UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação

Projeto e Avaliação de Portas Lógicas Complexas sem Restrições Topológicas

Maicon Schneider Cardoso

Pelotas, 2017

Maicon Schneider Cardoso

Projeto e Avaliação de Portas Lógicas Complexas sem Restrições Topológicas

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial para à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Felipe de Souza Marques

Coorientador: Prof. Dr. Leomar Soares da Rosa Junior

Pelotas, 2017

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

C268p Cardoso, Maicon Schneider

Projeto e avaliação de portas lógicas complexas sem restrições topológicas / Maicon Schneider Cardoso ; Felipe de Souza Marques, orientadora ; Leomar Soares da Rosa Junior, coorientador. — Pelotas, 2017.

113 f. : il.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2017.

1. Portas lógicas complexas. 2. Redes de transistores. 3. Geração automática de leiaute. 4. Libra. 5. Kernel finder. I. Marques, Felipe de Souza, orient. II. Rosa Junior, Leomar Soares da, coorient. III. Título.

CDD : 005

Elaborada por Maria Inez Figueiredo Figas Machado CRB: 10/1612

Agradecimentos

Gostaria de começar agradecendo aos professores Felipe de Souza Marques e Leomar Soares da Rosa Junior, fundamentais na orientação e no desenvolvimento da pesquisa e grandes amigos que fiz nessa caminhada. Obrigado pelas oportunidades que me foram dadas e pela confiança depositada em mim nesses últimos anos.

Estendo os agradecimentos aos colegas do Grupo de Arquitetura e Circuitos Integrados (GACI) por partilhar momentos bons e ruins do cotidiano do laboratório. Em especial, agradeço ao Regis Zanandrea e ao Renato de Souza pelo apoio na implementação de parte dos algoritmos propostos nessa dissertação, além do Gustavo Smaniotto e do Matheus Moreira, auxílios fundamentais na realização dos experimentos.

A todos os amigos que torceram por mim, meu muito obrigado. Pessoal da Ciência da Computação da UFPel, da Engenharia Elétrica do IFSul e ex-colegas do CEFET/RS, obrigado! Aos meus amigos de infância e aos que fiz durante todos esses anos, meus agradecimentos. Tenho certeza que todos seguem me apoiando, embora à distância.

Agradeço aos meus tios, dindos e primos que, mesmo não tendo o mesmo contato de antigamente, seguem torcendo pelo meu sucesso. Aproveito para estender os agradecimentos à família da minha noiva, em especial a minha sogra, cunhados, primos e afilhados.

À minha noiva, Pamela Acosta Gomes, agradeço pelo companheirismo, parceria, cumplicidade, carinho e amor. Obrigado por partilhar tudo da minha vida, por me apoiar independente do caminho que eu opte por seguir, pela paciência e compreensão nesses anos, em especial nos períodos onde a pesquisa fez com que eu me afastasse. Te amo.

À minha família, Mairon Schneider Cardoso, Dalva Farias Schneider, Algeu Vargas Cardoso e Darli Schneider Cardoso, obrigado por me dedicar tanto amor, pelos ensinamentos, por serem meu espelho de ética, moral e valores. Agradeço por todo suporte, por me encorajar e acreditar em mim sempre. Amo vocês.

Estendo meus agradecimentos à Universidade Federal de Pelotas (UFPel), ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo suporte financeiro durante o período de mestrado. Por fim, agradeço ao povo brasileiro por me proporcionar uma educação de qualidade em uma universidade pública.

“And in the end
The love you take
Is equal to the love
You make.”
The Beatles, The End.

Resumo

CARDOSO, Maicon Schneider. **Projeto e Avaliação de Portas Lógicas Complexas sem Restrições Topológicas**. 2017. 113f. Dissertação (Mestrado em Computação) – Programa de Pós-Graduação em Computação. Universidade Federal de Pelotas, Pelotas, 2017.

O projeto digital realizado através de portas lógicas complexas vem se demonstrando uma ferramenta eficaz na síntese de circuitos otimizados quando comparado com a tradicional metodologia *standard cell*. Isso ocorre pois a solução não fica limitada a um conjunto pré-determinado de células, possibilitando, assim, a minimização em área, potência e atraso. Neste contexto, uma das principais etapas do projeto de portas lógicas complexas é a de geração lógica, estágio responsável por prover a rede de transistores especializada que implementa a função Booleana. Com esse propósito, recentemente metodologias baseadas em grafos vêm apresentando resultados expressivos relativos à redução do número de componentes no arranjo lógico em comparação com os métodos tradicionais baseados em fatoração Booleana. No entanto, ainda que os dados inicialmente apontem para uma otimização do circuito digital composto por tais redes (já que há menos transistores por porta, em média), faz-se necessário uma maior investigação quanto aos impactos que estruturas não-planares e não-duais – as quais compõem boa parte das soluções obtidas por métodos baseados em grafos – ocasionam nos algoritmos e ferramentas automáticas de geração de células e no leiaute em si. Neste trabalho é apresentada a metodologia Libra, uma proposta para o projeto de portas lógicas complexas baseada nos métodos estado da arte de geração de redes de transistores – Kernel Finder – e de síntese automática de circuitos – ASTRAN. Para a avaliação das soluções desenvolvidas foram realizadas uma série de comparações relativas a metodologia de minimização lógica estado da arte através de fatoração Booleana – Composição Funcional –, a qual atua sob o paradigma amplamente empregado na indústria. Os resultados apontaram para uma expressiva redução em área e atraso para células com caminhos críticos pequenos. Para células com maiores caminhos críticos, os resultados obtidos indicaram uma melhora no atraso e piora relacionada ao consumo de potência. Os experimentos não apenas proporcionaram uma verificação quantitativa relativa às soluções produzidas através da metodologia proposta, mas, também, permitiram a identificação de diversos pontos que podem ser melhorados tanto na metodologia proposta quanto na própria ferramenta ASTRAN, motor para a geração de leiautes.

Palavras-chave: portas lógicas complexas; redes de transistores; geração automática de leiaute; Libra; Kernel Finder; ASTRAN

Abstract

CARDOSO, Maicon Schneider. **Design and Evaluation of Complex Logic Gates Without Topological Constraints**. 2017. 113f. Dissertação (Mestrado em Computação) – Programa de Pós-Graduação em Computação. Universidade Federal de Pelotas, Pelotas, 2017.

The digital design flow based on complex logic gates have gained relevance recently, becoming an important alternative to the quality design when compared to the standard cell methodology, widely used in the microelectronics industry. Due to its flexibility to generate cells on demand, this paradigm can achieve several optimizations in terms of area, power and delay when compared to the classical approach. Concerning the project of complex logic gates, the transistor network generation step is responsible for delivering an optimized arrangement to compute the Boolean function. In this scenario, graph-based methodologies have presented significative minimizations in the switch count when compared to the classical Boolean factoring paradigm. However, this new approach introduces some singular aspects in the transistor topology, such as non-planarity and non-duality, which impacts directly in the physical synthesis tools and in the layout itself. This work proposes Libra, a methodology to design complex logic gates through Kernel Finder and ASTRAN, state-of-art tools for network generation and cell design, respectively. To evaluate our proposal, a comparison with the cells provided by Functional Composition, the state-of-art methodology based on Boolean factoring to perform logic design, was performed. The obtained results showed significant gains in area and delay for the logic gates with relatively small critical paths. Other analyses have shown smaller optimizations in delay and an overhead in terms of power dissipation for cells containing larger critical paths. These experiments pointed that the proposed methodology achieves good results in general. As future works, we intend to improve the presented methodology to deal with these particular cases (including some optimizations in ASTRAN).

Keywords: complex logic gates; transistor networks; automatic layout generation; Libra; Kernel Finder; ASTRAN

Lista de Figuras

Figura 1	Gráfico apresentando o crescimento exponencial no número de transistores presentes em processadores Intel ao longo dos anos (Lei de Moore).....	19
Figura 2	Fluxo de síntese tradicional (*standard cell*).....	20
Figura 3	Circuitos lógicos implementando *f*. (a) *Standard cell*. (b) Porta lógica complexa. (c) Rede de transistores implementando (a). (d) Rede de transistores implementando (b).....	21
Figura 4	Redes de transistores implementando *f*. (a) Solução obtida por fatoração Booleana (FC). (b) Solução via minimização baseada em grafos (KF).....	23
Figura 5	Solução para *f* não-planar e não-dual provida pelo KF.....	24
Figura 6	Planos logicamente equivalentes compondo *f*. (a) Solução com 8 transistores proveniente da Equação (4). (b) Solução com 6 transistores descrita em (5). (c) Solução definida em (6).....	28
Figura 7	Construção iterativa de *buckets* na metodologia de Composição Funcional.....	30
Figura 8	Processo de criação de *buckets* para fatoração da função *f*.....	31
Figura 9	BDD de *f* (a). Rede de transistores correspondente (b).....	33
Figura 10	Duplicação de nodos através do LBBDD. (a) Rede original. (b) Solução com 9 transistores. (c) Solução com 8 transistores.....	35

Figura 11 Instâncias de execução do método de Kagaris, *et al*. (a) Grafo anterior à inserção do cubo f_{C4}. (b) Grafo após a inserção do cubo f_{C4}. (c) Rede de transistores resultante..... 36

Figura 12 *Kernels* implementados no Kernel Finder. (a) *NSP-kernel.* (b) *SP-kernel*..... 37

Figura 13 Fluxograma do Kernel Finder..... 38

Figura 14 Execução do Kernel Finder para a função *f* descrita pela Equação (12). (a) Resultado após o método *Kernel Identification*. (c) Solução após execução do método *Network Composition*..... 40

Figura 15 Previsão do *gap* de produtividade no projeto VLSI..... 42

Figura 16 Compartilhamento de difusão proposto em (UEHARA, *et al.*,1981). (a) Transistores NMOS em série compartilhando suas áreas ativas *B*. (b) Transistores NMOS em série não compartilhando suas áreas ativas *B* (conexão em metal 1)..... 43

Figura 17 Estilos de leiaute. (a) *Linear-matrix*. (b) 2D..... 44

Figura 18 Posicionamento por caminhos de Euler sem *gaps*. (a) Rede lógica. (b) Grafo correspondente. (c) Célula não contendo *gaps*.... 45

Figura 19 Posicionamento por caminhos de Euler com *gaps*. (a) Rede lógica. (b) Grafo e subgrafos correspondentes. (c) Célula contendo um *gap* na área ativa..... 46

Figura 20 Estrutura representativa da célula proposta em Iizuka *et al*..... 47

Figura 21 Estrutura gradeada utilizada para o posicionamento e leiaute produzido a partir da instância apresentada..... 52

Figura 22 Instância de execução do *maze router* em uma grade contendo obstáculos..... 55

Figura 23 Cellgen: gerador de células do ASTRAN..... 59

Figura 24 *Folding* de transistores. (a) Rede lógica original. (b) Leiaute posicionado resultante de (a). (c) Rede lógica equivalente a original com a aplicação de *folding* nos transistores b e c. (d) Leiaute resultante de (c)..... 60

Figura 25 Função de perturbação do método *Threshold Accepting* implementada no ASTRAN..... 63

Figura 26 Estrutura de grafos utilizada no roteamento *intra-cell*..... 64

Figura 27 Metodologia Libra proposta para a geração de portas lógicas complexas..... 66

Figura 28 Estrutura (*template*) da porta lógica complexa..... 68

Figura 29 Redes de transistores distintas geradas via Kernel Finder. (a) Solução obtida através do Algoritmo 4. (b) Solução proveniente de uma metodologia alternativa..... 68

Figura 30 Caminhos terminais da célula para o transistor *M1*..... 71

Figura 31 Dimensionamento da porta lógica complexa ilustrada na Figura 29 (a). (a) Dimensionamento relativo. (b) Dimensionamento real... 72

Figura 32 Porta lógica complexa não-série-paralelo apresentada na Figura 4 (b) e estratégias de posicionamento. (a) Rede de transistores implementando a Equação (2). (b) Posicionamento de transistores sob a estratégia CAA. (c) Leiaute sem quebras na linha de difusão e com quebras no polisilício. (d) Posicionamento de transistores sob a estratégia CPG. (e) Leiaute com quebras nas linhas de difusão e sem quebras no polisilício..... 74

Figura 33 Leiautes gerados através de diferentes estratégias de posicionamento para células com *mismatching* intrínseco. (a) Solução via estratégia CAA. (b) Solução via estratégia CPG..... 75

Figura 34 Leiaute da porta lógica complexa que implementa *f* ilustrada na Figura 29 (a)..... 77

Figura 35 Inversor de referência adotado no dimensionamento das células..... 80

Figura 36 Resultados dos comparativos geométricos entre as células geradas via Libra em relação as células providas por Composição Funcional para o catálogo *53 NSP Handmade Networks*..... 83

Figura 37 Maior ganho em área (*F31*, 45,54%) observado entre as células propostas e as obtidas por Composição Funcional. (a) Célula provida pela metodologia Libra. (b) Célula provida pelo FC..... 84

Figura 38 Resultados dos comparativos elétricos entre as células geradas via Libra em relação as células providas por Composição Funcional para o catálogo *53 NSP Handmade Networks*..... 85

Figura 39 Resultados dos comparativos geométricos entre as células geradas via Libra em relação as células providas por Composição Funcional para o subconjunto do catálogo *4-input P-Class*..... 89

Figura 40 Comparativos de área entre as células desenvolvidos via Libra e provides por Composição Funcional. (a) Caso de maior perda em área (*F1944*, com piora de 12,76%). (b) Caso de maior ganho em área (*F3900*, com redução de 14,74%)..... 90

Figura 41 Resultados do comparativo elétrico entre as células geradas via Libra em relação as células providas por Composição Funcional para o subconjunto do catálogo *4-input P-Class*..... 91

Lista de Tabelas

Tabela 1 Quantidade de transistores necessários para implementação das funções Booleanas presentes nos catálogos *53 NSP Handmade Networks* (LOGICS, 2012) e *4-input P-Class* (CORREIA, *et al.*, 2001) via Composição Funcional (MARTINS, *et al.*, 2012) e Kernel Finder (POSSANI, *et al.*, 2015)..... 23

Tabela 2 Aspectos topológicos referentes às redes de transistores obtidas via Kernel Finder (POSSANI, *et al.*, 2015): quantidade de redes planares e duais *versus* não-planares e não-duais..... 24

Tabela 3 Comparativo geométrico e elétrico das estratégias de posicionamento CPG relativo à CAA para os leiautes das Figuras 32 (a) e (b)..... 75

Tabela 4 Dados topológicos das redes geradas pelo Kernel Finder para o *benchmark 53 NSP Handmade Networks*..... 81

Tabela 5 Dados de *sizing* médio por transistor nas redes lógicas geradas por Composição Funcional e pelo Kernel Finder para o benchmark *53 NSP Handmade Networks*..... 82

Tabela 6 Dados topológicos das redes geradas pelo Kernel Finder para o subconjunto apresentando no Apêndice A representativo do *benchmark 4-input P-Class*..... 87

Tabela 7 Quantidade de transistores necessários para implementação das funções Booleanas presentes no subconjunto de portas lógicas complexas representativo do catálogo *4-input P-Class* obtidas através do FC e KF..... 88

Tabela 8 Dados de *sizing* médio por transistor nas redes lógicas geradas por FC e pelo KF para o *benchmark 4-input P-Class*..... 88

Lista de Abreviaturas e Siglas

ASIC	*Application-Specific Integrated Circuit*
ASTRAN	*Automatic Synthesis of Transistor Networks*
BDD	*Binary Decision Diagram*
BRC	*Boolean Representation Code*
CAA	*Continuous Active Area*
CI	Circuito Integrado
CMOS	*Complementary Metal-Oxide-Semiconductor*
CNF	*Conjunctive Normal Form*
CPG	*Continuous Polysilicon Gates*
DFS	*Depth-First Search*
E/S	Entrada e Saída
EDA	*Electronic Design Automation*
FC	*Functional Composition*
FPGA	*Field Programmable Gate Array*
ILP	*Integer Linear Programming*
ISOP	*Irredundant Sum-of-Products*
KF	Kernel Finder
LBBDD	*Lower Bound BDD*
NMOS	N-type Metal-Oxide-Semiconductor
NSP	Não-Série-Paralelo
OpBDD	*Optimized BDD*
PBF	*Pseudo-Boolean Form*
PD	*Pull-down*
PMOS	P-type Metal-Oxide-Semiconductor
PU	*Pull-up*
SA	*Simulated Annealing*
SAT	*Boolean Satisfiability*
SC	*Standard cell*
SCCG	*Static CMOS Complex Gates*

SOP	*Sum-of-Products*
SP	Série-Paralelo
TA	*Threshold Accepting*
VLSI	*Very-Large-Scale Integration*

Sumário

1 Introdução 18
1.1 Motivação 21
1.2 Objetivos 24
1.3 Organização da Dissertação 26
2 Geração de Redes de Transistores 27
2.1 Fatoração Algébrica Booleana 27
2.1.1 Mintz, *et al.*, 2005 29
2.1.2 Composição Funcional (MARTINS, *et al.*, 2012) 29
2.2 Metodologias Baseadas em Grafos 31
2.2.1 OpBDD e LBBDD (ROSA, *et al.*, 2006) (ROSA, *et al.*, 2007) 32
2.2.2 Kagaris, *et al.*, 2007 35
2.2.3 Kernel Finder (POSSANI, *et al.*, 2015) 37
2.3 Conclusões 40
3 Projeto Automático de Células 42
3.1 Posicionamento de Transistores 43
3.1.1 Posicionamento de Redes de Transistores Duais 44
3.1.1.1 Posicionamento via Caminhos de Euler (UEHARA, *et al.*, 1981) .. 45
3.1.1.2 Posicionamento via Satisfatibilidade Booleana (IIZUKA, *et al.*, 2004) 46
3.1.2 Posicionamento de Redes de Transistores Não-duais 48
3.1.2.1 Posicionamento via Métodos Heurísticos (KIRKPATRICK, *et al.*, 1983) (DUECK, *et al.*, 1990) 50
3.1.2.2 Posicionamento via Satisfatibilidade Booleana (IIZUKA, *et al.*, 2005) 52
3.2 Roteamento *Intra-cell* 54
3.2.1 *Maze Router* (LEE, 1961) 54
3.2.2 Algoritmo de Dijkstra (DIJKSTRA, 1959) 55

3.2.3 Algoritmo A* (HART, *et al.*, 1968) 56
3.3 Ferramentas de Síntese Automática 57
3.3.1 cellTK (KARMAZIN, 2013) 57
3.3.2 ASTRAN (ZIESEMER, *et al.*, 2015) 59
3.3.2.1 Estimativa da Área da Célula 60
3.3.2.2 *Folding* dos Transistores 60
3.3.2.3 Posicionamento 61
3.3.2.4 Roteamento 63
3.3.2.5 Compactação 65
3.4 Conclusões 65
4 Metodologia Proposta 66
4.1 Geração Lógica 67
4.2 Dimensionamento de Transistores 70
4.3 Geração de Leiaute 72
4.4 Conclusões 77
5 Resultados 79
5.1 Metodologia de Experimentação 79
5.2 *Benchmark 53 NSP Handmade Networks* (LOGICS, 2012) 80
5.2.1 Dados Topológicos 81
5.2.2 Dados Geométricos 82
5.2.3 Dados Elétricos 84
5.3 *Benchmark 4-input P-Class* (CORREIA, *et al.*, 2001) 86
5.3.1 Dados Topológicos 87
5.3.2 Dados Geométricos 88
5.3.3 Dados Elétricos 91
6 Conclusões e Trabalhos Futuros 93
Referências 96
Apêndice A Conjunto de Funções dos *Benchmarks* Utilizados 103
Apêndice B Dados dos Experimentos 105
Apêndice C Comparativo de Estratégias de Posicionamento para Redes Não-Série-Paralelo 110
Anexo A Lista de Publicações 112

1 INTRODUÇÃO

Desde o surgimento dos primeiros dispositivos transistorizados em meados da década de 1970, sistemas digitais estão cada vez mais presentes na sociedade contemporânea, sendo fundamentais para inúmeras áreas do conhecimento. Em sua maioria, esses sistemas são compostos por circuitos integrados (CIs), *chips* que contêm diversos componentes eletrônicos (tais como resistores, capacitores e transistores) integrados em uma mesma área. Considerando a linha do tempo do projeto de CIs, inicialmente tais dispositivos eram concebidos de forma manual – abordagem conhecida como *full-custom* –, os projetistas necessitavam posicionar e rotear todos os elementos na área disponível. No entanto, conforme previsto em (MOORE, 1965) e demonstrado na Figura 1, o número de transistores em um CI tende a crescer de forma exponencial duplicando a sua quantidade a cada 18 a 24 meses. Por esse fator, no contexto de projeto VLSI (*Very-Large-Scale Integration*), tal metodologia manual de desenvolvimento é inviável dada à enorme e crescente complexidade causada pelo elevado número de transistores em um *chip* (em que a ordem de grandeza chega ao bilhão em processadores atuais). Com isso, novos paradigmas de desenvolvimento de CIs surgiram, destacando-se, dentre eles, o *semi-custom design*, metodologia onde o projeto do circuito é realizado de forma assistida por ferramentas de EDA (*Electronic Design Automation*), assim diminuindo o esforço manual do projetista, aumentando a produtividade e, consequentemente, encurtando de maneira significativa o *time-to-market* do produto.

Dentre as metodologias *semi-custom*, a *standard cell* (SC) fora, historicamente, a mais amplamente utilizada para projeto de circuitos integrados de aplicação específica (*Application Specific Integrated Circuit*, ASIC), sendo adotada ainda hoje pela indústria de microeletrônica em geral. O paradigma consiste em utilizar bibliotecas de células pré-caracterizadas a fim de reduzir a complexidade nos estágios de desenvolvimento e teste de circuito, visto que tais células já estão previamente

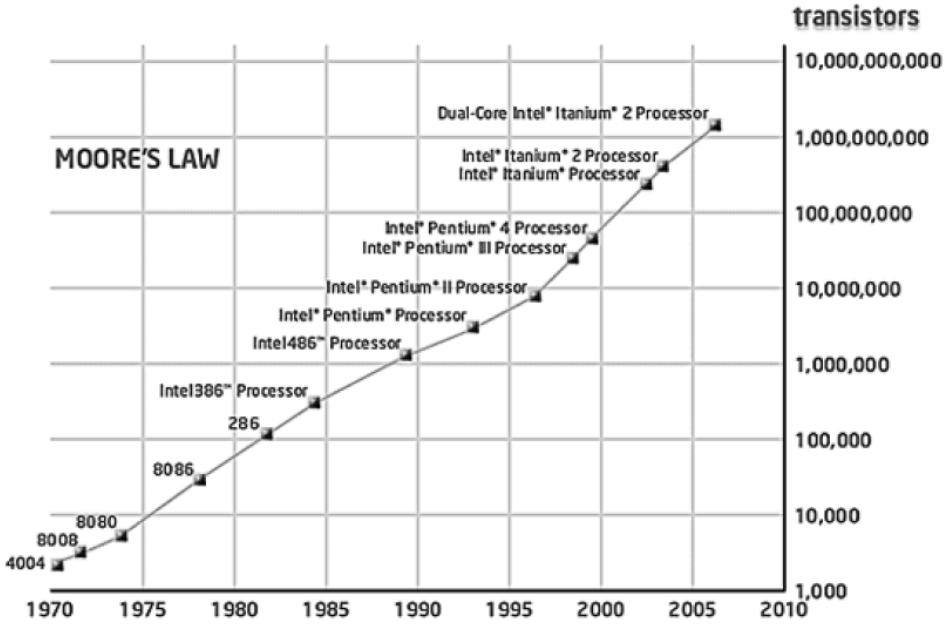


Figura 1 – Gráfico apresentando o crescimento exponencial no número de transistores presentes em processadores Intel ao longo dos anos (Lei de Moore).

Fonte: INTEL, 2007.

projetadas, verificadas e validadas.

As ferramentas de EDA são elementos importantes nos estágios de síntese lógica e física, etapas fundamentais do projeto SC (que tem seu fluxo resumidamente ilustrado na Figura 2). No entanto, ainda que o projeto seja em boa parte realizado de forma automática, a geração das células componentes da biblioteca ainda segue o paradigma *full-custom*. Por conta disso, existem limitações quanto ao número de funções Booleanas cobertas pela biblioteca, o que impede maiores otimizações nos circuitos gerados. Para ilustrar esse fato, tomemos como exemplo os dados apresentados em (DETJENS, *et al.*, 1987), onde o autor demonstra que apenas com a limitação de 5 transistores em série em cada plano é possível implementar 125803 funções diferentes. Considerando isso e o fato de que bibliotecas SC possuem, em geral, cerca de apenas 200 portas lógicas (muitas delas logicamente equivalentes entre si), torna-se evidente a limitação do paradigma *standard cell* no que tange a cobertura lógica oferecida pela biblioteca.

Para contornar os problemas relatados da metodologia SC, em (MORAES, *et al.*, 1995) e (REIS, 2011) propõe-se uma geração de circuitos digitais baseada em portas lógicas complexas (ou *Static CMOS Complex Gates*, SCCG). A principal vantagem dessa abordagem é produzir circuitos que implementem funções de múltiplas entradas utilizando apenas uma porta lógica, ao passo que, através do

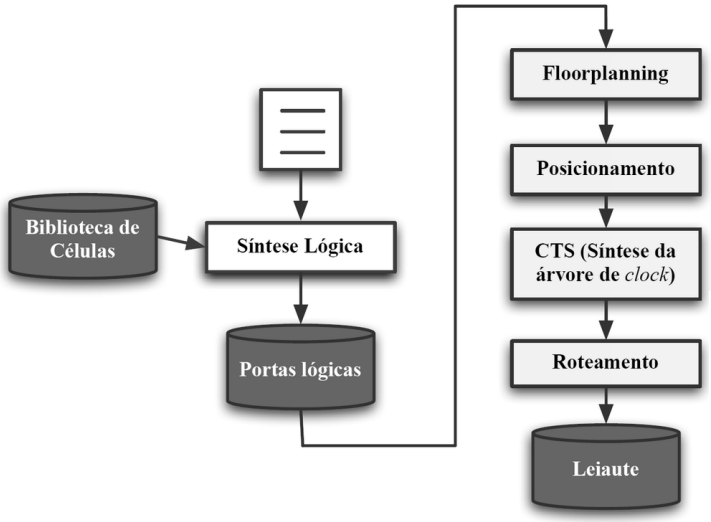


Figura 2 – Fluxo de síntese tradicional (*standard cell*).
Fonte: POSSER, 2011, p. 24.

paradigma SC, esses circuitos seriam obtidos pela combinação de portas lógicas fundamentais. A partir dessa abordagem é possível diminuir o número de transistores da rede lógica, melhorando, assim, aspectos físicos da célula. A Figura 3 ilustra a diferença dos paradigmas *standard cell* e SCCGs para o projeto de circuitos que implementem a função *f* descrita na Equação (1), onde (a) e (b) apresentam soluções obtidas por combinação de portas lógicas fundamentais e pelo projeto por porta lógica complexa, respectivamente, e (c) e (d) ilustram os arranjos de chaves dos circuitos (a) e (b), respectivamente. Nesse caso, a otimização obtida pela segunda abordagem foi de 6 transistores (42,86%, em comparação com a primeira).

$$f = \overline{a \cdot c + b \cdot d} \tag{1}$$

Nesse cenário, a geração de portas lógicas complexas é de grande importância para o *design* de CIs modernos, onde a corrente de *leakage* e a potência estática crescem de maneira inversamente proporcional ao encurtamento do canal do transistor (que chega a 7nm em processadores atuais). Motivado por isso, tem-se como principal objetivo dessa dissertação a proposta de uma metodologia de geração e investigação dos impactos de redes topologicamente irregulares relativo a aspectos físicos de portas lógicas complexas, realizando, para tanto, uma comparação com soluções obtidas através de métodos baseados em paradigmas clássicos e que geram arranjos topologicamente regulares.

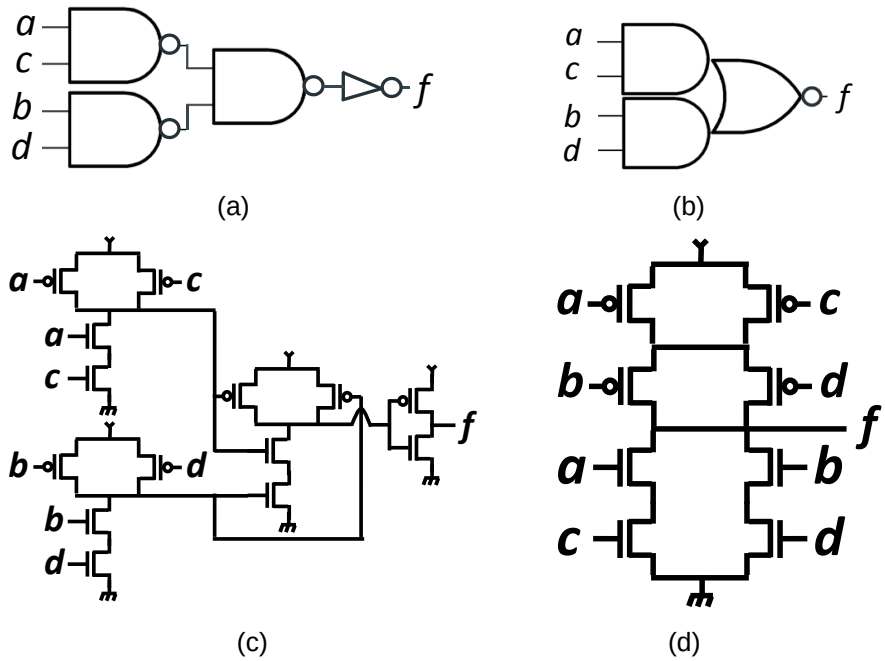


Figura 3 – Circuitos lógicos implementando *f*. (a) *Standard cell*. (b) Porta lógica complexa. (c) Rede de transistores implementando (a). (d) Rede de transistores implementando (b). ■

1.1 Motivação

Conforme descrito anteriormente, portas lógicas complexas são alternativas valiosas para o projeto de CIs digitais. Através desse paradigma, é possível realizar otimizações seja considerando aspectos lógicos (número de transistores) ou físicos (área e *leakage*, por exemplo), comparando-se a tradicional SC. Por esses motivos, é de suma importância investigar diferentes modelos e topologias de portas complexas em relação à complexidade inerente ao projeto das mesmas, bem como referente aos aspectos físicos das soluções obtidas através de diferentes paradigmas.

Especificamente na etapa de geração de arranjos de chaves, estágio inicial no projeto automático de portas complexas, destacamos as metodologias de otimização lógica, responsáveis por produzir redes de transistores especializadas a partir de uma determinada função Booleana de entrada. Proposta originalmente em (SHANNON, 1938), a geração otimizada de arranjos de chaves é um desafio de pesquisa ainda em aberto. Nesse contexto, dois diferentes paradigmas se destacam: a fatoração algébrica Booleana e as metodologias baseadas em grafos, constituindo as duas principais frentes para a minimização multinível.

Proposta originalmente em (LAWLER, 1964), a fatoração Booleana caracteriza-se por utilizar manipulações algébricas na função de entrada a fim de prover uma rede de chaves otimizada. Dado ao seu aspecto metodológico, que

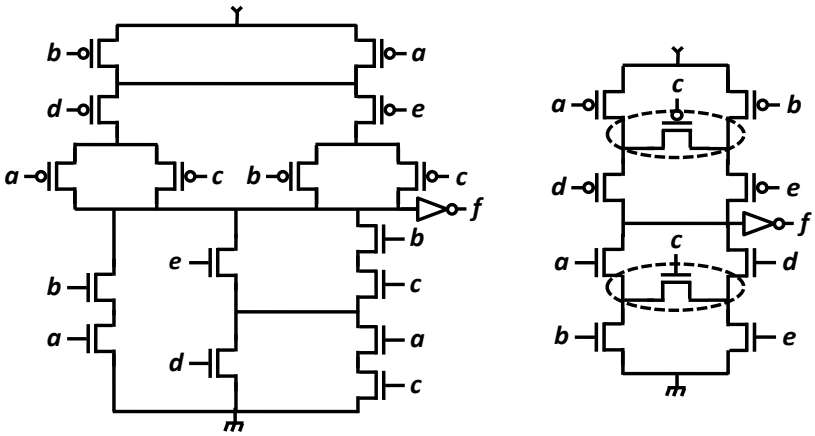
consiste na composição iterativa do arranjo lógico, as soluções obtidas por fatoração Booleana apresentam um comportamento topológico regular, não apresentando falta de planaridade, dualidade ou, ainda, ocorrência de ligações *bridge* na célula (ou seja, todos os componentes interconectam-se apenas através de ligações série-paralelo). O método de Composição Funcional (*Functional Composition*, FC) (MARTINS, *et al.*, 2012) é o estado da arte considerando o paradigma de fatoração Booleana. Baseando-se em combinações entre os elementos fundamentais da função alvo (literais) para composição da forma fatorada, a metodologia é capaz de prover redes série-paralelo mínimas ao custo de uma alta complexidade computacional.

