurísticas, como por exemplo a programação dinâmica. Com o auxílio da programação geométrica e a modelagem em corretas restrições, seria possível obter uma solução ótima. Desta maneira, uma ferramenta completa de síntese lógica utilizando-se programação geométrica poderia vir a ser elaborada.

REFERÊNCIAS

BERKELEY CAD GROUP. **Berkeley Logic Interchange Format (BLIF**). Disponível em: <http://www.cad.eecs.berkeley.edu/Respep/Research/vis/blif.ps>. Acesso em: out. 2009.

BEKELEY CAD GROUP. **SIS: A System for Sequential Circuit Synthesis**. Disponível em: <ftp://ic.eecs.berkeley.edu/pub/Sis/SIS_paper.ps.Z>. Acesso em: out. 2009.

BERKELEY LOGIC SYNTHESIS AND VERIFICATION GROUP. **ABC: A System for Sequential Synthesis and Verification**. Disponível em: <http://www.eecs.berkeley.edu/~alanmi/abc/>. Acesso em: nov. 2009.

BOYD, S. et al. **A tutorial on geometric programming**. Optimization and Engineering 8, 2007. p.67-127.

BOYD, S.; KIM, S. J. **Geometric Programming for Circuit Optimization.** In Prooceedings of International Symposium on Physical Design (ISPD), 2005. p.44-46.

BRGLEZ, F.; FUJIWARA, H. **A neutral netlist of 10 combinatorial benchmark circuits and a target translator in FORTRAN**. Int. Symposium on Circuits and Systems. Sessão especial sobre ATPG e Simulação de Falhas, 1985.

CADENCE INC. **Encounter RTL Compiler**. Disponível em: <http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx>. Acesso em: nov. 2009.

CHEN, W. C.; HSEIH, T.; PEDRAM, M. **Simultaneous gate sizing and placement**. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19(2), 2000. p.206-214.

CHU, C.; WONG, D. **VLSI circuit performance optimization by geometric programming**. Annals of Operations Research, 105, 2001. p.37-60.

COUDERT, O.; HADDAD, R. **New Algorithms for Gate Sizing: A Comparative Study**. Proc. of 33rd DAC, 1996. p.734-739.

DUFFIN, R. J.; PETERSON, E. L.; ZENER, C. **Geometric Programming: Theory and Applications**. Wiley, 1967.

ELMORE, W.C. **The Transient Analysis of Damped Linear Networks with**

Particular Regard to Wideband Amplifiers. J. Applied Physics, vol. 19(1), 1948.

FISHBURN J.; DUNLOP, A. **TILOS: a posynomial programming approach to transistor sizing**. IEEE international conference on computer-aided design: ICCAD-85. Digest of technical papers. IEEE Computer Society Press, 1985. p.326-328.

HANSEN, M.; YALCIN, H.; HAYES, J. **ISCAS High-Level Models** Disponível em: <http://www.eecs.umich.edu/~jhayes/iscas/>. Acesso em: out. 2009.

JOHNSON, S. C. **YACC: Yet Another Compiler Compiler**. Computing Science Technical Report 32, Bell Laboratories, Murray Hill, 1975.

KEUTZER, K. **DAGON: Technology Binding and Local Optimization by DAG Matching**. Proc. IEEE/ACM. DAC, 1987. P.341-347.

KUKIMOTO, Y. et al. **Delay Optimal Technology Mapping by DAG Covering**. Proc. IEEE/ACM DAC, 1998. p.348-351.

LESK, M. E. **Lex - a Lexical Analyzer Generator**. Computing Science Technical Report 39, Bell Laboratories, Murray Hill, 1975.

NAGEL, L. W.; PEDERSON, D. O. **SPICE (Simulation Program with Integrated Circuit Emphasis)**. Memorandum ERL-M382, University of California, Berkeley, 1973.

NANGATE INC. **Nangate 45nm Open Cell Library**. Disponível em: <http://www.nangate.com/index.php?option=com_content&task=view&id=137&Itemid=137>. Acesso em: nov. 2009.

