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

DANIEL SILVA GUIMARÃES JÚNIOR

Inserção de Células Geradas Automaticamente em um Fluxo de Projeto *Standard Cell*

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

Prof. Dr. Ricardo Augusto da Luz Reis
Orientador

Porto Alegre, setembro de 2016.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Guimarães Jr, Daniel

Inserção de Células Geradas Automaticamente em um Fluxo de Projeto *Standard Cell* / Daniel Silva Guimarães Junior – Porto Alegre: Programa de Pós-Graduação em Computação da UFRGS, 2016.

73 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2016. Orientador: Reis, Ricardo Augusto da Luz.

1.Geração Automática de Células. 2.Redes Neurais Artificiais 3.Fluxo de Projeto. 4. Caracterização de Biblioteca de Células. I. Reis, Ricardo Augusto da Luz. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Vladimir Pinheiro do Nascimento
Diretor do Instituto de Informática: Prof. Luís da Cunha Lamb
Coordenador do PPGC: Prof. Luigi Carro
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

Agradecimentos

Agradecendo primeiramente aos meus pais Daniel e Ana, por todo amor, carinho, dedicação e empenho, me possibilitando alcançar todos meus anseios. Obrigado.

Agradeço à minha esposa Greyce por todo o seu amor, dedicação, companheirismo e apoio nesta caminhada acadêmica, da vida e por ter me dado o maior presente que poderia receber, nossa linda filha Melissa. Obrigado.

Aos meus irmãos Vinícius e Gabriel por sempre servirem de inspiração e referência em minhas ações. Obrigado.

Continuando, gostaria de agradecer a todas as pessoas que contribuíram para a minha formação acadêmica e que sempre se propuseram a ajudar para que eu chegasse aqui. Sou grato ao meu orientador Ricardo Reis pela oportunidade que me foi dada, por todas as suas contribuições técnicas ao longo deste trabalho e principalmente por não ter desistido de mim em nenhum momento durante esta longa caminha.

Também agradeço a todos os colegas de mestrado que me ajudaram com este trabalho. Aos colegas que me ajudaram a entender o mundo da microeletrônica Adriel Ziesemer, Guilherme Flach, Cristina Meinhardt, Glauco Valim e Felipe Pinto. Ao Gustavo Wilke por ceder sua experiência e com a ajuda que me deu em delimitar o escopo deste trabalho.

Em especial gostaria de agradecer às colegas Graciele Posser e Julia Puget, pela excepcional ajuda em realizar experimentos e testes necessários utilizando as ferramentas necessárias neste trabalho. Ainda à Julia por ser incansável na realização e estruturação de testes, coleta de resultados e ainda revisão de textos. Muito obrigado.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ASIC	*Application Specific Integrated Circuit*
BIST	*Built-In Self Test*
CAD	*Computer-Aided Design*
CDF	*Cumulative Distribution Function*
CI	Circuito Integrado
CMOS	*Complementary Metal-Oxide Semiconductor*
CUT	*Circuit Under Test*
EDA	*Electronic Design Automation*
ELC	*Encounter Library Characterizer*
FLEX	*Fast Lexical Analyzer*
ILP	*Integer Linear Programming*
LEF	*Library Exchange Format*
MLP	*Multi Layer Perceptron*
NLDM	*Non Linear Delay Model*
NMOS	*N-channel Metal Oxide Semiconductor Field – effect transistor*
NN	*Neural Network*
OTC	*Over the Cell*
QCL	*Quick Custommizable Logic*
RAM	*Random Access Memory*
RNA	Redes Neurais Artificiais
ROM	*Read-Only Memory*
RTL	*Register Transfer Level*
SOC	*System On Chip*
SDC	*Synopsys Design Constraints*
SLA	*Statistical Leakage Analysis*
SPICE	*Simulation Program with Integrated Circuit Emphasis*
TA	*Threshold Accept*
UFRGS	Universidade Federal do Rio Grande do Sul
ULG	*Universal Logic Gates*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho apresenta o desenvolvimento de um fluxo de projeto de circuitos digitais integrados, visando a incluir células geradas automaticamente pela ferramenta ASTRAN.

Como parte integrante deste novo fluxo, desenvolveu-se uma nova técnica de comparação entre células, utilizando Redes Neurais Artificiais, para a modelagem das células ASTRAN, esta técnica se mostrou flexível ao se adaptar a diversos tipos de células e com resultados robustos tendo 5% de desvio padrão e 4% para o erro relativo.

Também, foi criada uma ferramenta capaz de substituir células comerciais por células ASTRAN, tendo como objetivo melhorar as características de potência consumida e área utilizada pelo circuito, e por fim gerando um circuito misto composto de células comerciais feitas à mão e células ASTRAN geradas automaticamente.

O foco principal deste trabalho encontra-se na integração do fluxo de geração de células geradas automaticamente a um fluxo de síntese comercial de circuitos digitais.

Os resultados obtidos mostraram-se promissores, obtendo-se ganhos em redução de área e potência dos circuitos analisados. Em média os circuitos tiveram uma redução de 3,77% na potência consumida e 1,25% menos área utilizada. Com um acréscimo de 0,64% por parte do atraso total do circuito.

Palavras-Chave: Rede neural artificial, CAD, geração automática, fluxo de síntese, microeletrônica.

New Approach to the Design Flow of Digital Integrated Circuits

ABSTRACT

This work presents the development of a design flow for digital integrated circuits, including cells generated automatically by the ASTRAN tool.

Moreover, a new technique, using Artificial Neural Networks, was developed to perform a comparison between two different cells, i.e. commercial and ASTRAN's cell. This technique proved to be flexible when adapting to several types of cells and with robust results having 5% of standard deviation and 4% for relative error.

Also, a new tool was developed, capable of performing cell replacement between ASTRAN and commercial cells, to improve power consumption an used area. Finally a mixed circuit composed of handmade commercial cells and cells automatically generated by ASTRAN was generated.

A target was to mix an automatic cell synthesis tool with commercial synthesis tools dedicated to standard cells.

Comparisons have shown that our approach was able to produce satisfactory results related area and power consumption. In average the circuits had a reduction of 3.77% in the power consumed and 1.25% less used area. With an increase of 0.64% due to the total delay of the circuit.

Keywords: Artificial neural network, CAD, automatic generation, synthesis flow, microelectronics.

1 INTRODUÇÃO

A evolução dos sistemas computacionais embarcados é um exemplo do aumento significativo na complexidade dos *chips*. Tais sistemas acumulam cada vez mais funcionalidades, e, em contrapartida, o custo final de mercado tem diminuído significativamente em relação ao conhecimento contido em cada CI, necessitando assim de técnicas que visam à diluição do custo de desenvolvimento de um novo *chip*.

No decorrer dos anos 80, surgiu o método de projeto de CI, conhecido como *Standard Cells*, que utiliza células previamente projetadas com funções lógicas bem definidas. Os leiautes dos transistores que compõem uma célula de funções lógicas básicas (NOR, NAND, LATCH D,…), são previamente projetados, e assim, considerando os diferentes cenários de funcionamento, algumas versões de leiaute são desenhadas para uma mesma função lógica. Após o desenvolvimento de um conjunto de leiautes, estes são agrupados, de modo a formar uma *Standard Cells Library*.

Uma biblioteca de células lógicas consiste em reunir, em formato textual, metamodelos físicos dos leiautes de cada célula, possibilitando a obtenção de dados importantes, tais como potência consumida e o atraso de propagação de uma dada célula.

Ao ser projetado um circuito utilizando a abordagem *Standard Cell*, as células são posicionadas em bandas de alturas iguais e interconectadas por canais de roteamento (RABAEY, 1996), ou por roteamento sobre as células – *Over-The-Cell (OTC) Routing* (SARRAFZADEH, 1996).

Alinhado com os avanços da indústria de semicondutores, a metodologia *Standard Cell* é responsável por permitir altos níveis de escalabilidade na confecção de *chips* aos projetistas, desde CIs de simples funções lógicas até complexos dispositivos contendo milhões de transistores (*SoC*). Esse aumento crescente na integração de transistores em um único *chip* demanda muitos recursos no desenvolvimento dos projetos. Esses custos partem desde a aquisição de ferramentas de *EDA* até custos para a utilização de bibliotecas de células, sendo observado que essas são construídas manualmente por projetistas experientes, e assim elevando demasiadamente o seu custo.

1.1 Motivação

Observando limitações relativas ao custo de um projeto *Standard Cell*, podem-se desenvolver diferentes abordagens que tenham como objetivo a redução de custos em alguma das etapas de um projeto ASIC. Abaixo, na Figura 1.1, um exemplo de fluxo de um projeto *Standard Cell.*

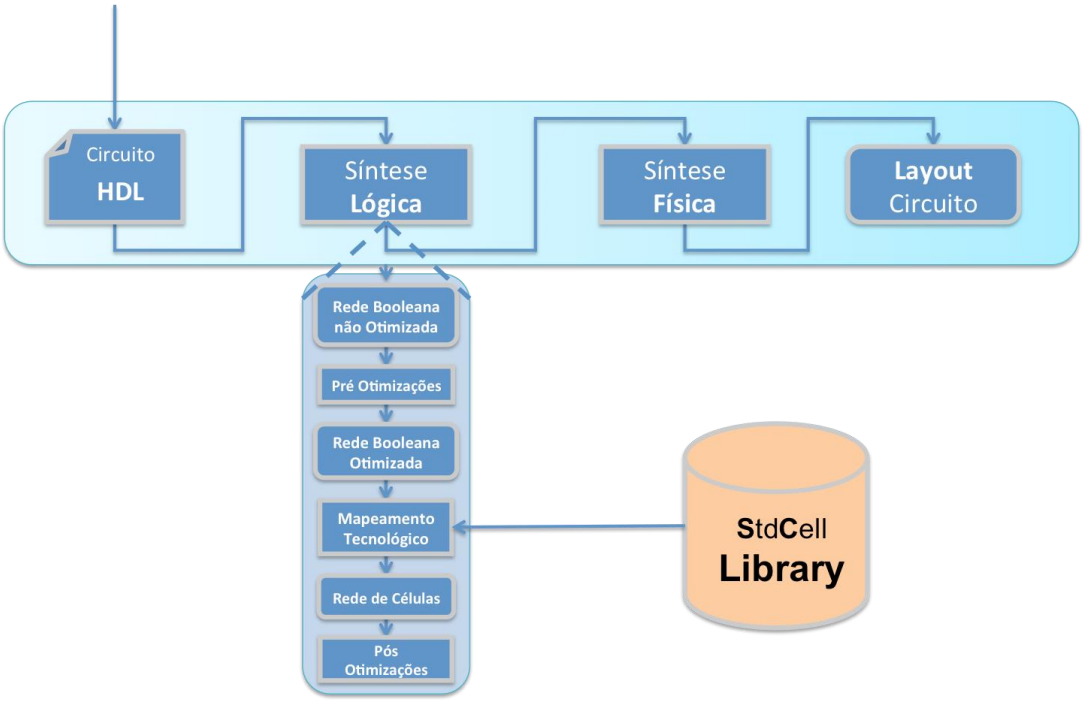


Figura 1.1: Fluxo de um projeto *Standard Cell*

Como pode ser observado é necessária, em um fluxo tradicional, a utilização de uma biblioteca de células. As vantagens de empregar uma biblioteca são devidas ao fato de todas as células contidas nela terem uma grande previsibilidade quanto ao seu comportamento em diferentes cenários. Por outro lado, é um item que não permite uma completa automação do processo, pois as células são projetadas manualmente.

O grupo de microeletrônica da UFRGS tem como uma das principais linhas de pesquisa o estudo de técnicas e ferramentas computacionais para o auxílio na automação da síntese de circuitos integrados. Neste segmento, está o ASTRAN (ZIESEMER, 2007), o qual é um exemplo de uma ferramenta sintetizadora de células. Isto conduz a um alto grau de automação na criação de células lógicas: o ASTRAN, a partir de um *netlist*, é capaz de gerar o leiaute das células de maneira eficiente, permitindo gerar qualquer função lógica ou rede de transistores (POSSER, 2010) e, consequentemente, de menor custo. Outra vantagem de um gerador automático de células lógicas é a maior facilidade no momento da migração das células para uma nova tecnologia. Além disso, há a possibilidade de gerar um número maior de células, cobrindo assim um amplo espaço de projeto antes não alcançado por bibliotecas manualmente desenvolvidas.