Uma alternativa eficiente à minimização por fatoração Booleana é a de otimização baseada em grafos, paradigma proposto originalmente em (ZHU, *et al.*, 1993). Tal abordagem tem como base o compartilhamento de arestas do grafo representativo do arranjo de chaves, fazendo com que um mesmo componente (transistor) possa ser sensibilizado a partir de diferentes caminhos da porta lógica. Destacados por sua relevância, citam-se aqui os trabalhos propostos em (ROSA JUNIOR, *et al.*, 2006), (ROSA JUNIOR, *et al.*, 2007) e (KAGARIS, *et al.,* 2007), todos apresentando bons resultados quando comparados com os métodos de fatoração Booleana (seja relativo ao número de chaves ou a complexidade dos algoritmos). Finalmente, em (POSSANI, *et al.*, 2015) é apresentado o Kernel Finder (KF), considerado o método estado da arte na geração de redes de transistores, ou seja, que apresenta os melhores resultados na tarefa de prover redes de transistores sob demanda para uma função Booleana. A metodologia, por sua vez, tem como base a busca de padrões lógicos na função Booleana de entrada para construção de arranjos de chaves a partir de *templates* pré-estabelecidos, o que garante bons resultados e relativamente baixa complexidade algorítmica.

A fim de comparar os diferentes paradigmas de geração lógica de portas complexas e as metodologias estado da arte citadas anteriormente, a Figura 4 ilustra duas redes de transistores que implementam a mesma função Booleana *f* definida na Equação (2). Em (a) tem-se a solução gerada por Composição Funcional, enquanto que (b) apresenta a solução obtida através do Kernel Finder. Pode-se observar que (b) apresenta um arranjo menor (em termos do número de chaves), solução obtida graças ao compartilhamento do transistor *c* ligado em *bridge* na rede (destacado na figura). Esse cenário é recorrente considerando-se conjuntos maiores de funções Booleanas, como fica evidente na Tabela 1 onde tem-se uma otimização de 26,28%

e 6,89% para as funções Booleanas dos catálogos *53 NSP Handmade Networks* (LOGICS, 2012) e *4-input P-Class* (CORREIA, *et al.*, 2001), respectivamente, no que tange a quantidade de transistores em soluções obtidas via KF em relação a redes produzidas por FC.

$$f = a \cdot b + a \cdot c \cdot e + d \cdot e + b \cdot c \cdot d \tag{2}$$



(a) (b)

Figura 4 – Redes de transistores implementando *f*. (a) Solução obtida por fatoração Booleana (FC). (b) Solução via minimização baseada em grafos (KF). ■

Tabela 1 – Quantidade de transistores necessários para implementação das funções Booleanas presentes nos catálogos *53 NSP Handmade Networks* (LOGICS, 2012) e *4-input P-Class* (CORREIA, *et al.*, 2001) via Composição Funcional (MARTINS, *et al.*, 2012) e Kernel Finder (POSSANI, *et al.*, 2015).

Benchmark	**Composição Funcional**	***Kernel Finder***	***Redução (%)***
53 NSP Handmade Networks	487	359	26,28
4-input P-Class	102668	95595	6,89

Conforme já observado, as otimizações relativas à quantidade de chaves presentes nas soluções obtidas via metodologias baseadas em grafos – tal como o KF – são decorrentes do compartilhamento de caminhos (transistores) na célula. No entanto, esses ramos não-série-paralelo levam a outras características relevantes no contexto da geração automática do leiaute: a ocorrência de arranjos topologicamente não-planares e não-duais. Como exemplo, a Figura 5 ilustra a rede de transistores obtida via KF que implementa a função Booleana *f* definida na Equação (3), a qual apresenta todas essas características particulares (ou seja, é não-planar e não-dual). É válido ressaltar que, devido a esses aspectos, a solução possui planos

desbalanceados, ou seja, com diferença no número de transistores PMOS e NMOS que compõem os planos *pull-up* (PU) e *pull-down* (PD), respectivamente. Novamente, essas particularidades estendem-se a boa parte do espectro de funções analisados pela Tabela 2, onde 32% do total das soluções geradas via KF apresentam falta de planaridade e dualidade. Nesse contexto, faz-se necessário compreender o impacto que a irregularidade topológica presente no *netlist* trará ao projeto da célula física, possibilitando, assim, uma análise mais precisa da qualidade das soluções apresentadas por metodologias baseadas em grafos quando comparadas as providas por métodos de fatoração algébrica (topologicamente regulares).

$$f = b \cdot c \cdot d + \overline{a} \cdot \overline{b} \cdot \overline{c} + a \cdot c + a \cdot b \tag{3}$$

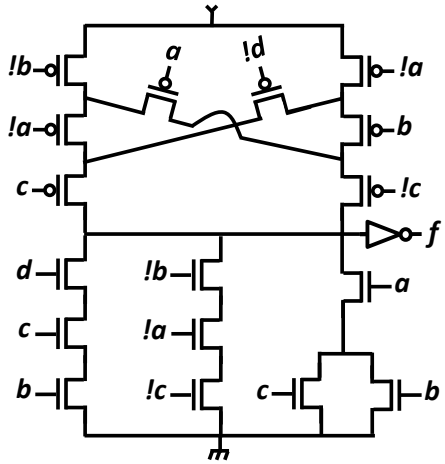


Figura 5 – Solução para *f* não-planar e não-dual provida pelo KF. ■

Tabela 2 – Aspectos topológicos referentes às redes de transistores obtidas via Kernel Finder (POSSANI, *et al.*, 2015): quantidade de redes planares e duais *versus* não-planares e não-duais.

Benchmark	***Planar e dual***	***Não-planar e não-dual***
53 NSP Handmade Networks	53 (100%)	0 (0%)
4-input P-Class	3183 (79,93%)	799 (20,07%)
NPN-class up to 5-input	416359 (67,58%)	199766 (32,42%)

1.2 Objetivos

Considerando o cenário descrito na seção anterior e as lacunas existentes relativas ao projeto físico de arranjos de transistores sem restrições topológicas, esse trabalho tem como objetivo principal propor a metodologia de projeto Libra, uma solução automática para geração de portas lógicas complexas obtidas a partir de redes de transistores providas pelo Kernel Finder. Com isso, torna-se possível realizar

um comparativo (em termos geométricos e elétricos) com células obtidas a partir da minimização lógica via Composição Funcional (abordagem tradicional).

Para que o objetivo principal seja alcançado, definiu-se um conjunto de objetivos específicos apresentados a seguir:

- Geração lógica das portas complexas: a partir da definição dos *benchmarks* utilizados, intenta-se projetar as redes de transistores sob demanda para implementação das funções Booleanas objetivo. Para isso, partiu-se da hipótese de que a qualidade das soluções (em termos geométricos e elétricos) é relacionada de forma direta ao número de transistores do arranjo lógico. Com isso, o Kernel Finder foi empregado como motor da pesquisa, visto que é uma metodologia que fornece redes otimizadas se comparadas as soluções providas por métodos que seguem o paradigma tradicional de fatoração;
- Dimensionamento da porta complexa: essa etapa consiste em definir os *sizings* dos transistores componentes da solução proposta. Para que a comparação dos diferentes paradigmas de minimização lógica no contexto do *design* de portas complexas seja realizada de forma justa, o mesmo método deve ser utilizado para dimensionamento das soluções obtidas pelo FC e através do método proposto (que utiliza o KF como base). Para isso, faz-se necessário a implementação de uma metodologia versátil que possibilite lidar com diferentes topologias de redes sem uma especificação em particular;
- Projeto físico da célula: com os *netlists* já dimensionados, a próxima etapa consiste em projetar o leiaute do circuito. Utilizou-se como motor desse estágio o ASTRAN, ferramenta *open-source* onde é possível realizar a síntese da célula (*folding*, posicionamento, roteamento e compactação) de forma automática e sem restrições topológicas;
- Experimentação: a última etapa consiste na realização de experimentos que comparem as soluções obtidas por diferentes paradigmas de minimização lógica. Para isso, definiu-se um cenário de testes para levantamento de dados elétricos (capacitância de entrada, tempos de transição e propagação e potências de *leakage*, interna e de chaveamento) e geométricos (área, comprimento de fio, número de contatos e densidade de roteamento) de cada célula projetada.

1.3 Organização da Dissertação

Este trabalho está organizado como segue. O capítulo 2 realiza uma revisão bibliográfica em métodos de geração de redes de transistores otimizadas e sob demanda. O capítulo 3 trata dos métodos e ferramentas de geração automática de leiaute. O capítulo 4 apresenta a metodologia desenvolvida nessa dissertação. No capítulo 5 têm-se os resultados dos experimentos realizados. Finalmente, no capítulo 6 são apresentadas as conclusões da dissertação, bem como os trabalhos futuros propostos.

2 GERAÇÃO DE REDES DE TRANSISTORES

Desde os trabalhos pioneiros de Shannon (SHANNON, 1938) a geração de redes de chaves otimizadas é tema frequente em pesquisas em computação devido a sua versatilidade de aplicações. Seja em redes de transistores – utilizados na microeletrônica moderna – ou em redes de relés e válvulas – utilizados em sistemas eletromecânicos e computacionais antigos –, as metodologias de geração e minimização lógica são elementos importantíssimos no projeto de circuitos eletrônicos otimizados, ou seja, contendo um menor número de elementos. São considerados métodos clássicos para minimização lógica de dois níveis os apresentados em (KARNAUGH, 1953), (QUINE, 1955) (MCCLUSKEY, 1956). Apesar da simplicidade relativa às suas implementações, tais algoritmos não são alternativas viáveis no projeto de dispositivos reais devido à alta complexidade computacional inerente aos mesmos. Uma alternativa para contornar esses problemas, ainda no cenário de minimização de dois níveis, é o Espresso (MCGEER, *et al.*, 1993).

Mais recentemente, dois paradigmas de minimização lógica vem ganhando destaque na literatura: a fatoração algébrica Booleana e as metodologias baseadas em grafos, ambas constituindo métodos de minimização multinível. As subseções a seguir tratarão mais detalhadamente desses paradigmas, visto que os mesmos serão diretamente utilizados nessa pesquisa.

2.1 Fatoração Algébrica Booleana

Segundo (BRAYTON, *et al.*, 1984), uma forma fatorada pode ser definida como um produto formado por apenas um literal, por produtos de formas fatoradas, como uma soma formada por apenas um literal ou como uma soma de formas fatoradas. Em outras palavras, uma função em sua forma fatorada pode ser vista como uma representação algébrica dessa função com a adição de parênteses. Como

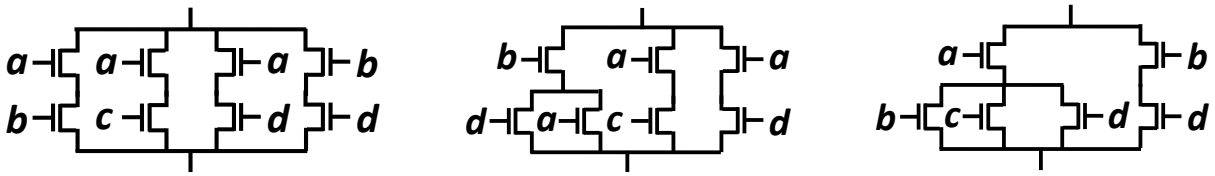
exemplo, as equações Booleanas definidas nas equações (5) e (6) são representações em formas fatoradas da função *f* descrita na Equação (4).

$$f = a \cdot b + a \cdot c + a \cdot d + b \cdot d \quad (4)$$

$$f = b \cdot (a + d) + a \cdot c + a \cdot d \quad (5)$$

$$f = a \cdot (b + c + d) + b \cdot d \quad (6)$$

Conforme exemplificado, as Equações (5) e (6) demonstram que existem pelo menos duas formas de se fatorar a função *f* definida em (4). De fato, em geral existem diversas formas de se fatorar algebricamente uma expressão, obtendo-se resultados diferentes no que tange ao número de literais e, consequentemente, na quantidade de transistores presentes no *netlist* da porta lógica complexa correspondente. Para demonstrar essa propriedade, considere a Figura 6, onde está ilustrado o plano *pull-down* das redes de transistores que implementam as funções descritas pelas Equações (4), (5) e (6). Nota-se que a função com o menor número de literais – Equação (6) – implica em um arranjo com um menor número de chaves – (c). A busca pelos métodos de fatoração que encontrem os melhores resultados relativos à quantidade de literais na função é o principal desafio na otimização de funções Booleanas via fatoração algébrica Booleana.



(a) (b) (c)

Figura 6 – Planos logicamente equivalentes compondo *f*. (a) Solução com 8 transistores proveniente da Equação (4). (b) Solução com 6 transistores descrita em (5). (c) Solução definida em (6). ■

Em (LAWLER, 1964) é proposto um método que garante solução mínima para o cálculo da menor forma fatorada de uma função Booleana. No entanto, devido a sua altíssima complexidade computacional, o cômputo dessas soluções torna-se inviável na prática. Em (BRAYTON, 1987) é apresentada uma heurística para otimizar as deficiências temporais do método de Lawler ao custo da perda da garantia de solução ótima. Mais recentemente, (MINTZ, *et al.*, 2005) (seção 2.1.1) introduz uma técnica

de fatoração recursiva através do particionamento de grafos que apresenta solução exata para funções *read-once*[1]. Finalmente, (MARTINS, *et al.*, 2012) (seção 2.1.2) propõe o algoritmo estado da arte do paradigma de fatoração Booleana a partir do emprego da técnica de composição funcional.

2.1.1 Mintz, *et al.*, 2005

O algoritmo proposto em (MINTZ, *et al.*, 2005) faz uso da técnica de particionamento de grafos para obter a forma fatorada de funções Booleanas. Partindo de uma função *f* em sua polaridade direita e inversa (*!f*), são construídos dois grafos G_f e $G_{!f}$ que representam as funções *on-set* e *off-set* de *f*, respectivamente. Após a construção desses grafos, calculam-se as intersecções entre cada cubo componente de *f* para que seja executado o particionamento de ambos os grafos. As subfunções obtidas pós-particionamento são utilizadas no passo recursivo do algoritmo, o qual compõe a função *f* a partir de somas ou produtos de suas subfunções. (GOLUMBIC, *et al.*, 2008) propõe uma rotina que, aliada ao método descrito, é capaz de garantir solução exata para os casos onde são encontradas subfunções *read-once* nos níveis mais internos da recursão.

2.1.2 Composição Funcional (MARTINS, *et al.*, 2012)

A metodologia de fatoração Booleana proposta em (MARTINS, *et al.*, 2012) é o algoritmo estado da arte no paradigma de fatoração algébrica de funções Booleanas. O método baseia-se no paradigma de composição funcional originalmente proposto em (MARTINS, *et al.*, 2010), que consiste na construção iterativa da forma mínima (em termos de números de literais) da função alvo.

De forma simplificada, o algoritmo parte do princípio da construção de *n* baldes (*buckets*), onde cada *bucket* B_n armazena todas as soluções com *n* literais geradas a partir das combinações entre as funções presentes nos *n-1* baldes já criados. Portanto, o método atua de forma iterativa: primeiramente, constrói-se o *bucket* B_1, responsável por representar todos os literais (tamanho $n = 1$) presentes na função *f* (onde as polaridades dos mesmos são levadas em consideração). Após, combinam-se os elementos em B_1 tomados dois a dois (através das operações lógicas *AND* e *OR*), formando o balde B_2 com soluções de tamanho $n = 2$. De forma análoga,

[1] Uma função é *read-once* se cada variável aparecer uma única vez na expressão lógica.

o *bucket* B_3 é gerado a partir das combinações algébricas entre B_1 consigo mesmo e B_1 e B_2. O processo segue com a composição do balde B_4 através das combinações de B_1 consigo mesmo, B_1 e B_3 e B_2 consigo mesmo. A cada iteração é verificada se a função alvo não foi atingida por um dos elementos do *bucket* computado, sendo esse o critério de parada do algoritmo (vale ressaltar que o método de verificação funcional implementado – *Boolean Representation Code*, BRC – possui complexidade computacional constante). A Figura 7 ilustra de forma geral o processo de criação de *buckets*. Relativo ao desempenho da metodologia, ainda que algumas otimizações relativas a combinações redundantes e descarte de *buckets* sejam propostas em (MARTINS, *et al.*, 2012), a aplicabilidade do método fica restrita a funções de até sete variáveis dado o alto custo computacional inerente ao armazenamento do conteúdo dos *buckets*.

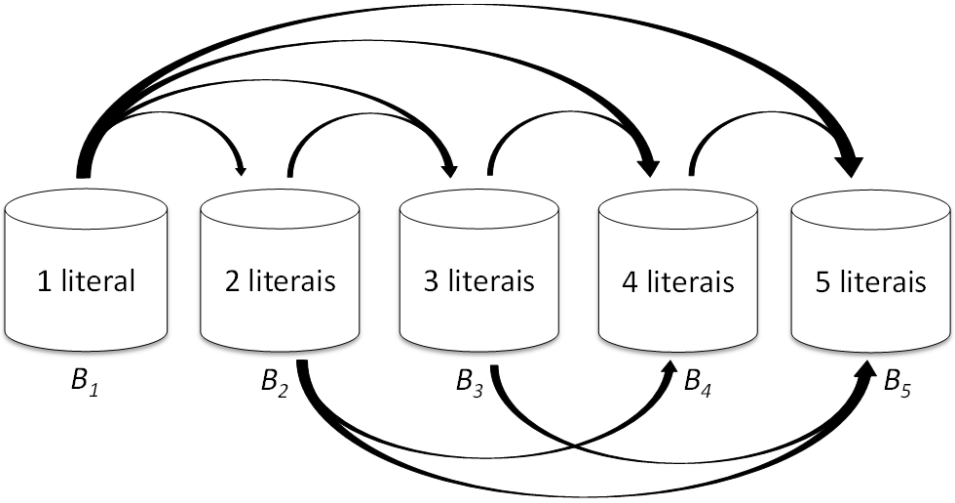


Figura 7 – Construção iterativa de *buckets* na metodologia de Composição Funcional.
Fonte: Adaptado de MARTINS, *et al.*, 2012, p. 5.

Para melhor entendimento da metodologia, considere a função *f* dada pela Equação (7) definida abaixo. A Figura 8 ilustra alguns dos *buckets* criados conforme descrito no parágrafo anterior, partindo do balde B_1 até B_8, onde *f* pode ser encontrada na sua forma fatorada ótima (em negrito). Nota-se que a função de entrada *f* – Equação (7) –, que possui 12 literais, pode ser representada, em sua menor forma, por uma função fatorada equivalente de 8 literais, o que significa um ganho bastante expressivo de 4 chaves por plano lógico.

$$f = (\bar{a} \cdot c \cdot d) + (\bar{a} \cdot b \cdot d) + (a \cdot \bar{b} \cdot \bar{c}) + (a \cdot \bar{b} \cdot \bar{d}) \quad (7)$$

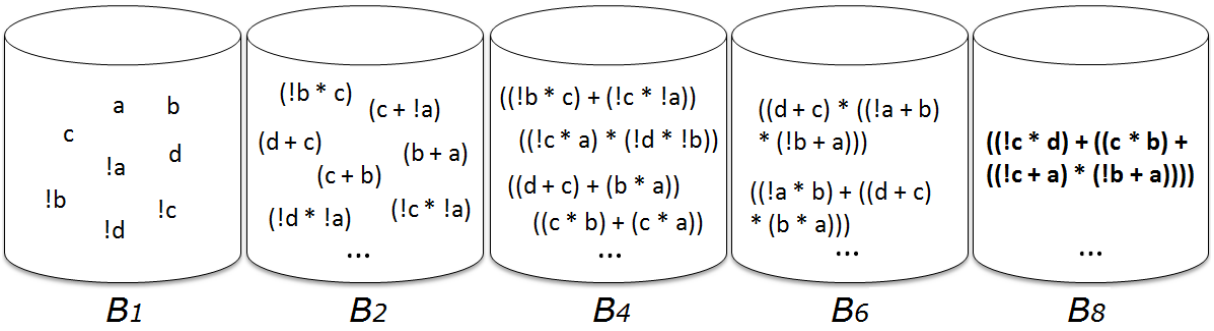


Figura 8 – Processo de criação de *buckets* para fatoração da função *f*. ■

Um aspecto importante para o contexto do trabalho desenvolvido nessa dissertação é o fato de que as combinações *AND* e *OR* realizadas no cômputo de novas soluções por Composição Funcional resultam em redes de transistores série-paralelo, ou seja, onde todos os componentes do arranjo lógico da porta complexa estão interconectados de forma série ou paralela entre si. Essa particularidade topológica influencia na escolha de metodologias de síntese física de células, especialmente de posicionamento de transistores, conforme detalhado na seção 3.1.

2.2 Metodologias Baseadas em Grafos

Recentemente, um paradigma vem ganhando destaque na geração de redes de transistores otimizadas: as metodologias baseadas em grafos. Sendo bastante presente em algoritmos e ferramentas de EDA nas várias etapas do fluxo VLSI, a estrutura de grafos é de grande valia devido a já consolidada teoria dos grafos, a qual contribui com um conjunto de regras e algoritmos bem conhecidos e amplamente desenvolvidos que podem ser aplicados em diversas áreas de conhecimento, incluindo no projeto de células sob demanda, escopo desse trabalho.

Conforme mencionado no capítulo introdutório dessa dissertação, as metodologias baseadas em grafos são capazes de gerar redes de chaves com aspectos topológicos singulares quando comparadas com os métodos de otimização baseados em fatoração algébrica Booleana. Devido ao compartilhamento de arestas presente em otimizações nesse paradigma, é comum a geração de soluções não-série-paralelo, onde há ao menos um componente da rede conectado em ponte (*bridge*) com seus vizinhos. Essa característica é a principal responsável pelo surgimento de arranjos topologicamente não-duais e não-planares, além de eventualmente causar falta de *matching* vertical entre os transistores nos planos. No

entanto, se por um lado essas metodologias causam irregularidades topológicas, por outro os métodos baseados em grafos geram soluções mais otimizadas em termos do número total de transistores em comparação com as obtidas via fatoração Booleana, o que potencialmente conduz a circuitos com melhores características físicas (elétricas e geométricas). Ademais, vale ressaltar que os algoritmos mais atuais no cenário de geração de redes através de manipulações em grafos possuem um ganho expressivo em tempo de execução e complexidade computacional quando comparados a outros métodos de diferentes paradigmas. Por conta disso, as metodologias apresentadas a seguir constituem uma eficiente ferramenta para a etapa de geração de portas lógicas complexas.

As otimizações lógicas baseadas em grafos surgem em (ZHU, *et al.*, 1993), que propõe uma metodologia de compartilhamento de arestas em grafos a partir das funções geradas pelo algoritmo (QUINE, 1955) (MCCLUSKEY, 1956). Buscando otimizar o número de literais na função, o método se caracteriza por não inserir caminhos falsos que alterem a função lógica original, evitando a reverificação da função, processo que consumiria um tempo significativo no fluxo de execução.

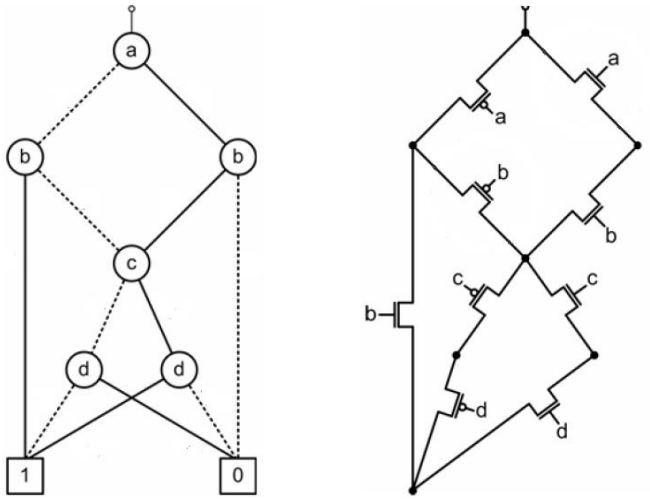
A partir desse trabalho, outras propostas surgiram buscando otimizar tanto o número de literais quanto a complexidade computacional das metodologias do paradigma baseado em grafos. Nas seções a seguir estão detalhadas as metodologias OpBDD (ROSA JUNIOR, *et al.*, 2006) e LBBDD (ROSA JUNIOR, *et al.*, 2007) (seção 2.2.1), as quais fazem uso de diagramas de decisão binária (*binary decision diagrams* – BDDs*)* para representação e otimização da rede de chaves. Após será apresentado o método de minimização lógica proposto por (KAGARIS, *et al.*, 2007) (seção 2.2.2) e, finalmente, o Kernel Finder (KF) (POSSANI, *et al.*, 2014) (seção 2.2.3), metodologia estado da arte e que fora utilizada diretamente nesse trabalho.

2.2.1 OpBDD e LBBDD (ROSA JUNIOR, *et al.*, 2006) (ROSA JUNIOR, *et al.*, 2007)

Optimized BDD (OpBDD) (ROSA JUNIOR, *et al.*, 2006) e *Lower Bound BDD* (LBBDD) (ROSA JUNIOR, *et al.*, 2007) são duas técnicas de minimização lógica que utilizam BDDs (grafos acíclicos direcionados) como estrutura de dados principal. Para revisar essa forma de representação de funções Booleanas, tome como exemplo a Figura 9 (a) que mostra o BDD da função *f* descrita na Equação (8), bem como a rede de chaves correspondente ao BDD (b). No exemplo, a aresta representada por uma linha cheia que parte de um nodo do BDD em (a) originará um transistor NMOS na

rede ilustrada em (b), enquanto que a aresta representada por uma linha tracejada no BDD servirá como representação a um transistor PMOS no arranjo lógico.

$$f = (\overline{a} \cdot b) + (a \cdot b \cdot \overline{c} \cdot \overline{d}) + (a \cdot b \cdot c \cdot d) + (\overline{a} \cdot \overline{b} \cdot c \cdot d) + (\overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d}) \qquad (8)$$



(a) (b)

Figura 9 – BDD de *f* (a). Rede de transistores correspondente (b). ■
Fonte: Adaptado de ROSA JUNIOR, 2008, p. 51.

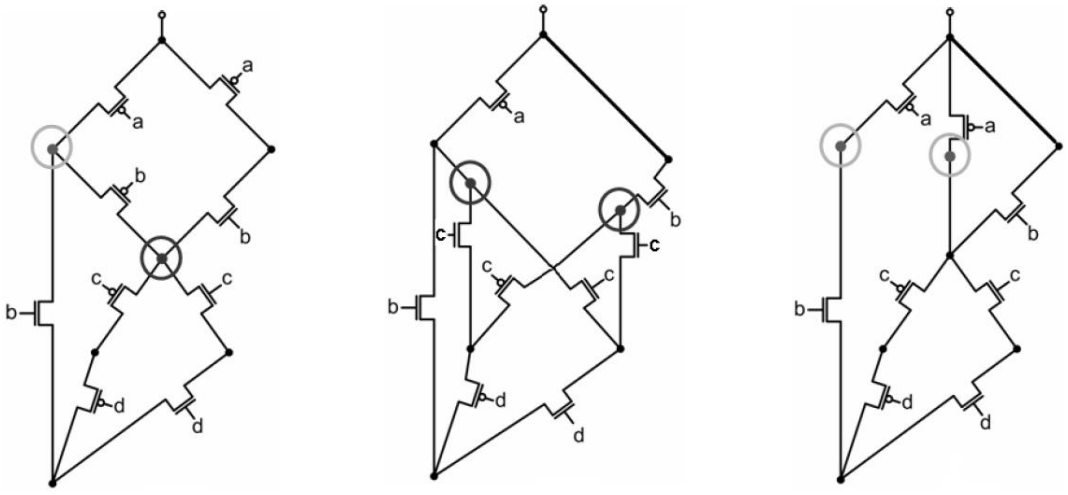
OpBDD é uma técnica que faz uso da propriedade *unateness* de funções Booleanas. Uma função é dita *unate* se todas as variáveis que a compõem aparecem em apenas uma polaridade. Caso a função tenha variáveis em ambas as polaridades, ela pode ser *non-unate* ou *binate*, onde a última caracteriza-se por ter todas as variáveis em suas polaridades direta e inversa. Através da avaliação dessa propriedade, o método introduz curtos-circuitos (fios) no lugar de chaves que não afetam a lógica da função. Ademais, (ROSA JUNIOR, *et al.*, 2006) define o conceito de dominância de caminhos numa rede lógica. Para ilustrar essa característica, considere a função *f*, descrita pela Equação (9) abaixo, composta por dois cubos $f_{C1} = (!b \cdot c \cdot d \cdot e)$ e $f_{C2} = (c \cdot d)$. Notemos que, nesse caso, $f_{C2} \subset f_{C1}$, ou seja, o caminho obtido pela parcela f_{C1} é dominante na função *f*. Na prática, essa propriedade faz com que os literais redundantes *c* e *d* da função *f* possam ser eliminados da solução sem alteração da sua lógica. ■

$$f = (\overline{b} \cdot c \cdot d \cdot e) + (c \cdot d) \qquad (9)$$

Uma propriedade importante na representação de redes de chaves através de BDDs é a ordem em que as variáveis são dispostas, a qual altera a árvore gerada. Por conta disso, existem diferentes possíveis soluções para minimização de uma mesma função, tornando o processo de ordenamento das variáveis uma etapa crítica na minimização lógica (NP-difícil). (ROSA JUNIOR, *et al.*, 2006) descreve dois métodos de ordenamento: o primeiro via força-bruta, onde *n!* grafos de *n* variáveis são gerados e o algoritmo segue a partir do grafo que apresenta o menor número de chaves; o segundo, fazendo uso da técnica de *sifting*, uma heurística originalmente proposta em (RUDELL, 1993) e que não garante solução ótima para o problema descrito.

LBBDD é uma extensão do algoritmo anterior. Como descrito, através da aplicação do método OpBDD, que introduz conceitos de *unateness* e de dominância da função lógica na otimização da rede de chaves, é possível substituir transistores redundantes por curtos-circuitos no arranjo de chaves. No entanto, notou-se que, para determinados casos, eliminar um transistor levava a uma mudança na função lógica original, assim invalidando a solução obtida. (ROSA JUNIOR, *et al.*, 2007) descreve métodos para correção de erros que possam ter sido gerados pela otimização via OpBDD. O LBBDD tem como principal característica a duplicação dos nodos conectados aos transistores candidatos a remoção, conforme ilustra a Figura 10, onde intenta-se remover o transistor *b* localizado entre os nodos destacados de (a). Para tanto, as alternativas (b) e (c) são computadas pelo LBBDD, escolhendo a solução que possua um menor número de chaves (nesse caso, a solução (c) é escolhida por apresentar 8 transistores, ao passo que (b) contém 9 chaves no total).

Ademais, o LBBDD inclui a possibilidade de buscar apenas por soluções que respeitem um determinado número de transistores em série, uma restrição importantíssima na geração de arranjos de chaves sob demanda devido a aspectos elétricos. No caso da rede obtida não respeitar o tamanho máximo do ramo série entre alimentação e saída, o algoritmo faz uma busca no grafo original pelas arestas e nodos que não foram candidatos anteriormente para remoção e verifica se a eliminação do elemento é possível (ou seja, se isso não irá causar erros funcionais).