NESTEROV, Y.; NEMIROVSKY, A. **Interior-point polynomial methods in convex programming**. Studies in applied mathematics, vol 13. SIAM, Philadelphia, 1994.

PATTANAIK, M.; BANERJEE, S.; BAHINIPATI, B. **GP based transistor sizing for optimal design of nanoscale CMOS inverter**. IEEE Conference on Nanotechnology, 2003. p. 524-527.

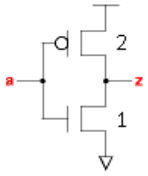
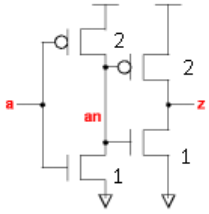
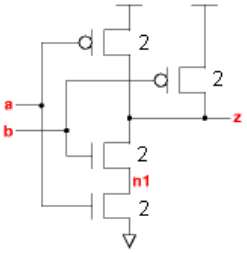
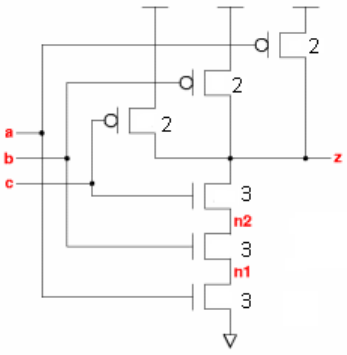
SUTHERLAND, I.; SPOULL, R. F.; HARRIS, D. **Logical Effort: Designing Fast CMOS Circuits**. Morgan Kaufmann, 1999.

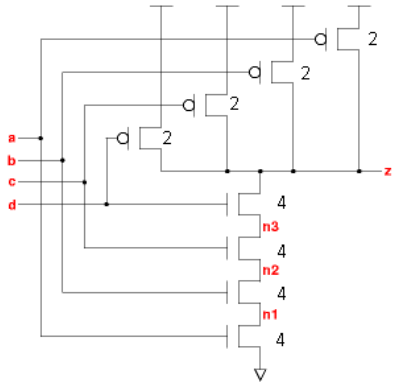
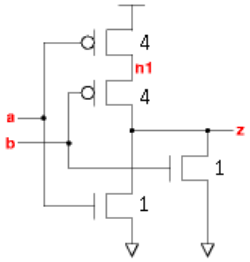
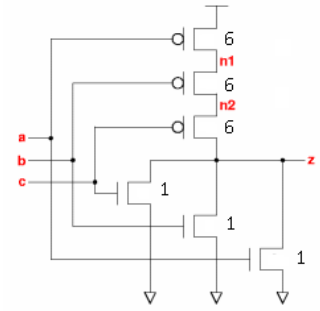
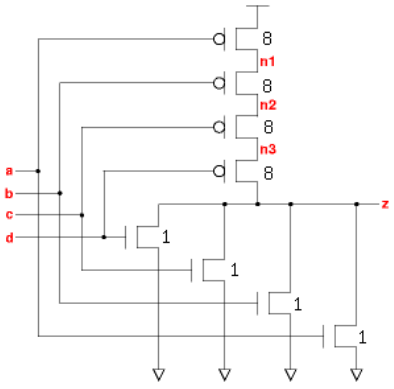
SYNOPSYS, INC. **Open Source Liberty.** Disponível em: <http://www.opensourceliberty.org/>. Acesso em: nov. 2009.

