UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

VINICIUS CALLEGARO

SwitchCraft: Um ambiente computacional para síntese e análise de redes lógicas

Trabalho de Graduação.

Prof. Dr. Renato Perez Ribas
Orientador

Prof. Dr. Leomar Soares da Rosa Junior
Co-orientador

Porto Alegre, dezembro de 2009.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

FPGA	Field Programmable Gate Array
CPLD	Complex Programmable Logic Device
CAD	Computer-Aided Design
BDD	Binary Decision Diagram
LBBDD	Lower-Bound BDD
SOP	Sum-of-Products
POS	Product-of-Sums
CMOS	Complementary Metal-Oxide-Semiconductor
NMOS	nFET Metal Oxide Silicon
DCVSL	Differential Cascade Voltage Switch Logic
PTL	Pass Transistor Logic
PMOS	pFET Metal Oxide Silicon
UML	Unified Modeling Language

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O ambiente SwitchCraft provê um conjunto de ferramentas para geração de redes de chaves lógicas. Estimativas para atraso de propagação de sinais, área e dissipação de energia (dinâmica ou corrente de fuga) também estão disponíveis.

A plataforma é amigável e permite a construção de scripts, agrupando seqüências de comandos. Redes de transistores correspondendo a funções lógicas alvo podem ser geradas de equações e de BDDs. Redes logicamente e topologicamente complementares podem ser derivadas através de métodos baseados em grafos duais.

Diferentes estilos lógicos CMOS podem ser obtidos, por exemplo, *single* ou *dual-rail*, topologias estáticas ou dinâmicas, com planos disjuntos (*pull-up* PMOS / *pull-down* NMOS) ou em uma estrutura tipo PTL (com estruturas *pull-up/pull-down* compartilhadas).

Palavras-Chave: CAD, redes lógicas, circuitos digitais, BDD, CMOS.

SwitchCraft: A computer environment for switch network synthesis and analysis.

ABSTRACT

SwitchCraft environment provides a set of tools for switch network generation. Estimators for delay, area and power dissipation (dynamic and leakage) are available.

The platform is easy-to-use and it allows the construction of scripts grouping a sequence of commands. Transistor networks corresponding to target logic functions can be generated from equations and from BDDs. Logically and topologically complementary networks can be derived through dual-graphs.

Different CMOS logic styles can be obtained, e.g. single and dual-rail, static and dynamic topologies, with disjoint planes (pull-up PMOS / pull-down NMOS) and in a PTL-like structure (with a shared pull-up/pull-down structure).

Keywords: CAD, switch network, digital circuits, BDD, CMOS.

1 INTRODUÇÃO

Circuitos digitais estão cada vez mais presentes em nossas vidas. Seja para uso pessoal ou profissional, é notório o aumento do número de equipamentos eletrônicos que nos rodeia. Essa explosão deve-se principalmente à redução nos custos de fabricação e miniaturização dos componentes eletrônicos, resultado do avanço da tecnologia na concepção de circuitos integrados.

Naturalmente espera-se que produtos cada vez mais complexos sejam gerados pelo aumento da capacidade de integração das novas tecnologias de fabricação. Circuitos estão utilizando cada vez mais elementos, fazendo com que passem a existir novos desafios, métodos e ferramentas que auxiliem no desenvolvimento e fabricação desses circuitos.

Atualmente, para o desenvolvimento de sistemas em hardware, existem três principais plataformas: microcontroladores, componentes programáveis (FPGAs e CPLDs) e de circuitos integrados dedicados. Circuitos integrados digitais podem ser compostos utilizando-se portas lógicas, as quais operam com uma ou mais entradas para produzir uma saída booleana verdadeira ou falsa.

Essas portas lógicas são constituídas por redes de transistores que seguem a matemática binária para representar uma função lógica, sendo que essas redes podem ser vistas como arranjos elétricos. Retirando-se apenas a informação lógica desses arranjos elétricos, criam-se redes lógicas. Assim, existe uma rede que representa a função lógica que uma porta lógica implementa, porém não depende mais da tecnologia empregada na construção da porta lógica.

O elemento básico utilizado para implementar uma rede lógica é a chave, a qual pode ser classificada em dois tipos: chave positiva, que permite a passagem da informação desde que seja aplicado o nível lógico alto em seu terminal de controle e chave negativa, que transmite informação desde que, em seu terminal de controle, seja aplicado o nível lógico baixo.

Assim, a rede lógica permite que se trabalhe com certo nível de abstração, pois suas chaves podem ser facilmente convertidas para qualquer estilo de tecnologia de fabricação, como por exemplo, transistores, formando assim uma rede de transistores. Essa abstração é importante, pois diversas ferramentas que auxiliam no desenvolvimento dos circuitos integrados trabalham no nível lógico, ficando assim, independente da tecnologia que será utilizada na fabricação dos circuitos.

Existem diversas ferramentas que auxiliam projetistas em geral no desenvolvimento de seus trabalhos. Essas ferramentas são conhecidas como CAD (*Computer-Aided Design*), e são geralmente utilizadas em projetos nas mais diversas áreas da engenharia. Com a atual demanda por projetos de circuitos integrados complexos crescendo

vertiginosamente, é imprescindível que sejam utilizadas tais ferramentas que automatizem e auxiliem o trabalho dos projetistas, uma vez que tais projetos não são mais humanamente factíveis, seja em relação a custos ou em relação a tempo de projeto.

Neste contexto, surge a necessidade da geração de novos algoritmos para geração de redes, bem como utilização de ferramentas CAD que implementem tais algoritmos e possam prover um fluxo de trabalho automatizado.

1.1 Histórico

Os primeiros trabalhos na área datam dos anos 30, quando Claude E. Shannon iniciou seus estudos (SHANNON, 1938). Sua principal idéia era utilizar elementos eletromecânicos, como chaves mecânicas e relés para compor redes lógicas. Com o passar do tempo, muitas descobertas foram feitas acerca de materiais que poderiam ser utilizados na fabricação de elementos mais confiáveis e de maior capacidade de miniaturização, como os materiais semicondutores. Desde então chaves mecânicas e relés foram sendo substituídos por elementos mais baratos e mais confiáveis, como válvulas, diodos e transistores. Essa evolução permitiu que circuitos maiores e mais complexos pudessem ser desenvolvidos.

Desde os anos 30, quando Shannon iniciou seus estudos até os dias de hoje, diversos métodos foram propostos na literatura. Em 1981, Uehara (UEHARA, 1981) propôs um método de geração de redes utilizando o conceito de dualidade de grafos. O método parte de um grafo planar que representa uma rede de chaves lógicas. Com isso, uma rede topologicamente e logicamente complementar pode ser gerada a partir do dual do grafo da rede original.

Em 1993, Zhu (ZHU, 1993) propôs um método de geração de redes baseado em grafos. A idéia principal de seu método consiste em verificar todos os caminhos do grafo apenas quando realmente fosse necessário. Esta operação é realizada com o objetivo de verificar qual é a função lógica que o grafo representa, sendo sua complexidade exponencial, portanto bastante custosa em relação ao processamento computacional. Assim, sua abordagem consiste em detectar arranjos que podem ser otimizados e ainda realizar operações que são bem estabelecidas e seguras, não necessitando de uma verificação dos caminhos do grafo, pois não alteram a funcionalidade da rede. Ainda existem outras otimizações (que são realizadas em menor número) que necessitam da verificação de todos os caminhos do grafo para saber se a otimização comprometeu ou não a funcionalidade da rede. Além disso, seu método pode ainda gerar arranjos não série-paralelo, também conhecidos como arranjos do tipo *Wheatstone Bridge*. Geralmente, esses arranjos necessitam de um número menor de chaves para representar a mesma função lógica quando comparados com arranjos puramente série-paralelos. Naquela época, os resultados de Zhu foram bastante positivos em comparação com os outros já encontrados na literatura.

Outro método já bem estabelecido na literatura é a geração de redes utilizando BDDs. Essa estrutura de dados é constituída por nodos, arcos e terminais, sendo utilizada para representação de funções lógicas. O método LBBDD, proposto por (DA ROSA JUNIOR, 2008), utiliza-se dessas estruturas para gerar arranjos que respeitem o número mínimo de chaves em série necessárias e suficientes para representar uma dada função lógica (*lower-bound*) (SCHNEIDER, 2007). Além disso, outra característica importante é que este método pode gerar arranjos não série-paralelo.

Atualmente, um dos estados-da-arte em geração automática de redes de chaves foi proposto por Kagaris (KAGARIS, 2007). Este trabalho propõe uma metodologia que consegue gerar redes não série-paralelo, tendo como objetivo minimizar o número total de chaves utilizadas para representar um circuito. O método parte de uma soma-de-produtos (SOP) de uma dada equação, e para cada produto, tenta aplicar certas transformações de modo a descobrir dentre elas qual é a que tem o menor custo (em número de chaves) para inserir o produto na rede. Por ser um algoritmo guloso, é importante ressaltar que este método gera uma solução heurística, mas que se mostrou bastante eficiente.

É importante dizer que até a conclusão deste trabalho, não se encontrava na literatura uma ferramenta de CAD que pudesse gerar redes de chaves lógicas e fornecer estimativas das mesmas. Este fato serve como motivação para a conclusão do trabalho, uma vez que diversos usuários poderiam utilizar a ferramenta e gerar resultados através dela, fomentando assim a pesquisa no meio acadêmico.

1.2 Motivação

Atualmente, o laboratório de pesquisa Nangate-UFRGS Research Lab detém diversos protótipos de ferramentas que manipulam redes lógicas. Essas ferramentas, por sua vez, funcionam de forma *stand-alone*, isto é, são protótipos que foram construídos com um propósito e executam sua tarefa sem interferir nem se integrar a outras ferramentas.

Existem diversos protótipos que geram redes lógicas utilizando uma série de metodologias distintas. Para tornar possível a comparação entre essas redes e até mesmo verificar a qualidade das mesmas, foram ainda criados protótipos que analisam a rede e extraem estimativas, como consumo de energia, área e atraso de propagação de sinais. Além desses protótipos, existem ainda ferramentas que fornecem uma maneira de inspecionar redes lógicas de forma visual. Todos esses protótipos geralmente utilizam-se de estruturas de dados próprias, sendo incompatíveis entre si.

Entender qual a metodologia por trás desses protótipos, o que utilizam como entrada e quais resultados podem gerar como saída são alguns desafios interessantes e que motivam o andamento do trabalho, bem como aprender os métodos e a forma como foram implementados.Além disso, o esforço de integrar trabalhos anteriores em uma única ferramenta é sem dúvida uma grande experiência a ser adquirida. Existem as mais variadas formas em que estes trabalhos encontram-se no repositório de dados: sem documentação alguma, outros bem documentados. Alguns trabalhos são entendidos facilmente, seja por meio da documentação ou até mesmo pela legibilidade do código.

A motivação para a realização deste trabalho consiste em aplicar a computação em uma área específica, a de microeletrônica, exercitando os conhecimentos adquiridos ao longo do curso em um problema real. Orientação a objetos, teoria dos grafos, engenharia de software, compiladores, etc. serão conhecimentos importantes e imprescindíveis para a realização dessa ferramenta.

Por fim, este trabalho visa fornecer subsídios para que futuras implementações de outros algoritmos e métodos de geração, análise e representação de redes e portas lógicas possam ser facilmente agregados ao ambiente. Para isso, serão utilizadas metodologias de engenharia de software para que seja facilitada a manutenção futura do código em todo o projeto. Devido a grande produção científica gerada pelo laboratório,

existe a previsão de que o ambiente seja expandido, englobando outras aplicações que hoje encontram-se em desenvolvimento.

1.3 Proposta

A proposta deste trabalho consiste em desenvolver um ambiente que integre todos os métodos de geração de redes existentes no laboratório, bem como métodos de estimativas de redes, visualizadores, verificação e comparação lógica entre redes.Além disso, o ambiente deverá prover tanto uma *GUI* (interface gráfica interativa com o usuário) quanto uma interface em modo texto. Esta última será utilizada para fazer a conexão com outras ferramentas que fogem da proposta do trabalho ou não podem ser integradas à ferramenta, como por exemplo, ferramentas comerciais e ferramentas de outros laboratórios e universidades.

Independente da interface utilizada, seja ela gráfica ou não, será implementada uma linguagem de *scripts*, de modo que o ambiente possa converter os comandos em ações. Com isso, será possível a execução de *scripts* em lotes, acelerando a utilização da ferramenta, sendo que se pode criar um *script* uma vez e depois chamá-lo novamente para realizar a tarefa. Além disso, *scripts* provêem flexibilidade à ferramenta e facilitam o uso da mesma.

Existem módulos que implementam diversas metodologias distintas e com características particulares de geração de redes. Logo, um comparador entre redes é um módulo imprescindível. Esse comparativo pode ser em relação à função lógica que a rede representa, o número total de chaves, número máximo de chaves em série, estimativas de consumo de energia, atraso de propagação de sinais e de área, entre outras características que possam ser comparadas.

Apesar de existirem vários métodos de geração de redes disponíveis no repositório de dados, ainda existe uma lacuna a ser preenchida com o desenvolvimento de outros dois módulos que implementem a metodologia proposta em Zhu (ZHU, 1993) e (KAGARIS, 2007). A implementação desses dois métodos fornece uma boa base para futuras comparações com metodologias de geração de redes que estão sendo desenvolvidas no laboratório.

Por fim, é desejável que a ferramenta disponibilize um módulo que permita a edição de uma rede de modo gráfico e interativo, isto é, o usuário da ferramenta poderia criar redes utilizando apenas o mouse, e ao mesmo tempo verificando qual é a função lógica que esta rede em questão está representando.

1.4 Estrutura

Até agora, apresentamos alguns dados históricos e discutimos brevemente o objetivo do trabalho. No Capítulo 2 apresentaremos uma breve revisão técnica sobre conceitos de geração de redes lógicas e suas diversas abordagens. Os detalhes da proposta e os objetivos serão discutidos no Capítulo 3. No Capítulo 4 iremos mostrar como o projeto foi implementado. No Capítulo 5 apresentaremos os resultados dessa implementação, e finalmente no Capítulo 6 trataremos das conclusões do projeto e indicaremos trabalhos futuros a serem realizados.

2 CONCEITOS BÁSICOS

Com o passar dos anos, diversos métodos de geração de redes lógicas foram propostos na literatura. Este capítulo irá apresentar alguns desses métodos, bem como o método atualmente considerado o estado-da-arte.

2.1 Portas Lógicas (estilos e desempenho/estimativas)

Até agora apresentamos algumas características da geração de redes lógicas. Existem inúmeros estilos lógicos e formas distintas de implementar essas redes lógicas (ROSA JR, 2009). Esses estilos podem ser classificados entre topologia estática e dinâmica. Como exemplo do estilo lógico dinâmico, podemos citar *Domino* e suas variantes: *Dual Domino*, *Multiple-Output Domino*, *NORA Domino* e *Zipper Domino* (WESTE, 2005). Por outro lado, alguns dos estilos lógicos estáticos mais utilizados são CMOS, Pseudo-NMOS, DCVSL e PTL (RABAEY, 2005).