Este trabalho explora as possibilidades na alteração do fluxo de projeto ASIC que correspondam a uma maior agilidade para o reprojeto de circuitos digitais, alterando o uso de técnicas convencionais, como o emprego de bibliotecas de células manualmente desenvolvidas, além da inserção, no fluxo, de ferramentas de EDA, que adicionando etapas, possibilitam a obtenção de melhores resultados.

1.2 Objetivo

O objetivo desta pesquisa é apresentar um fluxo de projeto para circuitos digitais com uma primeira etapa baseada em *Standard Cells* e uma segunda etapa baseada no uso de células geradas automaticamente, além de propor a investigação sobre abordagens alternativas para a utilização da ferramenta geradora de células.

O presente trabalho tem, como foco principal, propor um fluxo de projeto inovador, onde células geradas automaticamente são inseridas em um circuito já projetado, de modo a fazer análises no circuito original, a fim de trocar células da biblioteca padrão (manufaturada), que são avaliadas como responsáveis por acarretar em demasiada área ou consumo de potência, por células geradas pelo ASTRAN. Neste intuito, foi desenvolvida uma ferramenta capaz de realizar tal avaliação e de substituir as células já mencionadas. Tal ferramenta utiliza técnicas de aprendizado de máquina (Redes Neurais Artificiais), além de algoritmos de meta-heurística *(Simulated Annealing)* aplicadas ao problema computacional de otimização relativo à troca de células.

Abaixo, na Figura 1.2, é apresentado o fluxo desenvolvido com o objetivo de acoplar a ferramenta substituidora ao fluxo de síntese padrão *Standard Cell*. Como pode ser observado, a ferramenta atua com base nos relatórios relativos ao atraso e potência obtidos a partir do *software* responsável pela síntese lógica – e mapeamento tecnológico a partir das células geradas pelo ASTRAN.

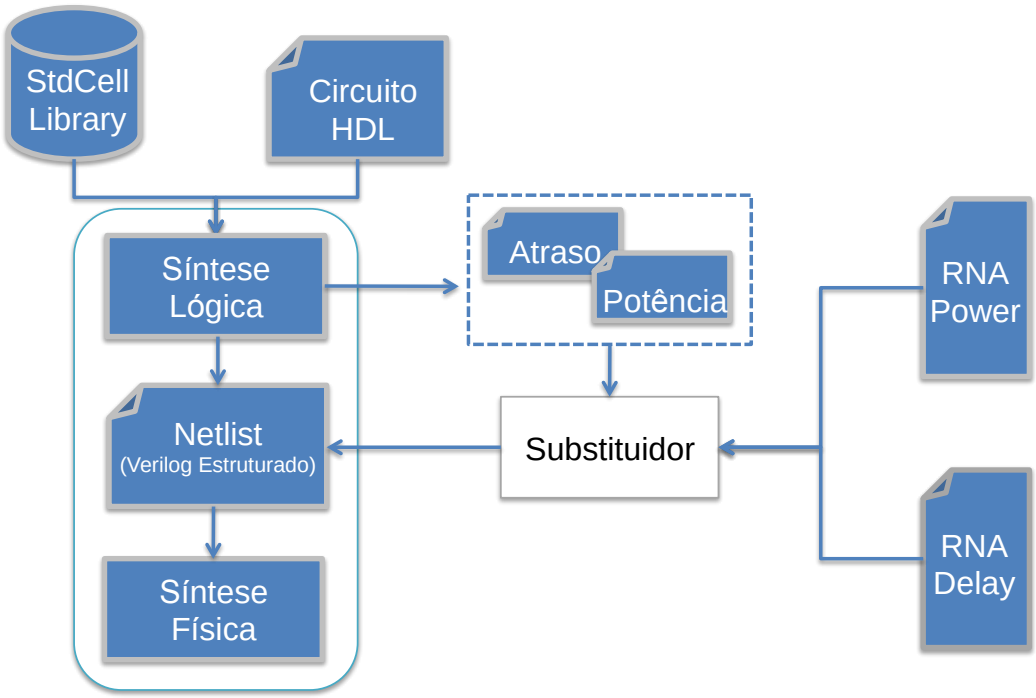


Figura 1.2: Fluxo Desenvolvido

1.3 Estrutura do Trabalho

Esta dissertação compreende 7 capítulos, onde cada um destes descreve, de maneira detalhada, as diversas partes do trabalho desenvolvido.

O segundo capítulo trata sobre técnicas para geração automática de células, em especial, a ferramenta ASTRAN, desenvolvida na UFRGS. Além dos conceitos envolvidos, também será abordado como a ferramenta sintetizadora de células está inserida no fluxo desenvolvido.

O Capítulo 3 apresenta conceitos sobre técnicas de aprendizado de máquina aplicadas a microeletrônica. Neste capítulo, é realizada uma revisão, principalmente, sobre RNAs (Redes Neurais Artificiais), além de explicitar o modo como foram concebidas as RNAs para *timing* e potência.

No Capítulo 4, há uma descrição sobre a ferramenta substituidora de células elaborada, mostrando conceitos e técnicas compreendidas na sua concepção e detalhamento quanto à inserção da ferramenta no novo fluxo.

O próximo capítulo, i.e. Capítulo 5, mostra o fluxo de projeto de circuitos digitais resultante. Informações relativas às ferramentas utilizadas também são apresentadas.

O Capítulo 6 apresenta os resultados obtidos com o novo fluxo, incluindo a ferramenta substituidora de células. Os circuitos são avaliados na síntese física, e são realizadas comparações entre área total do *chip*, consumo de potência e atraso.

O último, Capítulo 7, relata as conclusões sobre o trabalho e inclui considerações sobre possíveis trabalhos futuros.

2 GERAÇÃO AUTOMÁTICA DE REDE DE TRANSISTORES

A crescente evolução das tecnologias de fabricação de circuitos integrados demanda o desenvolvimento de novas ferramentas de EDA. Uma forma tradicional de projetar um circuito integrado em nível físico requer que o projetista, primeiramente, defina uma biblioteca de células lógicas e um modelo comportamental da funcionalidade de um circuito integrado. Essas bibliotecas de células oferecem um comportamento elétrico previsível devido à sua caracterização prévia (RABAEY, 1996). Ademais disso, diferentes versões para cada célula são requeridas, de forma que condições características como de atraso e consumo sejam atendidas, aumentando o número de células necessárias em uma biblioteca.

A geração automática de leiautes é uma alternativa cada vez mais importante para a síntese baseada em células. Esse método implementa transistores e conexões de acordo com padrões, que são definidos em algoritmos sem as limitações impostas pelo uso de uma biblioteca de células (LAZZARI, 2003, permitindo otimização dos circuitos sintetizados e objetivando a reduzir área, atraso e potência de um circuito integrado.

Normalmente, bibliotecas de células são feitas de forma manual, por projetistas experientes. Por esse processo ser extremamente demorado e suscetível a erros, pesquisas destinadas ao aperfeiçoamento de ferramentas responsáveis pela geração automática de leiaute são realizadas.

Outra vantagem das ferramentas responsáveis pela geração de leiaute é o continuo melhoramento, ou seja, a experiência de diversos usuários é preservada e compartilhada através de novas versões do *software*, de modo a colaborar com o amadurecimento das técnicas empregadas.

2.1 Geração Automática de Leiaute na UFRGS

Como citado anteriormente, uma das principais linhas de pesquisa do grupo de microeletrônica da Universidade Federal do Rio Grande do Sul é o desenvolvimento de ferramentas que atuem para alcançarmos um maior grau de automação do projeto físico de circuitos integrados.

Diversos trabalhos foram efetuados pelo grupo de microeletrônica. Faremos, a seguir, um resumo de algumas ferramentas e métodos desenvolvidos no decorrer dos anos.

Podemos tomar como exemplo trabalhos como de Meinhardt (MEINHARDT, 2006) e Menezes (MENEZES, 2004), onde é apresentado um resumo das teses e dissertações

do grupo de microeletrônica relacionadas à utilização de estruturas regulares como abordagem principal para o desenvolvimento de *chips*. Já Ziesemer (ZIESEMER, 2007) e Lazzari (LAZZARI, 2003) priorizam mostrar as ferramentas criadas na UFRGS, onde a geração automática de leiaute é o foco.

Esses trabalhos, abaixo descritos, têm como intenção mostrar diferentes abordagens desenvolvidas na UFRGS empregadas à microeletrônica. Deve-se levar em consideração que as soluções adotadas em cada ferramenta contêm uma forte relação com a época em que foram desenvolvidas, podendo assim atualmente não serem as soluções mais adequadas para os problemas discutidos.

Um dos passos iniciais no esforço para a automatização do processo de projeto de circuitos integrados vem com a metodologia TRANCA (*TRANsparent-Cell Approach*) (REIS, 1987). Essa, por sua vez, propõe uma série de procedimentos, visando a um melhor resultado em circuitos desenvolvidos com auxílio de ferramentas de CAD. Vale lembrar que, nesta época, o poder computacional era consideravelmente baixo em comparação com os dias atuais, sendo os modelos computacionais que descreviam o circuito, em muito, simplificados, ocasionando um decréscimo na qualidade dos circuitos gerados em comparação com as soluções manuais (*full custom*). Utilizando a análise do projeto de diversos microprocessadores da época como base, a metodologia TRANCA estipulou regras básicas para o desenvolvimento do leiaute, entre as quais, podemos destacar a utilização de uma estrutura de bandas, maleabilidade de blocos funcionais e células, transparência de células e blocos funcionais, além do gerenciamento de trilhas.

A primeira ferramenta desenvolvida segundo a metodologia TRANCA foi o sistema TRAMO (*TRAnca Module Generator*) (LUBASZEWSKI, 1990), que permite a geração automática de blocos de lógica aleatória a partir de células previamente desenhadas segundo a metodologia. O TRAMO é composto, fundamentalmente, por dois módulos: POTRANCA, o qual realiza o particionamento do circuito e também o posicionamento das células dentro das bandas, e o RETRANCA, responsável pelo roteamento e posicionamento final.

TRAGO (*TRAnca Automatic GeneratOr*) (MORAES, 1990), que foi desenvolvida logo a seguir, tem, com princípio, a síntese baseada na geração automática de leiaute. Um fato inovador na época do seu projeto deve-se ao roteamento, que é executado de maneira invertida, ou seja, sem nenhuma restrição, antes mesmo do posicionamento das células geradas.

A ferramenta MARCELA, desenvolvida por Güntzel (GUNTZEL, 1993), é um trabalho pioneiro na UFRGS quanto à utilização de estruturas regulares para a criação de circuitos integrados, ou seja, baseada no uso de matrizes pré-difundidas. Uma característica da ferramenta é a decomposição lógica dos circuitos a serem implementados, tornando mais flexível o posicionamento das células e assim permitindo uma distribuição mais equilibrada das conexões sobre a matriz.

Como evolução do TRAMO I, surgiu o TRAMO II (REIS, 1993). Uma das principais melhorias é a possibilidade de utilização de dois níveis de metal, que, consequentemente, demanda a adaptação dos algoritmos de roteamento e posicionamento das células.

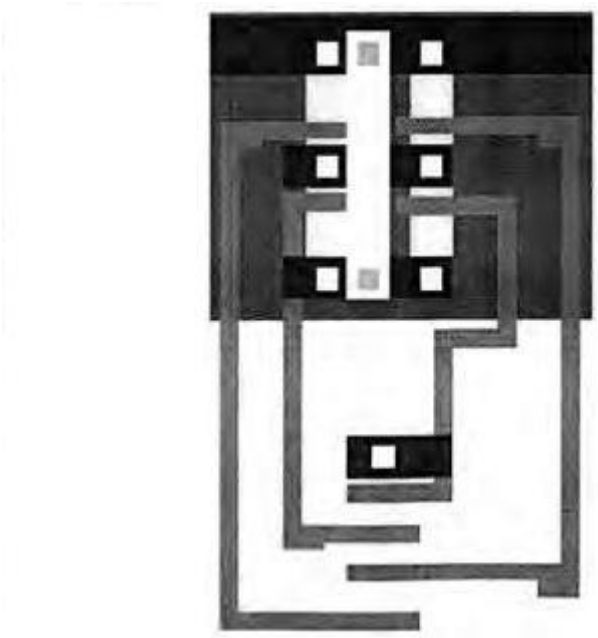


Figura 2.1: NAND4 gerado pelo *TRAMO II* (KINDEL, 1997)

O sistema TRAMO III (KINDEL, 1997), ao contrário do TRAMO II, não possui uma biblioteca de células; neste caso, as células são geradas automaticamente. Outra característica interessante do TRAMO III é o posicionamento das células: ao invés de serem dispostas em bandas, são colocadas na vertical (segundo Kindel, para aproximar-se dos resultados de projetos manufaturados).