(a) (b) (c)

Figura 10 – Duplicação de nodos através do LBBDD. (a) Rede original. (b) Solução com 9 transistores. (c) Solução com 8 transistores.

Fonte: ROSA JUNIOR, 2008, p. 53.

2.2.2 Kagaris, *et al.*, 2007

Em (KAGARIS, *et al.*, 2007) é proposto um método de geração de arranjo de chaves que busca compartilhar caminhos num grafo para se obter uma rede de transistores otimizada. Por conta dessa abordagem, o método é capaz de produzir redes topologicamente não-complementares e que contenham transistores ligados em *bridge*.

O método parte de uma soma-de-produtos para a construção do grafo. Na primeira iteração, o primeiro cubo f_{C1} da soma-de-produtos (*sum-of-products*, SOP) é lido e um vértice contendo esse cubo é criado. Na segunda iteração – atuando sobre o cubo posterior f_{C2} – é investigada a possibilidade de compartilhamento de literais entre f_{C1} e f_{C2}. Caso exista, então uma aresta que contém os literais compartilhados é criada, ligando-a as partes não semelhantes (ou seja, sem chaves controladas pela mesma entrada) das parcelas da função lida. Esse processo repete-se até que todos os cubos da SOP de entrada sejam computados.

Para exemplificar a metodologia, considere como exemplo a Figura 11 (a) onde está ilustrado o grafo resultante após a terceira iteração – portanto, posterior a leitura do cubo $f_{C3} = (f \cdot e \cdot a \cdot b)$ – da SOP *f* descrita na Equação (10).

$$f = (e \cdot g \cdot b \cdot a) + (a \cdot c \cdot k \cdot d) + (f \cdot e \cdot a \cdot b) + (h \cdot g \cdot e \cdot a) + (f \cdot a \cdot h \cdot e) \quad (10)$$

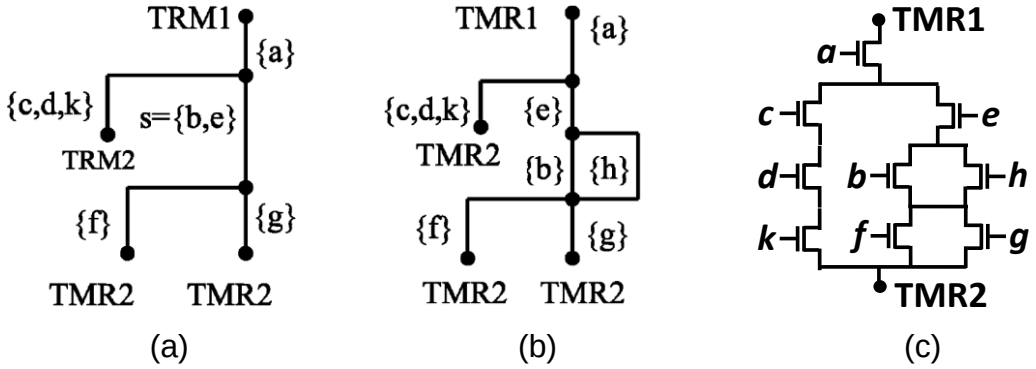


Figura 11 – Instâncias de execução do método de Kagaris, *et al*. (a) Grafo anterior à inserção do cubo f_{C4}. (b) Grafo após a inserção do cubo f_{C4}. (c) Rede de transistores resultante.

Fonte: Adaptado de KAGARIS, *et al.*, 2007, p. 2.

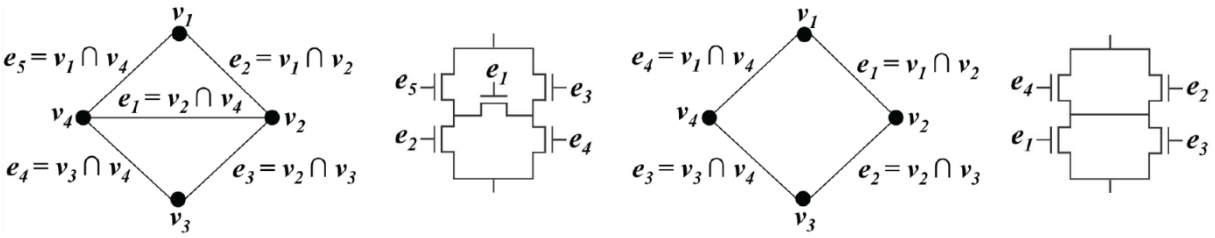
Percebe-se que o literal *a* é compartilhado por três diferentes caminhos terminais da célula (define-se caminho terminal como um caminho entre os nodos *TMR1* e *TMR2*), enquanto os literais *b* e *e* são compartilhados por dois diferentes caminhos. Numa quarta iteração, Figura 11 (b), onde pretende-se inserir o cubo $f_{C4} = (h \cdot g \cdot e \cdot a)$ no grafo, a aresta que contém *b* e *e* dará origem a duas novas arestas: uma contendo apenas o literal *b* e outra contendo somente *e*, sendo esse compartilhado por quatro caminhos terminais diferentes. Por fim, vale ressaltar que o cubo $f_{C5} = (f \cdot a \cdot h \cdot e)$ (ainda não computado) já possui um caminho válido no grafo gerado pelas construções anteriores. Essa característica pode criar falsos caminhos, sendo necessário um estágio de verificação funcional da rede que deve atuar a cada iteração realizada. Finalmente, a rede de transistores gerada a partir da solução produzida pelo grafo da Figura 11 (b) pode ser vista na Figura 11 (c). ■

Vale ressaltar que, pelo método descrito, a ordem em que os cubos são lidos na SOP influenciam na criação do grafo, tal como ocorre nas metodologias OpBDD e LBBDD descritas anteriormente. Kagaris, *et al*. adota uma solução para esse problema a partir de força-bruta, testando todas as entradas e escolhendo a que resulte em uma rede lógica com menor número de chaves. Um outro aspecto que deve ser observado é que a ordem em que os literais aparecem nos cubos não afeta a solução, visto que o algoritmo trata cada literal como um elemento do conjunto de entrada.

2.2.3 Kernel Finder (POSSANI, *et al.*, 2015)

O Kernel Finder (KF) é a metodologia de geração de redes de transistores sob demanda que atualmente apresenta os melhores resultados relativos à minimização da quantidade de chaves na rede lógica.

O algoritmo parte de uma soma-de-produtos irredundante (*irredundant sum-of-products* – ISOP) para combinar cubos formando *kernels, templates* usados para combinar os cubos da função a partir da relação que ocorre entre eles. Possani *et al*. define dois tipos de *kernels*: o *NSP-kernel* e o *SP-kernel*, apresentados nas Figuras 12 (a) e (b), respectivamente. Com essas estruturas pré-definidas, torna-se possível evitar a escolha gulosa que ocorre nas fases inicias da maioria dos algoritmos de otimização lógica (a qual leva a mínimos locais indesejados).



(a) (b)

Figura 12 – *Kernels* implementados no Kernel Finder. (a) *NSP-kernel.* (b) *SP-kernel*.
Fonte: POSSANI, *et al.*, 2015, p. 3, 4.

Conforme visto na Figura 12, as arestas dos grafos (*templates*) contêm os literais em comum dos cubos representados nos vértices dos mesmos. Para exemplificar, considere a aresta e_1 da Figura 12 (a) que possui os literais compartilhados dos cubos v_1 e v_2. Isso se estende para o restante das arestas, como pode ser visto na imagem.

O fluxograma geral do KF pode ser visto na Figura 13. Seguindo a figura, a metodologia pode ser decomposta em duas grandes etapas: *Kernel Identification* e *Network Composition*. Abaixo encontra-se a descrição detalhada do fluxograma:

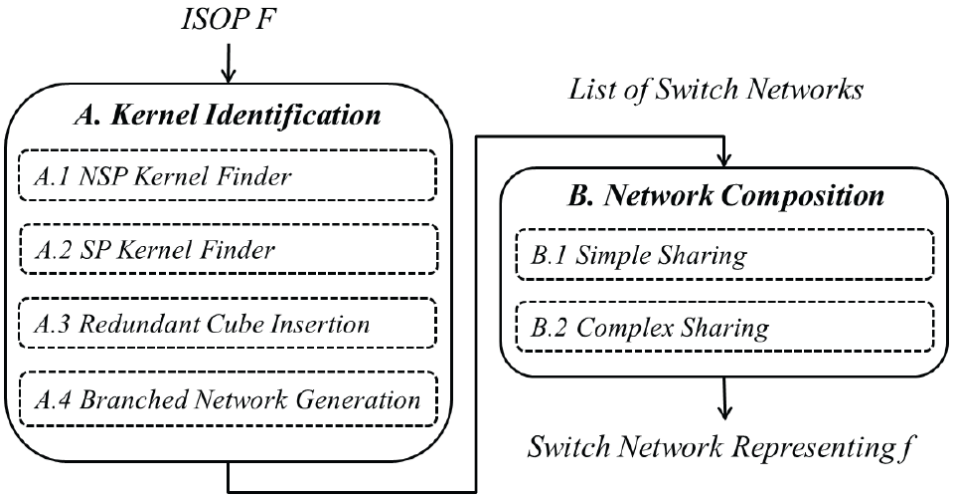


Figura 13 – Fluxograma do Kernel Finder.

Fonte: POSSANI, *et al.*, 2015, p. 2.

- *Kernel Identification*: o primeiro estágio é responsável por criar as sub-redes que comporão o arranjo de chaves final. São quatro principais partes, descritas nos itens abaixo:
  a) *NSP Kernel Finder*: a primeira etapa utiliza o *kernel* (grafo) ilustrado na Figura 12 (a) para a construção da rede de transistores. O método computa, para quatro cubos selecionados, quais os vértices que os mesmos ocuparão no *template* definido, considerando as relações entre literais em comum de cada cubo, conforme discutido anteriormente. Os termos remanescentes da função de entrada (ou seja, que não foram remanejados para os vértices do *kernel* NSP) são passados para a próxima etapa (item b);
  b) *SP Kernel Finder*: a segunda etapa atua de forma similar a anterior: são selecionados quatro cubos dentre os disponíveis, dispondo-os no *template* série-paralelo ilustrado na Figura 12 (b). Novamente, os cubos que não foram inseridos no *kernel* são repassados para a próxima etapa (item c);
  c) *Redundant Cube Insertion*: de forma análoga a descrita no estágio *NSP Kernel Finder*, a terceira etapa utiliza-se do *template* NSP, ilustrado na Figura 12 (a), com a diferença de que três cubos são selecionados, ao invés de quatro. No vértice que não corresponde a algum dos cubos insere-se um cubo falso que tem por objetivo casar a rede gerada com o *template* definido ao passo de não alterar a funcionalidade Booleana do arranjo;
  d) *Branched Network Generation*: a quarta etapa é responsável por tratar os cubos que não foram inseridos nas etapas anteriores, caso existam. Isso é

realizado conectando-os em paralelo com a solução desenvolvida pelos passos pré-computados considerando o fluxo da metodologia.

- *Network Composition*: o segundo estágio é responsável por compor a rede a partir das sub-redes geradas no módulo anterior (*Kernel Identification*). Partindo de uma associação em paralelo dos *kernels* e redes computadas anteriormente, executam-se os métodos *Simple Sharing* e *Complex Sharing*, descritos a seguir:
  a) *Simple Sharing*: a primeira etapa é responsável por percorrer a rede em busca de transistores equivalentes (controlados pelo mesmo literal). Ao encontrar chaves equivalentes, o arranjo de transistores é reestruturado de forma a fazer com que as instâncias equivalentes tenham um nó em comum, o que, por sua vez, torna possível o compartilhamento da chave em questão tal como efetuado em (KAGARIS, *et al.*, 2007) (seção 2.2.2);
  b) *Complex Sharing*: a última etapa recebe como entrada uma rede de transistores já pré-processada pela etapa anterior onde realizou-se o compartilhamento de transistores a partir do rearranjo da rede. No entanto, em determinadas situações, a depender da disposição dos transistores, pode não ser possível encontrar um nó comum de forma direta. Nesse caso, a etapa de *Complex Sharing* age de forma a buscar possíveis compartilhamentos de chaves que não foram encontrados pelo estágio anterior. Isso é feito através de compactação dos ramos série-paralelo da célula, seguido pela execução do *Simple Sharing*. Por fim, executa-se um método de descompactação, onde a solução final é obtida.

A Figura 14 mostra uma instância de execução do método descrito para a função *f* definida na Equação (11).

$$f = (\overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d}) + (\overline{a} \cdot b \cdot d) + (\overline{a} \cdot b \cdot c) + (a \cdot \overline{b} \cdot d) + (a \cdot \overline{b} \cdot c) + (a \cdot b \cdot \overline{c} \cdot \overline{d}) + (b \cdot c \cdot d) \quad (11)$$

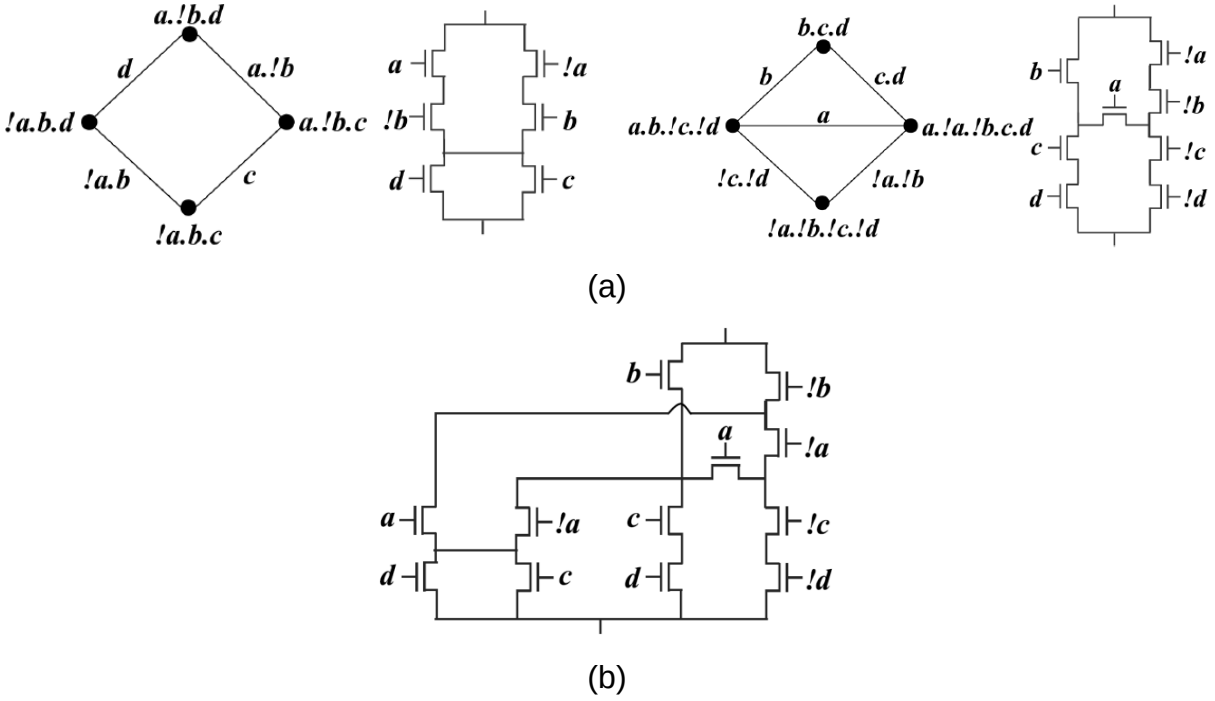


Figura 14 – Execução do Kernel Finder para a função *f* descrita pela Equação (11). (a) Resultado após o método *Kernel Identification*. (c) Solução após execução do método *Network Composition*.

Fonte: POSSANI, *et al.*, 2015, p. 7.

Em (a) tem-se as saídas dos estágios *NSP Kernel Finder* (esquerda) e *SP Kernel Finder* (direita). Finalmente, em (b) está ilustrada a rede após a execução das metodologias *Simple Sharing* e *Complex Sharing*. Nota-se que os *kernels* SP e NSP, quando somados, possuem um total de 14 transistores, enquanto a solução pós *Simple Sharing* e *Complex Sharing* é composta por 12 chaves. Ou seja, nesse caso houve uma otimização (através de compartilhamento de transistores equivalentes) de 2 chaves. ■

2.3 Conclusões

Nesse capítulo fez-se uma revisão dos principais métodos para a otimização de redes de transistores, apresentando-os separadamente em duas classes distintas: os métodos de minimização Booleana, algoritmos clássicos que apoiam-se em propriedades algébricas das funções para geração de uma forma fatorada com o menor número de literais possível, e as metodologias baseadas em grafos, meios de minimização lógica a partir de manipulações em estrutura de grafos as quais correspondem a redes de chaves.

Considerando esses dois diferentes paradigmas, destacou-se alguns dos métodos de maior importância na literatura. Em especial, foram abordadas as

metodologias de Composição Funcional (seção 2.1.2, sendo esse um método algébrico Booleano) e o Kernel Finder (2.2.3, metodologia baseada em grafo), algoritmos utilizados diretamente nesse trabalho no estágio de geração lógica. Como pode ser visto a seguir nesse texto, far-se-á um comparativo direto das redes de transistores geradas por esses diferentes paradigmas, onde as características topológicas das mesmas consistem em divergências bem delineadas de como a célula física será constituída (e os meios para a concepção da mesma).

3 PROJETO AUTOMÁTICO DE CÉLULAS

O projeto do leiaute de células é uma fundamental etapa na metodologia *standard cell* que atua no contexto do desenvolvimento de ASICs. Inicialmente projetados sob o paradigma *full-custom*, onde cada componente presente no *design* era dimensionado, posicionado e roteado a mão, o desenvolvimento de CIs tornou-se custoso ao passo que o número de transistores e as regras de projeto aumentaram. De fato, ao observarmos o gráfico da Figura 15, podemos notar que, enquanto a complexidade do *chip* cresce, em média, 58% por ano, a produtividade do projetista cresce apenas 21%. Isso evidencia a necessidade cada vez maior do emprego de metodologias automáticas em todas as frentes do projeto VLSI que garantam soluções em *time-to-market* e com boa qualidade. Nesse contexto, abordaremos nessa seção algumas metodologias que atuam nos principais estágios da geração de células, bem como apresentaremos ferramentas que incorporam essas metodologias e garantem soluções comparáveis as providas manualmente.

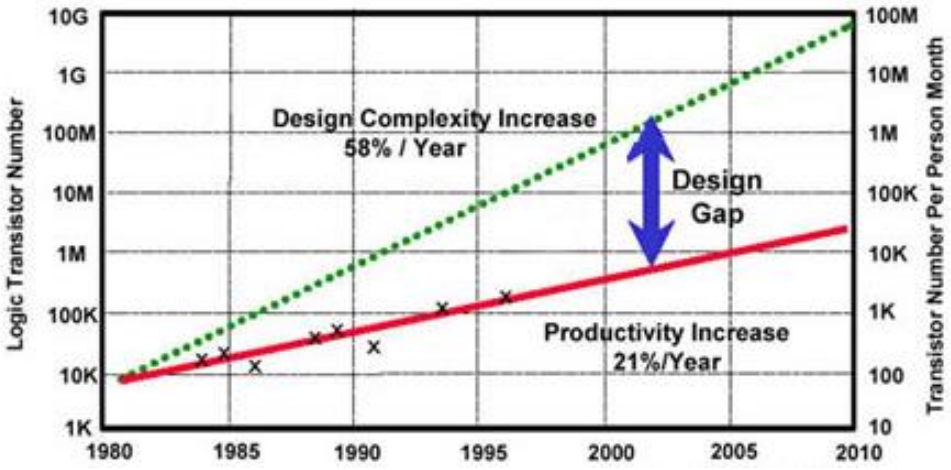
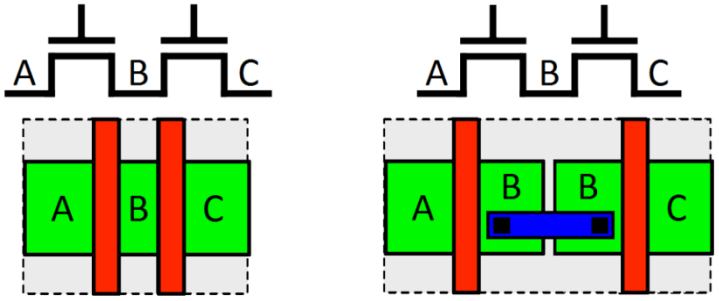


Figura 15 – Previsão do *gap* de produtividade no projeto VLSI.

Fonte: VICKER, 1997.

3.1 Posicionamento de Transistores

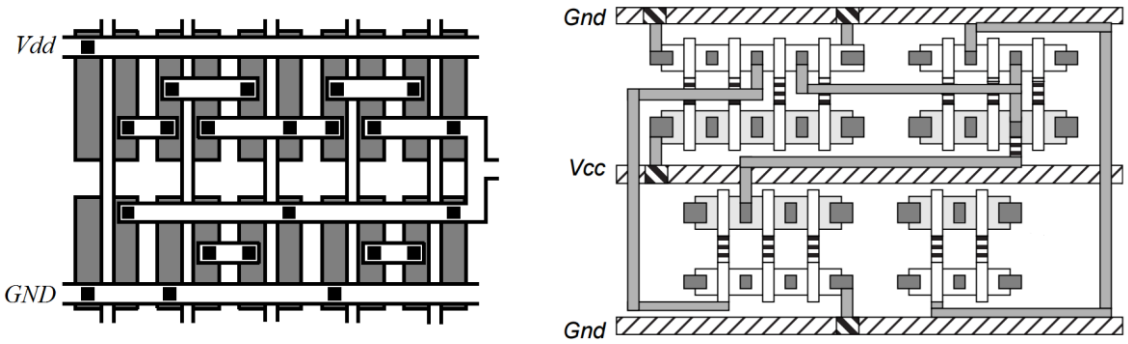
O posicionamento de transistores na célula é de grande importância para o *design* de circuitos integrados desde sua proposição originalmente realizada em (UEHARA, *et al.*, 1981). Nesse trabalho pioneiro, observou-se que é possível compartilhar áreas de difusão de transistores que sejam adjacentes logicamente, ou seja, que compartilhem conexões em nível de circuito. Essa técnica resulta, em geral, em leiautes mais compactos – conforme a Figura 16 exemplifica, onde fica clara a diferença entre as abordagens (a) (com compartilhamento da área ativa *B*) e (b) (sem o compartilhamento da área ativa *B* entre os transistores, realizando a conexão via metal 1) –, o que acarreta em um aumento da densidade de chaves no *chip* e contribui com a diminuição de capacitâncias de acoplamento, levando a uma diminuição média do atraso da célula.



(a) (b)

Figura 16 – Compartilhamento de difusão proposto em (UEHARA, *et al.*,1981). (a) Transistores NMOS em série compartilhando suas áreas ativas *B*. (b) Transistores NMOS em série não compartilhando suas áreas ativas *B* (conexão em metal 1).

Uma outra importantíssima contribuição de Uehara *et al*. foi a definição de um dos estilos de leiaute mais amplamente empregados na indústria de microeletrônica atual, conhecido como *linear-matrix* (1D). Esse estilo de desenho caracteriza-se por conter duas bandas de difusão, uma do tipo P, formada por transistores PMOS, e outra do tipo N, contendo transistores NMOS. Os polisilícios de *gate* de cada transistor são dispostos perpendicularmente à orientação da banda de difusão e podem ser compartilhados entre áreas P e N da célula, conforme mencionado no parágrafo anterior. A Figura 17 apresenta um leiaute com o estilo *linear-matrix* (a) em contraste com o estilo 2D (b), também utilizado pela indústria.



(a) (b)

Figura 17 – Estilos de leiaute. (a) *Linear-matrix*. (b) 2D.
Fonte: Adaptado de GUPTA, *et al.*, 2000.

Com o decorrer dos anos, várias outras abordagens surgiram para solucionar o problema de posicionamento de transistores na célula. Algumas delas estão presentes em ferramentas amplamente utilizadas pela indústria, tais como as ferramentas CELLERITY, utilizada na Motorola para síntese de células no estilo 2D (estilo alternativo ao *linear-matrix*, conforme ilustrado na Figura 17 (b) acima) (GURUSWAMY, *et al.*, 1997), e a XPRESS EDA, utilizada na Intel (GUPTA, *et al.*, 1996a). Para facilitar o entendimento dos algoritmos de posicionamento, nesse texto dividiremos os mesmos em duas categorias principais referentes a sua restrição topológica: algoritmos para posicionamento de redes de transistores duais e para redes não-duais.

3.1.1 Posicionamento de Redes de Transistores Duais

Redes duais (ou topologicamente complementares) são redes planares e que possuem o mesmo número de chaves em ambos os planos. Algoritmos de posicionamento para essa topologia são os mais comuns na literatura, alguns deles sendo abordados nos parágrafos e subseções seguintes.

Após a definição do posicionador clássico de (UEHARA, *et al.*, 1981) (detalhado em 3.1.1.1), (MAZIASZ, *et al.*, 1991) propõe o primeiro método exato de minimização em duas dimensões (altura e largura) de células no estilo *linear-matrix*. Em (GUPTA, *et al.*, 1996b) é proposto um método de minimização de largura de células no estilo 2D utilizando *Integer Linear Programming* (ILP). Ainda nesse trabalho, Gupta *et al*. propõe uma técnica de agrupamento de transistores em série para posicionar células com um elevado número de componentes. Em (GUPTA, *et al.*, 2000) é apresentada a ferramenta CLIP/HCLIP, que também faz uso de técnicas de

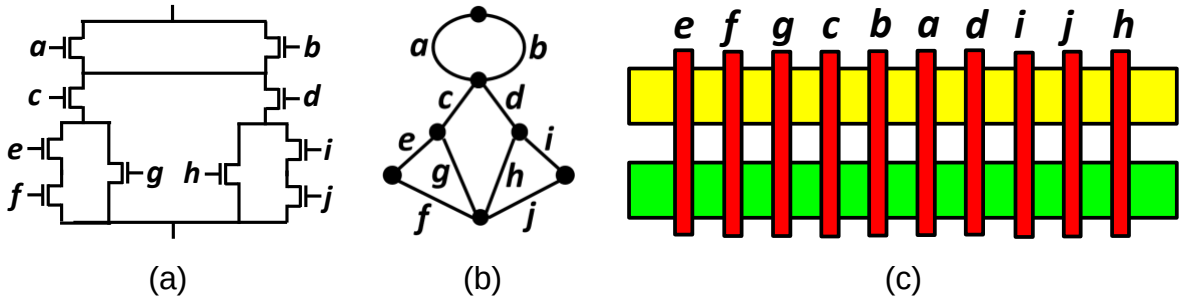
posicionamento via ILP para otimização de largura e altura em células 2D. Ademais, em (GUPTA, *et al.*, 1998) é proposta a ferramenta FCLIP que incorpora a técnica de *folding* ao posicionador. Finalmente, em (IIZUKA, *et al.*, 2004) é proposta a técnica considerada estado da arte em posicionamento de redes duais (seção 3.1.1.2).

3.1.1.1 Posicionamento via Caminhos de Euler (UEHARA, *et al.*, 1981)

Conforme descrito anteriormente, Uehara *et al*. propõe a primeira metodologia de posicionamento de células CMOS, bem como introduz o conceito de compartilhamento de difusão entre transistores (mostrado na Figura 16) e o estilo de desenho de células *linear-matrix* (ilustrado na Figura 17 (a) anteriormente).

O método de posicionamento proposto recebe uma descrição estrutural da rede (*netlist*) e cria um grafo corresponde a essa descrição, onde os vértices do mesmo correspondem aos nós da rede de chaves e as arestas representam os transistores no arranjo. Com essa estrutura de grafo construída, Uehara *et al.* propõe a verificação de compartilhamento de difusão na célula através da busca por caminhos de Euler no grafo. Encontrar um caminho de Euler significa ser possível construir um leiaute onde todos os transistores compartilham difusões, bastando utilizar o mesmo ordenamento no momento de posicionar os *gates* na célula.

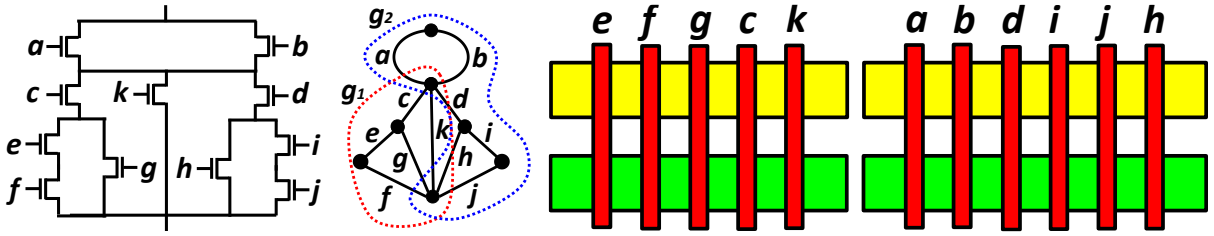
Para exemplificar o método de posicionamento, tomemos como exemplo a rede ilustrada na Figura 18 (a). Como podemos observar, existe um caminho de Euler C_1 = *(e, f, g, c, b, a, d, i, j, h)* no seu grafo corresponde (b). A partir disso, posicionou-se os *gates* na célula (c) mantendo-se o ordenamento obtido pelo caminho ordenado C_1. Conforme (c) deixa claro, nesse caso a célula não possuirá *gaps* na área de difusão (já previsto, pois o grafo possui um caminho de Euler).



(a) (b) (c)

Figura 18 – Posicionamento por caminhos de Euler sem *gaps*. (a) Rede lógica. (b) Grafo correspondente. (c) Célula não contendo *gaps*.

No caso onde não é possível encontrar caminhos eulerianos no grafo, o autor propõe que o mesmo seja dividido em dois ou mais subgrafos, gerando um *gap* na linha de difusão entre os transistores separados no *netlist*. Segundo Uehara *et al.*, uma boa heurística para divisão do grafo é escolher o ponto onde os dois grafos originados possuam o mesmo tamanho. Finalmente, para cada subgrafo originado, buscam-se novamente caminhos de Euler, ordenando os transistores de cada grafo para posterior posicionamento no leiaute conforme a solução encontrada, caso ela exista. Para a situação onde os subgrafos não admitam caminhos de Euler, dividem-se os mesmos recursivamente até que se obtenha uma rede euleriana. Vale ressaltar que, no pior caso, uma rede contendo apenas um transistor é uma rede euleriana por definição. A Figura 19 contém um exemplo desse processo, onde em (b) tem-se um grafo *G* – representando a rede lógica ilustrada em (a) – que não admite caminho de Euler dividido em dois subgrafos g_1 (circulado em vermelho) e g_2 (em azul), ambos contendo caminhos de Euler. Por fim, a Figura 19 (c) ilustra o leiaute posicionado, onde nota-se um *gap* na linha de difusão causado pela divisão do arranjo original.



(a) (b) (c)

Figura 19 – Posicionamento por caminhos de Euler com *gaps*. (a) Rede lógica. (b) Grafo e subgrafos correspondentes. (c) Célula contendo um *gap* na área ativa.

3.1.1.2 Posicionamento via Satisfatibilidade Booleana (IIZUKA, *et al.*, 2004)

Iizuka *et al.* propõe um método de posicionamento de transistores baseado em satisfatibilidade Booleana (SAT). O método apoia-se na qualidade dos algoritmos e ferramentas para resolução de problemas SAT, os quais vêm sendo propostos, amplamente discutidos e otimizados desde a formulação original do problema datada de meados da década de 1930.

(IIZUKA, *et al.*, 2004) transforma o problema de posicionamento em um problema SAT através de criações de cláusulas lógicas no padrão CNF (*Conjunctive Normal Form*), sendo essas resolvidas através do *SAT-solver* Chaff (MOSKEWICZ, *et al.*, 2001). O método proposto atua sobre uma grade representativa da célula onde os

transistores que estão em posições adjacentes partilham a difusão entre si, ao passo que *gaps* são representados por uma coluna vazia entre dois elementos. A Figura 20 ilustra esses dois casos, bem como dá uma visão geral da estrutura utilizada para representar o leiaute.