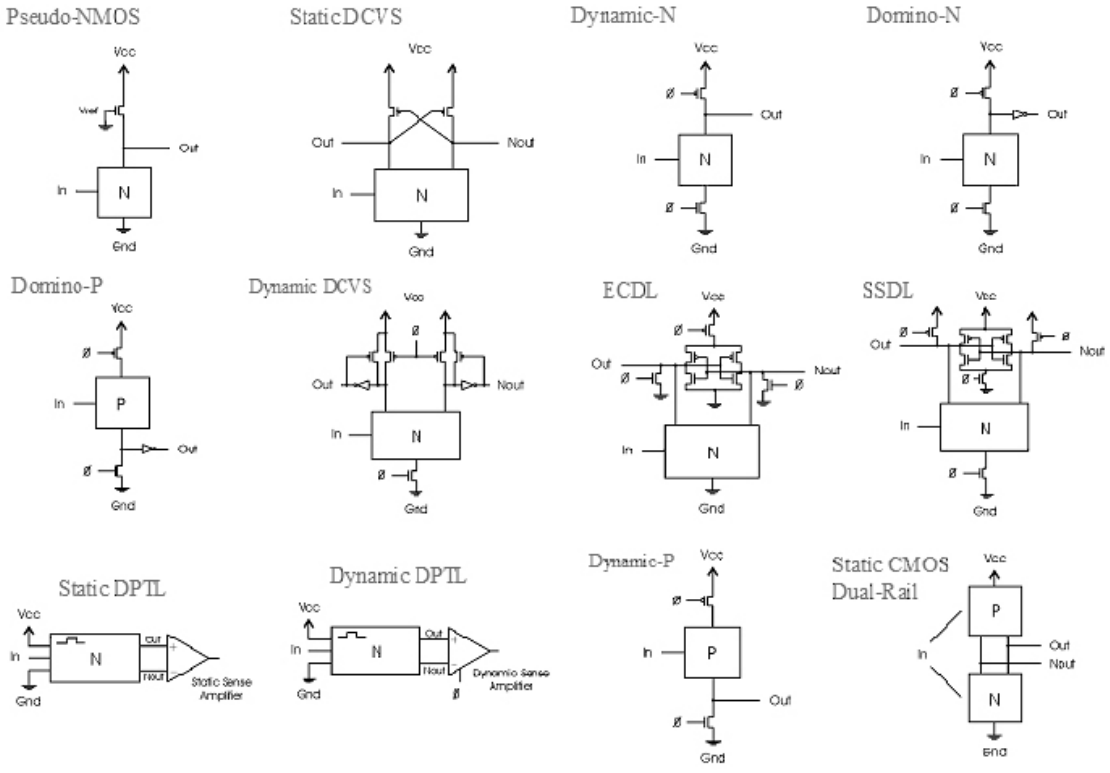


Figura 2.1: Alguns estilos lógicos disponíveis na literatura.

Esses estilos lógicos (Figura 2.1) apresentam, entre outras, diferentes características físicas e de desempenho elétrico. Uma forma de comparar esses estilos é realizar a verificação elétrica. Porém, tais verificações são custosas, e para amenizar esse problema, foram criadas estimativas para redes lógicas. Essas estimativas podem ser em relação à área, consumo e atraso que uma porta lógica possui.

2.2 Geração de Redes

2.2.1 Geração a partir da equação

A forma mais trivial de geração de redes é através da extração da rede lógica da equação que representa uma função lógica a ser implementada. Existem diversas formas de representar a equação de uma mesma função, sendo duas delas mais básicas: Como soma-de-produtos (SOP) ou como produto-de-somas (POS).

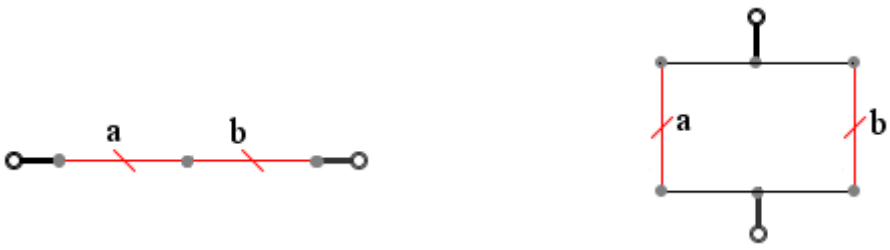
A equação (1) representa uma função arbitrária em forma de soma-de-produtos, e a equação (2) representa a mesma função lógica, porém, em forma de produto-de-somas.

$$f = (d*e)+(b*c*d)+(a*c*e)+(a*b) \quad (1)$$

$$f = (a+d)*(b+e)*(a+c+e)*(b+c+d) \quad (2)$$

Para a obtenção da rede lógica que a equação representa, uma tradução trivial para redes de chaves é baseada em três passos básicos:

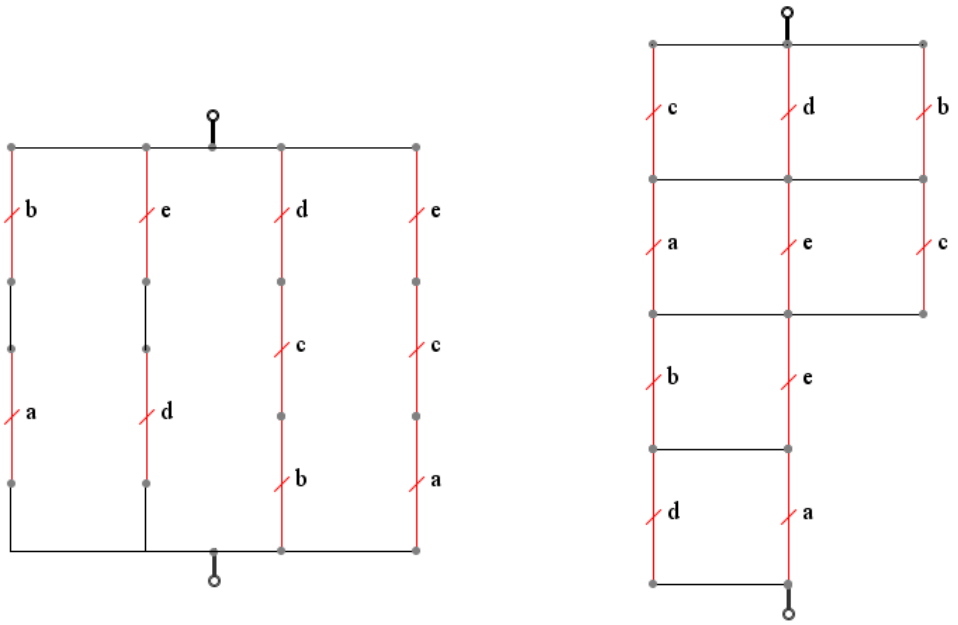
- Cada literal presente na equação é convertido em uma chave;
- Converter cada produto em arranjos de chaves em série (Figura 2.2-a);
- Converter soma em arranjo de chaves em paralelo (Figura 2.2-b).



(a) (b)

Figura 2.2: arranjo de chaves em série: (a) em série (b) em paralelo.

A rede obtida através da aplicação deste método na equação (1) e (2) pode ser vista na Figura 2.3-a e 2.3-b, respectivamente.



(a) (b)

Figura 2.3: rede lógica extraída da: (a) equação (1). (b) equação (2).

2.2.2 Fatoração e seu impacto

Existem ainda outras formas de representar uma mesma função lógica, porém, com o objetivo de reduzir o número total de literais na equação necessário para representar uma função. Esta abordagem é conhecida como fatoração de equações. Existem diversos trabalhos nessa área, propondo metodologias de fatoração, sendo que (REIS, 2009) é considerado atualmente um dos trabalhos estado-da-arte, juntamente com (GOLUMBIC, 2005). Utilizando como entrada a equação (1), aplica-se o método de fatoração proposto por (CALLEGARO, 2009) e chega-se ao resultado apresentado na equação (3).

$$f = d * (e + b * c) + a * (e * c + b) \quad (3)$$

Após a aplicação do algoritmo de fatoração, a equação que representa a mesma função lógica que (1) e (2) tem agora apenas 8 literais, contra 10 nas equações (1) e (2). Essa redução no número de literais é desejável nos casos em que se tem por objetivo reduzir o número total de chaves lógicas utilizadas para representar uma rede lógica, pois como apresentado anteriormente, o número de literais da equação é igual ao número de chaves utilizadas na rede. A rede gerada através da equação (3) pode ser vista na Figura 2.4.

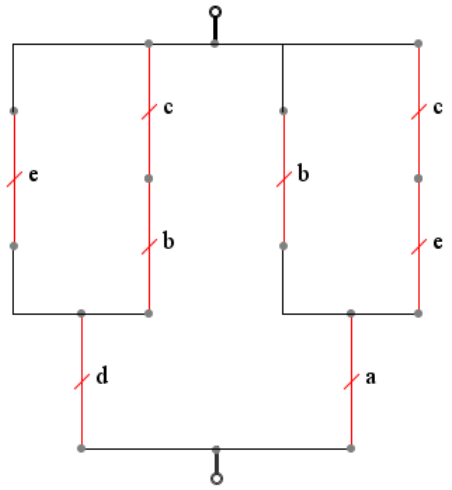


Figura 2.4: Rede lógica obtida através da equação (3).

Algoritmos de fatoração são largamente utilizados por gerarem boas soluções em um curto espaço de tempo. Porém, esta abordagem pode não atingir resultados ótimos em relação ao número de chaves necessárias para representar uma função lógica. Isto se deve ao fato de que a utilização de equações para o mapeamento de redes lógicas mencionado anteriormente não pode gerar arranjos do tipo não série-paralelo. Este tipo de arranjo tem a peculiaridade de apresentar chaves que não se pode afirmar se estão em série ou em paralelo.

Existem diversas funções para serem implementadas com um número mínimo de chaves necessitam de arranjos não série-paralelo para poderem ser representadas. Um exemplo é a função representada pela rede lógica presente na Figura 2.4, que pode ser representada utilizando apenas cinco chaves, como pode ser visto na Figura 2.6.

2.2.3 Geração de redes a partir de BDD

Uma outra forma de gerar redes de chaves é através da extração a partir de uma representação de grafo. Na abordagem de geração de redes a partir de BDDs, por exemplo, a ação de transformar o BDD em uma rede consiste em associar uma chave a cada arco de um nodo do BDD. Este conceito é ilustrado na Figura 2.5, que mostra um BDD (Figura 2.5-a) e uma das lógicas possível de ser gerada (Figura 2.5-b).

(a) (b)

Figura 2.5: (a) Nodo de um BDD. (b): uma possível associação para redes lógicas.

Este tipo de abordagem é interessante, pois pode apresentar como resultado tanto redes série-paralelo quando não série-paralelo (ROSA, 2006). Para que a função representada pela rede gerada por fatoração (Figura 2.4) possa ser reduzida em relação

ao número de chaves, pode ser utilizando o método LBBDD (DA ROSA JUNIOR, 2008). Por gerar tanto arranjos série-paralelo quanto não série-paralelo, este método consegue representar a rede utilizando apenas cinco chaves, como pode ser visto na Figura 2.6.

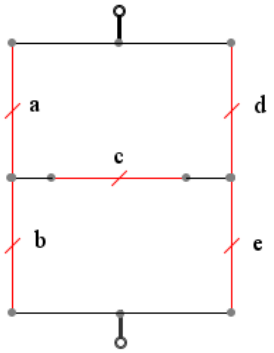


Figura 2.6: Rede lógica gerada por LBBDD que utiliza arranjo não série-paralelo.

Todas as chaves que compõe o arranjo caracterizam o arranjo como não série-paralelo (Figura 2.6), pois não se pode afirmar se as mesmas encontram-se em série ou em paralelo com as demais chaves da rede.

2.3 Redes Complementares

Apresentamos algumas formas de geração de redes, bem como suas principais características. Uma porta lógica é composta por um (redes não-disjuntas) ou dois planos (redes disjuntas). No estilo lógico CMOS os dois planos disjuntos possuem a característica de serem logicamente complementares. Planos logicamente complementares são aqueles que são dispostos em uma conFiguração tal que, quando um plano conduz o nível lógico alto para a saída, o outro não conduz o nível lógico baixo, e vice-versa. Caso os dois planos conduzam ao mesmo tempo, é gerado um curto-circuito na saída, comprometendo o funcionamento da porta lógica como um todo. Também é possível implementar arranjos de chaves com o intuito de nenhum plano conduzir dada uma determinada condição de entrada.

Existem diversas formas de gerar redes logicamente complementares, sendo que algumas se baseiam na estratégia de derivar a rede logicamente complementar utilizando os zeros da tabela verdade da função. Uma maneira alternativa consiste em utilizar uma rede existente para efetuar a extração da sua rede logicamente complementar

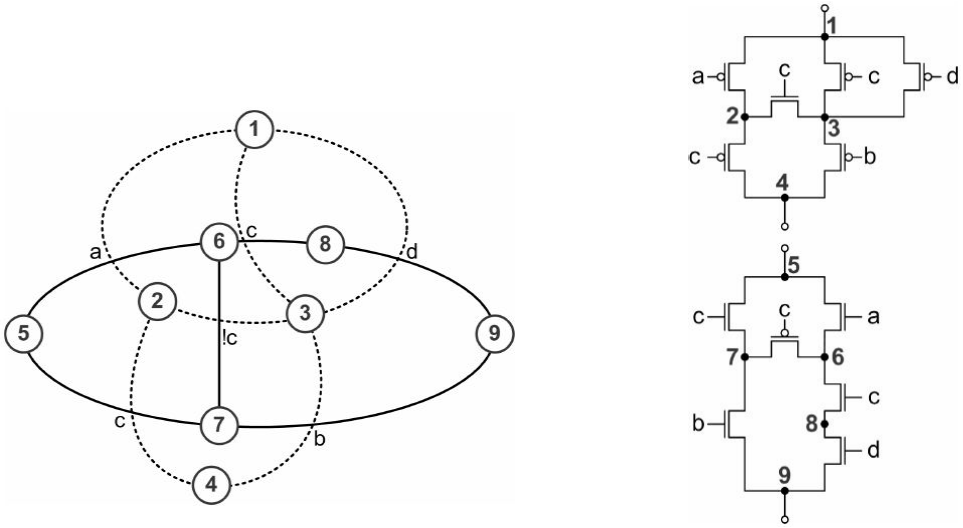
2.3.1 Geração de rede complementar puramente série-paralelo

Esta abordagem parte de uma rede pré-existente composta puramente por arranjos série-paralelo para gerar sua rede complementar. Para aplicação do método, todas as chaves que estão ligadas em série são convertidas para arranjos em paralelo na rede complementar, assim como chaves em paralelo são convertidas para arranjos em série. Esta é a abordagem mais simples de geração de redes complementares.