O TROPIC (MORAES, 1994) tem, como base, o TRAGO. De fato, é um gerador de estruturas regulares - macro células, e tem, como foco principal, a geração de qualquer rede de transistores.

Figura 2.2: Leiaute Gerado pelo *TROPIC* (KINDEL, 1997)

Outro trabalho relevante é o MARAGATA (LIMA, 1999), que aplica a metodologia *Quick Custommizable Logic* (QCL), isto é, tem *Universal Logic Gates* (ULGs) como elemento básico; É usado para compor uma matriz programável pelo último nível de metal.

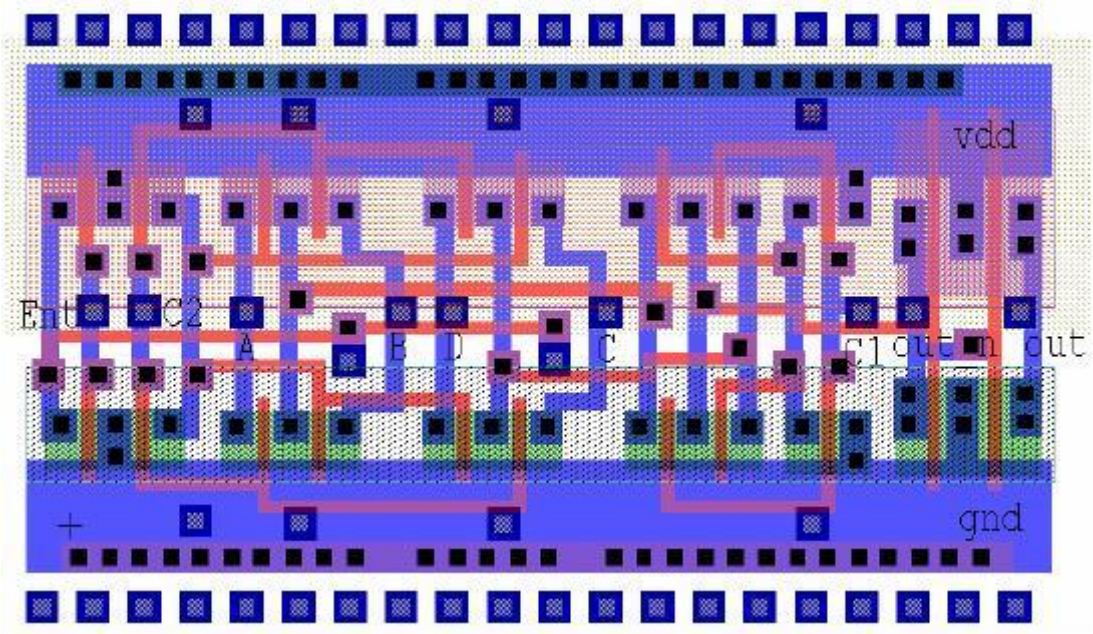


Figura 2.3: Leiaute da ULG3 utilizada no MARAGATA (LIMA, 1999)

O PARROT PUNCH (LAZZARI, 2003) não utiliza uma biblioteca de células, de modo que gera os leiautes das células apenas no momento da síntese. O sintetizador não restringe a geração a qualquer função lógica desejada, isto é, na prática, pode-se gerar o circuito todo desejado como uma única célula. A não-hierarquização do circuito remete em perda de desempenho, pelo tempo necessário para gerar circuitos complexos.

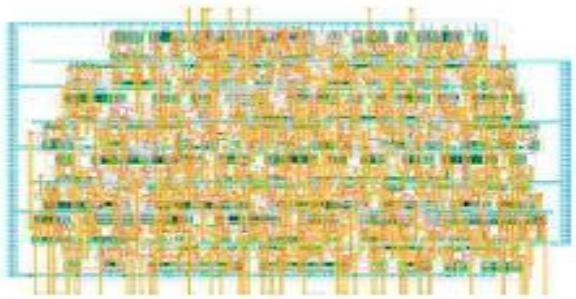


Figura 2.4: Leiaute Gerado pelo PARROT PUNCH (LAZZARI, 2003)

Nas ferramentas MARTELO (MENEZES, 2004) e R-CAT (MEINHARDT, 2006), apresenta-se uma abordagem de geração automática de matrizes de portas lógicas, com a intenção de obter aumento de previsibilidade do comportamento dos circuitos após a sua implementação.

O MARTELO utiliza portas NAND e inversores como base da sua matriz; já ao R-CAT, acrescenta-se a função lógica NOR, além de serem fornecidas múltiplas opções de posição para os pinos de entrada e saída das células, após se observar a influência desse fator no leiaute gerado e no roteamento do circuito.

2.2 ASTRAN – Gerador Automático de Redes de Transistores

Com o intuito de automatizar o processo de desenvolvimento do leiaute de células apresentado em Ziesemer (2007), foi projetada e implementada uma ferramenta capaz de

gerar leiautes 1-D, com capacidade de suportar células com diferentes redes de transistores e respeitando regras de projeto da tecnologia especificada.

O estilo de leiaute utilizado é baseado, fundamentalmente, no estilo 1D proposto em Uehara, (1981). Este modelo de leiaute visa, além do suporte a circuitos com diferentes redes de transistores individualmente dimensionados, também à construção contendo o poço dos transistores posicionados e dimensionados, de forma a permitir o espelhamento das células entre bandas pares e ímpares.

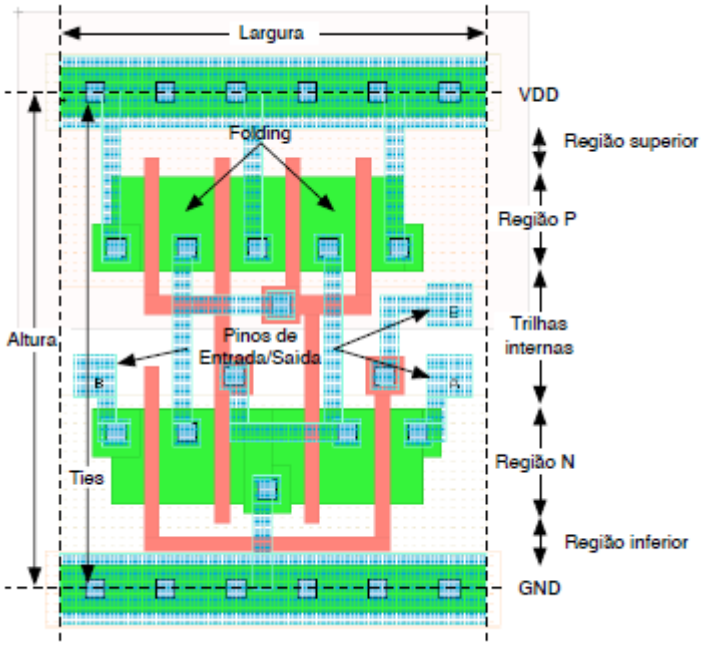


Figura 2.5: Estilo de leiaute 1-D

O roteamento interno das células é feito exclusivamente com polissilício, metal 1 e difusão, essa última para conexões de transistores adjacentes que possuam o mesmo sinal nos extremos. As linhas de alimentação estão localizadas nos limites inferiores e superiores das células em metal 1, contendo conexão com as células vizinhas por justaposição. Trilhas centrais constam para roteamento vertical e horizontal na região entre as difusões P e N, utilizando polissilício e metal 1; já as trilhas adicionais são utilizadas para roteamento sobre os transistores P/N utilizando metal 1.

Ziesemer (2007), em seu trabalho, utilizou padrões amplamente difundidos na indústria de microeletrônica, por exemplo, a especificação da estrutura interna das células é feita no formato Spice. Esse formato possui, para cada célula informada, a lista de seus transistores, o comprimento e a largura dos *gates*, as redes às quais pertencem e seus pinos de entrada/saída. A ferramenta recebe esse arquivo como entrada e gera o leiaute de cada uma das células, de acordo com outro arquivo de entrada, que define as regras da tecnologia especificada, e de um *template*, que define os parâmetros de geração.

O leiaute resultante é salvo em formato CIF (*Caltech Intermediate Format*) e LEF (*Library Exchange Format*). Esses dois formatos são também difundidos entre as principais ferramentas comerciais de EDA. Nos capítulos seguintes deste trabalho, será apontada a integração do sintetizador de células ASTRAN com o resto do fluxo elaborado, exemplificando a vantagem da utilização de padrões difundidos na indústria.

A geração automática das células segue um padrão que lhes permite a constituição de uma biblioteca e de, posteriormente, serem integradas a um fluxo de geração com posicionamento e roteamento automáticos, de forma similar ao fluxo para *standard cells*.

Na Figura 2.6 abaixo, é exibido o fluxo de projeto pertencente à ferramenta construída.

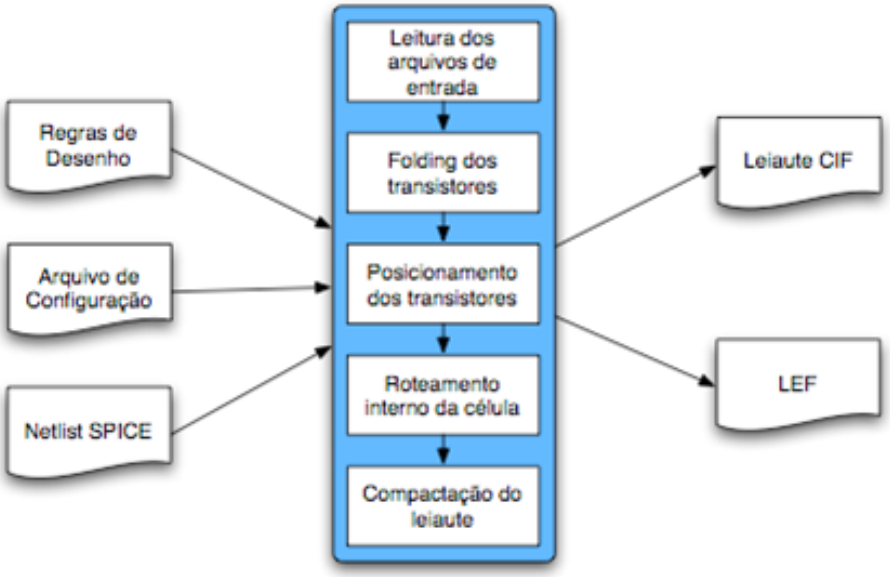


Figura 2.6: Fluxo do gerador de células ASTRAN (ZIESEMER, 2007)

Técnicas distintas foram empregadas na concepção do sintetizador ASTRAN, com o intuito de reduzir o dimensionamento dos transistores gerados (objetivo visado constantemente), a técnica de *folding* foi implementada, por ser para a produção de circuitos de alto desempenho, reduzindo o atraso e o consumo de energia (ZIESEMER, 2007). Essa técnica consiste em trocar a altura do transistor por largura, ou seja, para um transistor projetado que ultrapasse a altura da banda no projeto baseado em células, esse é dividido em dois outros transistores menores, então conectados paralelamente, mantendo a função lógica, diminuindo a altura e tendo um pequeno acréscimo em largura.

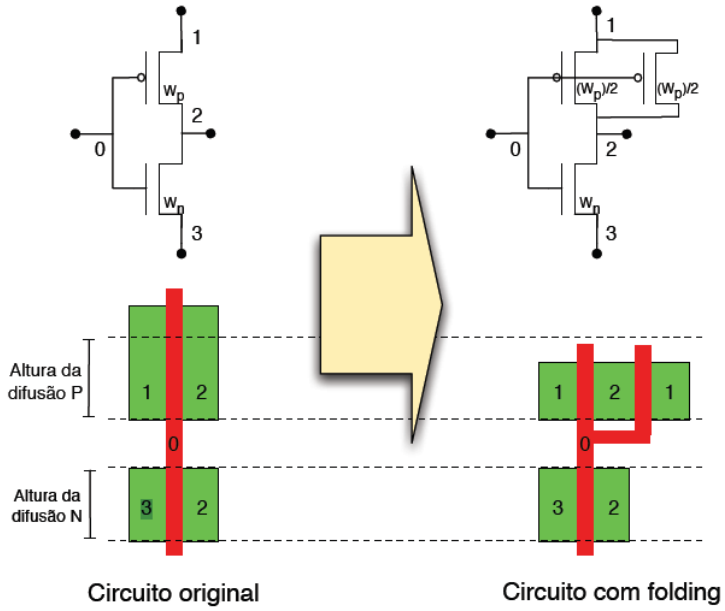


Figura 2.7: Aplicação do método de *Folding* (ZIESEMER, 2007)

Das metodologias para implementação da técnica de *folding*, a escolhida por Ziesemer (2007) é posicionamento dinâmico com *folding* estático, que, dados os limites para o *folding*, fragmenta os transistores e determina sua posição e orientação, tal que se minimize a área da célula. Um exemplo de aplicação pode ser visto na Figura 2.7.