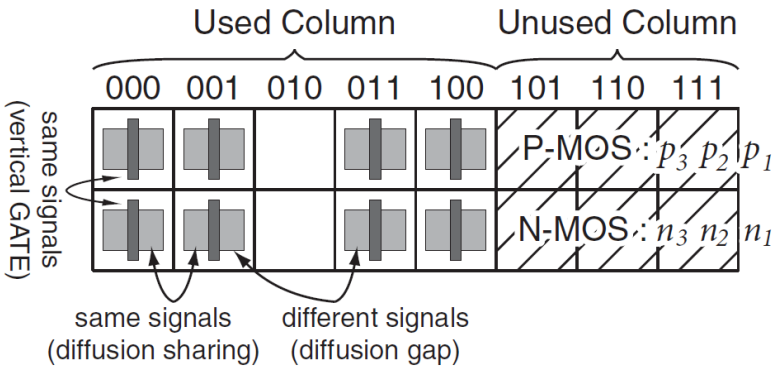


Figura 20 – Estrutura representativa da célula proposta em Iizuka *et al*.
Fonte: IIZUKA, *et al.*, 2004, p. 2B-3.

A metodologia propõe que cada transistor possua *P* variáveis para indicar o seu posicionamento, tal que $P = \lceil log_2 W \rceil + 1$, em que *W* representa o número de colunas da grade. Uma variável extra é adicionada para indicar a orientação (*flip*) do transistor (ou seja, indica qual terminal – *drain* ou *source* – está à esquerda do *gate*). A partir dessa definição de variáveis, Iizuka *et al.* descreve como as cláusulas para posicionamento são construídas. Abaixo estão destacadas as cláusulas propostas seguidos de sua equação geradora:

- Superposição de transistores: não deve haver mais de um transistor ocupando a mesma coluna;
  - Fórmula da CNF restritiva: $n_{i1} \oplus n_{j1} \vee n_{i2} \oplus n_{j2} \vee ... \vee n_{iP} \oplus n_{jP} = 1$. Vale ressaltar que há uma equação geradora de CNF restritiva para o plano PMOS análoga à apresentada nesse item.
- Colunas inutilizadas: se $W < 2^P$, então haverão colunas não utilizadas, como ilustra a Figura 20. Nessas colunas, nenhum transistor poderá ser posicionado;
  - Fórmula da CNF restritiva: $n_{i1} \oplus u_1 \vee n_{i2} \oplus u_2 \vee ... \vee n_{iP} \oplus u_P = 1$. Tal como anteriormente, há uma equação geradora de CNF restritiva para o plano PMOS análoga à apresentada.

- Restrições verticais de *gate*: a terceira cláusula define que os transistores que são posicionados na mesma coluna devem possuir os mesmos sinais de *gate* (ou seja, deve haver casamento – *matching* – de *gates* entre os planos);
  - Fórmula da CNF: $\bigvee_{j \in G_P^i} \overline{\left(n_{i1} \oplus p_{j1} \vee n_{i2} \oplus p_{j2} \vee \ldots \vee n_{iP} \oplus p_{jP} \right)} = 1$.
- Restrições de vizinhança de transistores: a última cláusula de posicionamento define que transistores que estão em colunas vizinhas na célula devem estar conectados no *netlist* de entrada (ou seja, compartilham a área de difusão). A Figura 20 ilustra essa restrição.
  - Fórmula da CNF restritiva: $GAP_n(i,j) \wedge (\bigvee_{k=0}^{W-2} C_n(i,k) \wedge C_n(j,k+1)) = 0$, tal que $C_n(i,k) = 1$ se existe um transistor *i* na coluna *k* e $GAP_n(i,j) = 1$ se o transistor *i* não pode compartilhar a difusão com um transistor *j* posicionado a sua direita. Há uma equação geradora de CNF restritiva para o plano PMOS análoga à apresentada, tal como anteriormente.

Caso não haja assinalamento válido para as variáveis de posicionamento, o autor propõe que se refaça o processo incrementando-se o número de colunas da grade ($W = W + 1$). Dessa forma, realizando buscas de soluções partindo de um *W* mínimo, o método garante que a célula posicionada apresentará largura mínima.

3.1.2 Posicionamento de Redes de Transistores Não-duais

Redes de transistores não-duais podem ser encontradas em diversos dispositivos digitais desde bibliotecas de *buffers tri-state*, *flip-flops*, circuitos assíncronos, redes que sofreram *folding* em seus transistores, dentre outros. Devido a isso, o posicionamento sem restrição de dualidade constitui um importante estágio no desenvolvimento de circuitos digitais topologicamente versáteis. Especialmente no contexto desse trabalho, onde busca-se flexibilidade para lidar com diferentes configurações construtivas da célula (incluindo não-dualidade e não-planaridade, características presentes em boa parte das redes geradas via Kernel Finder, como apresentado na Tabela 2), tais metodologias são fundamentais para a concepção de soluções automáticas de boa qualidade.

(BAR-YEHUDA, *et al.*, 1988) é um dos primeiros trabalhos relevantes a apresentar um método de posicionamento para redes não-duais no estilo *linear-matrix* (estilo demonstrado na Figura 17 (a) anteriormente). O método atua de maneira similar a (UEHARA, *et al.*, 1981) (seção 3.1.1.1), transformando as redes *pull-up* e *pull-down*

em grafos distintos. Após, executa-se uma busca em profundidade nesses grafos, computando os possíveis posicionamentos e escolhendo aquele que conduzir a uma menor quantidade de *gaps* como solução. Em (HSIEH, *et al.*, 1990) é proposta a ferramenta LiB, que descreve um método de divisão da rede lógica em *clusters* para posicionamento detalhado dos transistores, abordagem que permite tratar circuitos com um número elevado de chaves. (GUPTA, *et al.*, 1996a) apresenta a ferramenta XPRESS, onde o posicionamento também se dá a partir do ordenamento de *clusters* visando minimizar o número de quebras na linha de difusão e o número de trilhas de roteamento horizontais. (GURUSWAMY, *et al.*, 1997) introduz a ferramenta CELLERITY, usada na síntese de células sob o estilo de leiaute 2D. CELLERITY utiliza *Simulated Annealing* para posicionar os transistores na célula (método que será abordado na seção 3.1.2.1) tendo como função custo a minimização do número de quebras, o comprimento das conexões, a densidade do canal de roteamento e o desalinhamento entre os transistores nos planos lógicos. (SERDAR, *et al.*, 2001) apresenta um método capaz de posicionar transistores sem um estilo previamente definido e em diferentes orientações. O método também utiliza *Simulated Annealing* aliado a transformações em uma estrutura *O-tree* e tem como principais características a capacidade de lidar com largura fixa da célula e a possibilidade de gerar leiautes não retangulares. (RIEPE, *et al.*, 2003) propõe um gerador de células 2D (estilo ilustrado na Figura 17 (b) previamente) denominado TEMPO. A metodologia utilizada baseia-se no agrupamento de transistores em *clusters* para aplicação posterior do método de *Simulated Annealing* a fim de ordenar os grupos formados. (ZIESEMER, *et al.*, 2015) apresenta a ferramenta de síntese automática de redes de transistores ASTRAN. O posicionamento utilizado na ferramenta (apresentada na seção 3.3.2.3) baseia-se em uma abordagem de *Threshold Accepting* (discutida na seção 3.1.2.1). Por fim, (IIZUKA, *et al.*, 2005) apresenta o algoritmo considerado estado da arte para posicionamento de redes de chaves sem restrições de dualidade. Iizuka *et al.* propõe uma extensão ao método descrito em (IIZUKA, *et al.*, 2004) (seção 3.1.1.2) para topologias não regulares, também utilizando a resolução de cláusulas SAT como principal motor do posicionador (detalhado em 3.1.2.2).

3.1.2.1 Posicionamento via Métodos Heurísticos (KIRKPATRICK, *et al.*, 1983) (DUECK, *et al.*, 1990)

Os métodos heurísticos, em especial as meta-heurísticas *Simulated Annealing* e *Threshold Accepting*, estão presentes em diversas etapas do fluxo de síntese de célula e circuitos VLSI. Especialmente no problema de posicionamento de transistores no leiaute, tais abordagens permitem um projeto adaptável (aspecto fundamental no projeto *standard cell*), flexível (sem restrições topológicas) e com qualidade comparável a métodos exatos, normalmente mais complexos computacionalmente. Nessa seção apresentaremos uma visão geral desses dois paradigmas meta-heurísticos, enquanto que a seção 3.3.2.3 contém a aplicação do *Threshold Accepting* em uma ferramenta de síntese de células que será diretamente aplicada no contexto dessa pesquisa.

O *Simulated Annealing* (SA) é uma meta-heurística robusta, compacta e capaz de prover soluções para problemas de otimização envolvendo uma ou mais funções objetivo ao passo de uma complexidade computacional pequena. Proposto originalmente em (KIRKPATRICK, *et al.*, 1983), o SA pode ser entendido através de uma analogia relativa à como metais esfriam e aquecem-se seguindo as leis da termodinâmica, buscando, através desse processo, um estado de maior ou menor energia. O algoritmo parte de uma solução inicial e, a partir de iterações controladas por uma função custo e pela temperatura associada, converge para mínimos locais ou globais da solução. O pseudo-código disponível no Algoritmo 1 detalha o SA.

Algoritmo 1 Pseudo-código do algoritmo *Simulated Annealing*

```
1:  SA ( I,T )
2:      s_atual ← solucaoInicial ( )
3:      t ← T
4:      for i = 1:I do
5:          s_nova ← N ( s_atual )
6:          t ← t - 1
7:          if ( fCusto ( s_nova ) < fCusto ( s_atual ) ) then
8:              s_atual ← s_nova
9:              if ( fCusto ( s_nova ) < fCusto ( s ) ) then
10:                 s ← s_nova
11:             end if
12:         else if ( exp (( fCusto ( s_atual ) - fCusto ( s_nova ) / t ) > rand( ) )
13:             s_atual ← s_nova
14:         end if
15:     end for
16: return ( s )
```

O algoritmo recebe como entrada o número máximo de iterações *I* e a temperatura inicial (máxima) *T*. Na linha 2 é computada uma solução válida para servir como ponto de partida do método. Cabe ressaltar que tal solução pode ser obtida tanto de forma aleatória ou a partir de uma resposta qualquer pré-computada. O laço iterativo compreendido entre as linhas 4 e 15 atua de forma a calcular soluções com custo menor (linhas 7 a 11) do que a calculada em iterações prévias, podendo gerar mínimos locais através desse processo. Por fim, a linha 12 é a responsável por aceitar (menos frequentemente ao passo que a temperatura *t* decresce) soluções com um custo maior que a atual. Isso faz com que o algoritmo busque por mínimos globais mesmo que fique preso a um mínimo local.

A meta-heurística *Threshold Accepting* (TA), proposta originalmente em (DUECK, *et al.*, 1990) e demonstrada no Algoritmo 2 é bastante similar ao SA, descrita anteriormente. A principal diferença está na função de aceitação: enquanto no SA tal função é puramente probabilística (linha 12 do Algoritmo 1), o TA implementa a mesma função de forma determinística (linha 7 do Algoritmo 2). Na prática, essa diferença resulta em melhores soluções em favor da metodologia de TA, além de facilitar a implementação do algoritmo, conforme Dueck *et al.* discute.

Algoritmo 2 Pseudo-código do algoritmo *Threshold Accepting*

```
1:  TA ( I,Th )
2:      s_atual ← solucaoInicial ( )
3:      t ← Th
4:      for i = 1:I do
5:          s_nova ← N ( s_atual )
6:          Δf ← fCusto ( s_nova ) - fCusto ( s_atual )
7:          if ( Δf < t ) then
8:              s_atual ← s_nova
9:              if ( fCusto ( s_nova ) < fCusto ( s ) ) then
10:                 s ← s_nova
11:             end if
12:         end if
13:     t ← t - 1
14:     end for
15: return ( s )
```

Partindo de uma solução inicial e com um limite de iterações *I* e *threshold Th*, o TA realiza o cálculo de *Δf* subtraindo os custos da nova solução obtida na linha 5 com a solução atual. No caso de *Δf* estar abaixo do limiar calculado, então a nova solução é adotada. Caso contrário, a solução é descartada. Vale ressaltar que, da mesma forma que ocorre com a temperatura no SA, o limiar *Th* sofre um decréscimo

a cada iteração (linha 3 e 13), o que, na prática, faz com que a frequência das trocas de soluções caia conforme o número de iterações cresce.

Finalmente, vale ressaltar que a metodologia TA será diretamente utilizada nesse trabalho, visto que o posicionador presente na ferramenta ASTRAN é baseado nessa meta-heurística. A seção 3.3.2 apresentará com detalhes a ferramenta e o posicionador proposto.

3.1.2.2 Posicionamento via Satisfatibilidade Booleana (IIZUKA, *et al.*, 2005)

Nesse artigo o autor apresenta uma extensão à metodologia proposta em (IIZUKA, *et al.*, 2004), introduzida na seção 3.1.1.2 dessa dissertação. Por conta disso, algumas das formulações propostas no artigo original são mantidas, conforme pode ser visto a seguir. No entanto, além de ser possível tratar células não-duais, uma extensão a metodologia anterior, Iizuka *et al*. também propõe um meio alternativo para posicionamento eficiente de células no estilo 2D.

A Figura 21 ilustra a estrutura utilizada na metodologia proposta – análoga à usada no posicionador de redes duais apresentado anteriormente (Figura 20) –, bem como o leiaute produzido pelo método, onde pode-se notar o estilo 2D empregado e a não-dualidade da solução. Por conta dos bons resultados apresentados relativos à qualidade do leiaute (área) e tempo de execução, somado a versatilidade da metodologia que pode ser utilizada em células *linear-matrix* ou *multirow* (2D, por exemplo) e sem restrição de dualidade, (IIZUKA, *et al*., 2005) é considerado o método exato estado da arte para posicionamento de células com topologias irregulares.

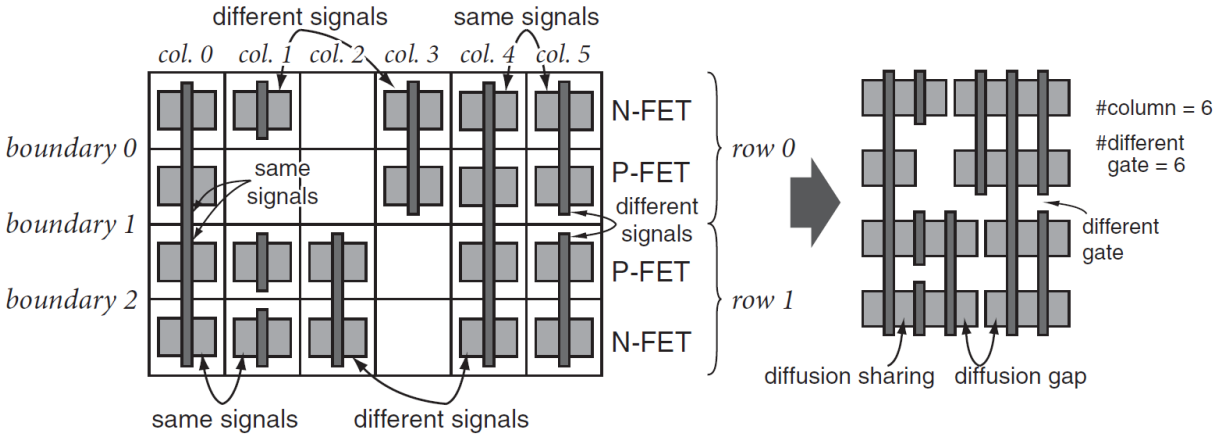


Figura 21 – Estrutura gradeada utilizada para o posicionamento e leiaute produzido a partir da instância apresentada.

Fonte: IIZUKA, *et al.*, 2005, p. 2.

A proposta de algoritmo também é baseada na solução do problema de satisfatibilidade Booleana (SAT), tal como no método anterior do mesmo autor. Primeiramente, $W + 1$ variáveis de posicionamento são criadas para cada X transistores tipo N e Y transistores tipo P, tal que C e R representam o número de colunas e linhas do leiaute, respectivamente. Posteriormente, são criadas as seguintes cláusulas nos formatos *Conjunctive Normal Form* (CNF) e *Pseudo-Boolean Form* (PBF) (inequações de variáveis Booleanas):

- Superposição de transistores: dois transistores não podem ocupar a mesma coluna;
  - Fórmula da PBF restritiva: $\sum_{i=0}^{X-1} C_n(i,k) \leq 1 \mid 0 \leq k \leq W$, tal que $C_n = 1$ se existe um transistor i do tipo N posicionado na coluna k. Vale ressaltar que há uma equação geradora de PBF restritiva para o plano PMOS análoga à apresentada nesse item.
- Instanciação de transistores: cada transistor deve ser instanciado separadamente. Essa cláusula é a responsável por permitir que diferentes tamanhos de planos sejam instanciados para uma mesma célula;
  - Fórmula da PBF restritiva: $\sum_{i=0}^{W-1} C_n(i,k) = 1 \mid 0 \leq i \leq X$. Há uma PBF análoga para o plano PMOS.
- Restrições de vizinhança horizontal: define que transistores que não compartilham um sinal de difusão entre si não podem ser posicionados em colunas adjacentes;
  - Fórmula da CNF restritiva: $GAP_n(i,j) \wedge (\vee_{k=0}^{W-2} C_n(i,k) \wedge C_n(j,k+1)) = 0 \mid i \neq j,\ 0 \leq i \leq X,\ 0 \leq j \leq X$. Há uma PBF análoga para o plano PMOS.
- Função Objetivo: define-se uma função D que representa o *mismatching* vertical entre os *gates* da célula. Busca-se minimizar essa função a fim de facilitar o roteamento *intra-cell através da* seguinte fórmula: $minimize(\sum_{k=0}^{W-1} D(k))$, tal que:

$$D = \bigvee_{i,j} \left(C_n(i,k) \wedge C_n(i,k) \right) \mid 0 \leq k \leq W \tag{12}$$

Analogamente ao método apresentado na seção 3.1.1.2, há a garantia de se obter células com largura mínima, visto que, caso não seja possível encontrar um

assinalamento válido para as variáveis de posicionamento utilizadas, incrementa-se iterativamente o número de colunas da grade e repete-se o processo.

3.2 Roteamento *Intra-cell*

O roteamento *intra-cell* é uma etapa fundamental no projeto de portas lógicas complexas. Seguindo o fluxo do projeto de células, tal estágio está localizado logo após o posicionamento dos transistores no leiaute, discutido na seção 3.1 dessa dissertação. De forma geral, o roteamento *intra-cell* é responsável por interconectar os transistores e as interfaces (pinos de entrada, saída e alimentação) da célula de forma a respeitar o *netlist*. Nessa seção abordaremos as principais metodologias que servem como motores para boa parte das ferramentas de roteamento disponíveis na literatura.

3.2.1 *Maze Router* (LEE, 1961)

Em (LEE, 1961) está proposto o primeiro algoritmo de roteamento para circuitos eletrônicos, metodologia presente no CELLERITY para roteamento de células 2D e amplamente utilizada em diversas aplicações ainda hoje. Nesse trabalho, Lee propõe um *maze router*, um método de roteamento genérico que atua sobre uma grade capaz de encontrar o caminho mais curto entre quaisquer dois pontos.

De maneira geral, o algoritmo funciona de forma a aumentar o espaço de busca por soluções de maneira iterativa, tal como ilustra a Figura 22, análogo à propagação de uma onda por um espaço de possíveis soluções. Primeiramente cria-se uma grade e faz-se a assinalação dos pontos de origem *S* e destino *T*. Então, a partir de *S*, expande-se gradualmente a busca, marcando nas células vizinhas à origem a distância da mesma para *S*. No momento em que *T* é encontrado, o algoritmo traça a conexão a partir da contagem decrescente das marcações realizadas partindo de *T* até *S*.

Como o exemplo da Figura 22 deixa claro, o *maze router* é capaz de contornar obstáculos na busca por soluções. Ademais, o algoritmo garante que o caminho mais curto (de menor *wirelength*) será encontrado (vale ressaltar que, conforme a Figura 22 demonstra, é possível encontrar múltiplas soluções de mesmo *wirelength* mínimo. Nesse caso, para o contexto de projeto eletrônico, normalmente opta-se pela solução com menor quantidade de mudanças de direção para evitar o efeito das pontas). Por

outro lado, são desvantagens do *maze router* de Lee a sua alta complexidade computacional – *O(MN)*, tal que *M* é o número de colunas e *N* o número de linhas.

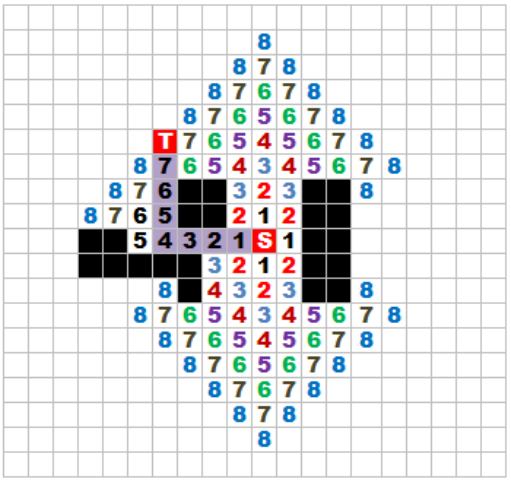


Figura 22 – Instância de execução do *maze router* em uma grade contendo obstáculos.

3.2.2 Algoritmo de Dijkstra (DIJKSTRA, 1959)

O algoritmo de Dijkstra para busca de menor caminho é um algoritmo de roteamento em um grafo aplicado em diversos problemas práticos de computação, tais como em sistemas de posicionamento global, jogos e redes de computadores, por exemplo. No contexto desse trabalho, esse algoritmo é a base do roteador utilizado no ASTRAN (ZIESEMER, *et al.*, 2015), ferramenta de síntese de células detalhada na seção 3.3.2.

O pseudo-código do algoritmo de Dijkstra está ilustrado em Algoritmo 3. De maneira simplificada, o método inicia marcando no grafo (ou na grade) os pontos de partida e chegada *S* e *T*, respectivamente, para, após, assinalar os nodos com a informação de distância característica do algoritmo: para o nodo *S*, define-se a distância zero, enquanto que para o restante é setado o valor infinito. Após, partindo de *S*, verifica-se a vizinhança ainda não visitada em relação ao custo da ida para cada nodo: soma-se o valor contido no nodo com o valor da aresta que conecta o nodo atual com o novo nodo. Por fim, seleciona-se o nodo com menor custo, realizando esse processo até *T*, encerrando o método e retornando o caminho encontrado *c*.

Algoritmo 3 Busca de Menor Caminho pelo Algoritmo de Dijkstra

```
1:  Dijkstra ( S,T,G )
2:      setaValoresNodos ( (S,0),(G,∞) )
3:      n_atual ← S
4:      c ← S
5:      while ( n_atual ≠ T )
6:          n_atual ← mínimo ( computaPesosVizinhos ( n_atual,G ) )
7:          c ← n_atual
8:      end while
9:  return ( c )
```

Como pode-se notar, o algoritmo de Dijkstra possui uma vantagem em relação aos roteadores *maze* ao permitir trabalhar com grades com pesos. No contexto de circuitos integrados, células valoradas podem significar áreas congestionadas (grande densidade de metal), por exemplo. Ademais, não se faz necessária a expansão do espaço de busca por caminhos com maior custo, o que diminui a complexidade assintótica do método.

3.2.3 Algoritmo A* (HART, *et al.*, 1968)

O algoritmo A* é um dos métodos mais populares e amplamente utilizados para solucionar o problema de encontrar o menor caminho entre dois pontos quaisquer, comum a diversas aplicações. No caso do roteamento no projeto de células, tal algoritmo é a base do roteador proposto no cellTK (KARMAZIN, *et al.*, 2013), ferramenta de síntese de leiautes para circuitos assíncronos detalhada na seção 3.3.1 desse texto.

Proposto originalmente em (HART, *et al.*, 1968) como uma otimização ao algoritmo de Dijkstra (DIJKSTRA, 1959) (seção 3.2.2), o A* é capaz de encontrar caminhos ótimos em uma grade ou grafo em tempo de execução e complexidade de memória menores que o *maze router*, também detalhado anteriormente (3.2.1). O algoritmo funciona de forma iterativa, tal como o anterior. No entanto, não há a necessidade de expandir toda a vizinhança do nodo origem, como realizado no método previamente apresentado. A proposta consiste em aumentar a complexidade do nodo em si adicionando informações a ele que guiem o método na busca pelo próximo nodo. Essa informação é atribuída ao vértice a partir da computação da função *f(n)* descrita na Equação (13), definida a seguir.

$$f(n) = g(n) + h(n) \tag{13}$$

Na Equação (13), *g(n)* é um custo associado à distância entre o nodo *n* e a origem *S*, enquanto *h(n)* é uma heurística que tem por finalidade estimar o custo do caminho entre *n* e o destino *T*. Exemplos de heurísticas utilizadas para *h(n)* são as distâncias euclidiana e de Manhattan (ambas computadas com complexidade assintótica constante) ou, no contexto do projeto de células, a priorização de determinadas conexões em detrimento de outras (uso de metal ao invés de polisilício, por exemplo).

O algoritmo prossegue realizando o cômputo da função *f(n)* para todos os vizinhos imediatos (distância unitária) do nodo atual, escolhendo aquele que possuir o menor valor de *f(n)* para uma nova iteração. Nota-se que, apesar de verificar todos os vizinhos imediatos de um nodo, essa computação consiste apenas no cálculo de *f(n)* e na escolha do menor valor mínimo dentre os candidatos, não necessitando expandir a área de busca como realizado no algoritmo *maze* e de Dijkstra. ■

3.3 Ferramentas de Síntese Automática

Ferramentas de apoio ao projeto eletrônico estão presentes em diversos estágios do desenvolvimento VLSI. Em especial, no contexto do projeto de células, essas ferramentas atuam de forma a prover, de forma otimizada, os blocos fundamentais que serão utilizados no *design* de circuitos através da metodologia *standard cell*. No entanto, apesar de aumentar a produtividade, tais ferramentas, em geral, não atingem a qualidade dos leiautes quando comparados aos obtidos manualmente (*full-custom*). Encontrar o ponto de equilíbrio no *trade-off* entre leiautes extremamente otimizados e a produtividade do processo é um desafio para a indústria e academia em geral.

São exemplos de ferramentas de EDA utilizadas no projeto de células sob demanda: LiB (HSIEH, *et al.*, 1990), XPRESS (GUPTA, *et al.*, 1996a), CELLERITY (GURUSWAMY, *et al.*, 1997), TEMPO (RIEPE, *et al.*, 2003), cellTK (KARMAZIN, *et al.*, 2013) e ASTRAN (ZIESEMER, *et al.*, 2015). As seções a seguir detalharão o funcionamento do cellTK e do ASTRAN por apresentarem resultados comparáveis ao projeto manual.

3.3.1 cellTK (KARMAZIN, 2013)

O cellTK é uma ferramenta concebida com o propósito de síntese física de células assíncronas, sem restrições topológicas, portanto. Devido a essa

característica, essa ferramenta pode ser aplicada no projeto de células não-duais e não-planares, tais como as derivadas do Kernel Finder, metodologia de geração lógica apresentada na seção 2.2.3 desse trabalho. Nessa seção serão apresentados os principais métodos implementados no cellTK

A metodologia de posicionamento de transistores do cellTK busca por soluções seguindo o estilo *linear-matrix* que contenham o menor número de quebras possível, métrica para avaliação de posicionamento proposta originalmente em (UEHARA, *et al.*, 1981) (seção 3.1.1.1). Convertendo o *netlist* para uma estrutura de grafos (tal que os vértices dessa estrutura são equivalentes aos nós do *netlist* e as arestas do grafo representem os transistores), é feita uma busca por caminhos de Euler que contemplem o grafo por inteiro. Esse processo é realizado a partir da busca em profundidade na estrutura criada, comparando os resultados obtidos com o *netlist* a fim de validar as possíveis soluções. Finalmente, é feita a escolha do caminho que resulta no menor número de quebras na difusão.

A etapa de roteamento da ferramenta pode ser desmembrada em quatro principais estágios, descritos abaixo:

- Roteamento de *gates* verticalmente alinhados: é feito o roteamento em polisilício nos *gates* que estão alinhados entre os planos PU e PD;
- Roteamento topológico: essa etapa consiste em rotear a célula em metal 1 a fim de obedecer a topologia descrita no *netlist*. Isso é realizado criando as interconexões que não estão cobertas pelo compartilhamento de área ativa;
- Roteamento de alimentação: o próximo estágio é responsável por conectar a difusão com as linhas de VCC/GND localizadas nos limites superior e inferior do leiaute. Nesse estágio o cellTK utiliza-se de metal 1 e metal 2;
- Roteamento de entrada e saída (E/S): o último estágio realiza o roteamento das interfaces da célula também através de metal 1 e 2.

Baseado no algoritmo A*, o roteador utilizado no cellTK é o Countour (DION, *et al.*, 1995), ferramenta que possibilita operar sob uma estrutura não gradeada de forma eficiente. Isso garante que as conexões possam ser feitas em qualquer espaço da célula, não estando atrelado a um *pitch* pré-determinado e constante.

Devido as características do roteamento adotado, o cellTK não segue o estilo de Maziasz (MAZIASZ, *et al.*, 1991), o qual restringe o uso de camadas de metal para a síntese de células. Ademais, é importante ressaltar que a ferramenta dá suporte ao *design* em até 90nm. Finalmente, comparando-se os resultados provenientes dos

experimentos realizados com o cellTK com soluções produzidas manualmente, a ferramenta foi capaz de gerar leiautes com incrementos médios de 51%, 12% e 9% em termos de área, potência e atraso, respectivamente.

3.3.2 ASTRAN (ZIESEMER, *et al.*, 2015)

O *Automatic Synthesis of Transistor Network* (ASTRAN) é uma ferramenta *open-source* de geração de células no estilo *linear-matrix* e circuitos digitais sob demanda proposta em (ZIESEMER, *et al.*, 2015). Tal ferramenta suporta redes de transistores sem restrições topológicas, uma importante característica no contexto desse trabalho. Ademais, vale ressaltar que, diferentemente da ferramenta cellTK (seção 3.3.1), o ASTRAN segue o estilo de Maziasz (MAZIASZ, *et al.*, 1991), o que proporciona uma integração das soluções projetadas com circuitos digitais complexos providos por outras metodologias, dentre outras vantagens. Outro aspecto importante a ser observado é relativo à qualidade das soluções providas pelo ASTRAN, capaz de desenvolver leiautes com redução média de 19,2% em área e desempenho semelhante em potência e atraso comparados a leiautes desenvolvidos a mão.

A Figura 23 ilustra os principais estágios do gerador de células (Cellgen) do ASTRAN, bem como seu fluxo de execução. A seguir serão detalhados os estágios principais da ferramenta.

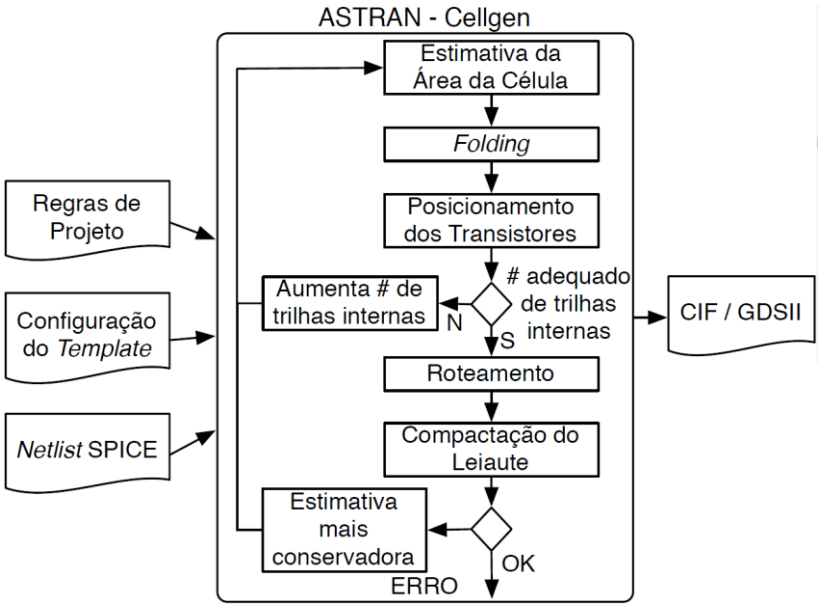


Figura 23 – Cellgen: gerador de células do ASTRAN.