2.3.2 Grafo dual

Existe uma abordagem que funciona tanto para arranjos puramente série-paralelo quanto para arranjos não série-paralelo. Esta abordagem parte de uma rede pré-

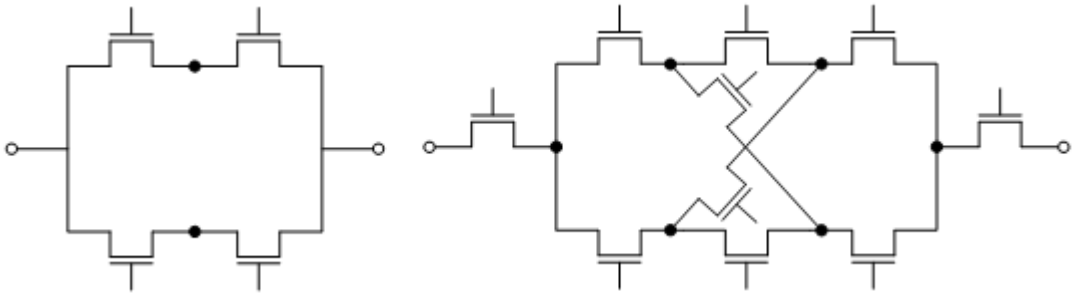
existente, traduzindo a rede em um grafo, sendo pontos de interconexão convertidos em nodos e chaves em arestas. Para que o método seja aplicado com sucesso, é necessário que o grafo resultante da tradução da rede seja um grafo planar. Após a transformação, é gerado o grafo dual do grafo original (CALLEGARO, 2008), que é obtido através do mapeamento de faces para vértices, conectando esses vértices através das arestas que separam as faces (Figura 2.7-a).



(a) (b)

Figura 2.7: (a) Geração de grafo dual. (b) Mapeamento de grafo para rede de chaves.

Após isso, aplica-se o caminho de tradução inversa, convertendo o grafo dual em uma rede de chaves. Esta nova rede é logicamente complementar a rede original, sendo ainda topologicamente complementar (Figura 2.7-b). É importante ainda ressaltar que para um dado grafo deve ser planar, não podem existir cruzamento de arestas no plano. No contexto de redes de chaves, ainda é importante que os nodos que representam os terminais da rede estejam na face externa do grafo, caso contrário o grafo não é considerado um grafo planar. A Figura 2.8 ilustra uma rede planar (Figura 2.8-a) e uma rede não-planar (Figura 2.8-b).

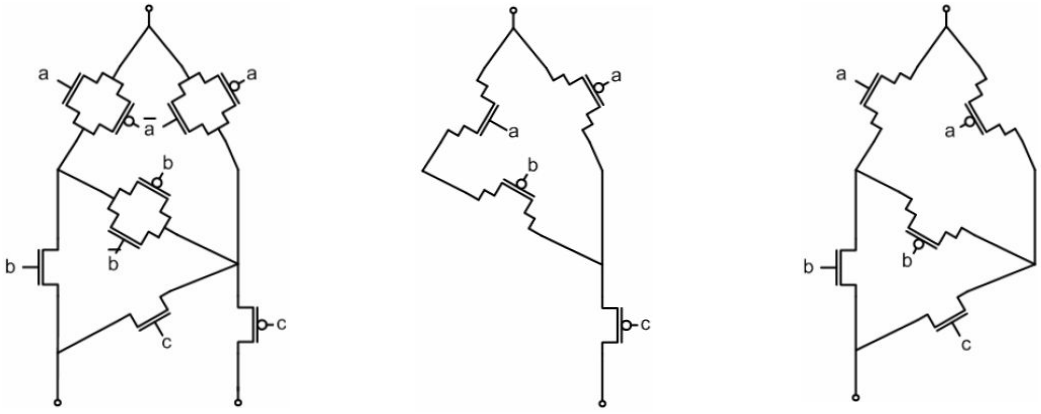


(a) (b)

Figura 2.8: Rede planar (a). Rede não planar (b).

2.3.3 BDD

Ainda é possível utilizar BDDs para criar redes disjuntas. Assim, pode-se criar redes para representar o ON-SET ou o OFF-SET de uma função. Este método pode ser visualizado na Figura 2.9, sendo que em 2.9-a a rede é não-disjunta e *dual-rail*, enquanto as redes de 2.9-b e 2.9-c são redes disjuntas e *single-rail*.

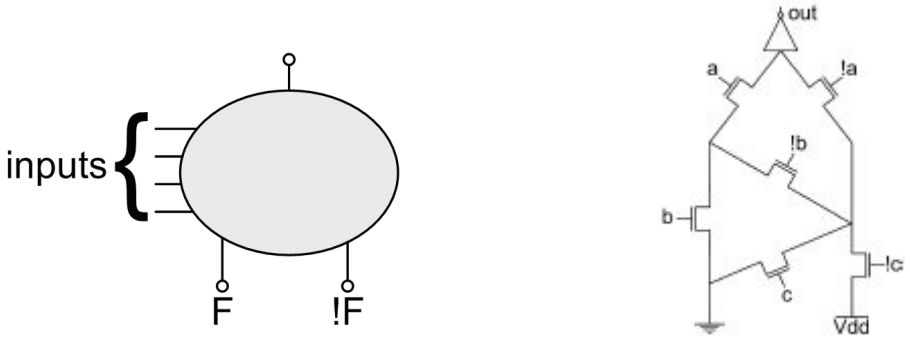


(a) (b) (c)

Figura 2.9: (a) Rede *dual-rail* não-disjunta. (b) e (c) Redes disjuntas e *single-rail*.

2.4 Redes Dual-Rail

Redes *dual-rail* são aquelas capazes de implementar dois terminais, sendo uma responsável por representar a função, e outra por representar a função negada. Essa lógica *dual-rail*(Figura 2.10-a) é geralmente utilizada na construção de circuitos assíncronos (ROSA JR, 2009). Essa lógica pode ser facilmente obtida através de BDDs, assim como a lógica PTL (Figura 2.10-b).



(a) (b)

Figura 2.10: (a) Representação da lógica *dual-rail* e (b) uma rede PTL.

2.5 Tipos de chaves

A utilização de redes de chaves para representar uma função lógica é largamente utilizada, pois permite abstrair a tecnologia que será utilizada na fabricação do circuito. Essas chaves podem ser convertidas em dispositivos tais como: chaves mecânicas, relés, transistores, etc. Essas chaves são divididas em chaves positivas e chaves negativas. Chaves positivas são aquelas permitem a passagem da corrente elétrica desde que seja aplicado o ‘1’ lógico em seu terminal de controle e chaves negativas conduzem a corrente elétrica desde que, em seu terminal de controle, seja aplicado o ‘0’ lógico.

2.6 Tradução de Redes de Chaves para Redes de Transistores

O mapeamento dessas redes de chaves para redes de transistores é feito simplesmente trocando-se chaves positivas por transistores NMOS e chaves negativas

por transistores PMOS. Um dos terminais da rede é conectado a fonte (seja ela GND ou VDD) e o outro é conectado à saída do circuito.

2.7 Outros métodos de Geração de Redes

Além das estratégias de geração previamente descritas, existem outros métodos de geração de redes que têm por objetivo reduzir o número total de chaves de uma rede que representa uma dada função lógica.

(ZHU, 1993) propôs um método de geração de redes baseado em grafos. A idéia principal de seu método consiste na procura de arranjos que podem ser convertidos em arranjos não série-paralelo. Ao encontrar tais arranjos, aplica-se a otimização e verifica-se a existência de falsos caminhos (*sneak paths*). Esta operação é realizada com o objetivo de verificar qual é a função lógica que o grafo representa. Porém, esta operação tem complexidade exponencial, sendo bastante custosa em relação a processamento. Existem ainda outras estratégias que consistem em detectar arranjos que podem ser otimizados, de modo a realizar operações que são bem estabelecidas e seguras, não necessitando de uma verificação dos caminhos do grafo.

(KAGARIS, 2007) propôs um método que parte de uma soma-de-produtos (SOP) de uma dada equação, e para cada produto (transformado em um conjunto de chaves), tenta colocá-lo na rede utilizando uma das quatro alternativas:

- Colocar o conjunto de chaves em série na rede, conectando os dois terminais.
- Procura o menor custo para acomodar o conjunto de chaves utilizando associações paralelo.
- Procura o menor custo para acomodar o conjunto verificando arranjos em série que sejam subsets do conjunto de chaves. Se tal arranjo for encontrado, verifica se caminhos que passem por esse arranjos podem ser compartilhados. Após isso, verifica e corrige possíveis *sneak-paths* e calcula o custo para o novo arranjo.
- Procura por arranjos que podem formar associações não série-paralelo, utilizando compartilhamento de chaves entre caminhos.

Após percorrer as quatro alternativas, verifica dentre elas qual é a que apresenta o menor custo (em número de chaves) para inserir o produto na rede. Por ser um algoritmo guloso, é importante ressaltar que este método gera uma solução heurística, mas que se mostrou bastante eficiente, sendo considerado até os dias atuais o estado da arte em geração de redes de chaves lógicas.

3 PROPOSTA

A proposta deste trabalho consiste em desenvolver um ambiente que integre métodos de geração de redes, bem como métodos de estimativas de redes, visualizadores, verificação e comparação lógica entre redes existentes no laboratório.

3.1 Ambiente "SwitchCraft"

Como objetivos principais têm-se a criação de um ambiente que reaproveite e integre todos os diversos módulos implementados no laboratório ao longo dos anos. Além disso, o ambiente deverá prover tanto uma interface gráfica interativa com o usuário quanto uma interface em modo texto. Esta última será utilizada para fazer a conexão com outras ferramentas que fogem da proposta do trabalho ou não podem ser integradas à ferramenta, como por exemplo, ferramentas comerciais e ferramentas de outros laboratórios e universidades.

É importante ressaltar a importância da integração e reutilização de códigos já feitos anteriormente. Existiam módulos que faziam tarefas específicas e disjuntas. O objetivo é integrar módulos de forma que estes possam cooperar entre si, sendo que a saída de um possa ser utilizada como entrada do outro, criando um fluxo de trabalho, o que não existia antes da criação da ferramenta.

Entre os inúmeros módulos que podem ser interligados, alguns terão maior prioridade, tais como:

- Geração de redes de chaves utilizando técnicas tais como:
  - Geração da rede diretamente da equação;
  - Técnicas de geração baseado em grafos duais (CALLEGARO, 2008);
  - Diretamente do BDD que representa a equação (ROSA, 2006);
  - LBBDD (DA ROSA JUNIOR, 2008);
  - OpBDD (POLI, 2003);
  - Geração de rede utilizando lógica Dual-Rail (DA ROSA JUNIOR, 2008);
- Obtenção de estimativas simples (*profile*):
  - Número total de chaves que compõe a rede;
  - Número máximo de chaves em série que compõe o arranjo de chaves da rede;
  - Número de nós de uma rede;

  - Número de chaves conectadas por nó;
  - Número de caminhos entre o nó de origem e o de saída;
  - Menor caminho na rede;
  - Maior caminho na rede;
  - Distância entre dois nós em uma rede.
- Obtenção de estimativas complexas de desempenho, tais como:
  - Atraso (ELMORE, 1948);
  - Consumo dinâmico (DA ROSA JUNIOR, 2008);
  - Consumo estático:
    - Leakage de Subthreshold (BUTZEN, 2007);
    - Leakage de Subthreshold + Gate (BUTZEN, 2008);
  - Área (DA ROSA JUNIOR, 2008);
- Um módulo de fácil visualização que seja responsável por comparar redes utilizando resultados colhidos pelas estimativas disponíveis na ferramenta.
- Um módulo de visualização de redes em duas dimensões que funcione para redes de todos os estilos lógicos. Este módulo será utilizado principalmente para que seja feita inspeção visual nas redes.
- Um módulo de verificação de consistência de redes. Este módulo consiste de uma engenharia reversa partindo da rede e chegando à equação lógica que a representa. Assim, permitiria a comparação da funcionalidade de redes (se ambas tem a mesma função lógica) assim como a comparação de uma equação com a rede.
- Será criada uma linguagem de scripts, de modo que o framework possa converter os comandos em ações. Com isso, será possível a execução de scripts em lotes, acelerando a utilização da ferramenta, uma vez que se pode criar um script uma vez e depois só chamá-lo para realizar a tarefa. Além disso, scripts provêem flexibilidade à ferramenta e facilitam o uso da mesma.

3.2 Objetivo Secundário

Apesar de existirem vários métodos de geração de redes disponíveis no repositório de dados, existem ainda lacunas a serem preenchidas com o desenvolvimento de outros dois módulos que implementem a metodologia proposta por (ZHU, 1993) e (KAGARIS, 2007). A implementação desses dois métodos fornece uma boa base para futuras comparações com metodologias de geração de redes que estão sendo desenvolvidas no laboratório.

Além desses módulos, é desejável a criação de um módulo que seja integrado ao módulo de visualização, que permita a edição de uma rede de modo gráfico e interativo, isto é, o usuário da ferramenta poderia criar redes utilizando o mouse, e ao mesmo tempo verificar qual é a função lógica que esta rede em questão está representando.

Um visualizador esquemático de redes remodelado, que consiga desenhar redes planares sem cruzamento de chaves também será implementado. Além disso, redes que

formam arranjos não série-paralelo devem ser visualizadas de forma fiel, não prejudicando a visualização por conta de cruzamentos ou mau posicionamento dos nodos de interconexão.

3.3 Metodologias

3.3.1 Ambiente SwitchCraft

Por ser um ambiente de software, a estratégia para o desenvolvimento do ambiente irá basear-se na metodologia Scrum (SCHWABER, 2004) que é uma das metodologias ágeis, processos que já são bem estabelecidos no meio da engenharia de software. Serão realizadas reuniões diárias (conferências por voz) de modo a atualizar a equipe de desenvolvimento sobre o estado atual do projeto. Foi definido um *sprint* de duas semanas, para que os usuários testem os *releases* e possam dar retornos sobre o *framework* SwitchCraft.

3.3.2 Implementação do método proposto por Kagaris

Estudar o método (KAGARIS, 2007) para entender a estratégia por de trás do mesmo, e recolher informações que ajudem a definir qual a melhor estrutura de dados a ser construída e após isso, criar um pseudocódigo que represente o fluxo do método. Implementar o método utilizando como *benchmark* funções e arranjos descritos no próprio trabalho, tanto para verificar a validade do método proposto quanto para testar o módulo.

3.3.3 Implementação do método proposto por Zhu

Após a implementação do método proposto por Kagaris (KAGARIS, 2007, será estudado e implementado o método proposto por Zhu (ZHU, 1993). Por utilizar procura por falsos caminhos, assim como Kagaris, alguns trechos de códigos para implementação da estratégia de Zhu poderão ser reutilizados. Além disso, outros métodos serão inseridos como módulos na ferramenta em trabalhos futuros.

4 IMPLEMENTAÇÃO

Há anos surgem novas metodologias de software para facilitar a tarefa árdua para integrar sistemas. Para a implementação do ambiente SwitchCraft, que tem como principal objetivo a integração de módulos, serão utilizados diversos *design patterns* (padrões de projetos) (FREEMAN, 2004) de engenharia de software, de modo a promover uma infra-estrutura robusta e principalmente de fácil manutenção.