Quanto ao posicionamento, o objetivo geral é de encontrar um arranjo de transistores de maneira a minimizar a área, o número de quebras de difusões e melhorar as características elétricas. O ASTRAN possui dois tipos de posicionadores, sendo o primeiro baseado no algoritmo de caminho de Euler, já o segundo utiliza um método heurístico que faz uso de *Threshold Accept* (DUECK; SCHEUER, 1990).

O posicionamento de transistores baseado no caminho de Euler é um método simples e rápido para encontrar uma organização de transistores com o objetivo de reduzir o número de GAPs no circuito. Além das vantagens visíveis na aplicação do algoritmo do caminho de Euler, esse apresenta algumas limitações na sua utilização; no caso para circuitos mais complexos, um único caminho de Euler pode não ser encontrado. Outra desvantagem é que apenas *netlists* de células com o mesmo número de transistores nas redes P e N –característica de circuitos CMOS estático complementar- podem ser posicionadas. A restrição definitiva para o uso do algoritmo de Euler, segundo Ziesemer (2007), é sua incompatibilidade juntamente à técnica de *folding*, causando o desbalanceamento das redes de transistores.

Devido às restrições na utilização do algoritmo do caminho de Euler, uma nova proposta de posicionador foi desenvolvida, baseado na heurística *Threshold Accept*. Basicamente, o funcionamento dessa técnica consiste em partir de uma solução inicial qualquer, com sucessivas injeções de perturbações, com o objetivo de encontrar soluções melhores, ou seja, com um menor custo. O *threshold* indica quando soluções piores devem ser aceitas, e esse limiar tem um decréscimo ao decorrer do processamento; no momento que este for igual a zero, apenas soluções melhores serão aceitas.

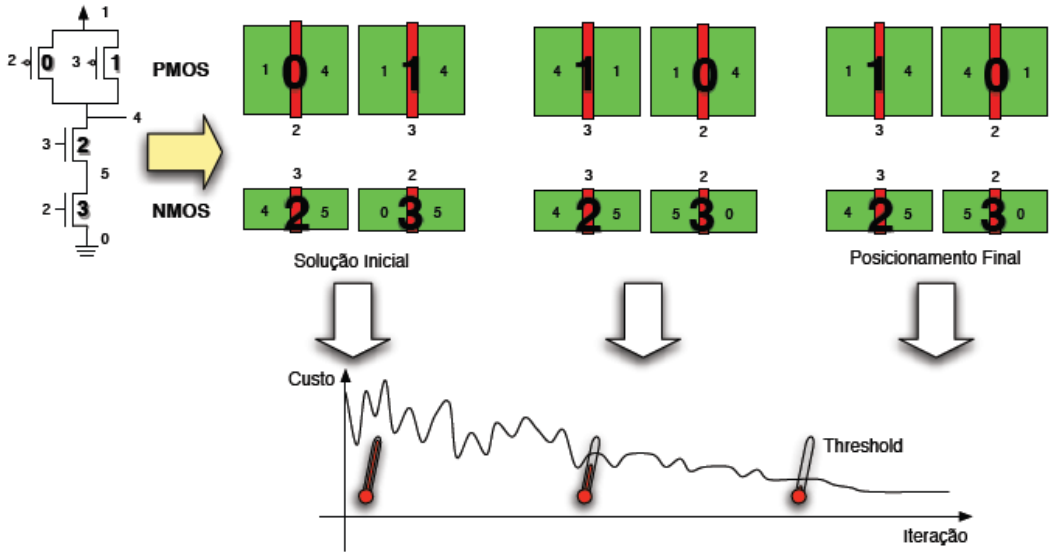


Figura 2.8: Exemplo de funcionamento do *Threshold Accept* (ZIESEMER, 2007)

Um aspecto importante para o funcionamento da heurística TA é a função de custo, esta, por sua vez, retorna uma medida qualitativa da solução ao longo das iterações do algoritmo. Para determinar a qualidade de um circuito, foram utilizadas quatro métricas distintas: largura da célula, número de GAPs, *Gate Mismatches* e o comprimento das conexões. Por algumas dessas métricas terem uma maior contribuição em relação a outras, a fórmula final para a função de custo possui pesos diferentes para cada elemento listado acima. Os pesos para cada métrica foram experimentalmente determinados.

Para convergir em soluções melhores, ou seja, com menor custo, é imperativo que seja usada uma função de perturbação, que tem, como objetivo, expandir o espaço de soluções e selecionar as melhores ao decorrer das iterações do algoritmo. O posicionador do ASTRAN, utilizando *Threshold Accept*, utiliza em sua função de perturbação, basicamente, dois movimentos: o primeiro consiste em deslocar os transistores em relação ao início ou final da lista de transistores; já o segundo movimento é o espelhamento, cujos transistores têm sua ordem e orientação invertidas.

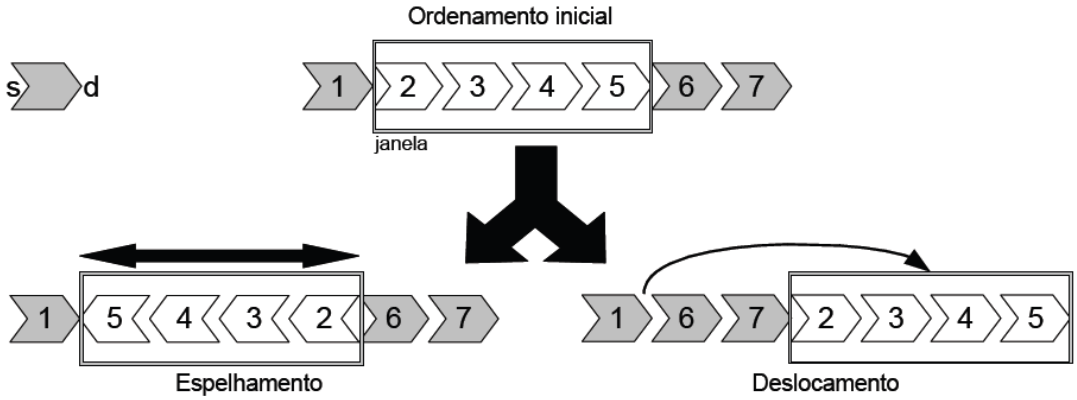


Figura 2.9: Movimentos da função de perturbação do ASTRAN. (ZIESEMER, 2007)

O roteamento das células sintetizadas pelo ASTRAN é realizado logo após o posicionamento dos transistores. O modelo computacional da célula utilizado para o

algoritmo de roteamento é uma estrutura de grafos, onde os nós representam os terminais dos transistores, e as arestas indicam os nós a serem conectados. Essa modelagem realizada por Ziesemer (2007) trata casos específicos na conjuntura dos nós se encontrarem em camadas diferentes, ou tratamentos especiais para criação de trilhas de alimentação em metal 1 e inserção pinos de entrada/saída, caso um nó em metal 1 esteja na região de trilhas (ZIESEMER, 2007). Outro aspecto importante é o uso de pesos diferentes para as arestas, a fim de priorizar o uso de conexões que possuam melhores características elétricas ou despendam menos área.

À necessidade de resolver os possíveis congestionamentos de redes roteadas, um algoritmo com base na negociação dessas concorrências faz-se necessário; nesse caso específico (ASTRAN), o algoritmo *PathFinder* (MCMURCHIE; EBELING, 1995) foi escolhido. A heurística utilizada leva em consideração o custo da aresta que é caminho para o nó de interesse, juntamente com um histórico de congestionamento para esse nó, além de um componente responsável por indicar o quanto tal nó já é utilizado. No fim, essa etapa é capaz de gerar redes roteadas sem conflito algum.

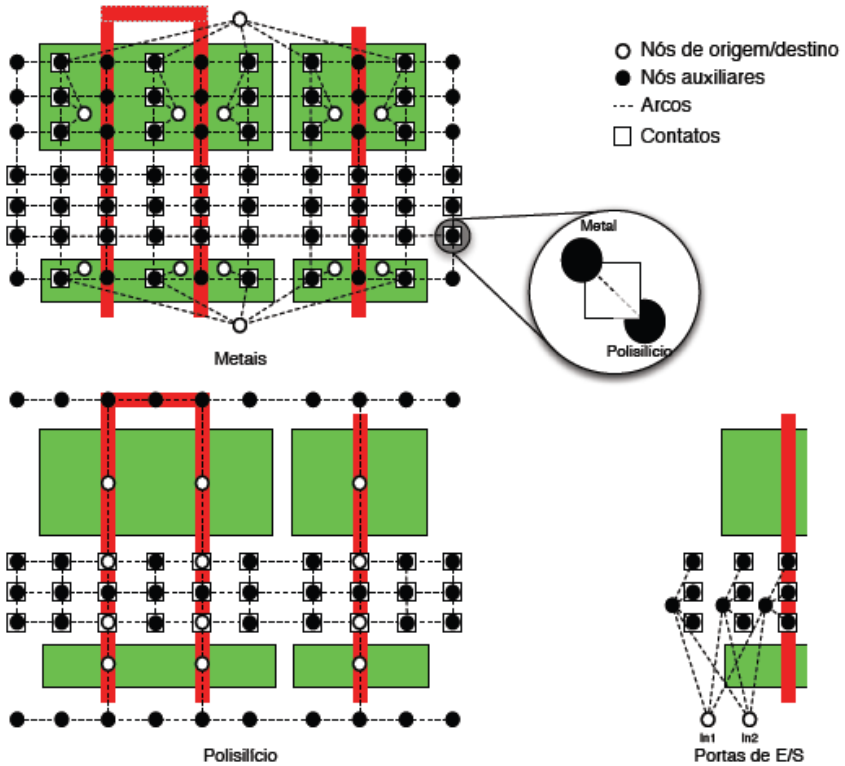


Figura 2.10: Exemplo da modelagem para o roteamento. (ZIESEMER, 2007)

Para a finalização do leiaute do circuito, a compactação é a última etapa. Nesse passo, os polígonos que compõem o desenho da célula são instanciados, de acordo com o resultado obtido nas etapas anteriores. Outro aspecto importante na compactação é a adição das restrições das regras de projeto, estabelecendo assim um circuito dentro dos padrões de uma determinada tecnologia.

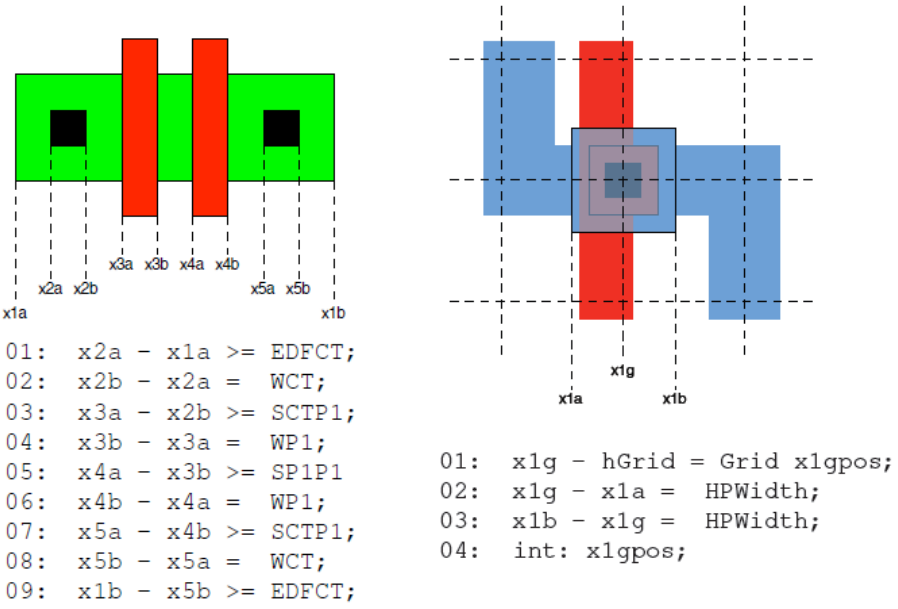


Figura 2.11: Exemplo de modelagem para o ILP. (ZIESEMER, 2007)

A compactação é realizada a partir do emprego da técnica de programação linear com inteiros (ILP), ou seja, as posições relativas entre os elementos que compõem a célula e as constantes que definem as regras de projeto são modelados de forma a compor um conjunto de equações lineares - Figura 13, assim, permitindo o uso de ILP. Para utilizar essa técnica, é necessário o uso de um resolvedor, que, nesse caso, é o LPSolve (LPSOLVE, 2010).