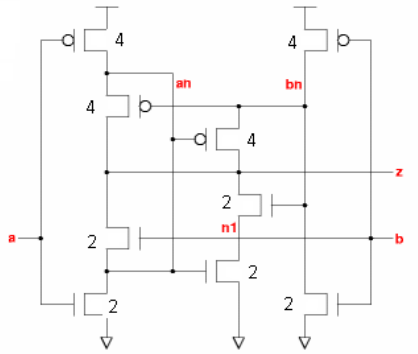
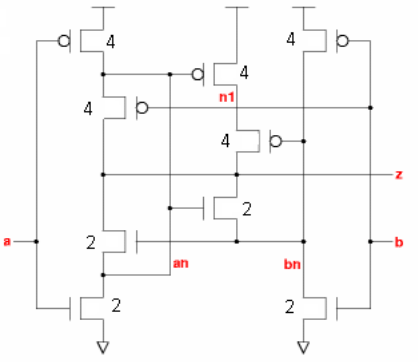
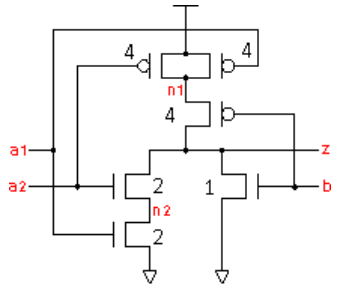
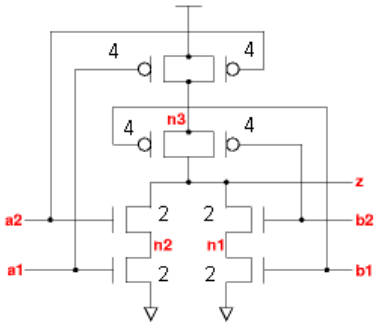
WESTE, N.; HARRIS, D. **CMOS VLSI Design A Circuits and Systems Perspective**. Terceira Edição. Addison Wesley, 2003.

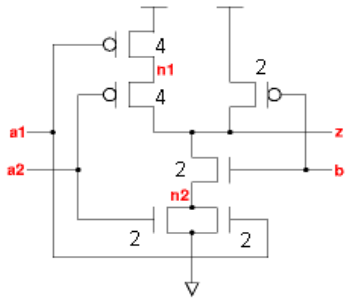
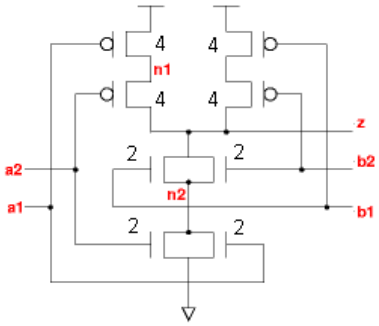
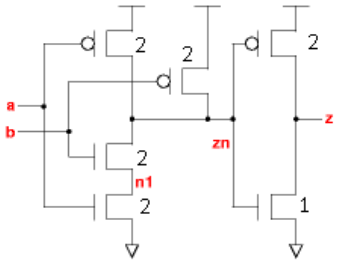
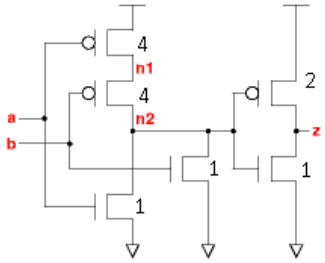
YANG, S. **Logic Synthesis and Optimization Benchmarks User Guide**. Technical Report 1991-IWLS-UG-Saeyang, MCNC, Research Triangle Park, 1991.

APÊNDICE A: CÁLCULO DO ESFORÇO LÓGICO

Célula	*Esforço Lógico*	*Atraso Parasita*	*Esquema Lógico*
inv1	a: 3/3	3/3	
buf1	a: 3/3	3/3	
nand2	a: 4/3 b: 4/3	6/3	
nand3	a: 5/3 b: 5/3 c: 5/3	9/3	

Célula	*Esforço Lógico*	*Atraso Parasita*	*Esquema Lógico*
nand4	a: 6/3 b: 6/3 c: 6/3 d: 6/3	12/3	
nor2	a: 5/3 b: 5/3	6/3	
nor3	a: 7/3 b: 7/3 c: 7/3	9/3	
nor4	a: 9/3 b: 9/3 c: 9/3 d: 9/3	12/3	

Célula	*Esforço Lógico*	*Atraso Parasita*	*Esquema Lógico*
xor2	a: 6/3 b: 8/3	12/3	
xnor2	a: 6/3 b: 10/3	12/3	
aoi21	a1: 6/3 a2: 6/3 b: 5/3	7/3	
aoi22	a1: 6/3 a2: 6/3 b1: 6/3 b2: 6/3	12/3	