Design Patterns são soluções genéricas e reutilizáveis para aplicar em problemas que são comuns em desenvolvimento de softwares. Um *design pattern* não é um design já acabado que pode ser transformado diretamente para código. Ele é uma descrição ou um *template* de como resolver um problema, e que pode ser utilizado em diversas situações distintas.

4.1 Plataforma e Contexto

Desde o ano de 2005, muitos alunos da graduação e de pós-graduação passaram pelo laboratório de pesquisa "Nangate/UFRGS Research Lab", deixando o conhecimento que lhes foi proporcionado em forma de publicações científicas e em códigos de programas utilizados para verificação de suas teorias.

Todos os códigos que foram desenvolvidos até o momento encontram-se armazenados no servidor que é responsável por fazer o versionamento dos códigos desenvolvidos pela Informática da UFRGS, sendo para isso utilizado o software CVS (*Concurrent Versions System*). Os códigos do laboratório que lá estão armazenados estão escritos utilizando a linguagem Java, que foi a linguagem escolhida para ser o padrão de desenvolvimento no laboratório. Para o contexto do laboratório, uma linguagem portável, podendo ser executada independente de plataforma, é imprescindível, uma vez que nem todas as máquinas têm o mesmo sistema operacional instalado.

4.2 Estrutura

Após quatro anos de pesquisa, o repositório de códigos do laboratório pode ser considerado de grande porte. São aproximadamente 2244 classes que implementam as mais diversas funções, desde árvores lógicas até visualizadores de redes em três dimensões.

Essas classes são agrupadas em módulos, de forma a realizar alguma tarefa e de modo que não interfiram no funcionamento de outros módulos. Entretanto, esse isolamento levou a algo indesejado: criaram-se diversas estruturas de dados que são utilizadas para representar o mesmo problema. Porém, as estruturas não são compatíveis

entre si. Por exemplo: Existem duas estruturas que são utilizadas para representar árvores lógicas. Uma reconhece os operadores AND e OR pelos símbolos '*' e '+' respectivamente. Já a outra, reconhece os respectivos operadores pelos símbolos '&' e '|'.

Este é um exemplo que ilustra diversos problemas semelhantes que se encontram no repositório de dados. Além do problema da incompatibilidade, essa duplicação de estrutura leva a outro problema como duplicação de código, sendo necessário que sejam feitas manutenções nos códigos que poderiam ser feitas em apenas um deles, bem como rotinas de testes para verificação da coerência dos códigos.

Como o principal objetivo do ambiente é a integração e a reutilização de códigos desenvolvidos, surgiu a necessidade de prover a intercomunicação entre módulos que até então estavam separados. Por exemplo, o módulo de geração de redes por BDDs e o módulo de extração de rede diretamente de uma equação apresentavam como saída estruturas de dados distintas. Com isso, surgiu um problema na construção do módulo que é responsável por visualizar o esquemático de redes. Não poderíamos optar apenas por uma das estruturas, senão isso implicaria em reescrever o código do outro módulo. Para resolver este e outros problemas que surgiram ao longo do desenvolvimento do projeto, recorremos aos já tradicionais *design pattens* (FREEMAN, 2004) (padrões de projetos).

A fim de resolver o problema anteriormente discutido, onde os módulos utilizam estruturas distintas e precisam convergir para uma estrutura comum, foi aplicado o *design pattern* chamado *adapter* (adaptador), que também pode ser encontrado na literatura pelo nome de *wrapper*. Este padrão é utilizado para converter a interface de uma classe para outra interface que o cliente espera receber. Esta estratégia permite que classes que eram incompatíveis possam enfim comunicar-se. A Figura 4.1 representa o padrão utilizando notação UML (*Unified Modeling Language*).

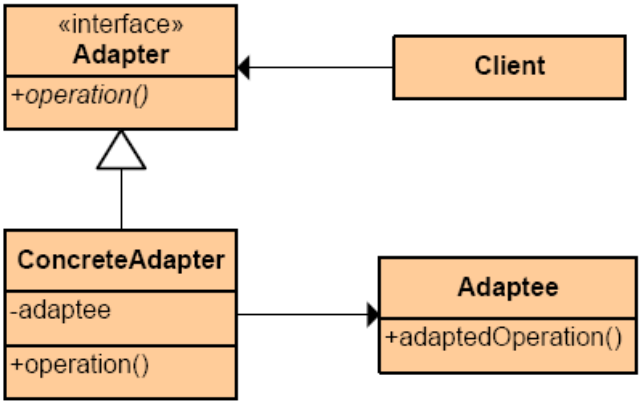


Figura 4.1: *Adapter* (*Design Pattern*).

Para resolver o problema das árvores lógicas, foi criada uma estrutura que tem por objetivo ser a mais simples e genérica possível. Reconhece os operadores AND por '*' ou '&' e o operador OR por '+' ou '|'. Além disso, houve a preocupação de prover um método que facilite a ação de percorrer a árvore. Para isso, utilizou-se o *design pattern* conhecido como *visitor (*visitante*)*, que tem por objetivo representar uma operação a ser aplicada sobre os elementos de uma estrutura, permitindo ao usuário definir uma nova operação sem mudar a classe dos elementos sobre os quais ele opera. A Figura 4.2 representa o padrão em notação UML.

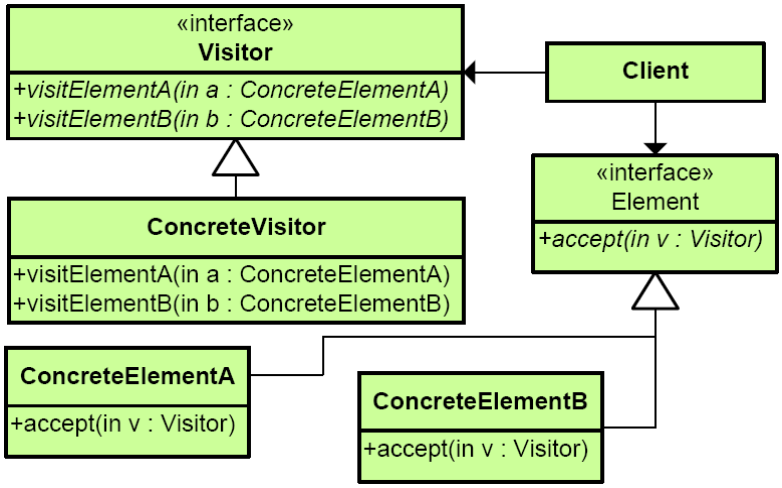


Figura 4.2: *Visitor* (*design pattern*).

Outra característica que foi implementada é a habilidade de realizar comandos tanto pela interface gráfica quanto por linhas de comando. Para isso, é importante que o comando chamado tanto pela interface gráfica quanto por linha de comando seja o mesmo, para evitar duplicação de código e outros problemas anteriormente mencionados. Para realizar esta tarefa, foi utilizado o *design pattern* conhecido por *command* (comando). Este padrão tem como objetivo encapsular uma requisição como um objeto, permitindo a parametrização de diferentes requisições. Uma aplicação bastante comum para este tipo de padrão é a geração de *logs*. Esta característica é importante, pois faz com que a ferramenta possa dar um *feedback* ao usuário do que está sendo executado. A Figura 4.3 representa o padrão em notação UML.

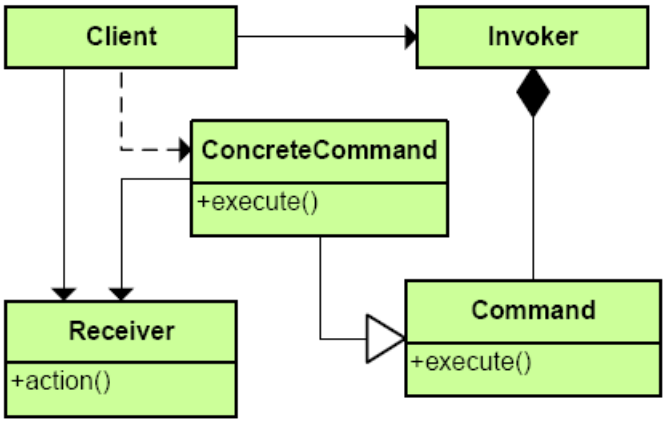


Figura 4.3: *Command* (*design pattern*).

Uma ferramenta de CAD deve, sempre que possível, possuir uma linguagem de *scripts* que possibilite realizar tarefas de modo automatizado, sem que o usuário precise interagir com a ferramenta durante todo o processo. Para isso, foi implementado uma linguagem de *script*, que possibilita a execução de diversos comandos em lote descritos através de arquivos.

Para dar ainda mais flexibilidade, criamos uma forma de composição de scripts, de modo que um script possa chamar outro. Para modelar essa característica, aplicamos o *design pattern* conhecido por *composite* que é ilustrado na Figura 4.4.

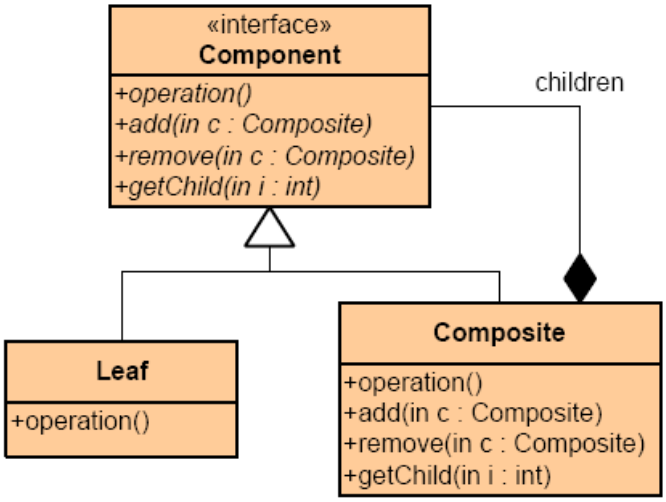


Figura 4.4: *Composite* (*design pattern*).

Para o armazenamento de toda a informação contida na ferramenta, foi criada uma classe que tem a função de armazenar as informações inseridas e obtidas através da ferramenta. Nesta classe (Figura 4.5), ficam armazenadas informações sobre as equações utilizadas como entrada, redes que foram geradas, etc. Atualmente, essa classe foi modelada aplicando o *design pattern singleton* (Figura 4.6), que é utilizado quando se deseja que apenas uma instância da classe esteja disponível para toda a aplicação.

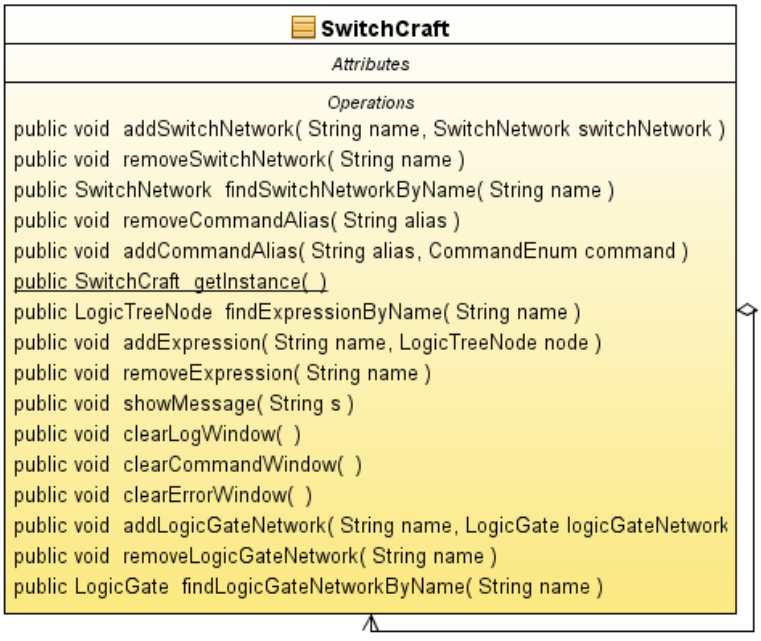


Figura 4.5: Classe que armazena informações da ferramenta.

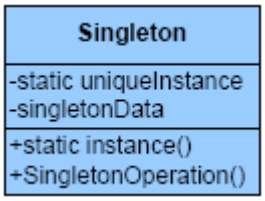


Figura 4.6: *Singleton* (*design pattern*).

5 RESULTADOS

Este capítulo apresentará os resultados obtidos ao longo da implementação da ferramenta. Apresentaremos um fluxo de trabalho, tendo como meta demonstrar a aplicação dos objetivos propostos no capítulo três.

5.1 Estrutura

O desenvolvimento da aplicação seguiu o modelo de três camadas, que consistem em:

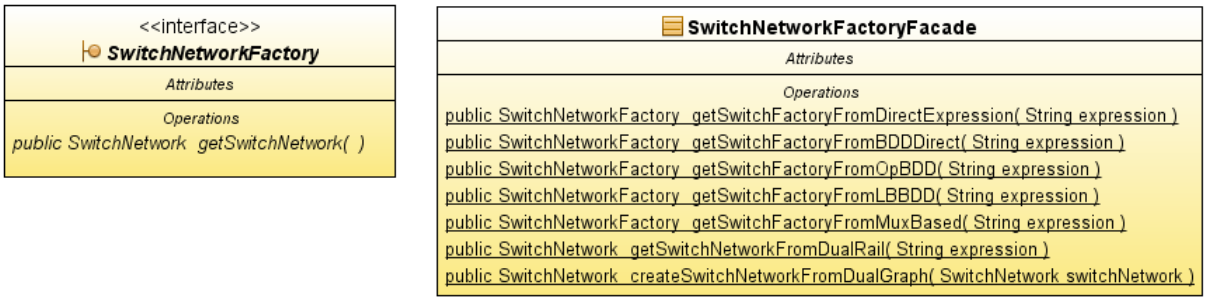
- Camada de dados;
- Camada de controle;
- Camada de apresentação.

Esta divisão é feita de modo a facilitar que novas funcionalidades sejam agregadas e que futuras manutenções no código possam ser feitas sem prejudicar as demais camadas. Apresentaremos a seguir as camadas que compõe a aplicação, bem como as principais características de cada uma.

5.1.1 Camada de dados

A camada de dados é responsável por manter as estruturas de dados e módulos que compõem a base da ferramenta. Nela ficam, por exemplo, a estrutura que representa uma rede de chaves, módulos que implementam algoritmos de geração de redes, etc.

Para facilitar que futuras abordagens de gerações de redes sejam agregadas à ferramenta, criou-se o conceito de fábrica de redes (*switch network factory*). Esta fábrica consiste em uma interface e uma classe que fornece acesso aos módulos de geração de redes. A interface possui apenas um método, que serve para acessar a rede que foi gerada. A classe que provê acesso aos módulos de geração de redes é a única classe publica do pacote, de modo que outras classes que estiverem dentro do pacote de geração de redes fiquem encapsuladas, não interferindo no resto da aplicação, sendo que o acesso aos módulos é feito através de uma única classe. Este é o conceito do *design pattern* chamado *facade*. A Figura 5.1-a ilustra a interface e a Figura 5.1-b, a classe que é utilizada para se ter acesso aos módulos de geração de redes.