Igualmente às etapas anteriores, a compactação trabalha objetivando à minimização de algumas variáveis, tais como: largura das células, já que, em uma biblioteca de células, todas tem a mesma altura; redução das difusões, para melhorar as características elétricas e definição da prioridade de roteamento de redes em polissilício ou metal, para menor atraso. Com a finalidade priorizar algumas destas metas, são utilizados pesos nas variáveis de minimização.

Por fim, é realizada a etapa de pós-otimização, no decorrer do processo de geração da célula. Muita informação redundante é armazenada à estrutura de dados do leiaute do circuito. De fato, é relevante remover tais redundâncias, assim, um algoritmo foi desenvolvido para unir retângulos pertencentes a uma mesma camada e que estejam sobrepostos ou justapostos. A execução desse algoritmo permite a redução de linhas no arquivo de saída.

Este trabalho utiliza as células do ASTRAN que são desenvolvidas para a tecnologia de 350*nm*. Atualmente encontra-se uma nova versão do sintetizador de células para tecnologias recentes, como a 65*nm.* Mesmo que a tecnologia de 350*nm* não pertença mais ao estado da arte, utilizar um gerador de células lógicas próprio apresenta diversas vantagens: além de favorecer o amadurecimento da ferramenta sintetizadora para versões futuras, o fluxo proposto não se mantém dependente de produtos externos para a geração de células, com a adição de colaborar, dessa maneira, para um conjunto mais completo de EDAs desenvolvidos pelo grupo de microeletrônica da UFRGS.

A ferramenta ASTRAN continua em constante evolução, com o aprimoramento de técnicas utilizadas na geração das células sintetizadas, bem como tecnologias de

fabricação mais modernas, que envolvem maiores restrições de modo a incrementar a complexidade da ferramenta no processo de criação de células.

Um novo caso de uso das células ASTRAN é descrito em Ziesemer (ZIESEMER, 2014), visando evidenciar a qualidade das células geradas, um conjunto de células assíncronas foram geradas pelo ASTRAN. Este trabalho utiliza como base comparativa a biblioteca ASCEnD – uma biblioteca de células assíncronas desenhadas manualmente. Foram comparadas seis diferentes características: área total da célula, capacitância parasita, pior caso de capacitância de entrada, atraso médio, média de energia por transição (EPT) e *leakage* médio.

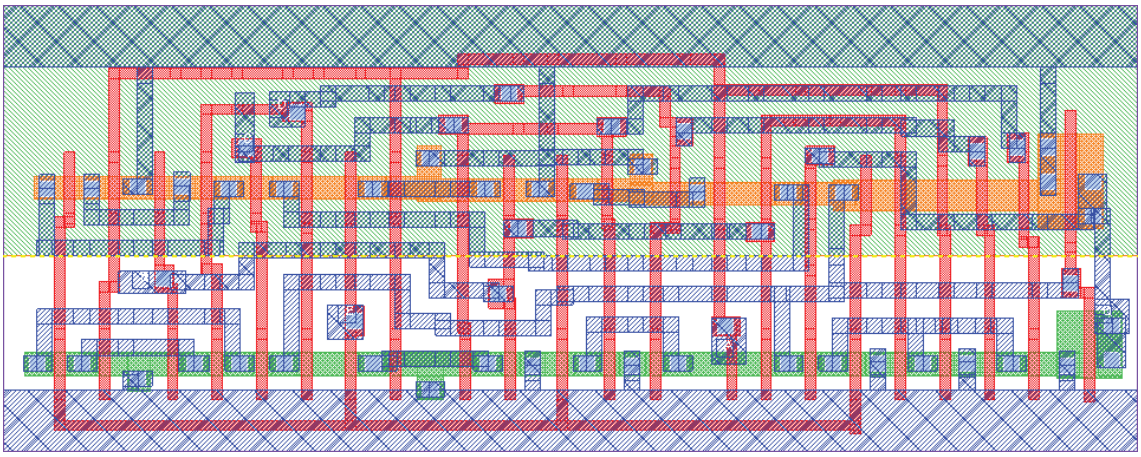


Figura 2.12 Layout da célula assíncrona NCL35X4, gerada pelo ASTRAN. (ZIESEMER, 2014)

No geral, este novo caso de uso do sintetizador ASTRAN resultou em células de 7% a 45% menores. Para capacitâncias parasitas e capacitâncias de entrada obteve melhoras significativas em torno de 12% a 60%. Com melhora substancial no atraso médio das células geradas, chegando a 46% no caso da célula NCL35X4. Já o EPT melhorou no mínimo 2% até 46%. Finalmente, o *leakage* médio se manteve ligeiramente parecido ao das células da biblioteca ASCEnD.

Outro trabalho relevante na evolução do ASTRAN, apresenta uma nova técnica de compactação para as células geradas (ZIESEMER; REIS, 2014). Nesta nova abordagem é utilizado o *Mixed-Integer Linear Programming* (MILP) para efetuar a compactação 2-D de leiaute, utilizando-se de variáveis binárias, assim não sendo necessário utilizar heurísticas classificadas como problemas NP-Hard, e em um tempo de execução aceitável.

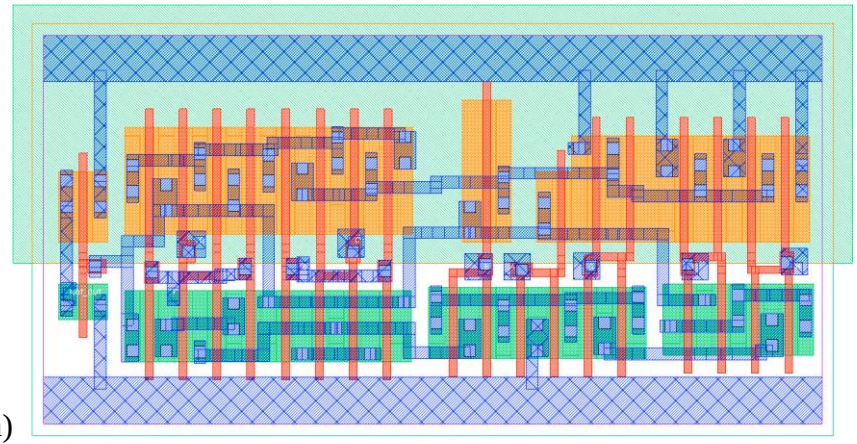
No trabalho citado, foram obtidos resultados interessantes, visto que na média a altura das células aumentou em 3,6%, sendo que em 71% das células analisadas a altura se manteve idêntica as células da biblioteca *standard cell*. Logo em relação ao tempo de execução para a criação das células foi pequeno mesmo para células grandes.

Na tabela abaixo são apresentados os resultados obtidos, em relação à altura das células geradas automaticamente pelo ASTRAN em comparação as células feitas manualmente pertencentes a biblioteca *standard cell*, bem como os tempos de execução para a criação de cada célula estudada.

Tabela 2.1: Resultados obtidos por Ziesemer (ZIESEMER; REIS, 2014).

Célula	**Altura** *(µm)*			**Tempo Exec.** *(s)*
	Std. Cell	Astran	%	
AND2X4	1	1.2	20	10
FAD1X4	3.6	4	12	750
FAD1X9	4.2	4.2	0	1800
HAD1X9	2.4	2.4	0	30
HAD1X18	2.8	2.8	0	205
INVX0	0.6	0.6	0	1
MUX21X9	1.8	2	11	80
NAND2X11	1.4	1.4	0	5
NAND2X21	2	2	0	8
NAND3X3	1.2	1.2	0	6
OAI211X3	1.4	1.4	0	7
OAI211X11	2.4	2.6	8	26
XOR2X3	1.6	1.6	0	16
XOR2X6	1.6	1.6	0	15
TOTAL	28	29	3,6	-

Em (SMANIOTTO, 2016) é realizado um aprimoramento na técnica de *folding* onde o método não é aplicado apenas em um único transistor e sim sobre um arranjo de transistores. Na implementação da técnica de *folding* em (ZIESEMER, 2007) é utilizado um *folding* estático e um posicionamento de dinâmico, exatamente como foi abordado acima neste trabalho. Primeiro é realizado o *folding* dos transistores para após eles serem organizados quanto a posição. Já no trabalho de Smaniotto (SMANIOTTO, 2016) foi utilizado um *folding* dinâmico – que significa a múltipla divisão do transistor, respeitando o limite mínimo da difusão, juntamente com o posicionamento dinâmico dos transistores, permitindo uma redução na área utilizada pelas células geradas.

a) 

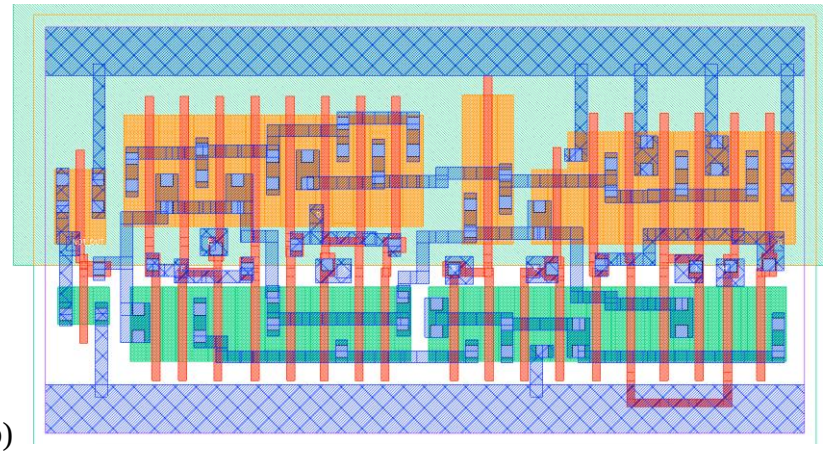
b) 

Figura 2.13: As diferenças entre layouts de células geradas pelo ASTRAN pela técnica original (a) em comparação a nova técnica de *folding* (b). (SMANIOTTO, 2016)

Esta nova abordagem empregada no ASTRAN para a *folding* conseguiu alcançar ganhos de aproximadamente 4,8% no decréscimo da área utilizada pelas células sintetizadas, assim como uma redução de 31,4% na difusão com contatos – devido à redução de conexão entre transistores.

Em um estudo realizado por Posser (2010) utiliza as células do ASTRAN para avaliar a qualidade dos leiautes desenvolvidos com células automaticamente geradas em comparação às células de bibliotecas comerciais. No trabalho citado, é possível observar as peculiaridades do projeto de sistemas digitais: os circuitos sintetizados, em um primeiro momento, pareceram ter resultados inferiores em razão do pequeno acréscimo de área, mas, em contrapartida, os resultados de consumo de potência e o atraso total do circuito diminuíram significativamente – com ganho na média de 16,46% e 11,66% respectivamente (POSSER, 2010).

Após investigação, concluiu-se que os resultados se deviam ao fato da menor capacitância de entrada, característica essa predominante nas células geradas pelo ASTRAN, em consequência do modelo de projeto empregado – roteamento interno feito, em sua maioria, em polissilício, não congestionando o interior das células com metal 1 e, assim, permitindo um melhor roteamento global.

Outro ponto foi o impacto das definições impostas para projeto das células, que se destacaram no projeto global dos circuitos sintetizados. Nesse caso, pode-se citar a maior facilidade de roteamento global em virtude da área maior das células e, assim, não sendo necessária a inserção de áreas em branco exclusivas para trilhas, o que resultou em redes de menor tamanho e também em menor atraso do caminho crítico dos circuitos.