Célula	*Esforço Lógico*	*Atraso Parasita*	*Esquema Lógico*
oai21	a1: 6/3 a2: 6/3 b: 4/3	8/3	
oai22	a1: 6/3 a2: 6/3 b1: 6/3 b2: 6/3	12/3	
and2	a: 4/3 b: 4/3	3/3	
or2	a: 5/3 b: 5/3	3/3	

APÊNDICE B: ARQUIVO C17.M

```
clear all; close all;
PLOT_TRADEOFF = 0;%Se estiver com o valor 1, a curva atraso vs. área é gerada

%*****************************Dados do Circuito*****************************

%Modelo RC para as células da biblioteca: arquivo 'table.txt'
buf1   = struct('Cin',0.46,'Cint',0.44,'Rdrv',21.73,'A',0.532);
inv1   = struct('Cin',0.48,'Cint',0.45,'Rdrv',51.42,'A',0.798);
nand2  = struct('Cin',0.52,'Cint',0.53,'Rdrv',26.22,'A',0.798);
nand3  = struct('Cin',0.59,'Cint',0.61,'Rdrv',28.98,'A',1.064);
nand4  = struct('Cin',0.66,'Cint',0.69,'Rdrv',30.05,'A',1.330);
nor2   = struct('Cin',0.57,'Cint',0.52,'Rdrv',29.25,'A',0.798);
nor3   = struct('Cin',0.68,'Cint',0.61,'Rdrv',37.07,'A',1.064);
nor4   = struct('Cin',0.72,'Cint',0.62,'Rdrv',43.26,'A',1.330);
and2   = struct('Cin',0.55,'Cint',0.51,'Rdrv',51.56,'A',1.064);
or2    = struct('Cin',0.63,'Cint',0.56,'Rdrv',58.45,'A',1.064);
xor    = struct('Cin',1.12,'Cint',1.01,'Rdrv',46.26,'A',1.596);
xnor   = struct('Cin',1.17,'Cint',0.99,'Rdrv',37.57,'A',1.596);
aoi21  = struct('Cin',0.59,'Cint',0.49,'Rdrv',41.59,'A',1.064);
aoi22  = struct('Cin',0.66,'Cint',0.57,'Rdrv',47.13,'A',1.330);
oai21  = struct('Cin',0.57,'Cint',0.45,'Rdrv',48.03,'A',1.064);
oai22  = struct('Cin',0.60,'Cint',0.54,'Rdrv',49.58,'A',1.330);
zero   = struct('Cin',0.01,'Cint',0.01,'Rdrv',1.00,'A',0.564);
one    = struct('Cin',0.01,'Cint',0.01,'Rdrv',1.00,'A',0.564);

m = 6;                                  %Número de Portas Lógicas
Amax = 14.364;                          %Área Máxima do Circuito

%Portas Lógicas lidas do arquivo 'c17_mapped.blif'
gates(1) = nand2;
gates(2) = nand2;
gates(3) = nand2;
gates(4) = nand2;
gates(5) = nand2;
gates(6) = nand2;

%Índices das entradas e saídas primárias
primary_inputs = [ 7 8 9 10 11 ]
primary_outputs = [ 12 13 ]

%M é o total de portas lógicas, entradas e saídas primárias do circuito
M = m + length( primary_inputs ) + length( primary_outputs );

%Matriz de Fan-in
FI = cell(M,1);
FI{1} = [ 9 7 ];
FI{2} = [ 10 9 ];
FI{3} = [ 2 8 ];
FI{4} = [ 3 1 ];
FI{5} = [ 2 11 ];
FI{6} = [ 5 3 ];
FI{7} = [ ];
FI{8} = [ ];
```

```
FI{9} = [ ];
FI{10} = [ ];
FI{11} = [ ];
FI{12} = [ 4 ];
FI{13} = [ 6 ];

%Capacitâncias de entrada das saídas primárias
Cin_po = sparse(M,1);
Cin_po(primary_outputs) = [ 5 5 ];

%Capacitâncias de carga das entradas primárias
Cload_pi = sparse(M,1);
Cload_pi(primary_inputs) = [ 5 5 5 5 5 ];

%*******************************Dados Obtidos*********************************
%Matriz de Fan-out obtida a partir da matriz de Fan-in
FO = cell(M,1);
for gate = [1:m primary_outputs]
  preds = FI{gate};
  for k = 1:length(preds)
    FO{preds(k)}(end+1) = gate;
  end
end

Cin_norm  = [gates.Cin]';              %Capacitâncias, resistência e área
Cint_norm = [gates.Cint]';             %para as células com implementação
Rdrv_norm = [gates.Rdrv]';             %unitária
A_norm = [gates.A]';