(a) (b)

Figura 5.1: (a): Interface de geração de redes. (b) Classe que provê acesso a módulos de geração de redes.

Com essa conFiguração, podemos agora encapsular módulos de geração de redes e padronizar o retorno dos mesmos, sendo nesse caso, a estrutura *SwitchNetwork*. Desta forma, módulos de visualização de redes esperam receber essa estrutura para representá-las de forma visual.

Esta camada ainda abriga diversas outras funcionalidades, como algoritmos de estimativas, equivalências entre redes, etc. Porém, neste nível o que temos são apenas dados, necessitando assim de outra camada que possa acessar e controlar os módulos presentes nessa camada.

5.1.2 Camada de controle

Esta camada é responsável por intermediar as ações vindas da camada de visualização e módulos da camada de dados. Quando o usuário entrar com uma equação na ferramenta, por exemplo, esta camada será responsável por verificar se a equação é uma equação válida e posteriormente armazená-la em memória para futuramente ser utilizada.

Uma outra característica que pode ser implementada nessa camada são os comandos e linguagem de *scripts*. Como as ações na ferramenta foram modeladas em forma de comandos, uma classe é responsável por agrupar esses comandos que ficam disponíveis na ferramenta. Com isso, bastou criar uma *parser* para reconhecer linhas de comandos digitadas pelo usuário, e após isso, evoluir para aceitar arquivos com diversos comandos encadeados, também conhecidos como *scripts*. Até o fechamento do trabalho, a ferramenta proporcionava ao usuário a possibilidade de utilizar 50 comandos distintos. Um exemplo de *script* é ilustrado na Figura 5.2, que foi utilizado para gerar uma rede utilizando o método LBBDD e visualizá-la.

```
> create_expression f1 (d*e)+(b*c*d)+(a*c*e)+(a*b)
> create_switch_network_from_lbbdd f1 f1_switch_network
> view_switch_network f1_switch_network
```

Figura 5.2: Exemplo de script para geração de uma rede arbitrária.

Para facilitar a utilização de comandos, foi criado um módulo que permite ao usuário criar atalhos, pois os nomes dos comandos foram criados de forma a serem

facilmente relacionados com sua função, deixando os mesmos por vezes muito extensos.

5.1.3 Camada de visualização

A camada de visualização foi implementada de forma que a ferramenta pudesse ser utilizada através de uma interface gráfica ou apenas em modo texto. Como o projeto foi dividido em três camadas, a separação da camada de visualização em duas foi bastante simples.

A interface gráfica consiste em uma janela principal, que podem conter outras janelas abertas. Logo abaixo da janela principal, existe uma caixa de texto onde podem ser digitadas as linhas de comandos. Acima desta caixa, existem três abas, onde aparecem os resultados, comandos ou erros gerados pela ferramenta. (Figura 5.3)

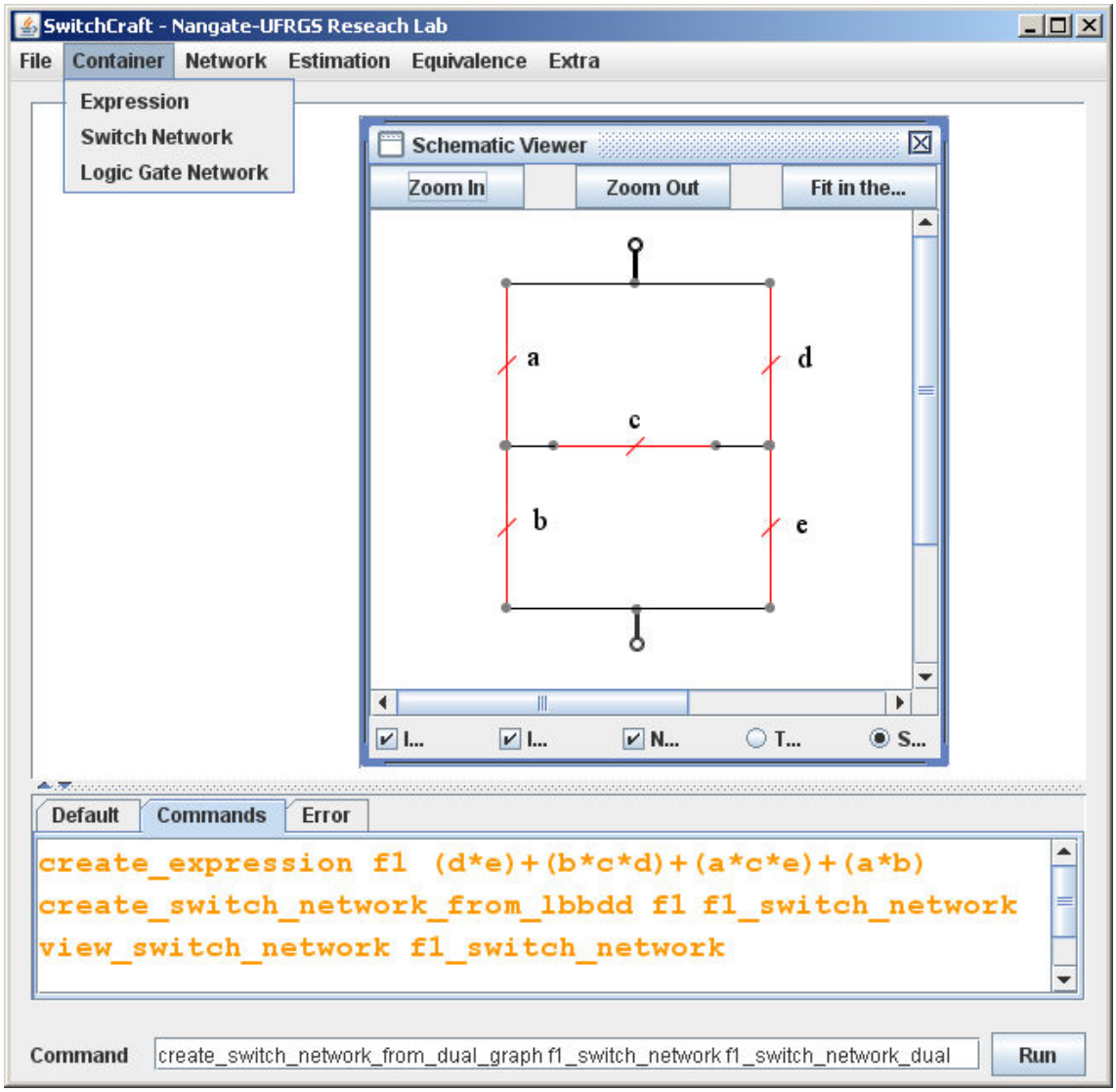


Figura 5.3: Interface gráfica do SwitchCraft.

As informações que ficam armazenadas em tempo de execução na ferramenta foram organizadas em módulos para serem encontradas facilmente. As informações que são armazenadas em tempo de execução são:

- Equações;

- Redes de chaves;
- Portas lógicas.

Foram criadas janelas específicas para cada uma dessas informações, de modo que as operações mais comuns são facilmente acessíveis através de botões. A Figura 5.4 ilustra a janela que organiza as redes de chaves lógicas (*Switch Networks*).

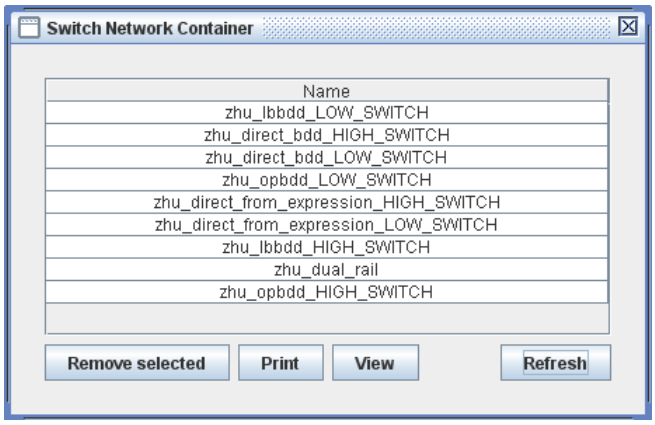
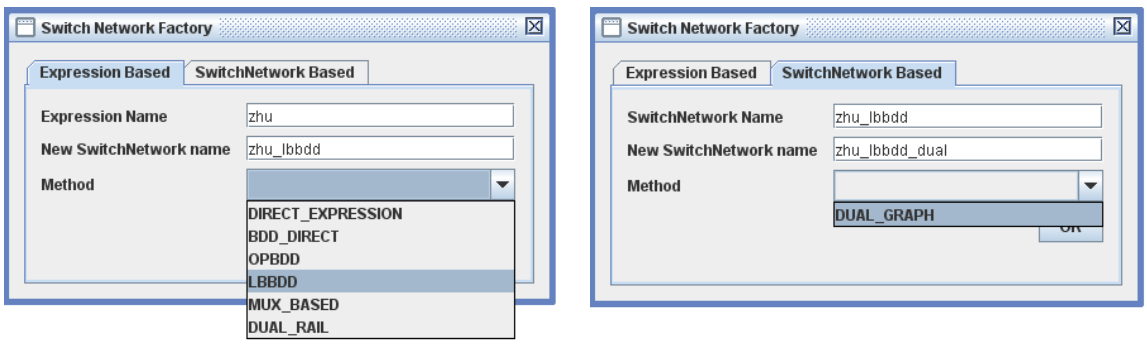


Figura 5.4: Janela que organiza as redes de chaves.

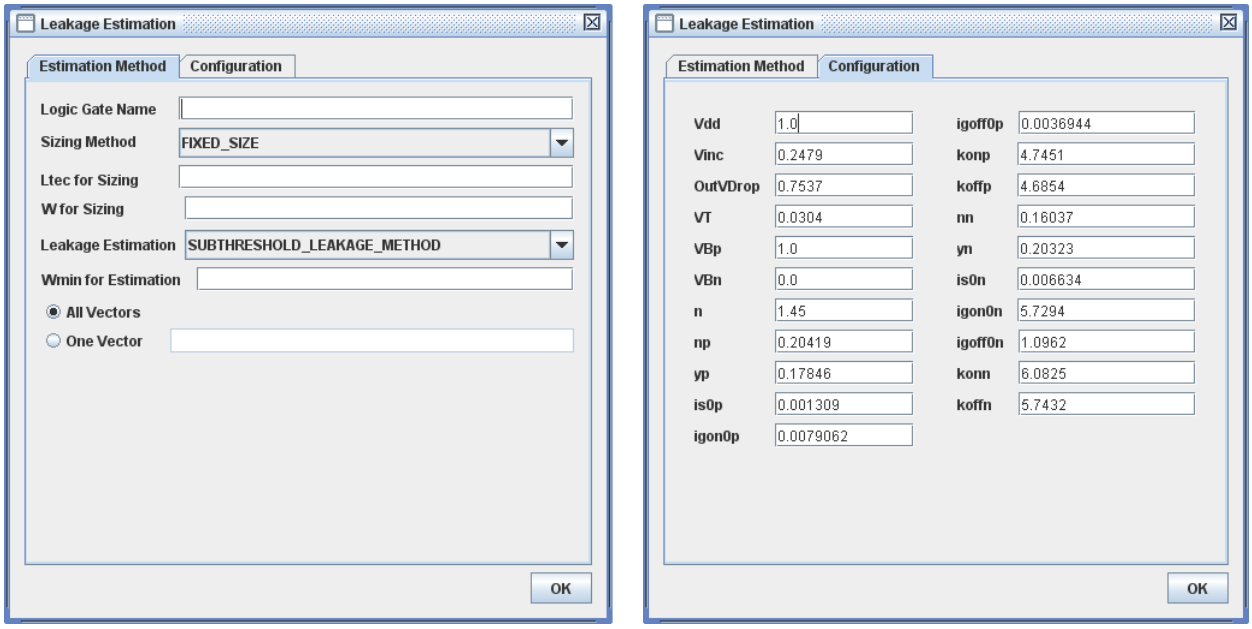
A janela que provê os diversos métodos de geração de redes é dividida em duas abas, sendo que uma delas fornece apenas métodos que partem de equações (Figura 5.5-a) e a outra, métodos que partem de redes já existentes (Figura 5.5-b). Todos os métodos que estão disponíveis em cada uma dessas abas estão dentro de uma *combobox*. Assim, o usuário preenche o nome da rede a ser gerada, o nome da equação ou da rede a ser utilizada, e seleciona qual o método que irá utilizar na geração.



(a) (b)

Figura 5.5: (a) Métodos baseados em equações. (b) Método baseado em redes.

Outros módulos que foram integrados são os módulos de estimativas. Cada um possui sua própria janela, pois exigem conFigurações próprias, como pode ser visto na Figura 5.6, que apresenta a janela de obtenção da estimativa de corrente de fuga (*Leakage*) de uma porta lógica.



(a) (b)

Figura 5.6: (a): Aba de informações básicas. (b) Aba de conFigurações do método.

Para fazer a comparação e verificação das redes, foram também agrupados módulos que realizam essas tarefas. Os comparativos que foram implementados são:

- Entre equações;
- Entre redes de chaves;
- Entre uma rede de chave e uma equação;
- Entre portas lógicas;
- Entre porta lógica e equação.

Além dessas comparações, outra forma de verificar a rede é através da inspeção visual. Para isso, utilizou-se o algoritmo descrito em (SILVA, 2009) para criar o módulo de visualização de redes e portas lógicas, que pode ser visto na Figura 5.3 (janela central).

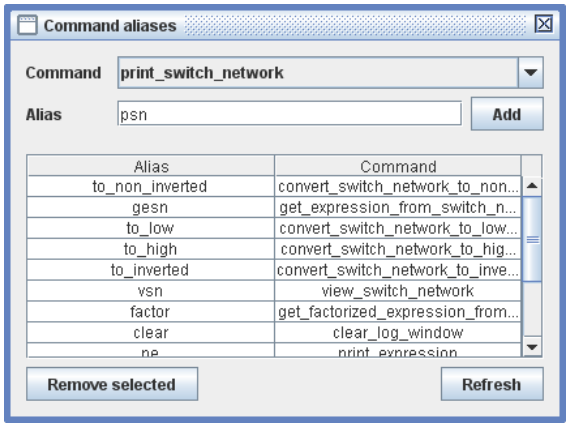


Figura 5.7: Janela que organiza atalhos para comandos.

Para a criação de atalhos para os comandos também foi criada uma janela que permite visualizar, adicionar e remover atalhos. Basicamente é um mapa com chave e

valor, que sempre que a chave (atalho) for encontrada no comando, será convertido pelo valor (comando) encontrado (Figura 5.7).