Uma possibilidade cabível, na geração de células pelo ASTRAN, é sintetizar células com a mesma função lógica, mas com diversas capacidades de condução. A experimentação dessa variedade de células é um dos principais pontos em que se concentra este trabalho, sendo explicitada nas seções seguintes.

3 APLICAÇÃO DO APRENDIZADO DE MÁQUINA

O aprendizado de máquina é uma técnica que consiste no projeto e desenvolvimento de algoritmos a que sejam permitidos adoção de comportamentos baseados na identificação de padrões ou regularidades de um certo processo (ALPAYDIN, 2004). Essa técnica de aprendizado é capaz de, partindo de dados amostrais desconhecidos, montar um modelo que represente tal processo.

Várias áreas do conhecimento são auxiliadas por técnicas de aprendizado de máquina, que são especificamente desenvolvidas para encontrar soluções para problemas complexos. Destas, podemos citar: aplicações financeiras para investimento em aplicações, detecção de fraude, análise de risco, entre outras; na medicina, linhas de produção; telecomunicações, além das áreas científicas, como física, astronomia e biologia. Por serem parte integrante da inteligência artificial, esses sistemas desenvolvidos têm, como característica básica, a habilidade de adaptação a ambientes dinâmicos, tornando, assim, os sistemas desenvolvidos robustos a constantes mudanças.

No âmbito deste trabalho foram utilizadas Redes Neurais Artificiais com o objetivo de modelar o comportamento das células, tanto geradas pelo ASTRAN quanto células comerciais feitas à mão. Nesta modelo adotado é capaz inferir, em uma célula específica, dependendo das condições em que se encontra tal célula, o comportamento quanto ao consumo de potência e ao atraso da mesma. Estas condições do ambiente que esta célula está inserida servem como informações de entrada para estas RNAs, que por sua vez irão fornecer como saída valores de potência e atraso. Estes aspectos citas serão melhores explicados e detalhados nas próximas seções – seção 3.2.

Dentre as diversas técnicas de aprendizado de máquina, as Redes Neurais Artificiais, ou simplesmente redes neurais – sendo essa uma analogia referindo-se à rede neural biológica como um grande sistema de processamento paralelo distribuído, capaz de adquirir conhecimento - têm sido amplamente usadas para solucionar diversos problemas de reconhecimento de padrões, bem como de aproximação de funções complexas (HAYKIN, 1994). Essas estruturas têm sido utilizadas, com sucesso, em aplicações de domínios extremamente diversificados, desde controle robótico até visão computacional, passando por aproximações de curvas complexas, entre outras (HAYKIN, 2008).

Dos tipos existentes de redes neurais, pode-se dizer que a rede neural do tipo *perceptron* de múltiplas camadas (MLP – do inglês *Multi Layer Perceptron*) é a mais popular. A Figura 14 apresenta a topologia genérica de uma rede neural do tipo MLP. Os elementos i_1, ..., i_n denotam os *n perceptrons* da camada de entrada, que, por definição, não realizam processamento sobre os dados, sendo então classificados como *transparentes*.

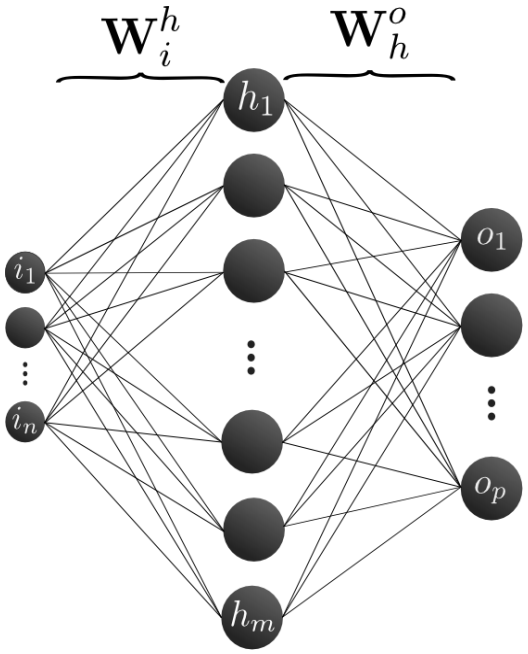


Figura 3.1: Topologia Genérica de uma Rede Neural do tipo MLP

Os *m* elementos da camada intermediária, ou escondida, são apresentados como h_1, ..., h_m. Essa camada, diferentemente da anterior, apresenta processamento sobre os dados de entrada assim como a terceira camada, ou camada de saída, de onde os resultados serão lidos. Os elementos o_1, ..., o_p denotam os *p* neurônios dessa última camada.

Para cada neurônio *ativo*, existe um conjunto de pesos associados, estes têm função de ponderar as saídas da camada anterior. Os pesos, ou *sinapses*, são os coeficientes que, efetivamente, aprenderão os exemplos apresentados à rede. Os elementos $\mathbf{W}_i^h$ e $\mathbf{W}_h^o$, por sua vez, representam as matrizes de pesos da camada de entrada para a camada oculta e da camada oculta para a camada de saída, respectivamente.

A saída de cada neurônio da camada escondida obedece a um formalismo matemático expresso pela equação:

$$h_j = net_h(b_j + \sum_{k=1}^{n} w_j^k \cdot i_k)$$

Nesta equação acima, net_h e h_j denotam: a função de ativação e um neurônio genérico da camada escondida, respectivamente, e b_j representa o viés (*bias*) do neurônio h_j.

Já a saída da rede neural, obtida dos neurônios da camada *o*, obedece à equação a seguir:

$$o_j = net_o(b_j + \sum_{k=1}^{m} w_j^k \cdot h_k)$$

Acima, net_o denota a função de ativação escolhida para a camada de saída, o_j representa um neurônio genérico nesta camada, e b_j representa o viés deste neurônio.

As funções de ativação dos neurônios podem apresentar diversos comportamentos, dentre as mais utilizadas, temos as funções: *linear* e *sigmoides (logística* e *tangente hiperbólica)*. Estas funções apresentam importante papel no que se refere ao tipo de dados a serem classificados ou preditos; com funções lineares, podemos alcançar maiores

magnitudes nas saídas, já com funções não lineares, como as sigmoides, temos curvas mais acentuadas (HAYKIN, 2008).

Em geral, bem como neste trabalho, as redes MLP são treinadas através da aplicação de *retropropagação do erro de aprendizado*, mais informações sobre este tema podem ser obtidas em HAYKIN (1994 e 2008).

Uma característica conhecida de redes neurais é a sua notável capacidade de lidar com padrões extremamente complexos, que dependam de diversas variáveis, apresentando predições com baixos índices de erros. Essas características sugerem que elas são uma ótima alternativa para a predição de características complexas e de comportamento não linear, como o atraso e consumo de potência de *standard-cells*.

Além da capacidade de boas predições para exemplos não apresentados, mas cujas variáveis de entrada estejam dentro da faixa de exemplos apresentados no treinamento (como em um processo de interpolação), existem trabalhos que comprovam que redes neurais também são capazes de realizar extrapolação na predição de exemplos não apresentados e que se mantenham dentro de intervalos de confiança.

3.1 Trabalhos Relacionados

A utilização de RNAs na microeletrônica é encontrada em diversas fases do desenvolvimento de um circuito eletrônico: na criação de metamodelos de análise de atraso, corrente de *leakage* e potência, além de técnicas de otimização de ferramentas de CAD, e até em sistemas de teste de circuitos.

3.1.1 Técnicas de Otimização

No projeto físico de circuitos integrados, várias etapas do processo são problemas complexos, entre eles, há a fase de posicionamento de *standard-cells*. Em Sriram (1991), é apresentada uma solução utilizando redes neurais de *Hopfield* em uma versão adaptada. É utilizada uma abordagem através do modelo *min-cut quadrisection*, para solucionar o posicionamento das células, implementado sobre as redes de *Hopfield*.

A solução para o *min-cut* consiste em manter uma distribuição uniforme de fios sobre o *chip*, unida à característica das redes de *Hopfield* de convergirem a sistemas de mínima energia, resulta em uma boa solução para essa abordagem. No entanto, uma desvantagem destas redes é a convergência para mínimos locais, que, em Sriram (1991), é evitada a partir da utilização de outra técnica antes citada: *Simulated Annealing*, que possibilita um aumento na faixa de aceitação de respostas, ou seja, existindo assim a possibilidade da resposta não convergir para mínimos locais, e, em consequência, é aumentada a chance de uma resposta próxima ao mínimo global.

A Figura 3.2 mostra a arquitetura modificada de uma rede de *Hopfield*, consistindo em duas redes, com os pesos das matrizes variando de acordo com o tempo, e também dependendo do estado da rede complementar. A rede definida como X é responsável por minimizar o número de redes que atravessam o eixo das abscissas, enquanto a rede Y minimiza a quantidade de redes no eixo vertical.

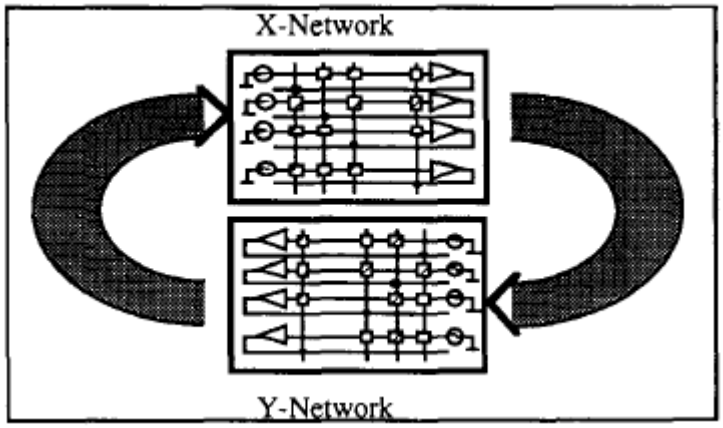


Figura 3.2: Modelo de uma Rede Neural de Hopfield adaptada. (Sriram, 1991)

Em Ohta (2000), é proposta a solução para o roteamento das redes intrachip a partir de redes neurais caóticas, problema também pertencente à classe dos problemas de otimização. Esse tipo de utilização para técnicas de aprendizado de máquina é uma forte tendência nas primeiras aplicações dentro da microeletrônica, atualmente, não tão aplicada, devido ao grande poder de processamento necessário, bem como à recorrência de resultar em soluções pertencentes a mínimos locais, ou seja, não alcançando a solução ótima.

3.1.2 Modelagem de Células

O roteamento de circuitos integrados utilizando redes neurais caóticas (OHTA, 2000) apresenta uma evolução do algoritmo de Funabiki-Takefuji (1992). Esse, por sua vez, consiste em um algoritmo de computação paralela, utilizando a Máxima Rede Neural, cuja origem é baseada nas redes de *Hopfield*.

O uso de *chaotic NN* é justificado por seu comportamento não-periódico, de tal forma que isto indica a característica de obter uma maior facilidade para escapar de mínimos locais.

Em trabalhos posteriores à primeira fase de utilização de redes neurais na microeletrônica como, por exemplo - Sriram (1991) e Ohta (2000), ocorrem uma mudança no modo de aplicação das RNAs. Ao contrário dos exemplos citados acima, que utilizam métricas simples para aplicação na função de custo, objetivando a minimização deste, trabalhos como o de Sait (2001) abordam o emprego de técnicas de aprendizado de máquina para simular o comportamento das diversas variáveis pertencentes ao projeto físico de um circuito integrado, *e.g.* a modelagem do atraso e consumo de potência do circuito.

A proposta de Sait (2001) consiste em utilizar um algoritmo de lógica *fuzzy* com simulação de evolução, com o objetivo de obter soluções de posicionamento de células com baixo consumo de potência e alto desempenho. Com o objetivo de selecionar a melhor solução, foram desenvolvidas algumas medidas de custo que representem todos os objetivos, de modo a fazer-se necessária a criação de dois diferentes operadores *fuzzy – OR Controlado* e *AND Controlado*, para a utilização nas regras *fuzzy* descritas.

No total, três regras foram criadas: a primeira, para definir se a solução em questão é aceitável; a segunda regra avalia o quão boa é a localização da célula; já a terceira regra é responsável pelo estágio de alocação de células, a partir da avaliação da segunda regra.

Trocas de posições podem ocorrer no circuito visando a reduzir as restrições globais de projeto.

Todas as regras adotam um equacionamento utilizando operadores de lógica *fuzzy*, que levam em consideração modelos que avaliam o atraso, potência, largura total do circuito e o tamanho das conexões das células do circuito.