Fonte: ZIESEMER, 2014, p. 48.

3.3.2.1 Estimativa da Área da Célula

O primeiro estágio é responsável por estimar a área do leiaute a ser desenvolvido, criando, assim, uma representação abstrata da célula que permita a compactação posteriormente. O método recebe como parâmetros para o cálculo a altura *W* dos transistores, a posição dos poços P e N (*P-well* e *N-well*), o número de trilhas internas, o *off-set* (relativo ao alinhamento da célula na grade de roteamento), o tamanho do *pitch*, a presença de TAPs (contatos de substrato) e as regras de projeto da tecnologia. Com esses dados, o algoritmo calcula a posição da trilha central (área localizada entre as linhas de difusão da célula), bem como a quantidade de trilhas que haverá nessa região. É válido ressaltar que, quanto maior essa zona da célula, maior será a tendência dos transistores sofrerem *folding* (detalhado a seguir em 3.3.2.2), já que o *W* disponível para as áreas PMOS e NMOS estará limitado.

3.3.2.2 *Folding* dos Transistores

O *folding* de transistores é uma técnica empregada na resolução do problema de desperdício de área do leiaute ocasionado por diferentes larguras (*W*) dos componentes da célula. Para exemplificar o problema, considere a Figura 24. Em (a) está presente um plano lógico composto por três transistores NMOS *a, b e c* de largura *w*, *2w* e *2w*, respectivamente. Em (b) tem-se o leiaute (simplificado, contendo apenas a solução posicionada) resultante dessa associação sem a aplicação do *folding*, onde podemos notar uma área (cinza) que não contém nenhum elemento do circuito (caracterizando um desperdício de área disponível do leiaute). Finalmente, em (c) tem-se uma rede lógica equivalente a (a) em que os transistores *b* e *c* foram submetidos à *folding*. Como podemos notar em (d), o leiaute (apenas posicionado, sem roteamento) não tem áreas vazias, aproveitando, assim, todo o espaço disponível.

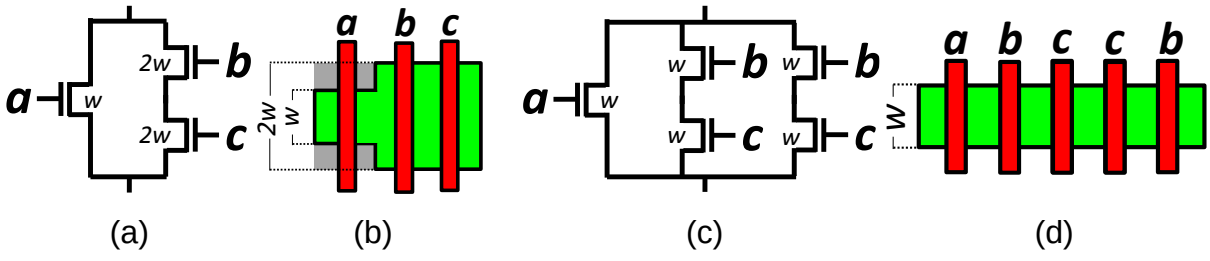


Figura 24 – *Folding* de transistores. (a) Rede lógica original. (b) Leiaute posicionado resultante de (a). (c) Rede lógica equivalente a original com a aplicação de *folding* nos transistores *b* e *c*. (d) Leiaute resultante de (c).

Recentemente, (SMANIOTTO, *et al.*, 2016) adicionou ao ASTRAN uma técnica de realização de *folding* conjunto, onde é aplicado o método sobre uma associação de transistores em série, em contrapartida com o descrito em (ZIESEMER, *et al.*, 2015), onde é prevista a aplicação do *folding* individual dos transistores. Como vantagens, essa técnica é capaz de diminuir interconexões da célula, o que leva a otimizações nas características geométricas e elétricas dos circuitos gerados.

Por fim, vale ressaltar que, através do *folding*, torna-se possível projetar células que respeitem uma largura *W* constante, o que facilita o posicionamento das linhas de alimentação e interconexão entre as células.

3.3.2.3 Posicionamento

Conforme discutido na seção 3.1 dessa dissertação, o posicionamento tem como principal objetivo definir o ordenamento dos transistores no leiaute. É através dessa etapa que torna-se possível o compartilhamento da linha de difusão entre transistores adjacentes, bem como o alinhamento vertical de *gates* entre planos lógicos distintos, facilitando o roteamento *intra-cell*. Para tanto, diversos algoritmos foram propostos nas últimas décadas, tais como os apresentados na seção 3.1.

O posicionador do ASTRAN, por sua vez, implementa um algoritmo de *Threshold Accepting*. Como descrito na seção 3.1.2.1, esse método é uma meta-heurística capaz de posicionar arranjos de chaves sem dualidade, um aspecto relevante para essa dissertação, dada a natureza das portas complexas geradas via Kernel Finder. Como vantagens, essa metodologia proporciona uma flexibilidade maior à ferramenta, sendo possível priorizar uma métrica de posicionamento em detrimento das métricas clássicas adotadas pela maioria dos algoritmos disponíveis (número de *gaps* e alinhamentos verticais de *gates*).

Conforme o que fora observado detalhadamente no Algoritmo 2, o TA possui uma função custo (linha 6) atrelada a solução, avaliando-a para a tomada de decisão do método (linha 7) que leva a novas soluções de menor custo. Portanto, faz-se necessário definir uma função de avaliação da solução – no caso do problema de posicionamento, essa função deve avaliar apenas o leiaute posicionado. Para isso, (ZIESEMER, 2014) define os seguintes parâmetros componentes da função custo:

- Largura da célula (w_w): para cálculo da largura da célula cria-se uma representação abstrata da mesma, onde cada transistor ocupa três colunas no leiaute (representando *source*, *gate* e *drain*). Caso haja compartilhamento de

área ativa entre transistores adjacentes, faz-se o reaproveitamento da última coluna do transistor posicionado anteriormente com a primeira coluna do transistor posicionado posteriormente. Ademais, leva-se em conta o alinhamento vertical de *gates* entre os planos PU e PD para o cômputo desse parâmetro;

- Número de quebras (w_g): contabiliza-se a quantidade de *gaps* na linha de difusão dos planos PU e PD da célula. É válido ressaltar que esse parâmetro independe das quebras estarem verticalmente alinhadas;
- *Mismatching* vertical de *gates* (w_{gm}): parâmetro que indica a presença de gates desalinhados verticalmente, o que impede o roteamento em polisilício para boa parte dos casos;
- Comprimento das conexões (w_{wl}): para cômputo dessa métrica faz-se uma estimativa do comprimento total das conexões internas da célula considerando apenas conexões horizontais. Para tanto, utiliza-se a mesma representação abstrata apresentada no cálculo de largura da célula. Células que possuem menor comprimento de suas conexões tendem a apresentar um maior número de linhas de roteamento entre difusão e a alimentação e melhores características elétricas;
- Densidade máxima de conexões (w_{md}): medindo-se a quantidade de linhas que cruzam ortogonalmente as colunas do leiaute tem-se a densidade de conexões daquela porção da célula (densidade local). O cômputo é feito tomando como base a representação abstrata da célula para todas as colunas da mesma, onde o valor máximo (a coluna com maior densidade) é tomada como base para o parâmetro. Esse aspecto é importante devido à limitação nos recursos de roteamento (número de camadas);
- Densidade local das conexões (w_{ld}): a densidade local das conexões é obtida através da soma dos quadrados das densidades locais. É uma medida importante para roteamento local da célula, levando a uma melhor qualidade do leiaute nesse quesito.

Empiricamente observou-se que algumas métricas impactam mais significativamente na qualidade do leiaute desenvolvido. Para que tais métricas sejam priorizadas na função custo, o cômputo dessa função se dá de forma parametrizada,

definindo pesos para cada uma das variáveis. A Equação (14) mostra a função custo e os pesos propostos por (ZIESEMER, *et al.*, 2015). ■

$$f = 100 \cdot (4w_{gm} + 4w_{md} + 3w_w + 2w_g + w_{wl}) + w_{ld} \quad (14)$$

Ainda relativo ao posicionamento via TA exposto no Algoritmo 2, há a necessidade de se definir uma função perturbação *N* (linha 5) capaz de encontrar novas soluções na vizinhança da solução atual. Ziesemer *et al*. define como função perturbação a seguinte abordagem (ilustrada na Figura 25):

- Primeiramente, seleciona-se de maneira randômica um conjunto de transistores posicionados adjacentemente no leiaute;
- Para esse conjunto, duas ações com a mesma probabilidade de execução são possíveis: o espelhamento ou o deslocamento da porção do circuito no leiaute.

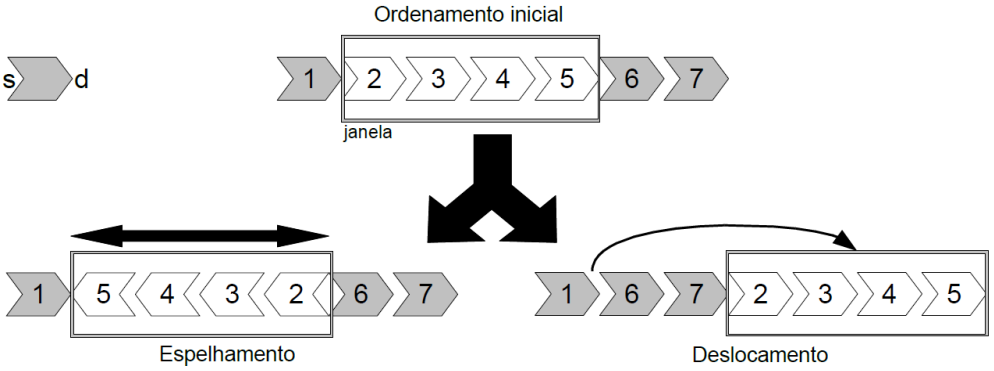


Figura 25 – Função de perturbação do método *Threshold Accepting* implementada no ASTRAN.
Fonte: ZIESEMER, 2014, p. 60.

3.3.2.4 Roteamento

O roteamento de célula do ASTRAN é a próxima etapa seguindo o fluxo ilustrado pela Figura 23. Essa etapa é responsável por realizar as conexões de sinal, interface e alimentação da porta desenvolvida respeitando o *netlist* de entrada.

Para realizar o roteamento, primeiramente é definida a estrutura de dados utilizada no processo. (ZIESEMER, *et al.*, 2015) define o uso de representação em forma de grafos do circuito digital, tal que os seus vértices representam os pinos de E/S da célula e as arestas representam os pontos que devem ser conectados durante a compactação (descrita na seção 3.3.2.5). A Figura 26 mostra os grafos utilizados para representação do circuito desenvolvido em suas respectivas camadas.

Definidos os pontos de conexão nos grafos a partir da configuração lógica da rede e do leiaute resultante da etapa de posicionamento, é realizado o roteamento através de uma versão modificada do roteador PathFinder (MCMURCHIE, *et al.*, 1995). O PathFinder é um roteador baseado no paradigma de negociação de congestionamento utilizado originalmente para roteamento em FPGAs. Tal característica faz com que vértices que estejam congestionados sejam disputados pelas redes de forma que as que estiverem com maiores dificuldades de traçar caminhos alternativos acabem elevando a sua prioridade, o que resulta em uma maior uniformidade na densidade do roteamento. É válido ressaltar que, além desse fator, cada conexão ainda possui um custo associado, evitando configurações que levem a dificuldades de compactação, por exemplo.

O PathFinder é composto de dois estágios: um roteamento de sinal e um roteamento global. O roteamento de sinal é realizado através do algoritmo de Dijkstra (DIJKSTRA, 1959) (detalhado na seção 3.2.2), escolhido em detrimento ao roteador *maze* (LEE, 1961) (seção 3.2.1) por conta da menor complexidade computacional e em detrimento ao A* (HART, *et al.*, 1968) (seção 3.2.3) pela dificuldade de se definir uma função *h(n)* (definida na Equação (13) apresentada previamente) que leve a bons resultados práticos. Por fim, o roteamento global tem como função realizar a chamada do roteamento de sinais até que todas as redes estejam devidamente conectadas.

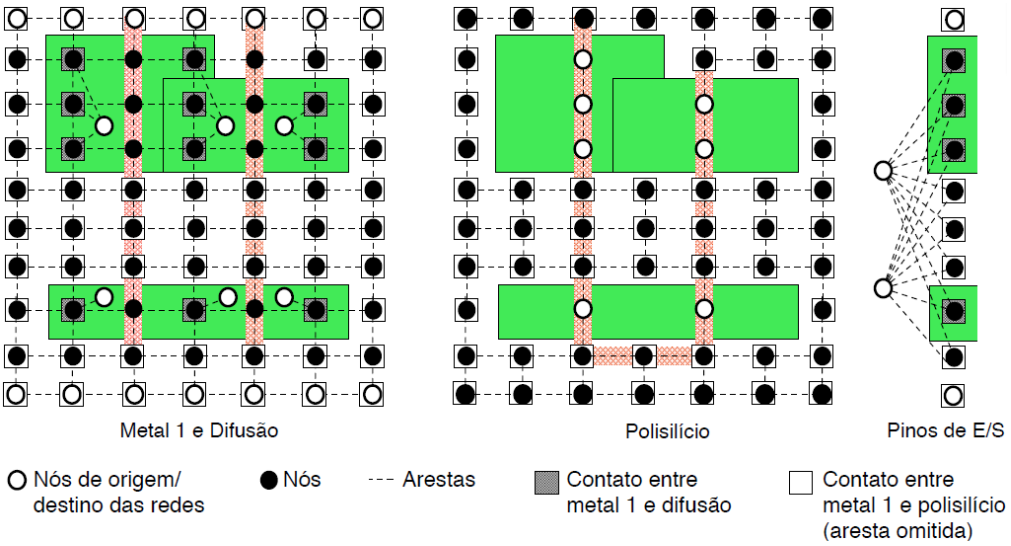


Figura 26 – Estrutura de grafos utilizada no roteamento *intra-cell*.

Fonte: ZIESEMER, 2014, p. 62.

3.3.2.5 Compactação

A etapa final do ASTRAN para a geração de células é responsável pela compactação do leiaute. A partir das informações de todos os polígonos obtidas após a computação das etapas de desenvolvimento anteriormente exploradas, esse estágio atua com o objetivo de diminuir as distâncias internas da célula de forma a otimizar a sua área e diminuir o comprimento das interconexões dos componentes da mesma.

Após os estágios de posicionamento e roteamento, o compactador associa as bordas de cada polígono da célula a uma posição, computando o tamanho de cada componente logo após. Com essas informações são criadas uma série de equações lineares que servem como base para aplicação da técnica de programação linear com inteiros (ILP), motor do estágio de compactação do ASTRAN.

3.4 Conclusões

Nesse capítulo foi realizado uma revisão dos métodos utilizados na síntese física no contexto de células. Foram abordados métodos clássicos e estado da arte relacionados ao posicionamento para redes duais e não-duais, bem como algoritmos aplicados para o problema de roteamento.

Finalmente, apresentou-se duas ferramentas que incorporam tais métodos para a geração automática do leiaute. A primeira, cellTK, apresenta bons resultados especialmente para células assíncronas. A segunda, ASTRAN, alcança resultados comparados ao leiaute produzido manualmente para células duais e não-duais, incluindo circuitos assíncronos e portas lógicas complexas, foco desse trabalho. Por conta disso e do fato da ferramenta ser *open-source*, o ASTRAN atuou como motor principal da síntese física da metodologia Libra, onde foram adicionados os métodos apresentados no próximo capítulo.

4 METODOLOGIA PROPOSTA

O objetivo principal desse trabalho é propor uma metodologia automática de geração de portas lógicas complexas para soluções providas pelo Kernel Finder, método estado da arte baseado em grafos para minimização de redes de transistores. Com isso, torna-se possível realizar um comparativo em nível de leiaute (ou seja, em termos físicos, estendendo o experimento realizado até então que consistia em apenas realizar o teste relativo ao número de chaves) com células geradas através do tradicional paradigma de fatoração Booleana.

Para tanto, propõe-se a metodologia Libra, apresentada resumidamente na Figura 27, onde estão presentes as interfaces (entradas e saídas) da metodologia (área branca) e os principais métodos aplicados em cada etapa que a compõem (área cinza). As próximas seções desse capítulo introduzirão todos os componentes presentes nessa metodologia.

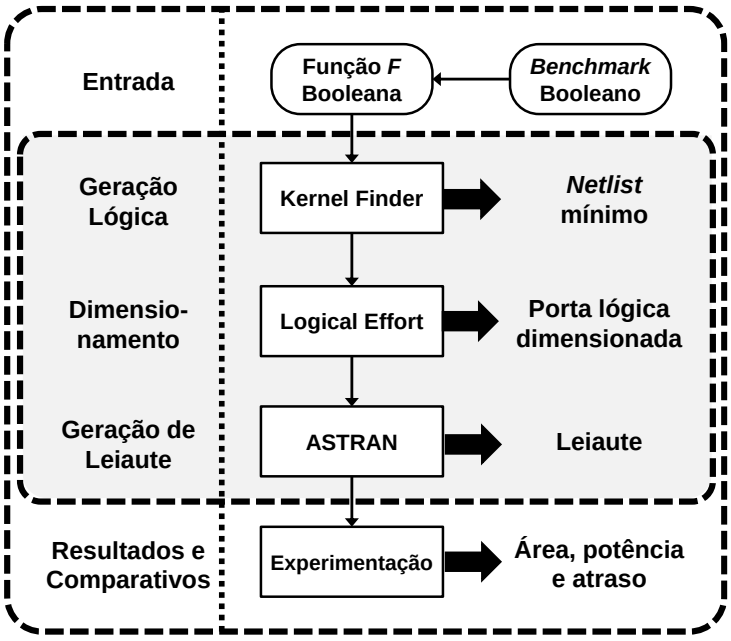


Figura 27 – Metodologia Libra proposta para a geração de portas lógicas complexas.

4.1 Geração Lógica

O estágio de geração lógica da metodologia proposta nesse trabalho tem como objetivo prover a rede de transistores que implemente a função Booleana de entrada. Como motor principal desse estágio, adotou-se o Kernel Finder (POSSANI, *et al*., 2015), metodologia baseada em grafos detalhada na seção 2.2.3. Conforme apresentado, o KF é considerado estado da arte na geração de arranjos de transistores sob demanda, provendo soluções otimizadas (relativo ao número de transistores) quando comparado com metodologias de mesmo propósito.

O Algoritmo 4 ilustra o método proposto para geração de *netlist* mínimo via KF. Os parágrafos que seguem descrevem o algoritmo e a prova da obtenção de solução mínima alcançável pelo Kernel Finder combinado ao método proposto.

Algoritmo 4 Geração Lógica Mínima via Kernel Finder

```
1:  GeraRedeLogica ( F )
2:      PU ← kernelFinder ( F̄ )
3:      PD ← kernelFinder ( F )
4:      if ( ( planar ( PU ) ) and ( PU.n <= PD.n ) ) then
5:          PD ← dual ( PU )
6:      else if ( ( planar ( PD ) ) and ( PD.n < PU.n ) ) then
7:          PU ← dual ( PD )
8:      end if
9:      return ( PU ∪ PD )
10: end
```

Partindo de uma função Booleana *F*, o algoritmo cria duas redes de transistores via Kernel Finder: a primeira, a partir da função *F* na sua forma complementar, representa o arranjo de *pull-up* (*PU*) (linha 2); a segunda, a partir de *F* na sua forma direta, representa o arranjo de *pull-down* (*PD*) (linha 3). Após, realizam-se testes relativos à planaridade (através do algoritmo Boyer-Myrvold (BOYER, *et al.*, 2004), método estado da arte para verificação de planaridade em grafos) e aos tamanhos de *PU* e *PD* (linhas 4 a 9, respectivamente). Caso a rede *PU* seja planar e tenha um número de transistores menor que a rede *PD*, atribui-se ao arranjo *pull-down* o dual da rede de *pull-up* (linhas 4 e 5). De forma análoga, a mesma operação é realizada a partir do teste da planaridade da rede *PD* (linhas 6 a 8). Por fim é retornada a rede composta pela união entre *PU* e *PD* adicionando-se um inversor na saída conforme o *template* adotado (ilustrado na Figura 28).

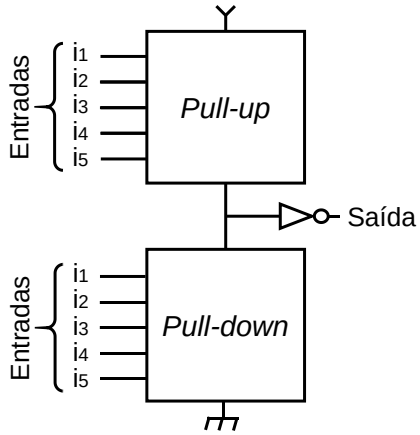


Figura 28 – Estrutura (*template*) da porta lógica complexa.

Conforme dito anteriormente, partiu-se da hipótese de que redes com um menor número de transistores levam a soluções (leiautes) mais otimizadas em termos físicos. Cabe ressaltar que essa premissa está de acordo com a literatura que trata da síntese de redes de transistores otimizadas (como as abordadas no capítulo 2). Portanto, é de suma importância garantir a geração da rede mínima através do KF visto que existem diferentes formas possíveis de geração lógica. Para exemplificar essa dependência entre a solução e os métodos de geração disponíveis utilizando o Kernel Finder, considere a função *f* descrita na Equação (15). Na Figura 29 (a) tem-se a solução obtida através do Algoritmo 4 contendo 16 transistores, enquanto (b) apresenta uma solução com 17 transistores gerada através de um método alternativo que consiste em gerar o plano PD e PU separadamente, unindo-os para formar a porta lógica (ou seja, atua como se apenas executasse as linhas 2, 3 e 9 do Algoritmo 4, não realizando consultas relativas a planaridade ou quanto aos tamanhos dos planos).

$$f = \overline{a} \cdot \overline{c} \cdot d + \overline{a} \cdot c \cdot \overline{d} + a \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot \overline{b} \cdot d \tag{15}$$

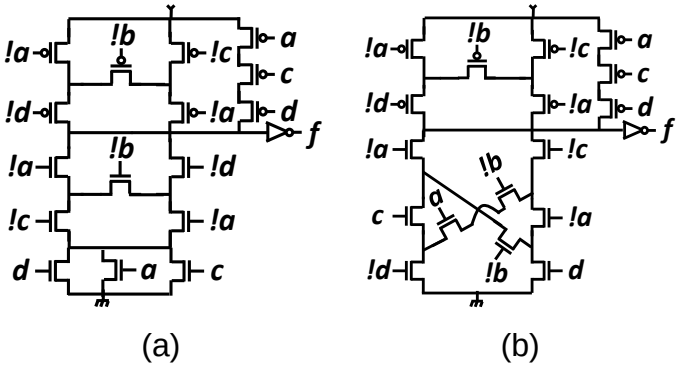


Figura 29 – Redes de transistores distintas geradas via Kernel Finder. (a) Solução obtida através do Algoritmo 4. (b) Solução proveniente de uma metodologia alternativa.

A seguir está a prova por exaustão demonstrando a capacidade do Algoritmo 4 em gerar as redes lógicas mínimas alcançáveis pelo KF. Considere como equivalentes as representações da célula em forma de grafo e como rede de transistores, conforme proposto em (UEHARA, *et al.*, 1981) (seção 3.1.1.1).

Teorema: o Algoritmo 4 provê a rede mínima alcançável via Kernel Finder tal que o número total de chaves do arranjo lógico é $n_{total} = n(PU) + n(PD)$ e $n(X)$ é o número de transistores num plano lógico *X* qualquer.

> **Lema:** um grafo *G* admite dual se, e somente se, *G* for planar. Portanto, uma rede de transistores *R* equivalente a *G* admite dual se, e somente se, *G* for planar.
>
> **Prova:** dada uma função Booleana em sua forma direta e complementar, as linhas 2 e 3 do Algoritmo 4 podem produzir quatro diferentes cenários para os planos lógicos (denotados por *X* e *Y* nessa prova) a depender da manipulação realizada pelo Kernel Finder. Os cenários após essa etapa do algoritmo são:
>
> 1. *X* é planar e *Y* é planar, tal que $n(X) < n(Y)$: seguindo o algoritmo, cria-se o plano *Y'* a partir da rede dual de *X* (linha 5 caso *X* represente o plano *PU* e linha 7 caso *X* represente o plano *PD*), tal que $n(Y') = n(X)$ e, portanto, $n(Y') < n(Y)$, resultando em $n_{total} = 2n(X)$;
> 2. *X* é não-planar e *Y* é planar, tal que $n(X) < n(Y)$: o algoritmo une os planos *X* e *Y* (apenas a linha 9 é executada), resultando numa solução tal que $n_{total} = n(X) + n(Y)$;
> 3. *X* é não-planar e *Y* é planar, tal que $n(X) > n(Y)$: o método cria o plano *X'* a partir da rede dual de *Y* (linha 5 caso *X* represente o plano *PU* e linha 7 caso *X* represente o plano *PD*, como no caso 1 dessa prova), tal que $n(X') = n(Y)$ e, portanto $n(X') < n(X)$. Isso resulta em $n_{total} = 2n(X') = 2n(Y)$;
> 4. *X* é não-planar e *Y* é não-planar: conforme caso 2, o algoritmo apenas une os planos *X* e *Y* (linha 9 é executada) Isso resulta em uma solução tal que $n_{total} = n(X) + n(Y)$.
>
> Portanto, para todos os casos a rede mínima é criada, provando o Teorema.

4.2 Dimensionamento de Transistores

A etapa de dimensionamento no contexto do projeto de portas lógicas complexas sob demanda é de fundamental importância para a qualidade da solução. Nessa etapa são computados os tamanhos (*sizings*) dos transistores que compõem o arranjo de acordo com uma métrica de especificação normalmente relacionada à área, atraso ou potência. Encontrar um ponto de equilíbrio relativo a esses três parâmetros é um desafio amplamente abordado na literatura. (FISHBURN, *et al.*, 1985), (BORAH, *et al.*, 1995), (BOYD, *et al.*, 2005), (BEECE, *et al.*, 2010) são exemplos de trabalhos que buscam por metodologias de dimensionamento cada vez mais precisas para suprir diferentes necessidades de projeto.

Para a etapa de dimensionamento desse trabalho, o método de esforço lógico (Logical Effort) (SUTHERLAND, *et al.*, 1999) foi implementado. Foi escolhida essa metodologia pela sua flexibilidade em lidar com topologias não regulares (como não-planares e não-duais), por gerar soluções sem privilegiar nenhum parâmetro de projeto específico e pela estrutura de dados compatível com a utilizada no método anterior (grafos).

O Logical Effort é uma metodologia de dimensionamento baseada em ganhos que permite calcular o esforço lógico de portas comparando-as com um inversor de referência. Tal esforço depende principalmente da topologia da rede de transistores, onde todos os caminhos presentes nos planos PU e PD devem contribuir com o mesmo fator para o cálculo da função lógica. Por conta disso, faz-se necessário computar os caminhos críticos de cada componente da célula. Define-se como caminho crítico do transistor *M* o caminho terminal (que liga a alimentação a saída da porta) que passa pelo maior número de transistores e por *M*.

Como exemplo, considere a Figura 30 que apresenta o plano *PD* da porta lógica complexa ilustrada na Figura 4 (b). Nota-se que para o transistor *M1* existem dois caminhos terminais possíveis (em vermelho e azul), sendo o caminho crítico aquele que passa por *M3* e *M5*, respectivamente (em vermelho). Portanto, o transistor *M1* deve receber um custo de esforço lógico $n_{M1} = 3$, valor equivalente ao tamanho do seu caminho crítico.

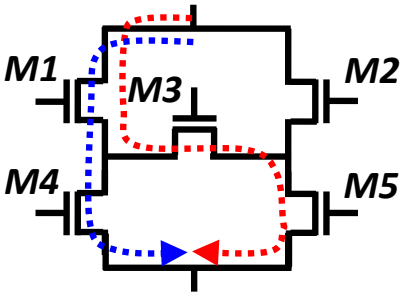


Figura 30 – Caminhos terminais da célula para o transistor *M1*.

O método ainda deve levar em conta a diferença tecnológica entre os transistores PMOS e NMOS: enquanto o inversor de referência possui um transistor NMOS com largura relativa (definida como a largura real em relação ao *W* mínimo descrito pelas regras de tecnologia) $w_N = 1$, o componente do plano PMOS do inversor terá largura $w_P = \lambda$, tal que $\lambda > 1$. Isso faz com que todos os transistores tipo P das portas desenvolvidas tenham um fator λ associado ao esforço lógico computado. Existem diversas formas de se calcular λ. Nesse trabalho adotou-se uma abordagem de iterações sucessivas em w_P do inversor padrão com w_N mínimo até haver equilíbrio das curvas de atraso de transição *high-to-low* e *low-to-high*.

O Algoritmo 5 abaixo apresenta o método implementado para busca de caminhos críticos e cálculo do Logical Effort em cada transistor. Para cada transistor pertencente ao plano P é realizado o cômputo do esforço lógico (identifica-se o caminho crítico do transistor) (linha 3). Caso o transistor seja do tipo PMOS o fator λ atua como multiplicador do valor obtido (linha 5).

Algoritmo 5 Dimensionamento via Logical Effort

```
1:  DimensionaTransistores ( P, λ )
2:      for i = 1:size( P.tr ) do
3:          P.tr[i].size ← LogicalEffort ( P.tr[i] )
4:          if ( P.tr[i].type == PMOS ) then
5:              P.tr[i].size ← P.tr[i].size * λ
6:          end if
7:      end for
8:      return ( P )
9:  end
```

O método Logical Effort presente no Algoritmo 5 (linha 3) foi implementado da seguinte forma: partindo-se da representação em forma de grafos do plano P (tal que cada transistor corresponde a uma aresta e cada nodo corresponda a um vértice, como descrito anteriormente), faz-se uma busca em profundidade (*Depth-First-Search*, DFS) pelos vértices correspondentes aos nós de *drain* e *source* do transistor

em questão. Localizado o transistor, é realizado o cômputo (também através de DFS) de todos os caminhos que passam pela aresta correspondente, tal que o valor do caminho crítico a ser retornado é igual ao tamanho do maior caminho (que passa por mais arestas) ligando o vértice de saída ao de alimentação.

Por fim, a Figura 31 ilustra o dimensionamento pelo método proposto dos transistores da porta lógica complexa apresentada na Figura 29 (a) que implementa *f*, função definida na Equação (15). Em (a) tem-se as larguras relativas *w* de cada transistor, enquanto (b) apresenta a largura real *W* projetada a partir das regras de projeto da tecnologia adotada nessa dissertação (STMicroelectronics 65nm). Na seção 5.1 serão explorados os detalhes relativos à experimentação, incluindo a apresentação de parâmetros básicos da tecnologia utilizada (W_{MIN}, λ e inversor de referência).

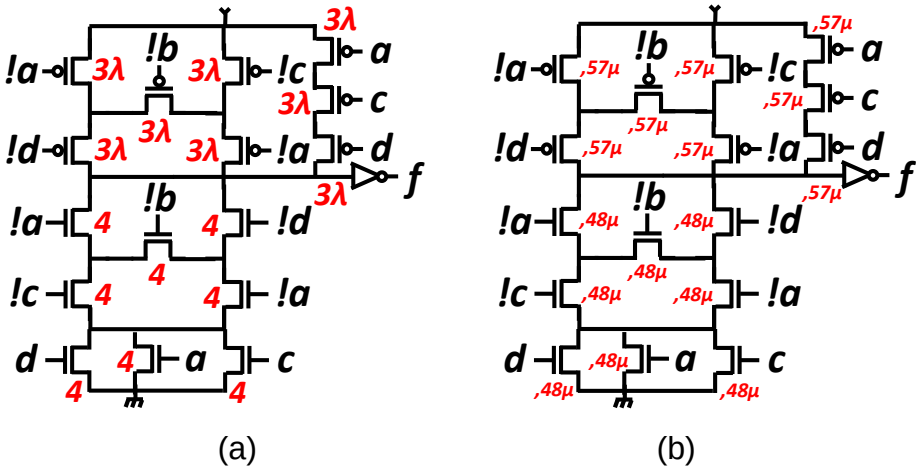


Figura 31 – Dimensionamento da porta lógica complexa ilustrada na Figura 29 (a). (a) Dimensionamento relativo. (b) Dimensionamento real.