5.2 Exemplo de fluxo

Apresentaremos nesta seção um fluxo de trabalho como exemplo de utilização da ferramenta. Partiremos de uma equação tendo como objetivo construir portas lógicas através de diversas abordagens distintas. Desta forma, apresentaremos os resultados intermediários, bem como o script gerado pela ferramenta.

Para geração dos resultados, utilizaremos unicamente a equação (4) extraída de (ZHU, 1993) como entrada na ferramenta.

$$f = a * b * h + d * e * !b + d * e * g + d * c * b * h + a * c * e * !b + a * c * e * g + a * b * !e * f + d * c * b * !e * f \quad (4)$$

Após essa etapa, iremos gerar diversas redes com o objetivo de obter uma porta lógica que represente a equação (4). Os métodos de geração que serão utilizados são:

- Mapeamento direto da equação;
- Fatoração da equação;
- Direto do BDD;
- OpBDD;
- LBBDD;
- LBBDD + Grafo Dual.

As Figuras das redes que foram geradas por esses métodos podem ser encontradas no apêndice.

Iremos extrair algumas informações das redes geradas, através do módulo que é responsável por fazer um *profile* das redes. Até o momento, estes resultados são apresentados em forma de texto puro. Porém, será construída uma interface onde possam ser comparadas as informações lado a lado, como é ilustrado na tabela 5.1.

Tabela 5.1: Resultados obtidos pelos métodos de geração de redes.

Método	# Chaves rede ON-SET	# Chaves rede OFF-SET	# Stack ON_SET	# Stack OFF_SET
Direto da equação	30	30	5	8
Fatoração da equação	13	13	5	4
Direto do BDD	23	24	7	7
OpBDD	16	20	6	6
LBBDD	10	17	5	4
LBBDD + Grafo Dual	10	10	5	4

É possível ainda gerar estimativas dessas redes, de modo que um projetista possa escolher qual a melhor rede dada certa característica (atraso, consumo dinâmico e

corrente de fuga). Os resultados obtidos através da ferramenta podem ser encontrados no apêndice.

Todo o fluxo que apresentamos pode ser atingido executando o seguinte *script*:

```
> create_expression zhu (a * b * h + d * e * !b + d * e *
g + d * c * b * h + a * c * e * !b + a * c * e * g + a * b
* !e * f + d * c * b * !e * f)

> factor_expression zhu zhu_factorized

> create_switch_network_from_direct_expression zhu
zhu_DIRECT

> create_switch_network_from_dual_graph zhu_DIRECT
zhu_DIRECT_DUAL

> create_switch_network_from_direct_expression
zhu_factored zhu_factored

> create_switch_network_from_dual_graph zhu_factored
zhu_factored_dual

> create_switch_network_from_bdd_direct zhu zhu_BDD

> create_switch_network_from_opbdd zhu oppBDD

> create_switch_network_from_lbbdd zhu zhu_lbbdd

> create_switch_network_from_dual_graph zhu_lbbdd
zhu_lbbdd_DUAL

> create_logic_gate direct zhu_DIRECT_DUAL zhu_DIRECT

> create_logic_gate factor zhu_factored_dual zhu_factored

> create_logic_gate bdd_direct zhu_BDD_LOW_SWITCH
zhu_BDD_HIGH_SWITCH

> create_logic_gate oppbdd oppBDD_LOW_SWITCH
oppBDD_HIGH_SWITCH

> create_logic_gate lbbdd zhu_lbbdd_LOW_SWITCH
zhu_lbbdd_HIGH_SWITCH

> create_logic_gate lbbdd_dual zhu_lbbdd_DUAL
zhu_lbbdd_HIGH_SWITCH

> print_logic_gate_profile direct

> print_logic_gate_profile factor

> print_logic_gate_profile bdd_direct

> print_logic_gate_profile oppbdd

> print_logic_gate_profile lbbdd

> print_logic_gate_profile lbbdd_dual
```

6 CONCLUSÃO

Este trabalho apresentou um *framework* de geração automática de redes de chaves lógicas e estimativas. O objetivo principal proposto e atingido por este trabalho consistia da integração de diversos módulos que antes deste *framework* não se relacionavam entre si.

Para atingir essa integração, diversos desafios foram encontrados e contornados ao longo do desenvolvimento do trabalho. A integração de diversos módulos e de diversas classes que não se relacionavam, além da falta de documentação foi sem dúvida a maior dificuldade encontrada. É importante mencionar que existiu ainda, a busca da informação diretamente com os autores de alguns módulos, levando a uma interação com diversos outros colegas que já passaram pelo laboratório. A experiência de trabalhar em equipe ainda dentro da graduação foi bastante importante, visto que interagir e compartilhar conhecimento com pessoas da área sempre é interessante.

Além dos módulos que foram integrados, existiu ainda a necessidade da implementação de alguns outros que não existiam. Esses módulos eram necessários para que um fluxo de trabalho pudesse ser inteiramente executado pelo ambiente. Módulos como geração de redes através de grafo dual, algoritmo de fatoração de funções booleanas e o método proposto por (KAGARIS, 2007) (parcialmente concluído) foram implementados ao longo desse trabalho e também integrados ao ambiente.

A integração de módulos foi uma tarefa bastante interessante em nível do conhecimento adquirido na área de engenharia de software. Essa tarefa levou ao estudo profundo de *design patterns*, pois diversos problemas que surgiram puderam ter suas soluções completamente embasadas nesses *patterns*, como apresentado no capítulo de implementação.

Como trabalhos futuros a serem incorporados ao ambiente, pode-se citar a conclusão do método de geração de redes proposto por Kagaris. Assim que esse módulo for concluído, será implementado o método proposto por (ZHU, 1993) sendo que com esses dois métodos chegaremos a oito abordagens distintas disponíveis no ambiente para geração de redes de chaves lógicas.

Após a conclusão desses novos módulos de geração de redes, pretende-se iniciar a implementação de um módulo de geração de redes através de visualização, o qual possibilitará ao usuário da ferramenta a liberdade de criar redes utilizando o *mouse*, por exemplo.

REFERÊNCIAS

SHANNON, C.E. A symbolic **Analysis of Relay and Switching Circuits**. Transactions American Institute of Electrical Engineers, New York, v.57, p. 38-80, May 1938.

UEHARA, T.; VANCLEEMPUT, W. M. **Optimal Layout of CMOS Functional Arrays**. IEEE Transactions on Computers, Los Alamitos, v.c-30, n. 5, p. 305-312, May 1981.

ZHU, J.; ABD-EL-BARR, M. **On the optimization of MOS Circuits**. IEEE Transactions on circuits and systems, [S.l.], v.40, n.6, p. 412-422, June 1993.

DA ROSA JUNIOR, L. S. **Automatic Generation and Evaluation of Transistor Networks in Different Logic Styles**. 2008. 147 f. Tese (Programa de Pós-Graduação em Microeletrônica) – Instituto de Informática, UFRGS, Porto Alegre.

SCHNEIDER, F.R. **Building Transistor-Level Networks Following the Lower Bound on the Number of Stacked Switches**. 2007. Master Deegre Thesis. Mestrado em Ciência da Computação. Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

KAGARIS, D.; HANIOTAKIS, T. **A Methodology for Transistor-Efficient Supergate Design**. IEEE Transactions on VLSI Systems, New York, USA, v.15, n.4, p. 488-492, Apr. 2007.

WESTE, N.H.E.; HARRIS, D. **CMOS VLSI Design: A Circuits and Systems Perspective**. 3rd ed. Boston: Pearson/Addison Wesley, 2005.

RABAEY, J.M.; CHANDRAKASAN. A.; NIKOLIC, B**. Digital Integrated Circuits: A Design Perspective**. 2nd ed. Upper Saddle River: Prentice Hall, 2005.

MINTZ, A.; GOLUMBIC, M.C. **Factoring Boolean functions using graph partitioning**. Discrete Applied Mathematics, [S.l.], n.149, p.131-135, May 2005.

REIS, A.I.; et al.. **Fast Boolean Factoring with Multi-Objective Goals**. In: INTERNATIONAL WORKSHOP ON LOGIC AND SYNTHESIS, IWLS, 18., 2009, Berkley, USA. Proceedings… Los Alamitos: IEEE, 2009.

CALLEGARO, V. Et al. . **A Kernel-based Approach for Factoring Logic Functions**. Microelectronics Students Forum (9. : 2009 Aug. : Natal, BR-RN). SForum 2009 [recurso eletrônico]. [S.l. : s.n., 2009?] CD-ROM [S.l.], Setembro 2009.

ROSA, L.S.; RIBAS, R.P.; REIS, A.I. **Fast Transistor Networks from BDDs**. Symposium On Integrated Circuits And System Design, SBCCI, 19., 2006, Ouro Preto, Brasil. Proceedings… New York: ACM, 2006. p. 137-142

CALLEGARO, V.; Et al. **A Graph-based Solution for Dual Transistor Network Generation**. VIII Student Forum on Microelectronics, 2008, Gramado. VIII Student Forum on Microelectronics CDROM. Porto Alegre : SBC, 2008.

DA ROSA JUNIOR, L. S. ; CALLEGARO, V. ; RIBAS, R. P. ; REIS, A. I. . **Redes de Transistores e Portas Lógicas CMOS.** Julio C.B. Mattos; Leomar S. da Rosa Jr.; Mauricio L. Pilla. (Org.). Desafios e Avanços em Computação: O Estado da Arte. 1 ed. Pelotas: Editora e Gráfica Universitária - PREC UFPel, 2009, v. 1, p. 197-224.

ELMORE, W. **The transient response of damped linear networks with particular regard to wideband amplifiers**. Journal Applied Physics, Woodbury, USA, v. 19, n. 1, p. 55- 63, Jan. 1948.

POLI, R. E. B.; RIBAS R. P.; REIS A. I. **Unified Theory to Build Cell-Level Transistor Networks from BDDs**. Symposium On Integrated Circuits And System Design, SBCCI, São Paulo, Brasil. Proceedings Los Alamitos: IEEE, 2003. p. 199–204. 2003.

BUTZEN, P. F. Et Al. **Modeling Subthreshold Leakage Current in General Transistor Networks**. ISVLSI. 2007

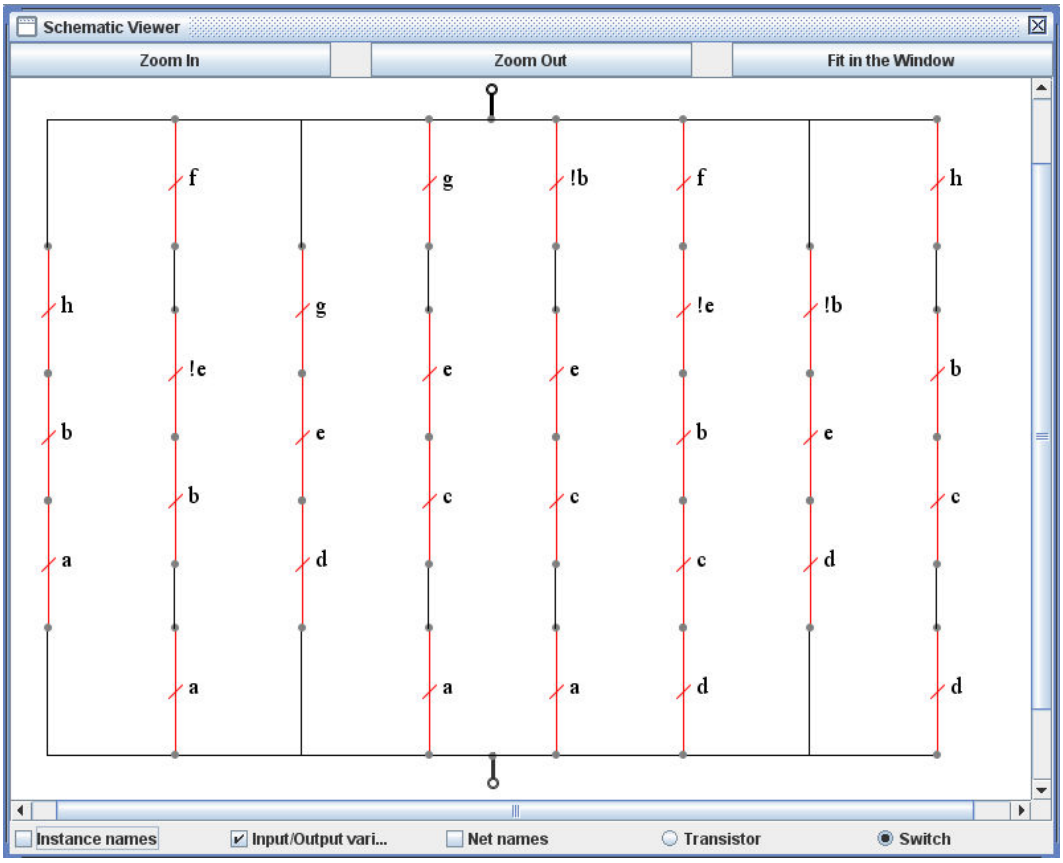
BUTZEN, P. F.; et al. **Simple and accurate method for fast static current estimation in CMOS complex gates with interaction of leakage mechanisms**. 18th ACM Great Lakes Symposium on VLSI (GLSVLSI), 2008, Orlando. Proceedings of the 18th ACM Great Lakes Symposium on VLSI. New York : ACM, 2008. p. 407-410.

SILVA, R. H. Et al. **Planar transistor network visualization algorithm** [recurso eletrônico]. Microelectronics Students Forum (9. : 2009 Aug. : Natal, BR-RN). SForum 2009 [recurso eletrônico]. [S.l. : s.n., 2009].