%*******************************Otimização************************************
gpvar x(m)                             %fator de dimensionamento
gpvar T(m)                             %arrival times

Cin  = Cin_norm.*x;                    %Capacitâncias e resistência são
Cint = Cint_norm.*x;                   %função do fator de dimensionamento
R = Rdrv_norm./x;                      %para cada célula

%Cálculo das capacitâncias de carga para cada célula
Cload = posynomial;
for gate = 1:m
  if ~ismember(FO{gate},primary_outputs)
    Cload(gate) = sum(Cin(FO{gate}));  %Soma das cap. das células no Fan-out
  else
    Cload(gate) = Cin_po(FO{gate});    %Exceção se for saída
  end
end
Cload = Cload';

D = 0.69*ones(m,1).*R.*(Cint+Cload);   %Atraso em cada célula
area = A_norm'*x;                      %Área total do circuito
x_constr = ones(m,1) <= x;             %Todas células maiores que 1
timing_constr = [];                    %Restrições de atraso em todas células

for gate = 1:m
  if ~ismember( FI{gate}, primary_inputs )
    for j = FI{gate}      %Atraso
      timing_constr =  [timing_constr; D(gate) + T(j) <= T(gate)];
    end
  else                    %Entrada: não é necessário analisar antes
    timing_constr = [timing_constr; D(gate) <= T(gate)];
  end
end

%Função-objetivo: O atraso máximo do circuito será o máximo entre os atrasos
nas células ligadas às saídas primárias
output_gates = [FI{primary_outputs}];
D = max( T(output_gates) );
```

```
%'constr' possui todas restrições (área máx, atraso na célula e tamanho mín)
constr = [area <= Amax; timing_constr; x_constr];

%Solucão do problema

tic;                                    %Solucionar o problema, com 'gpsolve'
[obj_value, solution, status] = gpsolve(D, constr);
assign(solution);                       %Dar valor à solução
toc;                                    %Tempo de execução

Amax_plot = Amax;
ckt_delay_plot = obj_value;
fprintf(1,'\nAtraso ótimo do circuito para Amax = %d Ã© %3.4f.\n', ...
        Amax, obj_value)                %Mostrar o atraso ótimo
disp('Fatores de escala ótimos são: ')
x                                       %Mostrar a área ótima para cada célula

%**************************Curva Atraso vs. Área*****************************
if( PLOT_TRADEOFF )
N = 20;                                 %Número de pontos da curva
Amax = linspace(7.182,28.728,N);        %Limites: metade e dobro da área máx.
min_delay = zeros(N,1);

global QUIET; QUIET = 1;                %Solucionar sem mensagens do solver

for n = 1:N
  constr(1) = area  <= Amax(n);         %Mudar o valor da área a cada iteração
  [obj_value, solution, status] = gpsolve(D, constr); %solucionar
  if strcmp(status,'Infeasible')
    error('Problema é infazível!')
  end
  min_delay(n) = obj_value;             %Armazenar o atraso obtido
end

global QUIET; QUIET = 0;                %Ativar novamente mensagens do solver

plot(Amax,min_delay); hold on;          %Mostrar a curva atraso vs. área
plot(Amax_plot, ckt_delay_plot,'o'); hold off;
xlabel('Amax'); ylabel('Dmin');
title(['Curva Dmin x Amax'])

end
% end tradeoff curve code
```