Em Das (2008) é apresentada uma técnica de modelagem comportamental de *standard-cells* utilizando RNA, em que uma única modelagem incorpora dados de: atraso, carga de saída da célula, além do tempo de subida ou descida (*slew*) da célula, tensão de alimentação, temperatura, e ainda parâmetros de variação, global e local, do processo de fabricação. A RNA que modela as células é treinada a partir de dados obtidos de simulações *spice*, cujo comportamento da célula obedece ao modelo elétrico.

A opção por uma modelagem através de aprendizado de máquina é devido ao alto grau de complexidade matemática da exata equação que descreve o atraso de uma porta lógica, e também por não ser uma função linear dos parâmetros subjacentes do processo, como todos os termos, que estão descritos acima, pertencentes à modelagem da RNA. Outro aspecto importante apresentado em Das (2008) é o fato de modelos existentes atualmente, tanto para análise estática, quanto estatística, serem baseados em tabelas, ou seja, são definidos, com índices da tabela, valores intermediários de carga do *gate* e *slew* de entrada da porta, e estes correspondem a um valor que indica o *delay* da porta lógica. Similarmente, o modelo estatístico de atraso, o qual é, normalmente, linear ou quadrático, usa tais tabelas para armazenar os coeficientes de sensibilidade do atraso. Dessa maneira, é evidente que, para os valores armazenados de uma forma discreta, e, dependendo da quantidade de índices da tabela, pode ser perdida muita informação quanto ao atraso de uma célula, pois, para valores de carga e *slew* que não são índices, ferramentas matemáticas, como a interpolação, são utilizadas. Expandindo esta abordagem para a utilização em múltiplos valores de tensão e temperaturas com uma granularidade adequada para tais análises, demandará um alto custo em termos de esforço de caracterização de células.

Os modelos baseados em redes neurais são criados com um menor número de simulações *spice* em comparação a modelos baseados em *lookup table*. Por conseguinte, estas podem ser utilizadas de maneira eficiente para, até mesmo, gerar modelos de atraso baseados em tabelas, e também para modelos estatísticos, linear e quadrático, para qualquer condição de tensão de alimentação, temperatura, carga e *slew*. Outro aspecto importante é a capacidade de análise das variações dos parâmetros de processo, possibilitando um maior controle no ambiente de projeto.

Como mencionado, Redes Neurais Artificiais têm mostrado uma grande capacidade para a modelagem de parâmetros dos circuitos para aplicações CAD. Em Janakiraman (2010), esse conceito é expandido para um *Statistical Leakage Analysis (SLA), Framework* capaz de trabalhar com modelos baseados em RNA de modo eficiente.

A construção do *framework* é justificada pelo fato de redes neurais com função sigmoidal de ativação e entradas Gaussianas não permitirem expressões analíticas para dados de média e variância.

Na Figura 16, é apresentado um exemplo de modelo de RNA baseado em um conjunto de transistores NMOS. Como é observado, o modelo RNA equivalente possui 4 entradas distintas – Temperatura, Tensão, Parâmetros Globais de Processo e Parâmetros Locais de Processo. Nesta RNA, os valores de temperatura, tensão e parâmetros globais de processo

são valores determinísticos (JANAKIRAMAN, 2010), enquanto a entrada correspondente aos parâmetros locais de processo é Gaussiana, ou seja, pertence a uma distribuição normal. A saída resultante da RNA consiste na corrente de *leakage* da célula.

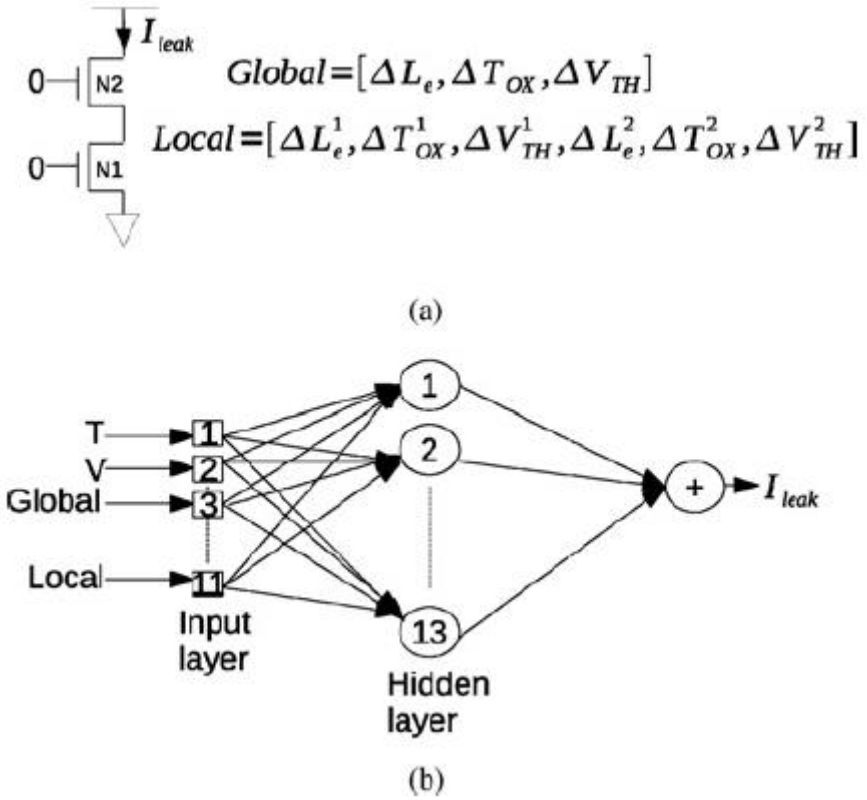


Figura 3.3: Ilustração de um modelo de RNA para dois transistores NMOS. (a) Representação elétrica de dois transistores NMOS. (b) Modelo RNA. (JANAKIRAMAN, 2010)

A função de ativação modificada utilizada por Janakiraman (2010) é chamada de função RC, pois se assemelha à função que descreve o comportamento de carga de um capacitor em um circuito RC. Utilizando a modelagem a partir de RNA com a modificação na função de ativação, o *framework SLA* desenvolvido é capaz de predizer a média, variância e CDF (*Cumulative Distribution Function*) da corrente de *leakage* de um circuito. Em comparação com outros *frameworks SLA* que utilizam modelos quadráticos, o *framework* utilizando RNA tem uma menor complexidade computacional, em comparação com o modelo de análise Monte Carlo, o *framework* chega a ser 100x mais rápido e com resultados que diferem entre 1% e 2%.

Em

3.1.3 Circuitos de Teste

Atualmente, testes online são um dos mais desafiadores temas em projetos de sistemas para a inspeção comportamental de circuitos digitais durante o seu período de trabalho. Em Hosseini (2010), é apresentada uma nova abordagem para a realização de testes *online* de maneira simultânea, para diversos tipos de circuitos combinacionais, utilizando uma RNA reconfigurável, implementada juntamente com o *hardware* do circuito. Para a realização de uma estrutura genérica em *hardware* com a possibilidade de alinhar-se ao comportamento de um circuito combinacional, são encontrados diversos desafios, de tal

modo que Hosseini (2010) apresenta o estudo e discussão de diversas técnicas para obter a configuração ideal para a realização de uma *Built-In Self Test* (BIST) a partir de uma Rede Neural Artificial em *hardware*, com o objetivo de testar de modo *online* circuitos combinacionais (CUT – *Circuit Under Test*).

O desenvolvimento em *hardware* de uma rede neural artificial completa necessita de atenção especial em algumas questões para possibilitar uma redução da complexidade e do *overhead* de área, pois uma RNA puramente combinacional e com alta precisão consome uma área inaceitável. Desta forma, Hosseini (2010) incorpora vários métodos para obter a mínima RNA capaz de modelar o sistema, através de aproximações de resultados, e evitar cálculos desnecessários por simulações prévias.

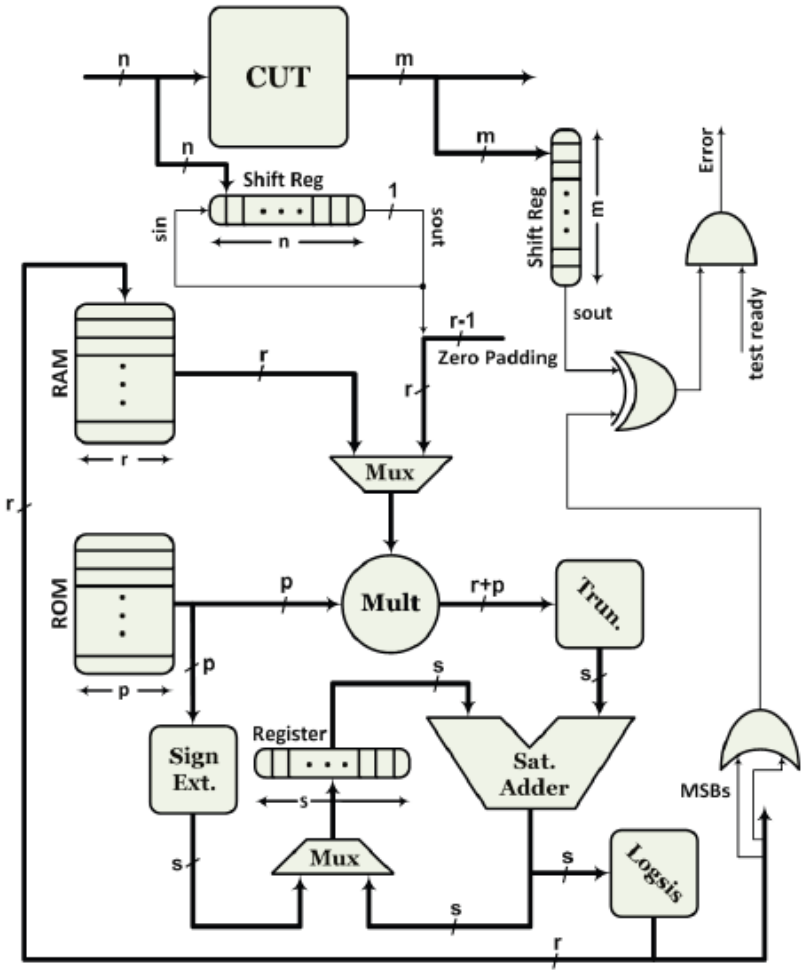


Figura 3.4: *Data-path* proposto para a realização da *BIST*. (HOSSEINI, 2010)

Na Figura 3.4, Hosseini (2010) apresenta o *data-path* responsável pela implementação em *hardware* da Rede Neural sequencial, isto é, com o objetivo de reduzir a área utilizada pela RNA, optou-se por uma implementação sequencial – ocasionando um atraso aceitável na obtenção da saída, e, por consequência, o descarte de algumas amostras. Outra adaptação realizada foi a adoção da aritmética de ponto fixo, pois esta necessita de menos recursos para implementação em *hardware*.

Alguns módulos são facilmente identificados: contém uma pequena RAM, para armazenar resultados intermediários da multiplicação, uma ROM, responsável por guardar os pesos dos neurônios, um somador saturado, que faz parte de um acumulador,

um multiplicador sequencial, utilizado na sensibilização das entradas, um módulo responsável por calcular a função de ativação *Logsis*, além de alguma lógica adicional para entrada e saída ou temporização do BIST.

Hosseini (2010) conclui afirmando que Redes Neurais Artificiais podem, perfeitamente, modelar um circuito combinacional qualquer, baseando-se em sua respectiva tabela verdade ou equações lógicas. Esta abordagem pode ser expandida para outras topologias de Redes Neurais Artificiais, tais como redes de *Hopfield*, ou mapas auto-organizáveis (redes de Kohonen), e também algumas melhorias podem ser aplicadas a circuitos específicos, como, por exemplo, analisar apenas alguns *bits* da saída do CUT - essa modificação seria conveniente para circuitos aritméticos onde os *bits* mais significativos são os mais importantes.

3.2 RNAs Para Caracterização de Células ASTRAN

Um fluxo de projeto ASIC é considerado desde a descrição comportamental em linguagem HDL (*Hardware Description Language*), sendo este um nível mais abstrato, até a sua síntese física, que é a fase que mais se aproxima do circuito pós-fabricado. Além de todos os refinamentos feitos nas fases de síntese, um fluxo de projeto é constituído de diversas etapas relacionadas à análise do circuito que está sendo desenvolvido.