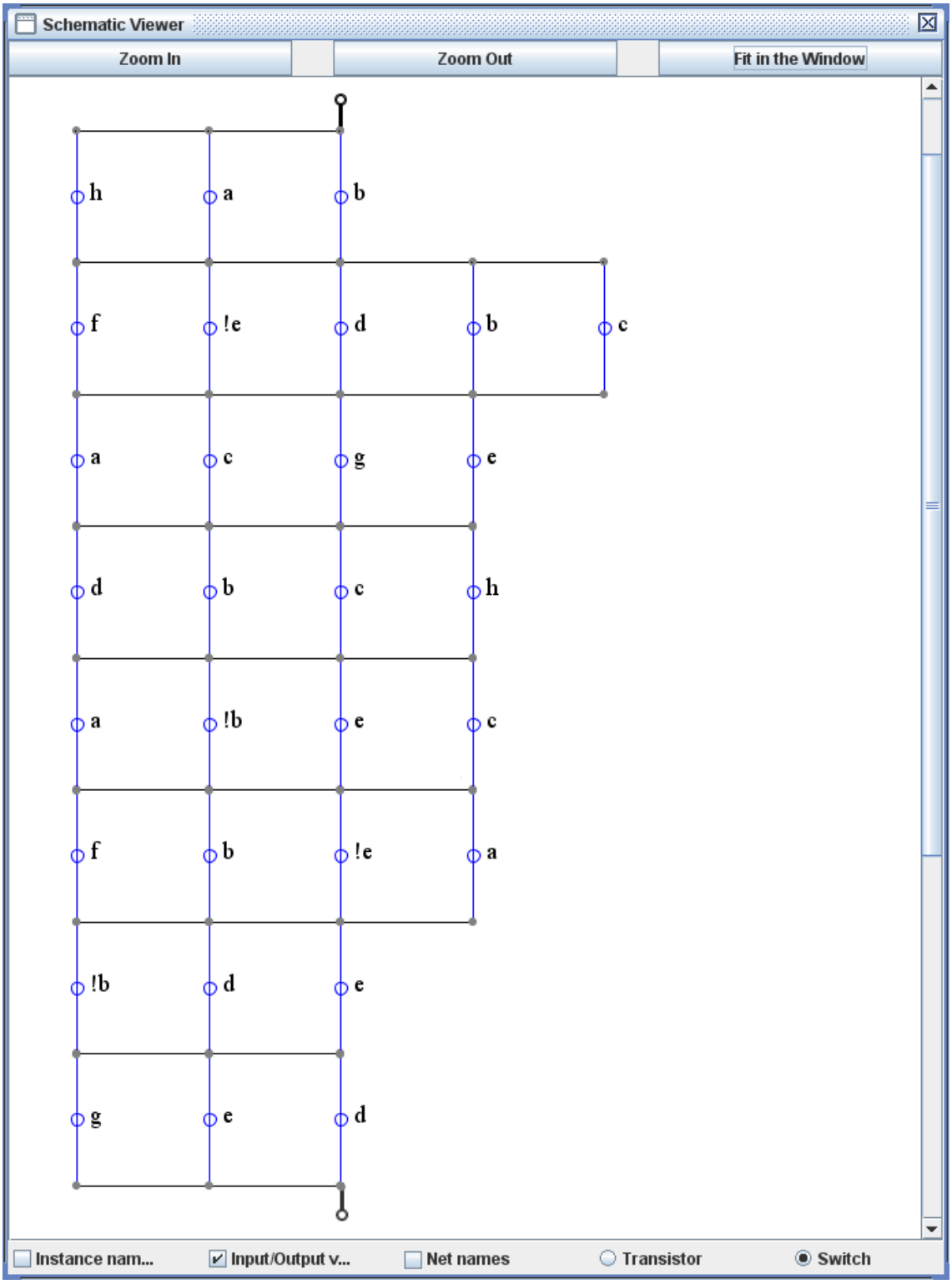
SCHWABER, K. **Agile Project Management with Scrum.**. Microsoft Press. ISBN 978-0-735-61993-7. February ,2004.

FREEMAN, E; et al. **Head First Design Patterns**. O'Reilly Media. ISBN 0-596-00712-4. 2004.

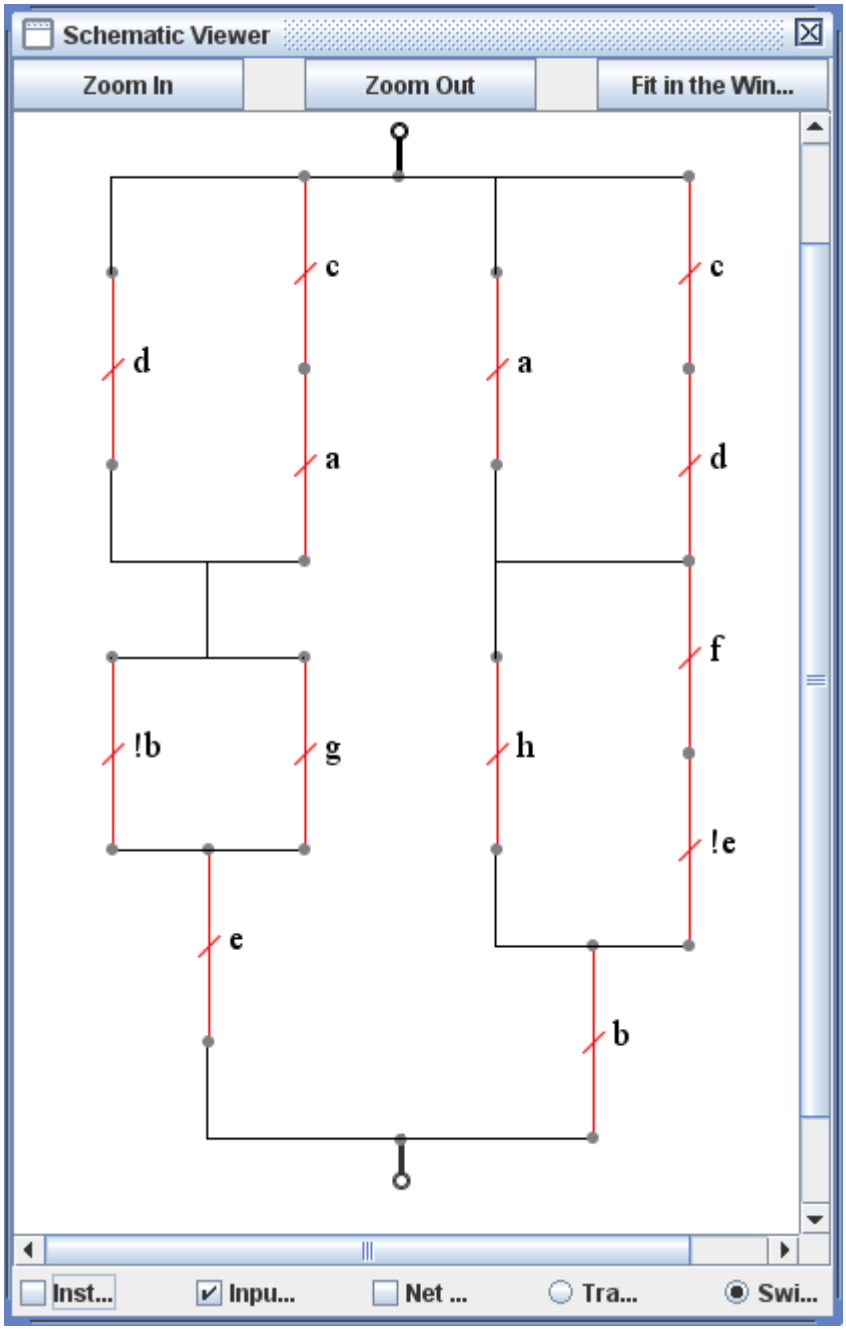
APÊNDICE <REDES DE CHAVES DE EXEMPLO >



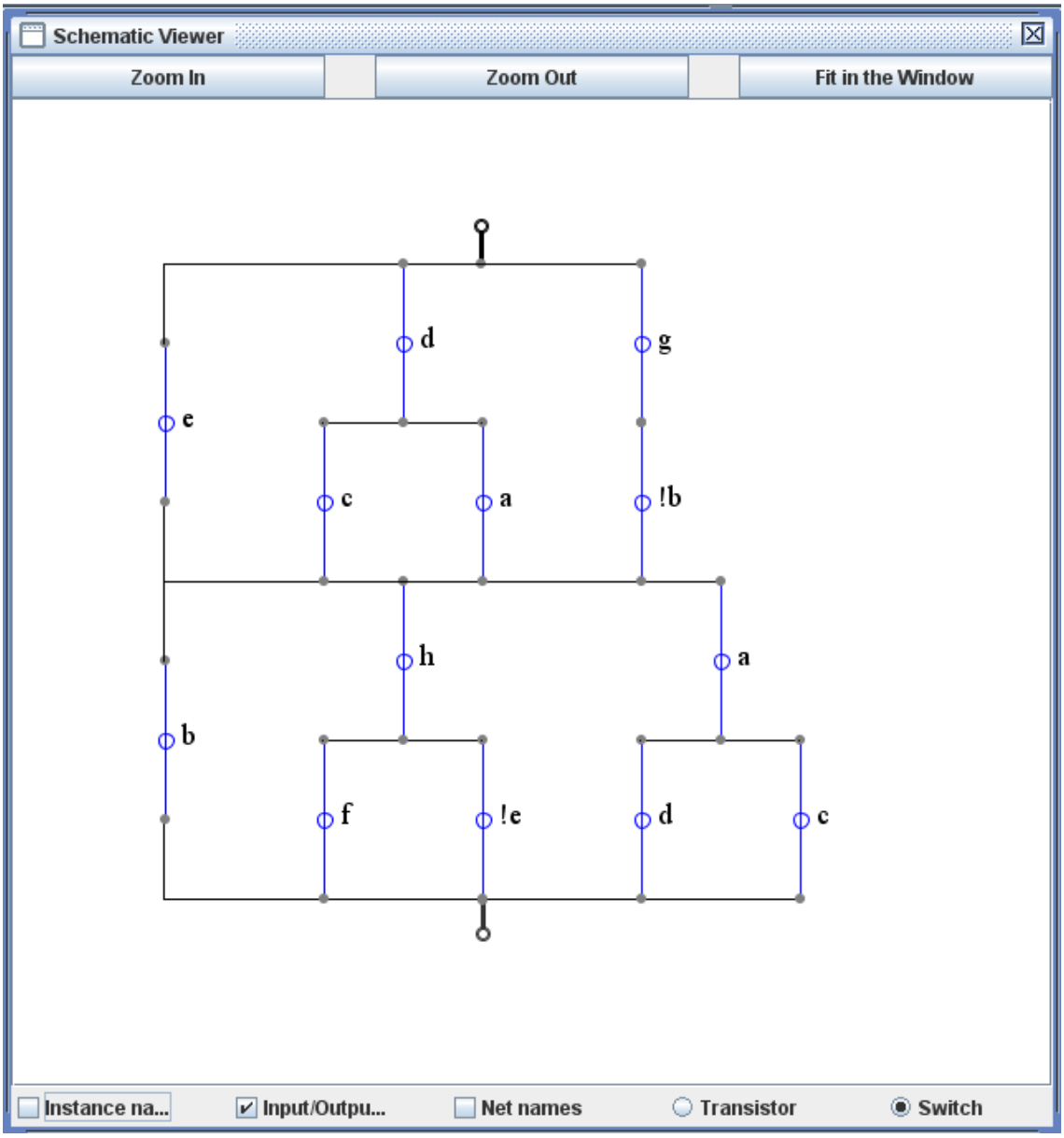
Apêndice: Rede gerada diretamente da equação em formato SOPl.



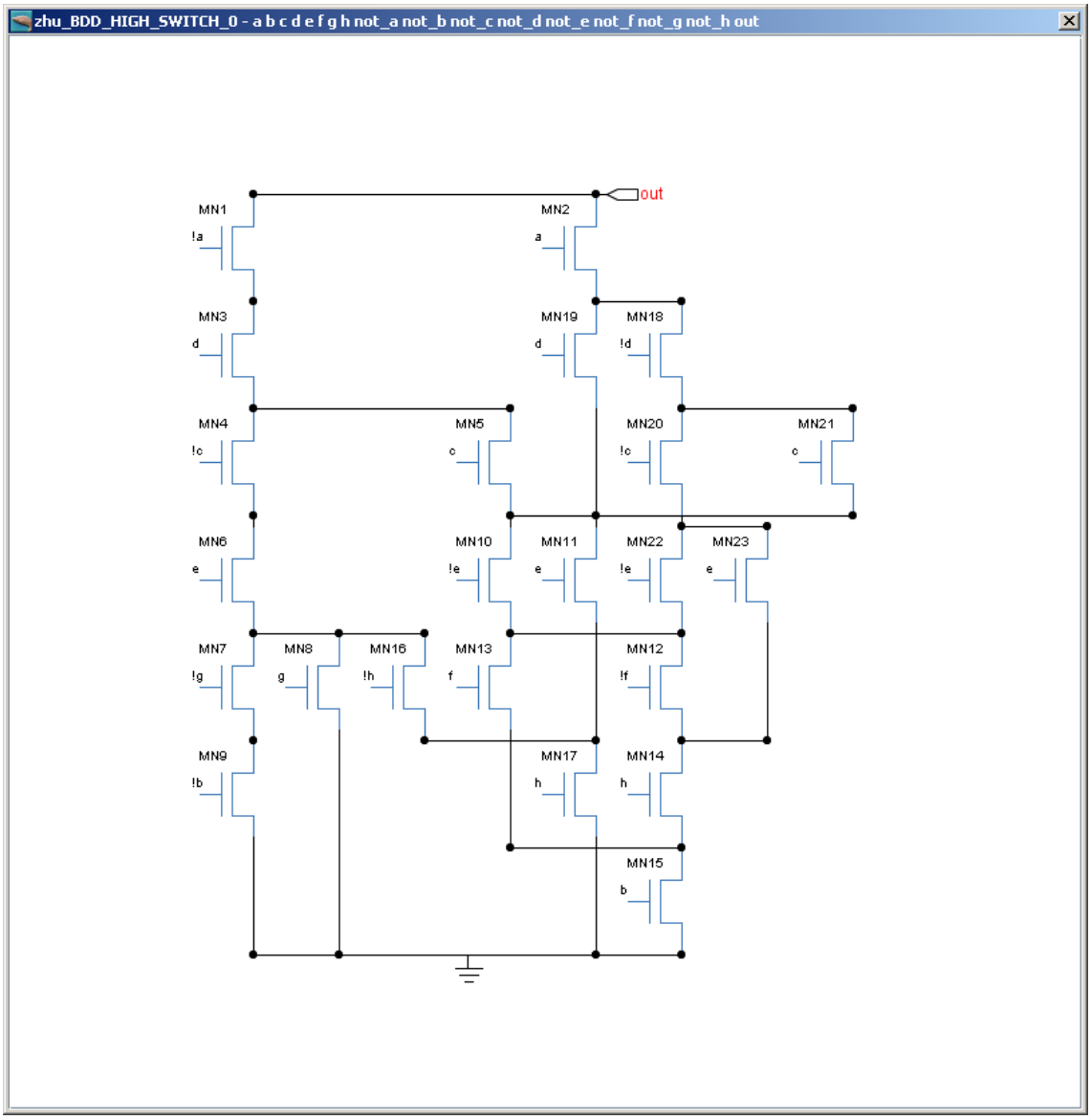
Apêndice: Rede complementar da equação em SOP.