4.3 Geração de Leiaute

A etapa de geração de leiaute da metodologia Libra é baseada no ASTRAN (ZIESEMER, *et al.*, 2015), ferramenta de geração automática de células apresentada detalhadamente na seção 3.3.2 dessa dissertação. Conforme ilustrado na Figura 23 da seção citada, a ferramenta é composta pelos seguintes estágios: estimativa de área do leiaute, *folding*, posicionamento de transistores, roteamento e compactação. Nesse trabalho focou-se no posicionamento visto que essa é a única etapa dependente da topologia da célula (contendo métodos próprios para redes com topologias duais e não-duais – seções 3.1.1 e 3.1.2, respectivamente).

Conforme previamente disposto, a principal métrica utilizada pelos métodos exatos de posicionamento presentes na literatura para gerar soluções de qualidade é a quantidade de *gaps* do leiaute, um aspecto que tem influência direta na área da célula devido à possibilidade do compartilhamento de difusão entre transistores adjacentes. No entanto, ainda que o posicionamento de redes não-duais não seja uma novidade para as metodologias que fazem parte da geração automática de células, a ocorrência de arranjos não-série-paralelo na tecnologia CMOS é um tema relativamente novo e um objeto de pesquisa ainda em aberto. Nessa seção será proposta uma nova abordagem para o posicionamento de transistores em redes não-série-paralelo.

Para caracterizar o problema e o algoritmo proposto, considere a Figura 32. Em (a) tem-se representada a porta lógica complexa definida na Figura 4 (b), enquanto na Figura 32 (b) estão ilustrados dois possíveis caminhos de Euler das redes de *pull-up* (amarelo) e *pull-down* (verde) a partir da aplicação da metodologia de posicionamento de Uehara *et al*. (UEHARA, *et al.*, 1981), base para a grande maioria dos posicionadores disponíveis. Finalmente, em (c) tem-se o leiaute resultante posicionado e que, pelo fato dos arranjos de transistores de PU e PD serem cobertos por um único caminho euleriano, não possuem nenhum *gap* na linha de difusão. Nota-se que, independente da escolha de qual caminho de Euler adotar, haverá um *mismatching* entre os polisilícios de *gate* da célula, ou seja, a célula apresentará quebras no polisilício. Definimos as portas lógicas com essa característica como células com *mismatching* intrínseco, já que, independente de otimizações que possam ser feitas no *netlist*, haverá pelo menos uma quebra nos *gates*.

A alternativa proposta para o posicionamento de células com *mismatching* intrínseco é apresentada na Figura 32 (d) e (e). Em (d) estão ilustrados dois caminhos de Euler para cada plano, tal que, conforme previsto por (UEHARA, *et al.*, 1981), esse aspecto introduzirá uma quebra (*gap*) na linha de difusão da célula. No entanto, como (e) deixa claro, essa quebra na área ativa faz com que seja possível obter uma solução sem quebras nos polisilícios dos *gates*, diferentemente do que ocorre no leiaute posicionado em (c).

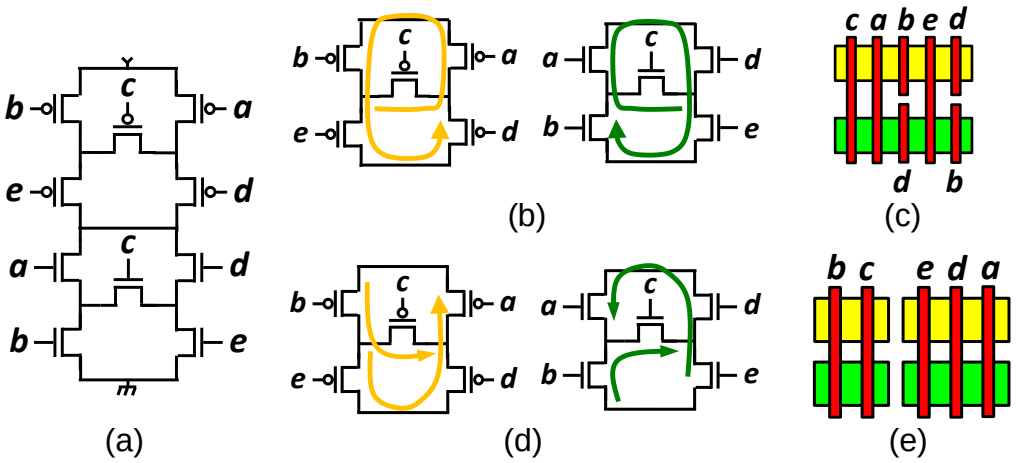
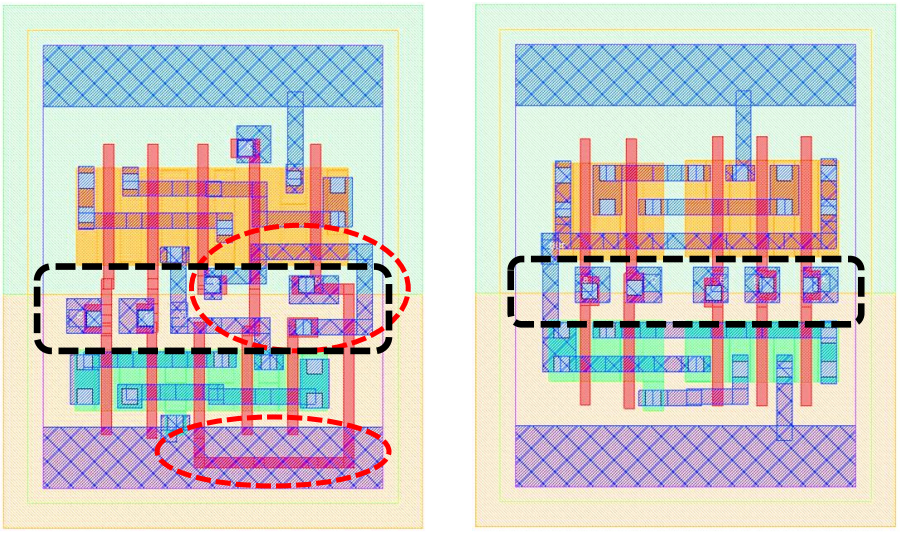


Figura 32 – Porta lógica complexa não-série-paralelo apresentada na Figura 4 (b) e estratégias de posicionamento. (a) Rede de transistores implementando a Equação (2). (b) Posicionamento de transistores sob a estratégia CAA. (c) Leiaute sem quebras na linha de difusão e com quebras no polisilício. (d) Posicionamento de transistores sob a estratégia CPG. (e) Leiaute com quebras nas linhas de difusão e sem quebras no polisilício.

Portanto, duas estratégias para posicionamento de células não-série-paralelo com *mismatching* intrínseco estão disponíveis: a primeira, definida aqui como CAA (*Continuous Active Area*) e que engloba as soluções obtidas pelas metodologias de posicionamento tradicionais, leva a um leiaute posicionado com quebras nos polisilícios de *gate* e linhas de difusões contínuas; a segunda, denominada CPG (*Continuous Polysilicon Gates*), acarreta em leiautes posicionados com quebras nas linhas de difusão e polisilícios contínuos.

Analisou-se tais estratégias para o caso do leiaute da porta lógica ilustrada na Figura 32: na Figura 33 (a) está presente a solução obtida seguindo a estratégia de posicionamento CAA, enquanto em (b) tem-se o leiaute da porta seguindo a estratégia de posicionamento CPG. Analisando ambas as soluções, nota-se que (a) fez maior uso de polisilício para roteamento dos pinos de E/S (com a linha de polisilício que percorre a célula horizontalmente circulada em vermelho na imagem), além de realizar uma ligação entre polisilícios a partir de metal 1 (também destacada em vermelho). Ambas as situações não ocorrem em (b). Outra observação que pode ser feita é em relação à densidade de roteamento entre difusões, maior em (a) (como demonstra as áreas circuladas em preto nas figuras). Isso faz com que o estágio de roteamento necessite de um maior espaço entre as difusões, potencialmente aumentando a área da célula. Vale ressaltar que esse aspecto em específico não é observado pelas metodologias de posicionamento disponíveis visto que a presença de *gaps* na difusão leva, em geral, a incrementos na área do leiaute em células sem *mismatching* intrínseco (que corresponde a maioria das células CMOS propostas até então).



(a) (b)

Figura 33 – Leiautes gerados através de diferentes estratégias de posicionamento para células com *mismatching* intrínseco. (a) Solução via estratégia CAA. (b) Solução via estratégia CPG.

Para justificar a escolha da estratégia CPG, A Tabela 3 ilustra a otimização alcançada pela célula da Figura 33 (b) (estratégia CPG) em relação à porta complexa da Figura 33 (a) (estratégia CAA). Ainda que a área não tenha sido otimizada nesse caso, a célula apresentou melhores características de potência e capacitância de entrada nos experimentos realizados, ao passo que o *delay* sofreu um acréscimo. Novos experimentos foram realizados para investigar o posicionamento proposto aplicado a um *benchmark* de células onde ficam claras as otimizações geométricas e elétricas que a técnica consegue alcançar. Os resultados desses experimentos podem ser conferidos no Apêndice C dessa dissertação.

Tabela 3 – Comparativo geométrico e elétrico das estratégias de posicionamento CPG relativo à CAA para os leiautes das Figuras 32 (a) e (b).

Métrica	***Otimização relativa (%)***
Área	0
Wirelength	11,79
Capacitância de entrada	6,46
Potência de leakage	1,90
Potência interna (internal power)	1,71
Potência de chaveamento (switching power)	1,79
Atraso de propagação	-2,71
Atraso de transição	-4,86

A fim de verificar o *mismatching* intrínseco das células produzidas na etapa de geração lógica da metolodogia Libra, propôs-se o algoritmo de detecção de *mismatching* a seguir (Algoritmo 6). O método consiste em verificar todos os caminhos de Euler disponíveis em ambos os planos da rede (linhas 2 e 3) (utiliza-se o algoritmo de Fleury para esse fim) comparando-os até que seja encontrado um par equivalente, o que caracteriza uma célula sem *mismatching* intrínseco (linhas 5 a 11). Caso não seja possível obter dois caminhos passando pelos mesmos transistores na mesma ordem, então conclui-se que a célula terá *mismatching* intrínseco (retorna a variável com o valor lógico definido na linha 4).

Algoritmo 6 Detecção de *Mismatching* Intrínseco

```
1:  DetectaMismatching ( PU, PD )
2:      caminhosPU ← computaEuler ( PU )
3:      caminhosPD ← computaEuler ( PD )
4:      mismatching ← true
5:      for i=1:caminhosPU.tamanho do
6:          for j=1:caminhosPD.tamanho do
7:              if ( caminhosPU[i] == caminhosPD[j] ) then
8:                  mismatching ← false
9:              end if
10:         end for
11:     end for
12:     return ( mismatching )
13: end
```

O Algoritmo 6 serve como guia para o ASTRAN, principal componente da geração de células da metodologia proposta. O ASTRAN recebe como entrada a *flag* computada pelo algoritmo, que informa se a porta lógica contém *mismatching* intrínseco. Em caso positivo, é realizada uma rotina de testes após a concepção do leiaute pela ferramenta a fim de investigar se a solução foi obtida pela estratégia CAA ou CPG, apenas aceitando aquelas que foram produzidas por essa última (ou seja, todos os leiautes gerados não terão quebras nos polisilícios dos *gates*).

Por fim, dando continuidade ao projeto da porta lógica ilustrada na Figura 29 (a) e com dimensionamentos expostos na Figura 31, a Figura 34 apresenta o leiaute computado pelo ASTRAN da célula que implementa *f*. Vale notar que, ainda que suprimidos nas figuras anteriores, as entradas complementadas deram origem a inversores no leiaute da porta complexa.

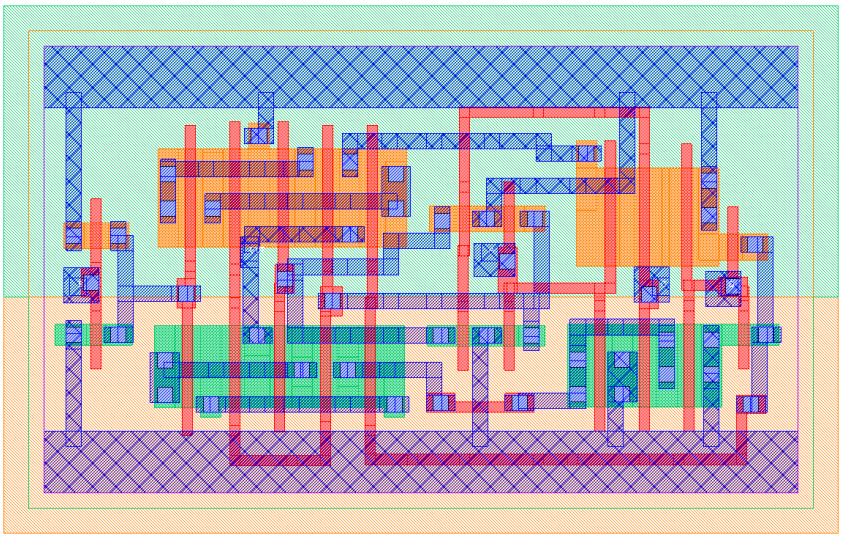


Figura 34 – Leiaute da porta lógica complexa que implementa *f* ilustrada na Figura 29 (a). ■

4.4 Conclusões

Nesse capítulo foi apresentada a metodologia Libra, proposta desse trabalho. Tendo como objetivo a geração de portas lógicas complexas sem restrições topológicas, a solução desenvolvida baseia-se nos principais métodos de geração lógica e física para portas complexas, provendo circuitos otimizados com qualidade comparável aos produzidos manualmente.

A metodologia pode ser dividida em três principais partes: geração da rede de transistores, dimensionamento e concepção automática do leiaute. Na primeira fase é adotado o Kernel Finder (método apresentado na seção 2.2.3) como motor de otimização incorporando-o a um método de otimização que garante resultados mínimos na ferramenta. Após, apresentou-se o algoritmo de dimensionamento Logical Effort, capaz de lidar com células sem restrições topológicas e que não privilegia determinados parâmetros da célula em detrimento a outros, tornando a comparação mais justa ao analisar os três eixos de otimização de circuitos (área, potência e atraso) com a mesma importância. Finalmente, utilizou-se o ASTRAN (ferramenta apresentada na seção 3.3.2) para a etapa de geração de leiautes. Na ferramenta foi adicionado o método de detecção de *mismatching* intrínseco a fim de guiar o posicionamento das células que possuem essa particularidade, levando-as a melhores resultados em geral, como apresentado no Apêndice C.

A partir do que fora proposto, torna-se possível realizar experimentos a fim de validar e testar as portas desenvolvidas. O próximo capítulo apresentará os resultados levantados a partir desses testes.

5 RESULTADOS

Esse capítulo é responsável por apresentar os comparativos entre as portas lógicas complexas geradas pela metodologia Libra e as soluções tradicionais providas por Composição Funcional. Os catálogos de funções adotados para esse propósito e apresentados a seguir são amplamente utilizados como *benchmark* na literatura (inclusive em (POSSANI, *et al.*, 2015) e (MARTINS, *et al.*, 2012), trabalhos onde são apresentados o Kernel Finder e a Composição Funcional, respectivamente).

5.1 Metodologia de Experimentação

Conforme proposto ao decorrer do texto, o objetivo principal dessa dissertação consiste em prover uma metodologia automática de geração de portas lógicas complexas via Kernel Finder, ou seja, sem restrições topológicas. Para avaliação das soluções geradas pela metodologia proposta foram realizadas análises de natureza geométrica (área, comprimento de fio – *wirelength* –, número de contatos e densidade de roteamento) e elétrica (capacitância de entrada, potência de *leakage*, potência interna, potência de chaveamento, atraso de propagação e atraso de transição), comparando as células providas pela metodologia Libra com soluções derivadas da Composição Funcional (seção 2.1.2), método estado da arte de geração de portas lógicas complexas baseado em fatoração Booleana (paradigma clássico amplamente adotado na indústria para minimização multinível).

A fim de tornar mais justo o comparativo, foram realizados os dimensionamentos das redes providas pelo FC através do mesmo método de *sizing* descrito no capítulo anterior (seção 4.2), ou seja, ambas as células projetadas possuem uma capacidade elétrica similar. Por fim, também deve-se ressaltar o uso do ASTRAN na etapa de geração de leiaute das redes providas por Composição Funcional.

Relativo à tecnologia adotada nos experimentos, utilizou-se a STMicroelectronics 65nm em todos os testes devido a limitações da ferramenta de geração de leiaute. Para propósitos do método de dimensionamento foi definido o inversor de referência ilustrado na Figura 35 com $W_N = 0{,}12\mu m$ e $\lambda = 1{,}58$. O valor W_N corresponde a largura de transistor mínima (informação disponível nas regras de projeto da tecnologia), enquanto λ foi obtido de forma empírica ajustando-se a largura do transistor PMOS de forma a equilibrar as curvas de transição da saída do inversor (*high-to-low* e *low-to-high*).

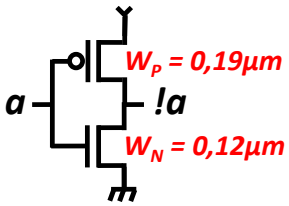


Figura 35 – Inversor de referência adotado no dimensionamento das células.

Os experimentos elétricos foram realizados nos ambientes das ferramentas Calibre PEX (extração de capacitâncias) e Cadence Spectre (demais dados elétricos), amplamente utilizadas para esses fins. Todos os arcos de transição e estados estáticos foram computados para diferentes cenários de capacitância de saída (definidos a partir de *fan-out* 4) e *slope*. Vale ressaltar que os resultados de potência e atraso apresentados nesse capítulo correspondem ao dado de pior caso. Por fim, a simulação assumiu um processo típico com as portas operando em tensão e temperatura padrão (1V e 25ºC, respectivamente).

5.2 *Benchmark 53 NSP Handmade Networks* (LOGICS, 2012)

O *53 NSP Handmade Networks* é um *benchmark* proposto em (LOGICS, 2012) composto por 53 redes de transistores não-série-paralelo desenhadas manualmente. Esse conjunto de células foi provido com o objetivo de apresentar soluções mínimas para determinadas funções lógicas.

O Apêndice A apresenta o conjunto de funções Booleanas de entrada correspondente ao catálogo utilizado. Como pode ser visto no apêndice, o *benchmark* compreende apenas funções *unate* de 4 a 7 variáveis.

5.2.1 Dados Topológicos

Nessa seção serão apresentados os dados obtidos a partir da etapa de geração lógica da metodologia Libra relativo às topologias das soluções geradas. Com o intuito de caracterizar o *benchmark* quanto ao seu aspecto construtivo (número de transistores, *sizing* e ocorrência de não-planaridade ou não-dualidade, por exemplo), essa informação torna-se útil para realizar a correlação entre outras métricas do leiaute com o conjunto de redes de transistores correspondente.

A Tabela 4 estende os dados apresentados na Tabela 2. Estão presentes os resultados topológicos obtidos para os arranjos gerados pelo Kernel Finder aplicados ao *benchmark* em questão. Vale ressaltar que todas as soluções providas por Composição Funcional são planares e duais (por conterem apenas arranjos série-paralelo). Os dados contidos na tabela foram apresentados em relação ao número total de funções do catálogo (53, tal que todas foram incluídas nesse experimento).

Tabela 4 – Dados topológicos das redes geradas pelo Kernel Finder para o *benchmark 53 NSP Handmade Networks*.

53 NSP Handmade Networks	***Aspecto topológico***		
	Não-série-paralelo	***Não-planar e não-dual***	***Diferença de transistores entre planos***
Total (%)	100	0	0

Como fica evidente, apesar da grande variedade do catálogo em relação à complexidade da função de entrada (número de variáveis), as redes produzidas pela geração lógica via Kernel Finder (apresentada na seção 4.1 e no Algoritmo 4) são não-série-paralelo, planares e duais. Ou seja, é esperado que o leiaute produzido apresente um aspecto regular no que tange à área das linhas de difusão (visto que os planos têm o mesmo número de transistores e não existem entradas complementadas).

Por fim, é importante ressaltar a otimização lógica e o *sizing* médio das portas lógicas complexas obtidas pelo Kernel Finder para que se tenha uma referência para a análise dos dados geométricos e elétricos levantados. Conforme apresentado anteriormente na Tabela 1 e abaixo na Tabela 5, as redes de chaves obtidas pela geração lógica da metodologia proposta apresentam uma redução no número de transistores de 26,28% em relação à solução proveniente do método de Composição Funcional e um *overhead* de apenas 0,75% em relação ao *W* médio por transistor.

Tabela 5 – Dados de *sizing* médio por transistor nas redes lógicas geradas por Composição Funcional e pelo Kernel Finder para o *benchmark 53 NSP Handmade Networks*.

Métrica	***Composição Funcional***	***Kernel Finder***	***Aumento (%)***
W por transistor (µm)	0,5582	0,5624	0,75

5.2.2 Dados Geométricos

Essa seção tem como objetivo apresentar o comparativo geométrico entre os leiautes providos pela metodologia Libra e por Composição Funcional. As métricas avaliadas são: área, *wirelength*, número de contatos e densidade de roteamento. Os resultados podem ser conferidos na Figura 36 (onde as barras azuis significam uma melhora na solução provida pela metodologia Libra em relação a célula obtida via FC e as barras em vermelho denotam uma piora na solução proposta também em relação ao leiaute FC) e no Apêndice B (onde estão condensados todos os resultados obtidos pelas simulações realizadas).

Conforme os dados apontam, as células geradas pela metodologia proposta alcançaram bons resultados geométricos quando comparadas as soluções providas pelo método de Composição Funcional. Relativo à área de leiaute, a Libra foi capaz de obter células otimizadas para todas as funções do *benchmark*. No pior caso (função *F19*), o ganho foi de 6,12%, enquanto, no melhor caso (*F31*), o ganho em área foi de 45,54% (para ilustrar essa grande otimização, as Figuras 36 (a) e (b) apresentam os leiautes das células obtidas via Libra e por Composição Funcional, respectivamente). Em média, a biblioteca de células provida via Libra alcançou ganhos de 20,12% e desvio-padrão de 8,90% em área em relação aos leiautes FC. Considerando que o KF é capaz de produzir uma redução de 26,28% no número de transistores comparado ao método de Composição Funcional para o *benchmark 53 NSP Handmade Networks*, pode-se concluir que a metodologia obteve resultados bastante satisfatórios (ou seja, seguiu a tendência da geração lógica).

Considerando *wirelength*, é esperado um ganho visto que essa métrica está diretamente relacionada à área e a quantidade de transistores da porta lógica. Em média, as soluções providas pela metodologia proposta apresentaram uma otimização de 27,79% nesse parâmetro com desvio-padrão de 9,74%.

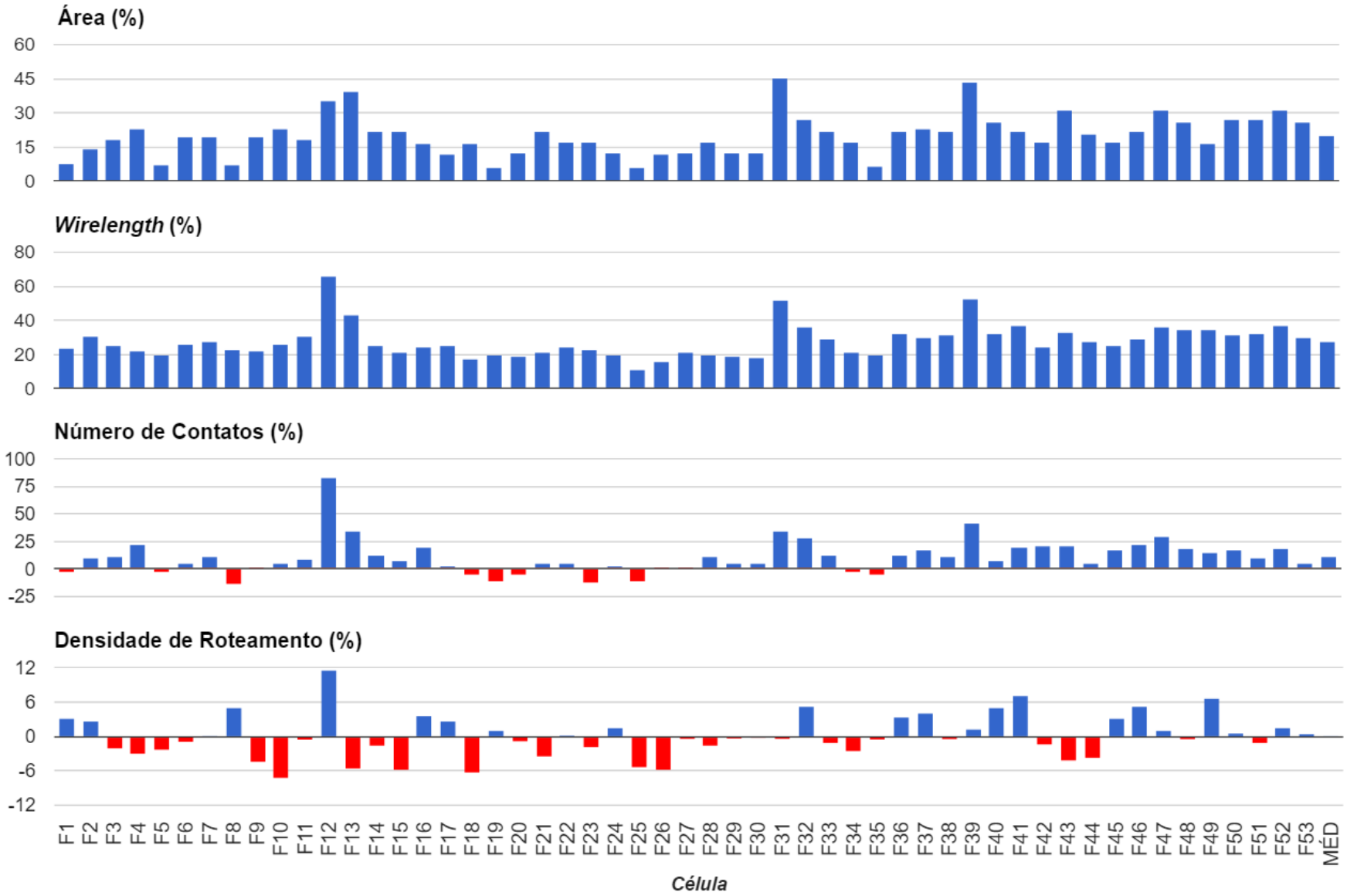
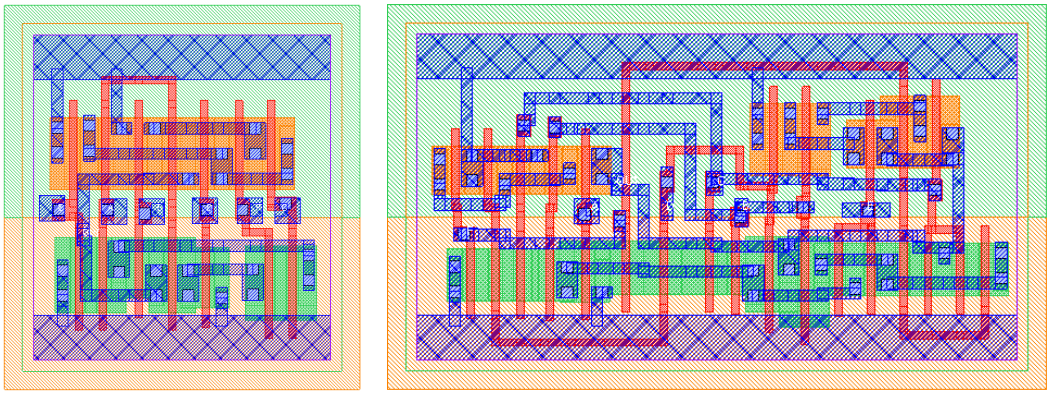


Figura 36 – Resultados dos comparativos geométricos entre as células geradas via Libra em relação as células providas por Composição Funcional para o catálogo *53 NSP Handmade Networks*.

Em relação ao número de contatos, a solução proposta apresentou uma redução média de 11,02% e desvio-padrão de 15,89%. Pode-se concluir que o ganho relativo ao número de contatos não seguiu à mesma taxa de diminuição do *wirelength*. Isso indica que a célula NSP gerada pela metodologia proposta apresenta linhas de roteamento menores (em média) do que a célula SP provida por FC. Finalmente, relativo à densidade de roteamento, obteve-se uma piora de apenas 0,05% com desvio-padrão de 3,81%.



(a) (b)

Figura 37 – Maior ganho em área (*F31*, 45,54%) observado entre as células propostas e as obtidas por Composição Funcional. (a) Célula provida pela metodologia Libra. (b) Célula provida pelo FC.

5.2.3 Dados Elétricos

Nessa seção realizaram-se as comparações elétricas das células desenvolvidas seguindo a metodologia de experimentação descrita na seção 5.1. Os dados levantados são referentes à capacitância de entrada, potência de *leakage*, potência interna, potência de chaveamento, atraso de propagação e atraso de transição. A Figura 38 ilustra os resultados obtidos.

Em relação à capacitância de entrada, uma métrica importante principalmente no contexto de circuitos, obteve-se um ganho médio de 29,68% com desvio-padrão de 3,53% e nenhum caso em que houve piora. Essa constante otimização deve-se principalmente ao fato da expressiva diminuição no número de transistores nas soluções provenientes do Kernel Finder (26,28%, conforme apresentado na Tabela 1) e um *W* médio constante (0,75% de diferença, segundo a Tabela 5)

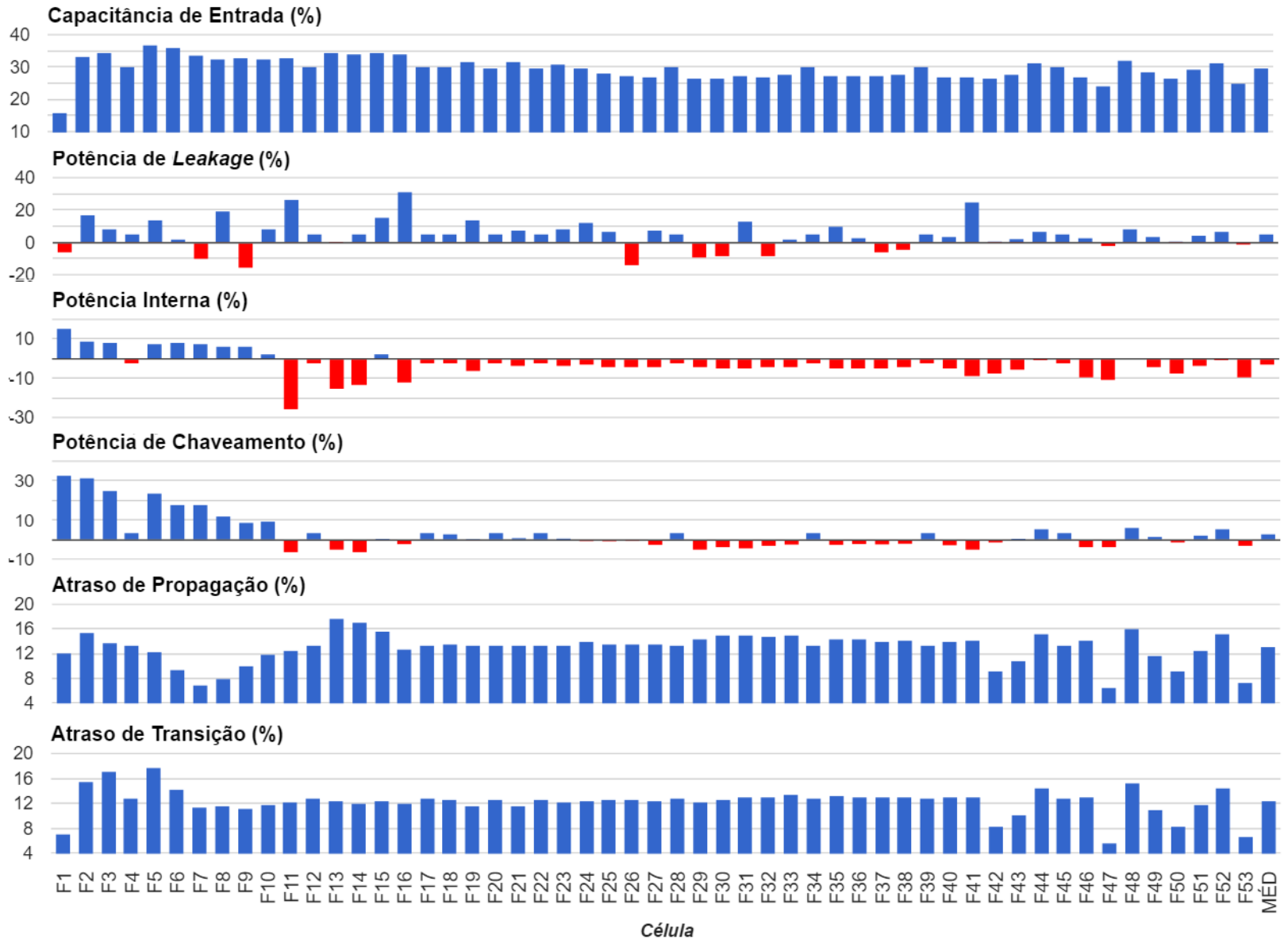


Figura 38 – Resultados dos comparativos elétricos entre as células geradas via Libra em relação as células providas por Composição Funcional para o catálogo *53 NSP Handmade Networks*.