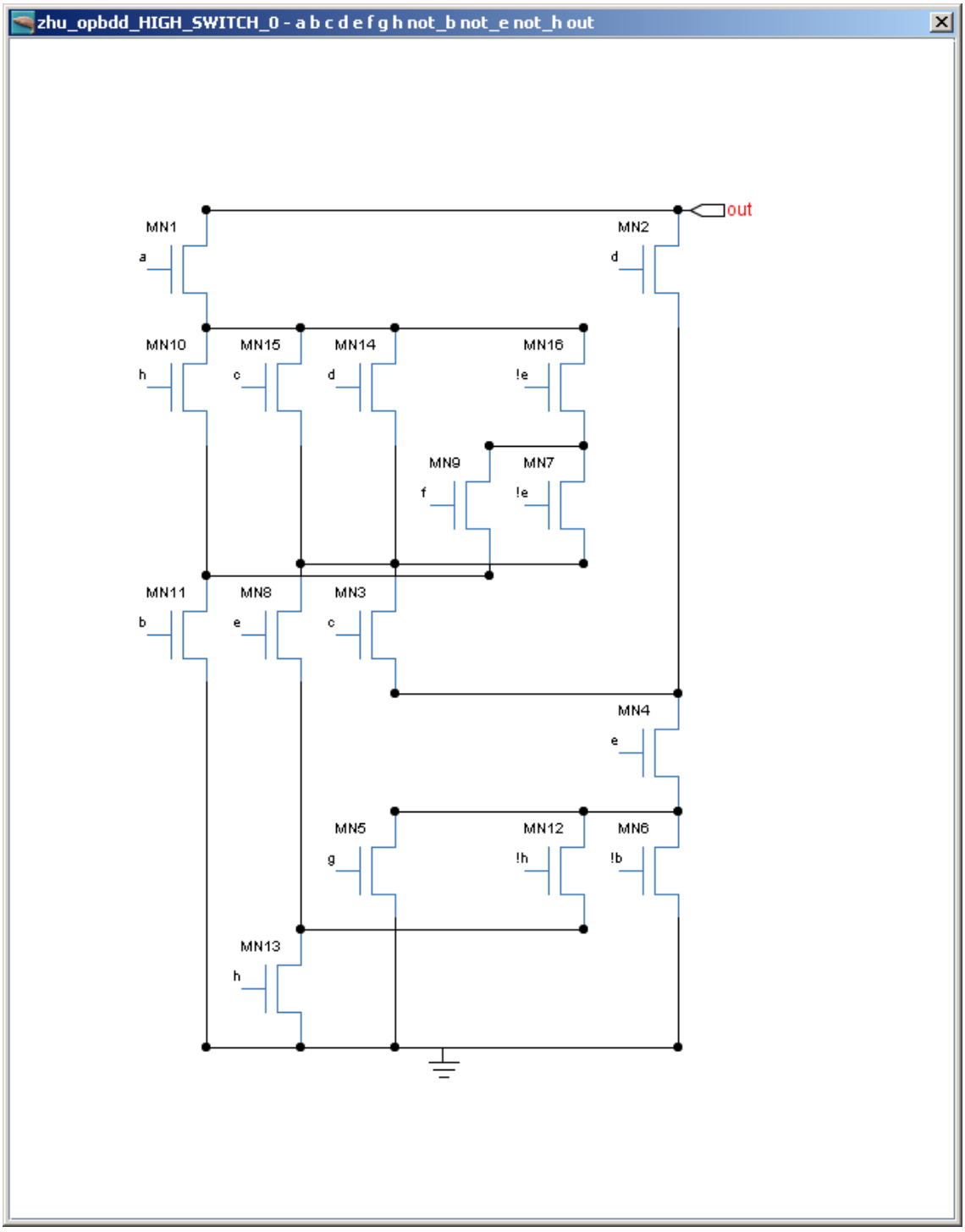
Apêndice: Rede de chaves que representa equação fatorada.



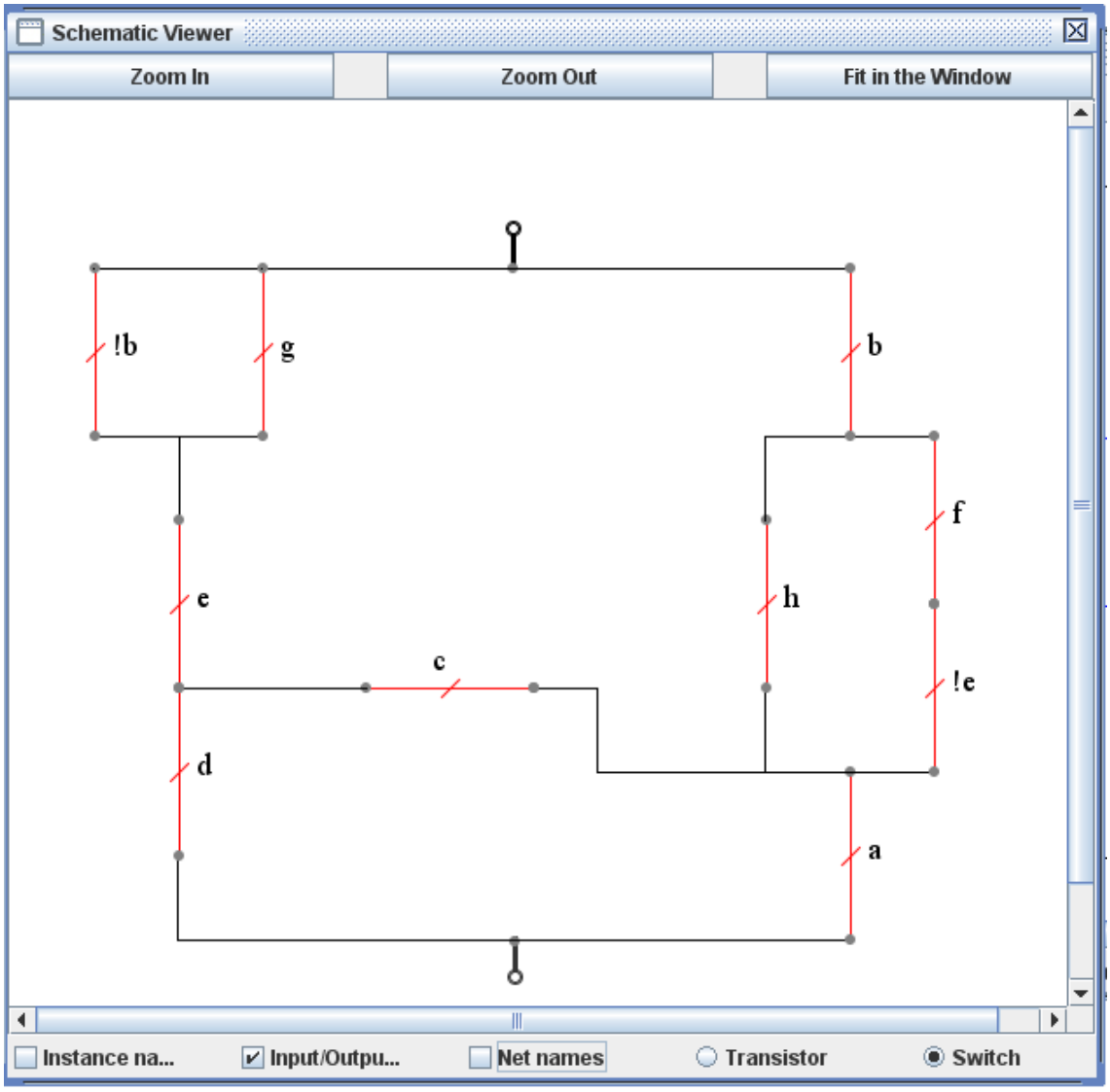
Apêndice: Rede complementar da equação fatorada.



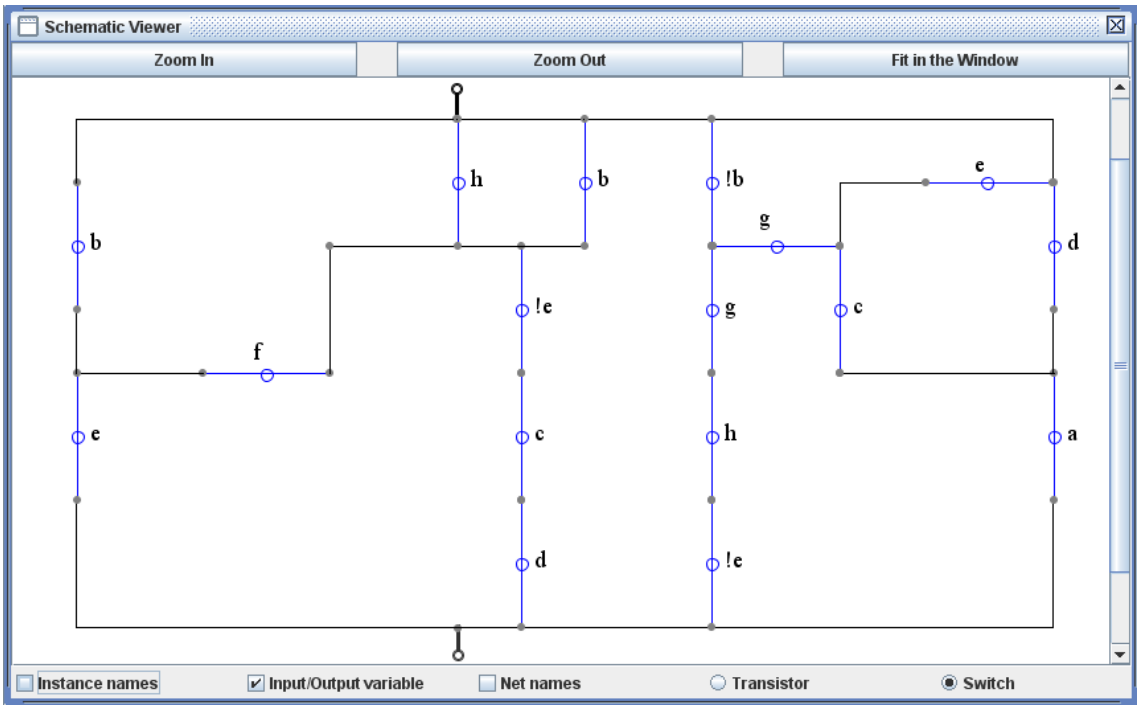
Apêndice: Rede gerada direto de um BDD.



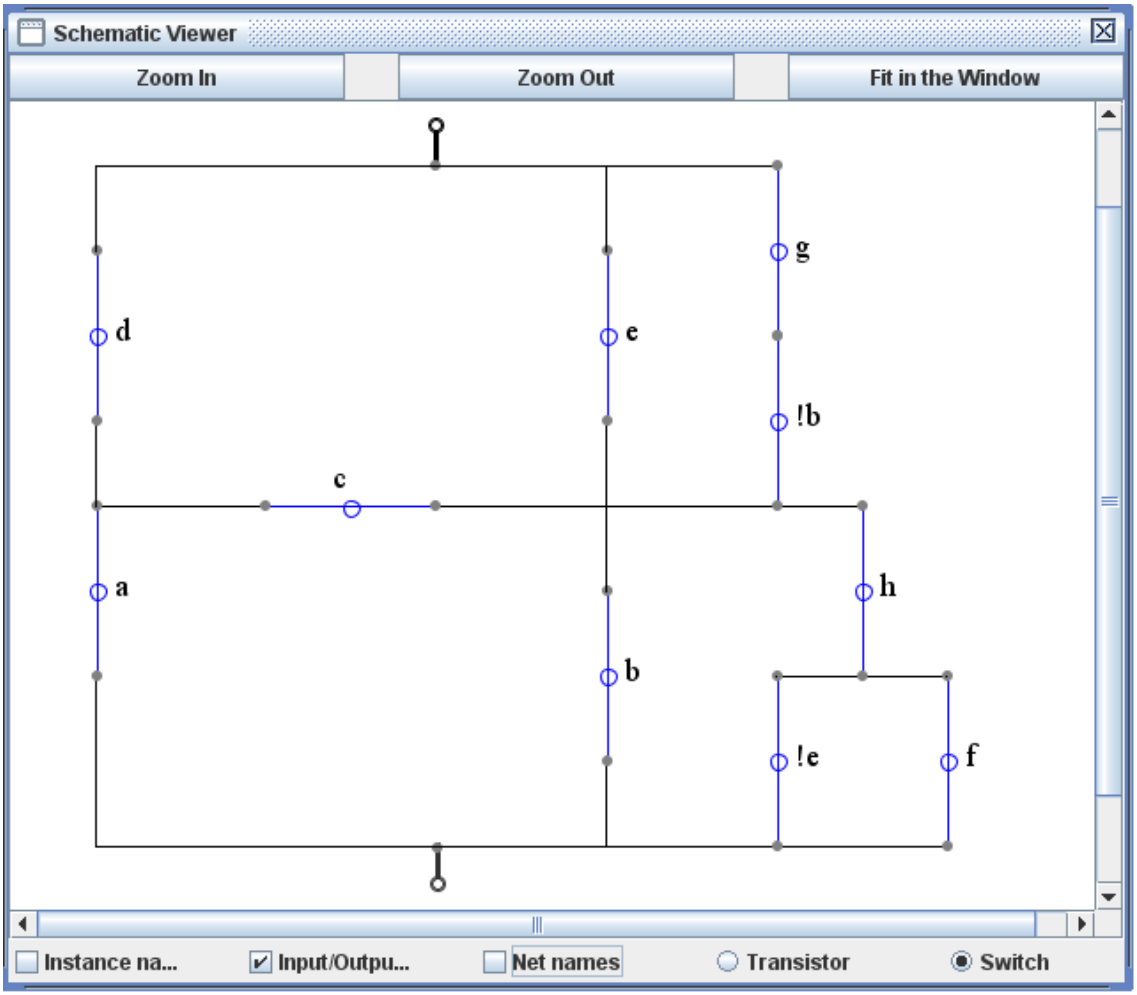
Apêndice: Rede gerada pelo método OpBDD.



Apêndice: Rede gerada pela aplicação do método LBBDD.



Apêndice: Rede complementar obtida de uma rede gerada pela aplicação do método LBBDD.



Apêndice: Rede resultante da aplicação do método que utiliza grafo dual para gerar a rede complementar.

Apêndice: Tabela de resultados da estimativa de atraso

Delay (Elmore)			
Method	**Min**	**Max**	**Average**
Factored	1.10E-11	1.02E-10	4.53E-11
Lbbdd	1.35E-11	1.11E-10	4.03E-11
Lbbdd Dual	8.07E-12	1.07E-10	3.76E-11
Opbdd	1.32E-11	1.42E-10	5.26E-11
Bdd Direct	8.94E-12	2.60E-10	1.03E-10
Direct	9.41E-14	5.87E-10	5.91E-11

Apêndice: Tabela de consumo dinâmico

Dynamic (Load) Consumption			
Method	**Min**	**Max**	**Average**
Factor	1.25E-15	6.58E-15	3.56E-15
Lbbdd	1.56E-15	5.55E-15	3.41E-15
Lbbdd Dual	1.13E-15	5.41E-15	3.30E-15
Opbdd	2.17E-15	7.00E-15	4.61E-15
Bdd Direct	9.19E-16	9.83E-15	4.90E-15
Direct	3.48E-15	1.69E-14	8.51E-15

Apêndice: Tabela de corrente de fuga

Leakage estimation			
Method	**Min**	**Max**	**Average**
Factor	2.32E-9	2.42E-7	8.17E-8
Lbbdd	3.26E-9	1.81E-7	8.34E-8
Lbbdd Dual	2.73E-9	1.93E-7	8.16E-8
Opbdd	2.14E-9	3.29E-7	1.02E-7
Bdd Direct	4.65E-9	2.39E-7	8.06E-8
Direct	1.39E-10	3.76E-7	9.37E-8