Relativo aos valores de potência, foram obtidas pequenas otimizações em termos de potência de *leakage* – média de 4,87% com desvio-padrão de 9,45% – e potência de chaveamento – média de 2,95% e desvio-padrão de 9,13% – e uma perda em potência interna – média de 3,34% e desvio-padrão de 6,85%. Esse resultado já era esperado devido ao método de dimensionado desenvolvido na metodologia proposta considerar apenas o caminho crítico no cômputo do *W* dos transistores, o que acaba por superdimensionar as células (especialmente as não-série-paralelo visto que há maior flexibilidade para encontrar caminhos críticos maiores dado essa topologia da rede, como abordado na seção 5.3).

Por fim, em relação ao atraso da célula, as otimizações referentes aos *delays* de propagação e transição ficaram, em média, em 13,04% e 12,32% com desvio-padrão de 2,39% e 2,14%, respectivamente. Novamente, tal como descrito para a métrica de potência, esse era um resultado esperado devido ao superdimensionamento das células não-série-paralelo.

5.3 *Benchmark 4-input P-Class* (CORREIA, *et al.*, 2001)

O *4-input P-Class* é um benchmark apresentado em (CORREIA, *et al.*, 2001) equivalente a todas as funções Booleanas de até 4 variáveis (65536, no total). O catálogo foi computado a partir do agrupamento dessas funções em classes equivalentes considerando permutações nas entradas, reduzindo o *benchmark* a 3982 soluções funcionalmente distintas.

Para a experimentação do *4-input P-Class*, definiu-se randomicamente um subconjunto de tamanho *n* do catálogo completo visto que é inviável realizar os testes para o grupo inteiro de 3982 funções. Para o cômputo do tamanho do subconjunto *n* utilizou-se a fórmula de cálculo do tamanho da amostra apresentada na Equação (16) (KREJCIE, *et al.*, 1970). Definiu-se $Z_{\alpha/2} = 1{,}645$ (correspondente a uma confiança de 90% nos resultados) e erro $e = 5\%$. Para se obter o valor de desvio-padrão σ buscou-se o pior caso ($\sigma_{MÁX}$) dentre os experimentos realizados no *benchmark* anterior (ou seja, considerando todos os cenários de teste). Obteve-se $\sigma_{MÁX} = 15{,}01\%$ para o comparativo relativo à potência de chaveamento no cenário que compreendeu uma capacitância de saída $C_{OUT} = 1pF$ e *slope* de entrada fixado em 5ps. Por fim, *N* representa o tamanho total do benchmark ($N = 3982$).

$$n = \left\lceil \frac{Z_{\alpha/2}{}^2 \sigma_{MÁX}{}^2 N}{(N-1)e^2 + Z_{\alpha/2}{}^2 \sigma_{MÁX}{}^2} \right\rceil \quad (16)$$

A partir dos valores definidos acima para cada variável, tem-se que $n = 25$. No Apêndice A estão ilustradas as funções randomicamente selecionadas para representar o benchmark *4-input P-Class* nos experimentos realizados nas seções que seguem.

5.3.1 Dados Topológicos

Diferentemente do *benchmark* anterior, o catálogo *4-input P-Class* possui uma maior variedade topológica, fato que pode ser observado na Tabela 2 desse texto. Além disso, cabe ressaltar que o mesmo contempla funções *unate*, *non-unate* e *binate*, o que aumenta a complexidade da geração lógica e de leiaute devido à necessidade de inclusão de inversores no projeto. Ademais, vale ressaltar que as redes lógicas desse catálogo contêm, em média, mais transistores que as do *benchmark* anterior (considerando apenas arranjos providos pelo KF, tem-se 24,00 transistores por porta complexa enquanto as soluções do *benchmark* anterior apresentam 6,77 chaves por plano lógico, conforme a Tabela 1 aponta). A Tabela 6 apresenta os dados topológicos extraídos das 25 funções analisadas.

Tabela 6 – Dados topológicos das redes geradas pelo Kernel Finder para o subconjunto apresentando no Apêndice A representativo do *benchmark 4-input P-Class.*

Subconjunto de 25 funções do catálogo 4-input P-Class	***Aspecto topológico***		
	Não-série-paralelo	***Não-planar e não-dual***	***Contém diferença de transistores entre planos***
Total (%)	64,00	16,00	12,00

Analisando os resultados acima pode-se concluir que apenas 64,00% das redes geradas são NSP, ou seja, 36,00% das redes são SP. Para esse subconjunto SP o FC leva vantagem visto que é capaz de produzir solução exata mínima, diferentemente do KF. É válido ressaltar que 16,00% das redes analisadas são não-planares e não-duais. Ademais, referente à diferença no número de transistores entre os planos PU e PD, o resultado obtido foi de 12,00%. Como os dados levantados acerca das métricas de planaridade/dualidade e de diferença entre os planos são diferentes (16,00% e 12,00%, respectivamente), pode-se concluir que existe ao

menos um caso onde ambos os planos lógicos são não-planares e possuam o mesmo número de chaves.

Por fim, conforme dito anteriormente, é de fundamental importância considerar a otimização lógica e o dimensionamento médio dos transistores das soluções obtida via Kernel Finder para que se tenha a possibilidade de realizar uma análise mais acurada dos resultados geométricos e elétricos levantados a seguir. A Tabela 1 apresenta os dados referentes a todo o catálogo *4-input P-Class* onde pode-se observar uma otimização lógica de 6,89% através do KF. Para o subconjunto utilizado nos experimentos a seguir, os seguintes resultados relativos ao número de transistores das células foram observados (Tabela 7). Nota-se uma redução na quantidade de chaves de 5,99% obtida pelo KF utilizado na etapa de geração lógica da metodologia proposta. Quanto ao *sizing* médio por transistor, observou-se um *overhead* de 11,76% (Tabela 8).

Tabela 7 – Quantidade de transistores necessários para implementação das funções Booleanas presentes no subconjunto de portas lógicas complexas representativo do catálogo *4-input P-Class* obtidas através do FC e KF.

Benchmark	**Composição Funcional**	***Kernel Finder***	***Redução (%)***
Subconjunto de 25 células do catálogo 4-input P-Class	434	408	5,99

Tabela 8 – Dados de *sizing* médio por transistor nas redes lógicas geradas por FC e pelo KF para o *4-input P-Class.*

Métrica	***Composição Funcional***	***Kernel Finder***	***Aumento (%)***
W por transistor (µm)	0,5208	0,5820	11,76

5.3.2 Dados Geométricos

Como realizado para o *benchmark* anterior (seção 5.2.2), fez-se o levantamento dos dados geométricos considerando as métricas de área, *wirelength*, número de contatos e densidade de roteamento nas células geradas. Os resultados estão presentes na Figura 39.

Em relação à área do leiaute, observou-se uma otimização média de 1,00% e desvio-padrão de 7,70% nas soluções providas pela metodologia proposta. Diferentemente do que aconteceu no experimento anterior, houve uma grande variação no comparativo de área, incluindo soluções em que há perda nesse quesito.

As Figuras 39 (a) e (b) mostram os leiautes com maior perda (*F1944*, com perda de 12,76%) e maior ganho (*F3900*, com ganho de 14,74%) no comparativo de área realizado, respectivamente.

Conforme ilustra a Tabela 8, o *W* dos transistores gerados pelo KF tende a ser maior visto que redes NSP acabam por flexibilizar a busca por caminho crítico dado a sua topologia. Esse fator, aliado ao *folding* aplicado no ASTRAN para larguras que ultrapassem o limite da célula (que possui uma altura máxima pré-determinada), fazem com que as células NSP necessitem de uma largura maior para alocar os componentes da porta complexa. Ademais, conforme ilustrado nas Tabelas 1 e 6, é

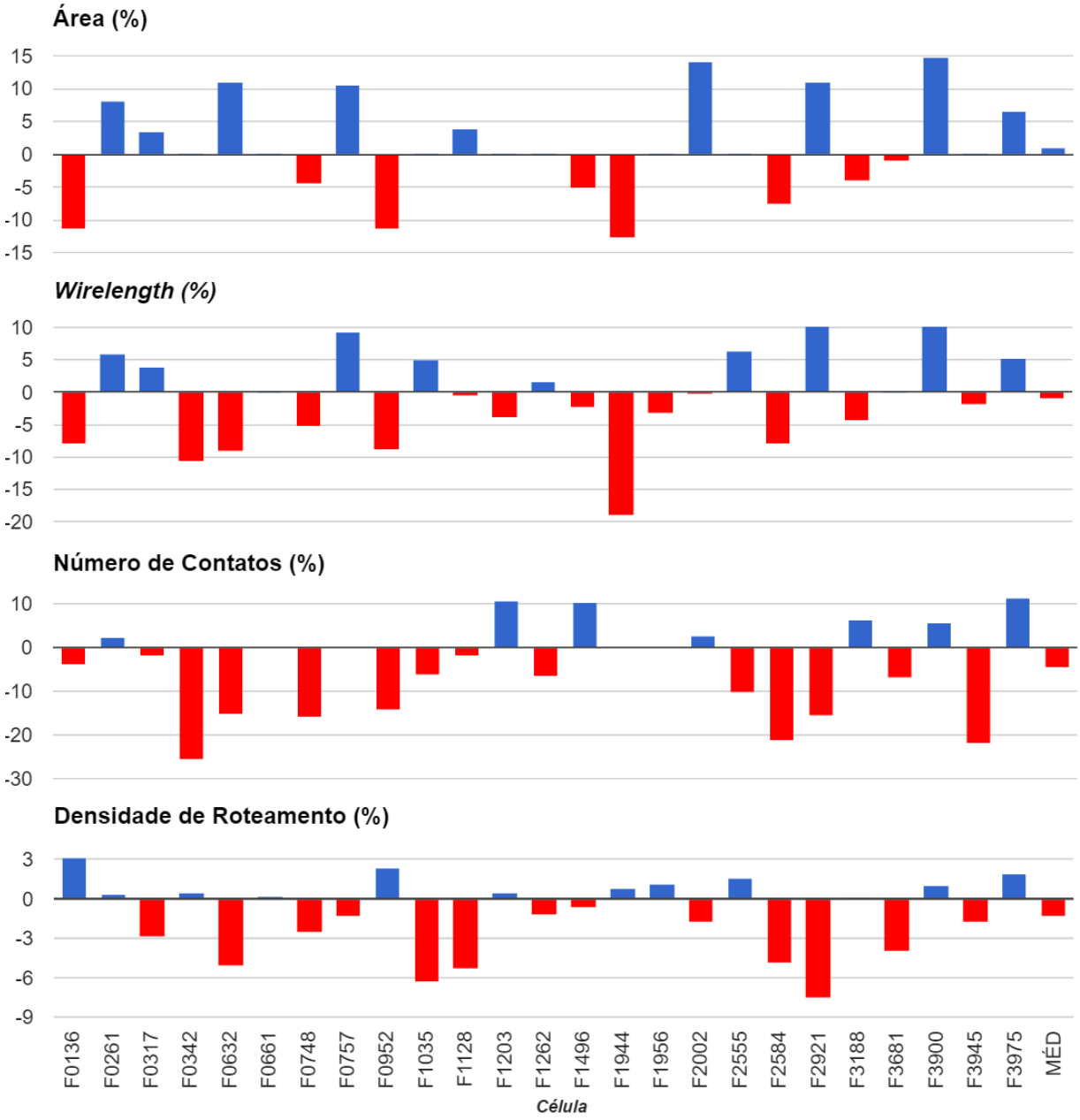


Figura 39 – Resultados dos comparativos geométricos entre as células geradas via Libra em relação as células providas por Composição Funcional para o subconjunto do catálogo *4-input P-Class*.

válido ressaltar que a redução no número de transistores das células do catálogo *4-input P-Class* não é tão significativa quanto a que ocorre no *benchmark* anterior, o que já indica uma menor otimização em quesitos geométricos.

Os resultados de *wirelength* apresentaram uma tendência semelhante à otimização em área, como esperado. Em média houve perda de 1,10% e desvio-padrão de 7,30% para as células geradas via Libra. Relativo ao número de contatos também foi observada uma perda média de 4,77% com desvio-padrão de 10,52% nas portas complexas propostas. Isso se deve principalmente a aplicação de *folding* nas células providas pela metodologia que adiciona novos contatos devido à necessidade de interconexão de mais elementos no leiaute.

Finalmente, em relação à densidade de roteamento houve uma perda de 1,29% com desvio-padrão de 7,30% (um grande contraste comparando-se ao mesmo dado levantado no experimento anterior). Essa diferença de densidade indica uma pouca utilização de metal 1 no roteamento da célula proposta, o que é esperado visto que o *folding* adiciona elementos que, em sua maioria, compartilham a área de difusão (ou seja, não necessitam de roteamento explícito).

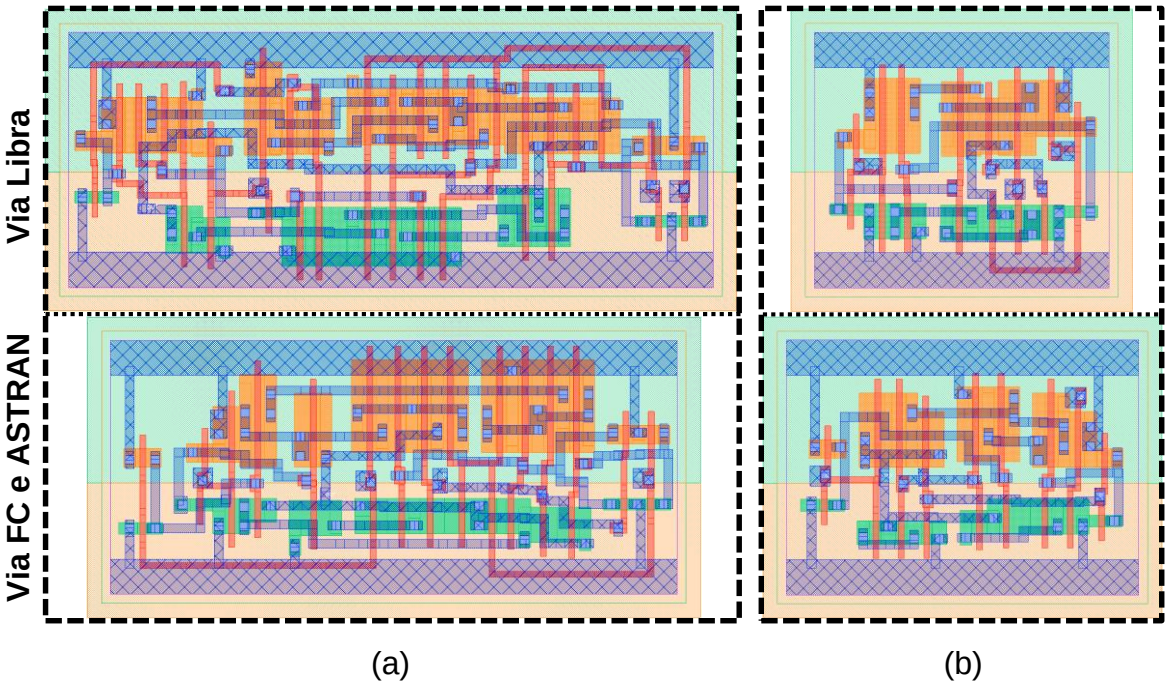


Figura 40 – Comparativos de área entre as células desenvolvidos via Libra e provides por Composição Funcional. (a) Caso de maior perda em área (*F1944*, com piora de 12,76%). (b) Caso de maior ganho em área (*F3900*, com redução de 14,74%).

Figura 41 – Resultados do comparativo elétrico entre as células geradas via Libra em relação as células providas por Composição Funcional para o subconjunto do catálogo *4-input P-Class*.

5.3.3 Dados Elétricos

O último experimento realizado é referente às características elétricas das células desenvolvidas em relação às células produzidas pelo método de Composição

Funcional. Os mesmos parâmetros do *benchmark* anterior foram investigados e os resultados estão apresentados na Figura 41 acima.

Como pode-se observar, a capacitância de entrada sofreu uma piora média de 8,43% e desvio-padrão de 11,43% na solução proposta. Essa perda ocorre devido ao maior *W* (Tabela 8) submetido a cada entrada da célula NSP com caminho crítico muito maior que a célula SP concorrente somado a baixa otimização relativa ao número de chaves (fato observado nesse *benchmark*, como já discutido).

Em relação à potência de *leakage* houve uma piora de 6,89% com desvio-padrão de 15,82%. Isso pode ser explicado pela mesma razão apresentada na Figura 39 (redes NSP tendem a ter transistores com *W* maior que suas versões SP).

Por razão semelhante a potência interna e a potência de chaveamento sofreram uma piora média de 8,65% e 5,83% com desvios-padrão de 16,99% e 21,77%, respectivamente.

Finalmente, em relação ao atraso de propagação e de transição houveram ganhos médios de 0,60% e 11,92% com desvios-padrão de 1,81% e 15,80%. Esse ganho também era esperado devido ao maior *W* (em média) das células providas pela metodologia proposta.

6 CONCLUSÕES E TRABALHOS FUTUROS

Nas últimas décadas o projeto baseado em portas lógicas complexas vem se mostrando uma eficiente alternativa para o desenvolvimento de circuitos integrados digitais. Nesse contexto, tal metodologia consegue prover soluções otimizadas para problemas específicos, alcançando melhores resultados se comparado às obtidas através do tradicional fluxo *standard cell*.

No entanto, ainda que se tenha um ganho expressivo em qualidade, a tarefa de projetar, dimensionar, validar e testar circuitos digitais através desse método é complexa, necessitando de um grande tempo de projeto, fator que acaba prejudicando a produtividade e inviabilizando o *design* sob demanda. Por conta disso, a aplicação dessa metodologia fica limitada ao desenvolvimento de células para bibliotecas *standard cell ou* focada em pequenas porções do sistema digital.

Esse trabalho propõe uma solução automática para o desenvolvimento de portas lógicas complexas sem restrições topológicas visando o projeto de leiautes com qualidade comparável a manual. Nesse contexto foram investigados os principais métodos e ferramentas disponíveis para o *design* digital sob demanda. Em relação à síntese de redes de transistores, etapa fundamental no desenvolvimento de portas complexas, dois paradigmas se destacam: o primeiro, amplamente utilizado na indústria, baseia-se na técnica de fatoração Booleana para o cômputo do arranjo lógico que implementa a função objetivo; o segundo, baseado em manipulações em estruturas de grafos, recentemente vem apresentando resultados promissores quanto ao número de chaves necessárias para computar as funções Booleanas de entrada. Por outro lado, tal paradigma promove a inserção de características topológicas singulares à porta (como o surgimento de estruturas não-planares e não-duais, por exemplo), o que acaba por influenciar diretamente nas soluções desenvolvidas (além de inviabilizar a síntese automática do leiaute em muitos casos).

Nesse cenário, o objetivo desse trabalho é propor uma solução versátil para o projeto de portas lógicas complexas a partir de um motor de geração lógica baseado em grafos. Assim, torna-se possível realizar comparações em nível físico dos resultados obtidos em contraste a soluções provenientes de minimizações baseadas na técnica de fatoração Booleana (empregadas no contexto do projeto de portas complexas até então).

A metodologia proposta foi denominada Libra e consiste em três estágios principais: o primeiro é responsável por gerar a rede lógica para uma dada função Booleana de entrada, etapa realizada pelo Kernel Finder (metodologia estado da arte para geração de redes de chaves otimizadas); a segunda etapa tem como objetivo dimensionar a célula de forma flexível e genérica, não especializando-a para um parâmetro específico pré-estabelecido; por fim, o terceiro estágio consiste na síntese de células provida pelo ASTRAN, uma ferramenta *open-source* estado da arte capaz de produzir leiautes de forma automática a partir de um *netlist* de entrada.

Para avaliação das soluções desenvolvidas foram realizados comparativos com células provenientes do método de Composição Funcional, metodologia estado da arte sob o paradigma de fatoração Booleana (amplamente adotado na indústria). Nos leiautes projetados foram avaliadas métricas de natureza geométrica (área, *wirelength*, número de contatos e densidade de roteamento) e elétrica (capacitância de entrada, potência de *leakage*, potência interna, potência de chaveamento, atraso de propagação e atraso de transição) considerando dois *benchmarks* bem conhecidos na literatura.

Para o primeiro *benchmark*, *53 NSP Handmade Networks*, observou-se um ganho expressivo nas células projetadas pela metodologia proposta em relação à área e atraso e uma pequena piora em relação à potência da célula. Referente ao segundo *benchmark*, *4-input P-Class*, observaram-se pequenas otimizações em área e atraso e uma ligeira piora nas métricas de potência, novamente.

Pode-se concluir que a metodologia proposta nesse trabalho alcançou resultados satisfatórios para os experimentos realizados, superando a tradicional abordagem baseada em fatoração Booleana na maior parte dos casos analisados.

Como trabalhos futuros, pretende-se:

- Propor métodos de dimensionamento que tenham maior precisão especialmente para topologias não-série-paralelo (evitando, assim, o superdimensionamento da porta);

- Criar perfis de dimensionamento para atender a especificações pré-determinadas, ou seja, realizar o projeto de células dedicadas para aplicações específicas (tal como para *low power*, por exemplo, uma característica importantíssima para circuitos digitais modernos);
- Investigar novos perfis de *folding* (diferentes alturas de circuito) que ofereçam um ponto de *trade-off* mais equilibrado entre os resultados de área, potência e atraso para as células desenvolvidas;
- Implementar e realizar um comparativo entre diferentes métodos de posicionamento para redes não-duais na ferramenta ASTRAN;
- Analisar o comportamento das soluções desenvolvidas integradas a um sistema digital complexo.

Referências

BAR-YEHUDA, R.; FELDMAN, J. A.; PINTER, R. Y.; WIMER, S. Depth First Search and Dynamic Programming Algorithms for Efficient CMOS Cell Generation. In: FIFTH MIT CONFERENCE ON ADVANCED RESEARCH IN VLSI, 1988, Cambridge, MA, USA. **Anais...** MIT Press, 1988. p.215–228.

BEECE, D. K.; XIONG, J.; VISWESWARIAH, C.; ZOLOTOV, V.; LIU, Y. Transistor sizing of custom high-performance digital circuits with parametric yield considerations. In: DAC, 2010. **Anais...** ACM, 2010. p.781–786.

BORAH, M.; OWENS, R. M.; IRWIN, M. J. Transistor Sizing for Minimizing Power Consumption of CMOS Circuits Under Delay Constraint. In: INTERNATIONAL SYMPOSIUM ON LOW POWER DESIGN, 1995, 1995, New York, NY, USA. **Anais...** ACM, 1995. p.167–172. (ISLPED ’95).

BOYD, S. P.; KIM, S. J. Geometric Programming for Circuit Optimization. In: INTERNATIONAL SYMPOSIUM ON PHYSICAL DESIGN, 2005, 2005, New York, NY, USA. **Anais...** ACM, 2005. p.44–46. (ISPD ’05).

BOYER, J. M.; MYRVOLD, W. J. On the Cutting Edge: Simplified O(n) Planarity by Edge Addition. **Journal of Graph Algorithms and Applications**, v.8, n.3, p.241–273, 2004.

BRAYTON, R. K.; SANGIOVANNI-VINCENTELLI, A. L.; MCMULLEN, C. T.; HACHTEL, G. D. **Logic Minimization Algorithms for VLSI Synthesis. Norwell**, MA, USA: Kluwer Academic Publishers, 1984.

BRAYTON, R. K.; Factoring logic functions. **IBM J. Res. Dev. 31**, p.187-198, 1987.

CORREIA, V. P.; REIS, A. I. Classifying n-Input Boolean Functions. **7th Workshop IBERCHIP**, p.58–66, Mar. 2001.

DETJENS, E.; GANNOT, G. Technology Mapping In Mis. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN, 1987. **Anais...** 1987.

DIJKSTRA, E. W. **A Note on Two Problems in Connexion with Graphs. Numer. Math**., Secaucus, NJ, USA, v.1, n.1, p.269–271, Dec. 1959.

DION, J.; MONIER, L. Contour: A tile-based gridless router. **Digital, Western Research Laboratory**, 1995.

DUECK, G.; SCHEUER, T. Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. **J. Comput. Phys**., San Diego, CA, USA, v.90, n.1, p.161–175, Aug. 1990.

FISHBURN, J.; DUNLOP, A. TILOS: A Posynomial Programming Approach to Transistor Sizing. **Int. Conference on Computer Aided Design**. Las Vegas, NV, USA, p.326–328, 1985.

GOLUMBIC, M. C.; MINTZ, A.; ROTICS, U. An improvement on the complexity of factoring read-once Boolean functions. **Discrete Applied Mathematics**, v.156, n.10, p.1633 – 1636, 2008.

GUPTA, A.; THE, S.-C.; HAYES, J. P. XPRESS: a cell layout generator with integratedtransistor folding. In: ED TC EUROPEAN DESIGN AND TEST CONFERENCE, 1996a. **Anais...** 1996a. p.393–400.

GUPTA, A.; HAYES, J. P. Width Minimization of Two-dimensional CMOS Cells Using Integer Programming. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1996, 1996, Washington, DC, USA. **Anais...** IEEE Computer Society, 1996b. p.660–667. (ICCAD ’96).

GUPTA, A.; HAYES, J. P. Optimal 2-D cell layout with integrated transistor folding. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN. DIGEST OF TECHNICAL PAPERS (IEEE CAT. NO.98CB36287), 1998, 1998. **Anais...** 1998. p.128–135.

GUPTA, A.; HAYES, J. P. CLIP: Integer-programming-based Optimal Layout Synthesis of 2D CMOS Cells. **ACM Trans. Des. Autom. Electron. Syst**., New York, NY, USA, v.5, n.3, p.510–547, July 2000.

GURUSWAMY, M.; MAZIASZ, R. L.; DULITZ, D.; RAMAN, S.; CHILUVURI, V.; FERNANDEZ, A.; JONES, L. G. CELLERITY: A Fully Automatic Layout Synthesis System for Standard Cell Libraries. In: ANNUAL DESIGN AUTOMATION CONFERENCE, 34, 1997, New York, NY, USA. **Anais...** ACM, 1997. p.327–332. (DAC ’97).

HART, P.; NILSSON, N.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE Transactions on Systems, Science, and Cybernetics**, v.SSC-4, n.2, p.100–107, 1968.

HSIEH, Y.-C. LiB: A cell layout generator. In: ACM, 1990. **Anais...** ACM, 1990.

IIZUKA, T.; IKEDA, M.; ASADA, K. High Speed Layout Synthesis for Minimum-Width CMOS Logic Cells via Boolean Satisfiability. Asia and South Pacific Design Automation Conference, Los Alamitos, CA, USA, v.0, p.149–154, 2004.

IIZUKA, T.; IKEDA, M.; ASADA, K. Exact Minimum-width Transistor Placement Without Dual Constraint for CMOS Cells. In: ACM GREAT LAKES SYMPOSIUM ON VLSI, 15, 2005, New York, NY, USA. **Anais...** ACM, 2005. p.74–77. (GLSVLSI ’05).

INTEL. **Intel Processors:** Quick Reference Guide. 2007. Disponível em http://www.intel.com/pressroom/kits/quickreffam.htm.

KAGARIS, D.; HANIOTAKIS, T. A Methodology for Transistor-Efficient Supergate Design. **Very Large Scale Integration (VLSI) Systems, IEEE Transactions on**, v.15, n.4, p.488–492, 2007.

KARMAZIN, R.; OTERO, C.; MANOHAR, R. cellTK: Automated Layout for Asynchronous Circuits with Nonstandard Cells. In: ASYNCHRONOUS CIRCUITS AND SYSTEMS (ASYNC), 2013 IEEE 19TH INTERNATIONAL SYMPOSIUM ON, 2013, **Anais...** 2013. p.58–66, 2013.

KARNAUGH, M. The map method for synthesis of combinational logic circuits. **Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics**, v.72, n.5, p.593–599, Nov 1953.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **SCIENCE**, v.220, n.4598, p.671–680, 1983.

KREJCIE, R. V.; MORGAN, D. W. Determining Sample Size for Research Activities. **Educational and Psychological Measurement**, v.30, n.3, p.607–610, 1970.

LAWLER, E. L. An Approach to Multilevel Boolean Minimization. **J. ACM**, New York, NY, USA, v.11, n.3, p.283–295, July 1964.

LEE, C. An Algorithm for Path Connections and Its Applications. **Electronic Computers, IRE Transactions on**, v.EC-10, n.3, p.346–365, Sept 1961.

LOGICS. **53 NSP Handmade Networks**. Logic Circuit Synthesis Labs, Universidade Federal do Rio Grande do Sul. Acesso em: 20 de novembro de 2016, DisponÍvel em http://www.inf.ufrgs.br/logics/docman/53 NSP Catalog.pdf.

MARTINS, M. G. A.; ROSA, L.; RASMUSSEN, A. B.; RIBAS, R. P.; REIS, A. I. Boolean factoring with multi-objective goals. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, 2010, 2010. **Anais...** 2010. p.229–234.

MARTINS, M. G. A.; RIBAS, R. P.; REIS, A. I. Functional composition: A new paradigm for performing logic synthesis. In: THIRTEENTH INTERNATIONAL SYMPOSIUM ON QUALITY ELECTRONIC DESIGN (ISQED), 2012. **Anais...** 2012. p.236– 242.

MAZIASZ, R. L.; HAYES, J. P. Exact Width and Height Minimization of CMOS Cells. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 28, 1991, New York, NY, USA. **Anais...** ACM, 1991. p.487–493. (DAC '91).

MCCLUSKEY, E. J. Minimization of Boolean functions. **The Bell System Technical Journal**, v.35, n.5, p.1417–1444, Nov. 1956.

MCGEER, P.; SANGHAVI, J.; BRAYTON, R.; VINCENTELLI, A. S.; ESPRESSO-SIGNATURE: A New Exact Minimizer for Logic Functions. 30th ACM/IEEE Design Automation Conference, Los Alamitos, CA, USA, **Anais...** v.00, p.618–624, 1993.

MCMURCHIE, L.; EBELING, C. PathFinder: A Negotiation-based Performance-driven Router for FPGAs. In: ACM THIRD INTERNATIONAL SYMPOSIUM ON FIELDPROGRAMMABLE GATE ARRAYS, 1995, 1995, New York, NY, USA. **Anais...** ACM, 1995. p.111–117. (FPGA '95).

MINTZ, A.; GOLUMBIC, M. C. Factoring Boolean functions using graph partitioning. **Discrete Applied Mathematics**, v.149, n.1–3, p.131 – 153, 2005. Boolean and Pseudo-Boolean FunctionsBoolean and Pseudo-Boolean Functions.

MOORE, G. E. Cramming more components onto integrated circuits. **Electronics**, v.38, n.8, April 1965.

MORAES, F.; REIS, A.; M. R. D. A.; REIS, R. Towards Optimal Use of CMOS Complex Gates in Automatic Layout Synthesis. In: CONGRESS OF THE BRAZILIAN MICROELECTRONICS SOCIETY, SBMICRO, 10, 1995. **Anais...** 1995. p.11–20.

MOSKEWICZ, M. W.; MADIGAN, C. F.; ZHAO, Y.; ZHANG, L.; MALIK, S. Chaff: Engineering an Efficient SAT Solver. In: ANNUAL DESIGN AUTOMATION CONFERENCE, 38, 2001, New York, NY, USA. **Anais...** ACM, 2001. p.530–535. (DAC '01).

POSSANI, V. N.; CALLEGARO, V.; REIS, A. I.; RIBAS, R. P.; SOUZA MARQUES, F. de; ROSA JR., L. S. da. Graph-Based Transistor Network Generation Method for Supergate Design. **IEEE Trans. VLSI Syst**., v.24, n.2, p.692–705, 2015.

POSSER, G. **Dimensionamento de portas lógicas usando programação geométrica**. 2011. Dissertação (Mestrado em Ciência da Computação) — UFRGS, Porto Alegre.

QUINE, W. A Way To Simplify Truth Functions. [S.l.]: American **Mathematical Monthly**, 1955.

REIS, R. Design automation of transistor networks, a new challenge. In: CIRCUITS AND SYSTEMS (ISCAS), 2011 IEEE INTERNATIONAL SYMPOSIUM ON, 2011. **Anais... 2011**, p.2485–2488, 2011.

RIEPE, M. A.; SAKALLAH, K. A. Transistor Placement for Noncomplementary Digital VLSI Cell Synthesis**. ACM Trans. Des. Autom. Electron. Syst**., New York, NY, USA, v.8, n.1, p.81–107, Jan. 2003.

ROSA JUNIOR, L. S.; MARQUES, F. S.; CARDOSO, T. M. G.; RIBAS, R. P.; SAPATNEKAR, S. S.; REIS, A. I. Fast Disjoint Transistor Networks from BDDs. In: ANNUAL SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 19, 2006, New York, NY, USA. **Anais...** ACM, 2006. p.137–142. (SBCCI ’06).

ROSA JUNIOR, L. S.; REIS, A. I.; RIBAS, R. P.; MARQUES, F. d. S.; SCHNEIDER, F. R. A Comparative Study of CMOS Gates with Minimum Transistor Stacks. In: ANNUAL CONFERENCE ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 20, 2007, New York, NY, USA. **Anais...** ACM, 2007. p.93–98. (SBCCI ’07).

ROSA JUNIOR, L. S. **Automatic Generation and Evaluation of Transistor Networks in Different Logic Styles**. 2008. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

RUDELL, R. Dynamic variable ordering for ordered binary decision diagrams. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN (ICCAD), 1993, 1993. **Anais...** 1993. p.42–47.

SERDAR, T.; SECHEN, C. Automatic datapath tile placement and routing. In: DESIGN, AUTOMATION AND TEST IN EUROPE. CONFERENCE AND EXHIBITION 2001, 2001. **Anais...** 2001. p.552–559.

SHANNON, C. E. **A Symbolic Analysis of Relay and Switching Circuits**. 1938.

SMANIOTTO, G. H.; MOREIRA, M. T.; ZIESEMER, A. M.; MARQUES, F. S.; ROSA, L. S. da. Toward better layout design in ASTRAN CAD tool by using an efficient transistor folding. In: IEEE 59TH INTERNATIONAL MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS (MWSCAS), 2016, 2016. **Anais...** 2016. p.1–4.

SUTHERLAND, I.; SPROULL, B.; HARRIS, D. **Logical Effort: Designing Fast CMOS Circuits**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

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

VICKER, K. Advanced Process Control in the Fab. In: SEMATECH ADVANCED EQUIPMENT AND PROCESS CONTROL WORKSHOP, 1997. **Anais...** 1997. p.338–51.

ZHU, J.; ABD-EL-BARR, M. On the optimization of MOS circuits. **IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications**, v.40, n.6, p.412–422, Jun 1993.

ZIESEMER, A. **Síntese Automática do Leiaute de Redes de Transistores**. 2014. Tese (Doutorado em Ciência da Computação) — UFRGS, Porto Alegre.

ZIESEMER, A. M.; REIS, R. Physical Design Automation of Transistor Networks. **Microelectron. Eng.**, Oxford, UK, UK, v.148, n.C, p.122–128, Dec. 2015.

Apêndices

Apêndice A – Conjunto de Funções dos *Benchmarks* Utilizados

Funções Booleanas do Benchmark 53 NSP Handmade Networks
F1 = a*b + a*c + a*d + b*c*d
F2 = a*b + a*c*e + d*e + d*b*c
F3 = e*a*b + e*a*c + e*a*d + e*b*c*d
F4 = e + a*b + a*c + a*d + b*c*d
F5 = a*e*b + a*e*c + d*a + d*b*c
F6 = a*b + a*d + a*c*e + b*d*c*e
F7 = a*b + a*c + d*e*a + d*e*b*c
F8 = a*b + a*c + a*d + b*c*d + b*e
F9 = a*b + a*c + a*d + a*e + b*c*d + b*c*e
F10 = a*f*b + a*f*c*e + d*b*c + d*e
F11 = a*b + a*c*e + f*b + f*c*e + d*b*c + d*e
F12 = c*f*a*e + c*f*b*d + a*b + d*e
F13 = c*a*e + c*b*d + f*a*e + f*b*d + a*b + d*e
F14 = e*f*a*b + e*f*a*c + e*f*a*d + e*f*b*c*d
F15 = e*a*b + e*a*c + e*a*d + e*b*c*d + f*a*b + f*a*c + f*a*d + f*b*c*d
F16 = e*f + e*a*b + e*a*c + e*a*d + e*b*c*d
F17 = e*f + a*b + a*c + a*d + b*c*d
F18 = e + f + a*b + a*c + a*d + b*c*d
F19 = f*a*e*b + f*a*e*c + f*d*a + f*d*b*c
F20 = f + a*e*b + a*e*c + d*a + d*b*c
F21 = f*c*e*a + f*c*e*b*d + f*a*b + f*a*d
F22 = f + c*e*a + c*e*b*d + a*b + a*d
F23 = a*e*f*b + a*e*f*c + d*a + d*b*c
F24 = d*a + d*b*c + a*b + a*c + e*b + f*b + f*c*a + e*c*a
F25 = a*e*b + a*e*c + a*f*b + a*f*c + d*a + d*b*c
F26 = d*b*c + a*b + a*c + d*a + e*f*b + e*f*c*a
F27 = c*e*f*a + c*e*f*b*d + a*b + a*d
F28 = c*a + c*b*d + e*a + e*b*d + f*a + f*b*d + a*b + a*d
F29 = c*e*a + c*e*b*d + c*f*a + c*f*b*d + a*b + a*d
F30 = c*a + c*b*d + e*f*a + e*f*b*d + a*b + a*d
F31 = a*e*b + a*e*c*f + d*a + d*b*c*f
F32 = d*b*c*f + a*b + d*a + a*c*f + e*b + e*c*f*a
F33 = a*e*b + a*e*c + a*e*f + d*a + d*b*c + d*b*f
F34 = a*b + a*c + a*f + e*b*a + e*c + e*f + d*a + d*b*c + d*b*f
F35 = a*e*b + a*e*c + d*f*a + d*f*b*c
F36 = a*b + a*c + e*b*a + e*c + d*f*a + d*f*b*c
F37 = a*e*b + a*e*c + d*a + d*b*c + f*a + f*b*c
F38 = a*b + a*c + e*b*a + e*c + d*a + d*b*c + f*a + f*b*c
F39 = a*e*b*f + a*e*c + d*a + d*b*c*f
F40 = a*b*f + a*c + e*b*f*a + e*c + d*a + d*b*c*f
F41 = a*e*b + a*e*f + a*e*c + d*a + d*c*b + d*c*f
F42 = a*f + a*b + e*f + d*a + e*c*a + d*c*f + d*c*b + a*c + e*b
F43 = a*e*b + a*e*c*f + d*a*f + d*b*c
F44 = a*b + a*c*f + e*c*f + d*a*f + d*b*c

F45 = a*e*b + a*e*c + a*e*c*f + d*a + d*f + d*b*c
F46 = a*d + a*b + a*c + b*c*d + b*e + d*f + c*e*f + a*c*f
F47 = a*b*e + a*c + d*f*a + d*f*c*b*e
F48 = a*b + a*e + a*c + d*f*a + d*f*c*b + d*f*c*e
F49 = a*b*e + a*c + d*a + d*c*b*e + f*a + f*c*b*e
F50 = a*b + a*e + a*c + d*a + d*c*b + d*c*e + f*a + f*c*b + f*c*e
F51 = a*b*e + a*c*f + d*a + d*b*c*e*f
F52 = a*b + a*e + a*c*f + d*a + d*c*f*b + d*c*f*e
F53 = a*b + a*e + a*c + a*f + d*a + d*c*b + d*c*e + d*f*b + d*f*e

Subconjunto de Funções Booleanas do Benchmark 4-input P-Class
F0136 = !a*!b*c + !a*b*!c + a*!b*!c*!d + !a*c*d
F0261 = !a*!d + !a*c + a*!b*!c*d
F0317 = !a*!b*!d + !a*!c*!d + !a*b*c*d + a*!b*!c
F0342 = !b*!c*d + !a*b*c + a*!b*!c
F0632 = !a*b*d + !a*b*c + !b*c*d
F0661 = !a*!b + !a*!c*!d + !a*c*d + !b*!c*!d + !b*c*d
F0748 = !a*!b*c + !a*b*!c + !a*d + !b*d
F0757 = !a*!b*!c*!d + !a*b*c + a*!b*d + !a*c*d
F0952 = !b*!c*d + !b*c*!d + b*!c*!d + !a*d + !a*c
F1035 = !a*!b + !a*d + !b*c*d + a*b*!c*!d
F1128 = !a*d + !a*b*c + a*!c*!d + !b*c*d
F1203 = d*!b + !d*!a + c*!a + !d*!c*b
F1262 = !b*d + !a*c + !a*b*!d + a*!c*!d
F1496 = !a*b*c + a*!c*d + !b*c*d
F1944 = !a*c + a*!b*!c + b*c*!d + !b*c*d + b*!c*d
F1956 = !c*d + c*!d + !a*b*d + a*!b*d
F2002 = !a*!b*d + !a*!b*c + a*b*c*d
F2555 = !b*!c + !b*!d + b*c*d + !a*b
F2584 = !a*!b*d + !a*!b*c + !a*b*!c*!d + c*d
F2921 = !a*!c*!d + !a*c*d + a*!b*!c*d + a*!b*c*!d + b*!c*!d + b*c*d
F3188 = c*d + a*!b*d + b*!c*!d + !a*b*!d
F3681 = !a*!b*!d + b*d + a*d + !a*c*!d + !b*c*!d
F3900 = c*d + !a*b*!d + a*d + b*c
F3945 = !a*!b*!c*!d + b*d + b*c + a*d + a*c
F3975 = !a*!b*!c*!d + c*d + b*d + b*c + a*d + a*c + a*b

Apêndice B – Dados dos Experimentos

Abaixo estão presentes os dados geométricos obtidos na comparação da solução proposta em relação à Composição Funcional para o *benchmark 53 NSP Handmade Networks*.

Célula	*Área (%)*	*Wirelength (%)*	*N. de Contatos (%)*	*D. de Roteamento (%)*
F1	7,49065955	23,38759	-3,44828	3,122295
F2	13,9373218	30,4343	10,34483	2,672981
F3	18,3486695	25,44773	10,81081	-2,26437
F4	23,0548091	22,34417	22,5	-3,17311
F5	6,96866089	19,6371	-3,33333	-2,51102
F6	19,5440258	26,1089	5,882353	-0,94696
F7	19,5440258	27,10319	11,11111	0,00606
F8	6,96866089	22,63949	-14,2857	4,979901
F9	19,5440258	21,78745	0	-4,47237
F10	23,0548091	25,70444	5,555556	-7,25113
F11	18,3486695	30,74205	8,823529	-0,57225
F12	35,3706998	65,88749	82,55814	11,5566
F13	39,312118	43,10502	34,09091	-5,6555
F14	21,7984135	24,96033	12,82051	-1,8143
F15	21,7984135	20,99814	8,108108	-5,92696
F16	16,3488101	24,20124	19,56522	3,527578
F17	11,5274046	24,85826	2,857143	2,771756
F18	16,3488101	17,10193	-5,71429	-6,50209
F19	6,11622318	19,74355	-11,4286	1,008995
F20	12,2324464	18,47336	-5,88235	-0,85107
F21	21,7984135	20,81142	5,263158	-3,67231
F22	17,2911069	24,15199	5,714286	0,101231
F23	17,2911069	22,63166	-12,5	-1,86459
F24	12,2324464	19,76082	2,857143	1,587937
F25	6,11622318	10,75013	-11,7647	-5,39773
F26	11,5274046	15,56152	0	-5,96967
F27	12,2324464	21,23909	0	-0,43876
F28	17,2911069	19,94418	11,42857	-1,79012
F29	12,2324464	18,7319	5,714286	-0,36482
F30	12,2324464	17,95723	5,555556	-0,23329
F31	45,5408673	51,58893	34,69388	-0,43504
F32	27,2480169	35,99565	28,57143	5,301844
F33	21,7984135	29,04322	13,15789	-1,30675
F34	17,2911069	21,04235	-3,125	-2,61658
F35	6,51467526	19,94436	-5,88235	-0,54714
F36	21,7984135	32,11909	12,82051	3,364634
F37	23,0548091	30,03707	17,14286	4,057565
F38	21,7984135	31,20926	10,81081	-0,47746

F39	43,3925746	52,58186	41,30435	1,386818
F40	25,8398477	32,46814	8,108108	5,117407
F41	21,7984135	36,81858	19,5122	7,12779
F42	17,2911069	24,06622	20,93023	-1,43098
F43	31,0078172	32,56851	21,05263	-4,35968
F44	20,6718781	27,54142	5,128205	-3,89264
F45	17,2911069	24,90096	17,94872	3,073526
F46	21,7984135	29,06876	22,5	5,171121
F47	31,0078172	35,93382	30	0,957006
F48	25,8398477	34,42255	19,04762	-0,4891
F49	16,3488101	34,51178	15	6,557441
F50	27,2480169	31,51482	17,64706	0,494346
F51	27,2480169	31,90253	10,52632	-1,16826
F52	31,0078172	37,16652	18,42105	1,504234
F53	25,8398477	30,1861	5,714286	0,440688
MÉD	20,12412966	27,7894	11,0233	-0,04729
DESV	8,902555742	9,739843787	15,88493565	3,816803086
MÁX	45,5408673	65,88749	82,55814	11,5566
MIN	6,11622318	10,75013	-14,2857	-7,25113

Abaixo estão presentes os dados elétricos obtidos na comparação da solução proposta em relação à Composição Funcional para o *benchmark 53 NSP Handmade Networks*.

Célula	***Cap. (%)***	***Leak. (%)***	***Int. (%)***	***Ch. (%)***	***Prop (%)***	***Tr (%)***
F1	15,75	-6,76	15,64	32,76	12,2	6,98
F2	33,13	16,72	8,64	31,86	15,44	15,5
F3	34,66	8,4	8,22	25,01	13,74	17,12
F4	30	5,29	-2,87	3,39	13,43	12,72
F5	36,84	13,69	7,75	23,92	12,23	17,68
F6	36,21	1,82	8,28	17,7	9,48	14,3
F7	33,6	-10,9	7,9	17,91	6,93	11,48
F8	32,43	19,28	6,25	12,09	7,89	11,64
F9	32,76	-16,16	6,33	8,53	9,99	11,12
F10	32,67	8,34	2,57	9,63	11,9	11,77
F11	32,79	26,47	-26,06	-7,02	12,48	12,29
F12	30	5,29	-2,87	3,39	13,43	12,72
F13	34,28	0,32	-15,52	-5,18	17,66	12,39
F14	34,25	5,21	-13,54	-6,51	17	12,06
F15	34,41	15,55	2,73	0,21	15,74	12,51
F16	33,91	31,43	-12,17	-2,3	12,8	12,02
F17	30	5,29	-2,87	3,39	13,43	12,72
F18	30,1	5,61	-2,77	3,29	13,46	12,64
F19	31,81	14,06	-6,58	0,06	13,33	11,68
F20	29,88	5,22	-3	3,47	13,39	12,56
F21	31,8	7,56	-4,24	0,8	13,4	11,6

F22	29,92	5,47	-2,96	3,54	13,43	12,61
F23	30,87	8,17	-3,94	0,6	13,35	12,13
F24	29,93	12,31	-3,57	-0,78	13,93	12,44
F25	28,14	6,97	-4,42	-0,85	13,6	12,64
F26	27,23	-14,25	-4,48	-0,69	13,46	12,57
F27	26,8	7,81	-4,45	-2,67	13,48	12,42
F28	30	5,29	-2,87	3,39	13,43	12,72
F29	26,63	-9,79	-4,98	-5,59	14,44	12,19
F30	26,65	-9	-5,22	-4,21	14,99	12,62
F31	27,26	13,42	-5,09	-4,65	14,95	13,06
F32	26,99	-9,23	-4,76	-3,38	14,84	12,94
F33	27,84	1,74	-4,85	-2,99	14,91	13,44
F34	30	5,29	-2,87	3,39	13,43	12,72
F35	27,3	9,72	-5,47	-2,67	14,36	13,33
F36	27,44	3,16	-5,47	-2,25	14,29	12,97
F37	27,27	-6,84	-5,16	-2,43	14,04	12,99
F38	27,61	-5,37	-4,55	-2,1	14,12	12,95
F39	30	5,29	-2,87	3,39	13,43	12,72
F40	26,98	3,36	-5,09	-2,89	14,05	12,95
F41	26,87	24,69	-9,25	-5,66	14,12	12,95
F42	26,5	0,56	-8,01	-1,44	9,11	8,36
F43	27,9	2,45	-5,95	0,49	10,84	10,1
F44	31,4	7,19	-0,81	5,32	15,17	14,47
F45	30	5,29	-2,87	3,39	13,43	12,72
F46	26,95	2,74	-10,11	-4,44	14,19	13,12
F47	24,4	-2,29	-11,1	-4,34	6,51	5,74
F48	32,1	8,13	0,22	6,29	16,03	15,34
F49	28,6	3,4	-4,93	1,46	11,7	10,98
F50	26,5	0,56	-8,01	-1,44	9,11	8,36
F51	29,3	4,34	-3,9	2,42	12,57	11,85
F52	31,4	7,19	-0,81	5,32	15,17	14,47
F53	25,1	-1,34	-10,07	-3,37	7,37	6,61
MÉD	29,68	4,87	-3,34	2,95	13,04	12,32
DESV	3,53517742	9,4516307	6,84810158	9,12636361	2,395541	2,141777625
MÁX	36,84	31,43	15,64	32,76	17,66	17,68
MIN	15,75	-16,16	-26,06	-7,02	6,51	5,74

Abaixo estão presentes os dados geométricos obtidos na comparação da solução proposta em relação à Composição Funcional para o *benchmark 4-input P-Class*.

Célula	*Área (%)*	*Wirelength (%)*	*N. de Contatos (%)*	*D. de Roteamento (%)*
F0136	-11,40221431	-7,95025881	-4	3,055907619
F0261	8,213566096	5,884779604	2,3255814	0,281037171
F0317	3,342191257	3,949636403	-1,9607843	-2,93466008
F0342	0	-10,7288749	-25,714286	0,44322456

F0632	10,89920675	-9,06482554	-15,384615	-5,0760263
F0661	0	0,079669981	0	0,147199236
F0748	-4,474281082	-5,38999517	-15,909091	-2,50015049
F0757	10,58202578	9,362168789	0	-1,30597751
F0952	-11,40388214	-8,90675966	-14,285714	2,365908593
F1035	0	5,087311962	-6,25	-6,34350922
F1128	3,795072278	-0,59684756	-2,0408163	-5,3840603
F1203	0	-3,95388337	10,6382979	0,408072564
F1262	0	1,671292414	-6,8181818	-1,2280413
F1496	-5,167969531	-2,33202466	10,2564103	-0,61489917
F1944	-12,75918723	-19,2077943	0	0,803662658
F1956	0	-3,39871108	0	1,119793765
F2002	14,05154905	-0,23902223	2,63157895	-1,79019197
F2555	0	6,345608199	-10,25641	1,600488218
F2584	-7,765527578	-8,08188634	-21,276596	-4,91161398
F2921	10,89920675	10,6394763	-15,625	-7,56702427
F3188	-4,106783048	-4,43329712	6,38297872	0,032297786
F3681	-0,977367545	0,104961963	-6,9767442	-3,9952619
F3900	14,74204424	10,21553357	5,55555556	0,996732702
F3945	0	-1,95288123	-21,95122	-1,79551922
F3975	6,623935537	5,191101864	11,3207547	1,874498797
MÉD	1,003663411	-1,10822084	-4,773532	-1,29272448
DESV	7,702203135	7,30372882	10,52380692	2,885412637
MÁX	14,74204424	10,6394763	11,3207547	3,055907619
MIN	-12,75918723	-19,2077943	-25,714286	-7,56702427

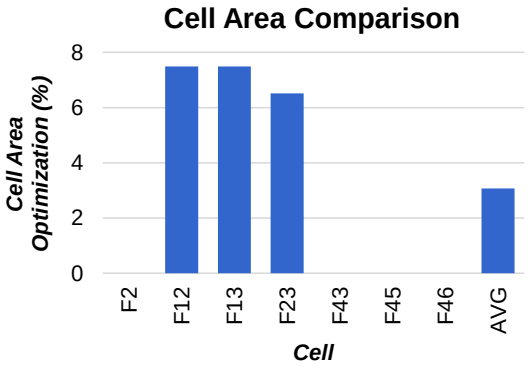
Abaixo estão presentes os dados elétricos obtidos na comparação da solução proposta em relação à Composição Funcional para o *benchmark 4-input P-Class*.

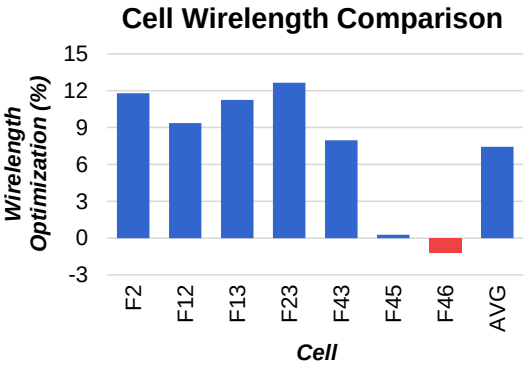
Célula	***Cap. (%)***	***Leak. (%)***	***Int. (%)***	***Ch. (%)***	***Prop (%)***	***Tr (%)***
F0136	-33,6772	-12,5085	-16,2264	-31,5375	-2,98151	-0,44063
F0261	-48,6013	-10,2239	-8,22908	4,268259	1,2613	-0,242
F0317	-1,59219	-47,5797	-23,9784	-21,4309	2,1604	23,35
F0342	-0,1299	-16,6353	14,09426	-27,4703	2,4845	34,134
F0632	-9,82017	-27,2755	-6,35174	23,87063	1,1833	20,907
F0661	-17,9098	-0,10963	1,396468	-6,25684	-1,117	17,99
F0748	7,228852	-6,36061	-1,51718	-44,2341	-1,014	4,0063
F0757	-13,7411	-7,29113	2,780758	7,434518	2,4637	25,926
F0952	-5,52883	-31,0517	-13,5957	-17,1822	0,7167	34,567
F1035	-1,83302	-6,68082	-24,8938	27,97513	-1,966	1,466
F1128	-3,51835	-11,6126	-3,78209	8,863333	1,488	19,82
F1203	-3,53014	-9,30107	-23,1547	-13,9168	-0,79	-0,539
F1262	-1,20757	7,146365	-24,3819	11,62135	1,7796	18,194
F1496	8,049409	-24,568	-37,8167	-28,0343	3,1622	21,218
F1944	-15,429	1,780326	-43,3728	8,294112	3,1948	22,325
F1956	-3,13485	7,859683	6,774317	-19,5385	-0,145	-14,6
F2002	-4,79134	8,310249	6,885895	-43,8647	0,9917	27,604

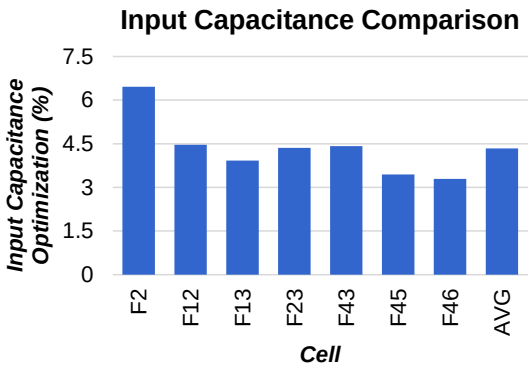
F2555	-25,0719	-7,86789	-13,1429	-0,66257	0,2962	-31,61
F2584	-0,35999	-14,5874	-8,57618	-38,8989	-0,447	0,6789
F2921	-6,78726	-0,7044	9,00365	7,903936	0,2333	4,7305
F3188	-7,14489	11,17932	-13,2388	-5,77085	1,8567	-4,377
F3681	-0,73237	7,944897	-9,80032	11,91428	0,4681	10,235
F3900	11,43064	-13,068	-8,92063	7,831635	3,2079	26,736
F3945	-6,77574	30,10678	37,68834	30,27259	-0,015	22,309
F3975	-26,0884	0,897839	-13,9199	2,796868	-3,354	13,546
MÉD	-8,4278563	-6,8880276	-8,6510212	-5,8300727	0,604755	11,9173628
DESV	13,4795511	15,828724	16,9956398	21,7758024	1,815057	15,80589537
MÁX	11,43064	30,10678	37,68834	30,27259	3,2079	34,567
MIN	-48,6013	-47,5797	-43,3728	-44,2341	-3,354	-31,61

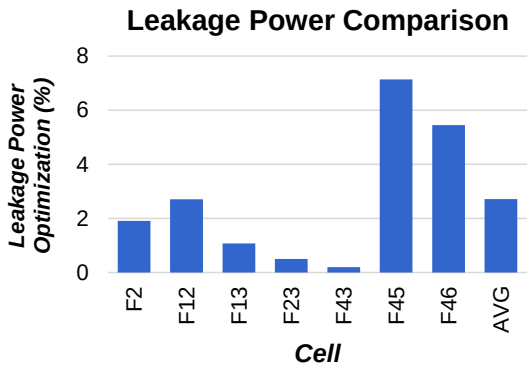
Apêndice C – Comparativo de Estratégias de Posicionamento para Redes Não-Série-Paralelo

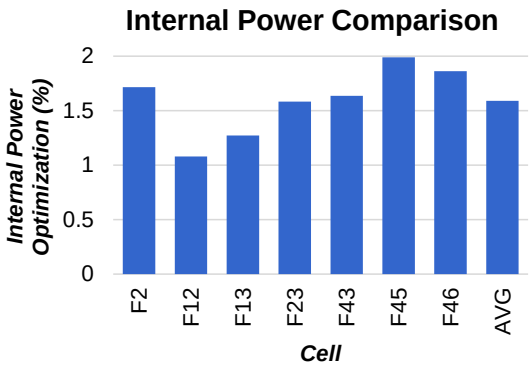
Abaixo estão os comparativos geométricos e elétricos da estratégia de posicionamento CPG em relação à CAA.

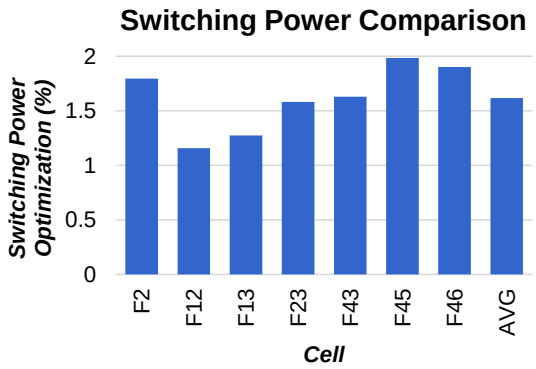


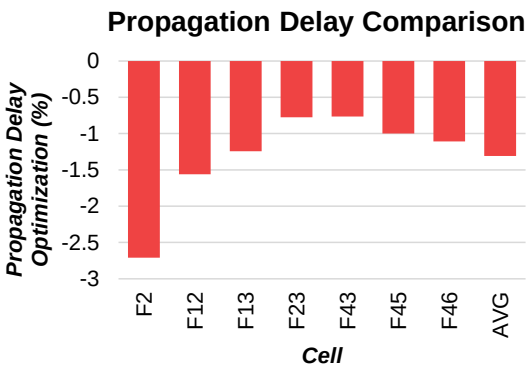


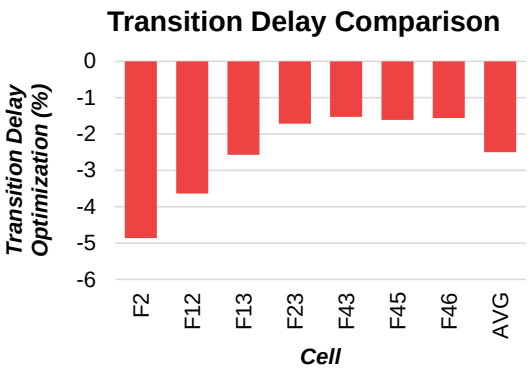












Anexos

Anexo A – Lista de Publicações

Artigos publicados:

CARDOSO, M. S.; ROSA JUNIOR, L. S.; MARQUES, F. S. Evaluating Non-Series-Parallel Cells Concerning Area and Wirelength Aspects. In: 30th South Symposium on Microelectronics, 2015, Santa Maria. **Proceedings 30th SIM**. 2015.

CARDOSO, M. S.; ZANANDREA, R.; SOUZA, R. S.; ROSA JUNIOR, L. S.; MARQUES, F. S.. Topological Aspects of Non-Series-Parallel Transistores Networks. In: 15th Microelectronics Students Forum, 2015, Salvador. **Proceedings 15th SForum. 2015**.

CARDOSO, M. S.; ROSA JUNIOR, L. S.; MARQUES, F. S.. Evaluating Geometric Aspects of Non-Series-Parallel Cells. In: 28th Symposium on Integrated Circuits and Systems Design, 2015, Salvador. **Proceedings of the 28th Symposium on Integrated Circuits and Systems Design**. New York: Association for Computing Machinery (ACM), 2015. p. 1 – 6.

CARDOSO, MAICON SCHNEIDER; ZANANDREA, REGIS; DE SOUZA, RENATO SOUZA; DA SILVA MACHADO, JOAO JUNIOR; DA ROSA, LEOMAR SOARES; DE SOUZA MARQUES, FELIPE. Topological characteristics of logic networks generated by a graph-based methodology. In: 2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS), Florianopolis. **Proceedings of the 2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS)**. p. 343 – 346.

CARDOSO, M. S.; SMANIOTTO, G. H.; ZANANDREA, R.; SOUZA, R. S.; ROSA JUNIOR, L. S.; MARQUES, F. S.. Physical Design of Supergate Cells Aiming Geometrical Optimizations. In: 2016 IEEE 59th International Midwest Symposium on Circuits and Systems, 2016, Abu Dhabi. **Proceedings of the 2016 IEEE 59th International Midwest Symposium on Circuits and Systems**. 2016.

Artigos aceitos para publicação:

SMANIOTTO, G. H.; ZANANDREA, R.; CARDOSO, M. S.; SOUZA, R. S.; MARQUES, F. S.; ROSA JUNIOR, L. S.. Post-Processing of Supergate Networks Aiming Cell Lay-out Optimization. In: 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore. **2017 IEEE International Symposium on Circuits and Systems (ISCAS)**.