Tanto para síntese como para a análise, componentes de um circuito real são modelados matematicamente e fisicamente, de forma a construir um cenário computacional que corresponda, aproximadamente, ao circuito de fato fabricado. Este modelo computacional é fundamental para manter a consistência entre as etapas de síntese, principalmente para fluxos mais flexíveis, que permitem a integração de novas características, ferramentas ou até mesmo novos algoritmos, devendo, obrigatoriamente, estar inseridos no mesmo cenário, considerando o mesmo meta-circuito. Caso esta regra não seja mantida, a consistência do fluxo será quebrada, e os dados de sínteses e análises gerados serão ainda mais distantes dos valores reais para o chip pós-fabricado.

Uma grande dificuldade encontrada no meio acadêmico no desenvolvimento de ferramentas CAD é, justamente, a integração do fluxo proposto na academia com o fluxo comercial - amplamente aceito que, ao ser possível efetuar tal conexão, o trabalho realizado no âmbito acadêmico pode ser validado de modo conclusivo.

No ambiente mostrado acima, faz-se necessária a utilização de técnicas computacionais capazes de possibilitar, com uma maior facilidade, a integração de fluxos de projeto distintos, de tal forma a unir diferentes modelos computacionais responsáveis por descrever as características do circuito em questão. Assim, o presente trabalho apresenta uma proposta na utilização de RNAs (Redes Neurais Artificiais) aplicadas na caracterização de redes de transistores, geradas automaticamente pela ferramenta ASTRAN (ZIESEMER, 2007), utilizadas no fluxo ASIC, sendo que as RNAs são capazes de estimar os valores de potência e atraso das células empregadas no circuito.

Como é apresentado primeiramente na Figura 1.2 e mostrado em detalhe pela Figura 3.5, para utilizar a ferramenta Substituidora, é necessário que as células geradas automaticamente estejam em um mesmo contexto das células a serem retiradas de uma biblioteca comercial.

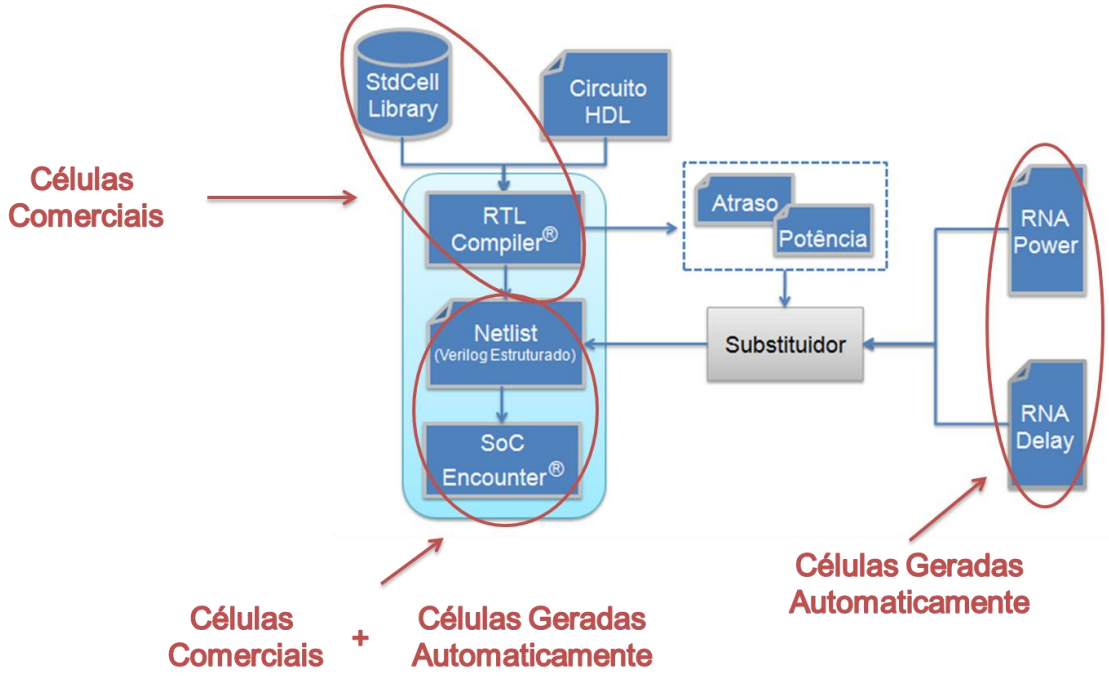


Figura 3.5: Fluxo de Projeto proposto utilizando a ferramenta Substituidora.

A ferramenta Substituidora tem, como entrada, os relatórios, gerados após a síntese lógica, dos caminhos dos circuitos, juntamente com seus respectivos atrasos e o consumo de potência de cada célula, sendo que cada célula comercial é comparada com um modelo baseado em RNA das células geradas automaticamente, modelo que, por sua vez, também é gerado a partir dos relatórios gerados após a síntese física, como pode ser observado na Figura 3.6.

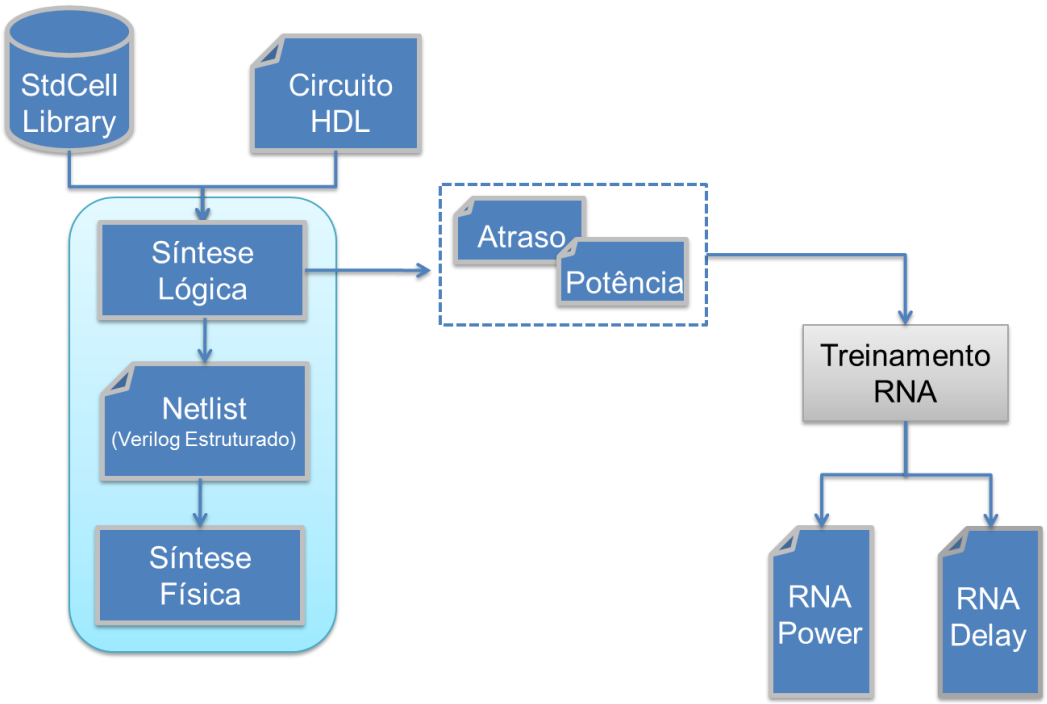


Figura 3.6: Processo de treinamento das Redes Neurais Artificiais.

Para obter estes dados finais de potência e atraso, respectivos a cada célula, são necessários diversos passos. A princípio, é indispensável que as células geradas sejam inseridas em um fluxo de projeto ASIC, o que implica na criação de uma biblioteca onde as células construídas pela ferramenta ASTRAN sejam descritas de uma maneira textual, possuindo todas suas características elétricas, conforme é feito para uma biblioteca de células comercial, onde o padrão adotado é formato LIBERTY, que possui, como particularidade, o detalhamento da área das células, bem como dados de atraso e potência.

Para construção de um LIBERTY, é necessária a realização de alguns passos importantes. O aspecto fundamental deste processo é o desenho do leiaute das células. Geralmente, essa etapa é realizada por projetistas experientes, capazes de desenhar células com uma boa relação entre as restrições, e.g. área, potência e atraso, mas, no caso do corrente trabalho, esse desenho é gerado de modo automático pela ferramenta ASTRAN conforme apresentado Capítulo 2.

Após o desenho concluído, o leiaute gerado passa por um processo de extração - Figura 3.7, ou seja, o desenho é analisado a partir de parâmetros referentes à tecnologia em questão, e esse processo valida o leiaute, além de fornecer detalhes elétricos, como a inserção de capacitâncias parasitas.

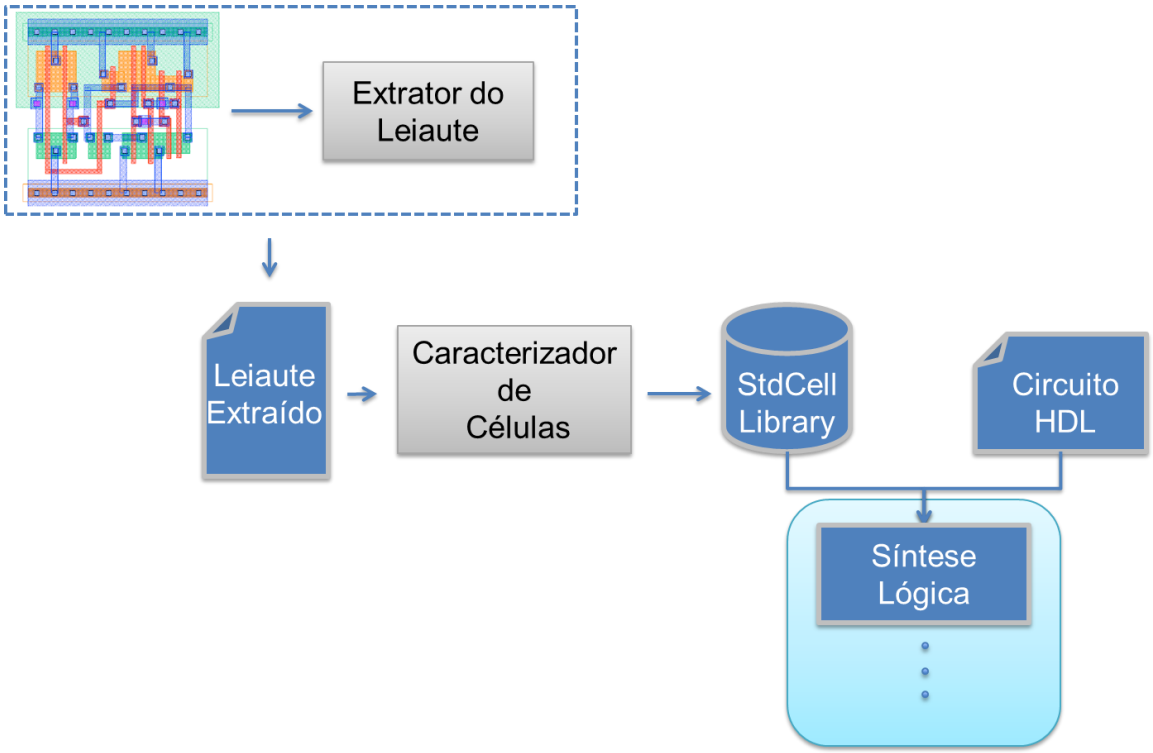


Figura 3.7:Fluxo para Extração do leiaute gerado automaticamente pelo ASTRAN e consequente criação de uma Biblioteca de Células.

O passo seguinte consiste na caracterização das células e consequente geração do arquivo texto no formato LIBERTY. Um passo intermediário à caracterização é a descrição do leiaute extraído em formato texto, com todos os parâmetros do modelo elétrico da tecnologia, possibilitando, desta maneira, dados suficientes para a fase seguinte, que consiste em realizar simulações a partir destes dados.

A caracterização de células consiste no cumprimento de diversas simulações no nível elétrico. Cada simulação é responsável por verificar o comportamento da célula segundo

um ambiente específico, comumente, definido pela variação da carga de saída e mudança na inclinação da onda de entrada da célula em questão. Neste ponto, já é capaz de avaliar características de atraso e potência da célula provenientes das simulações elétricas; já a área ocupada é obtida logo após o desenho do leiaute.

Em seguida, todos os dados adquiridos nas simulações elétricas são agrupados de forma a consolidar o arquivo no formato LIBERTY. Atualmente, existem algumas técnicas específicas para a construção de uma biblioteca, entre elas, temos desde o modelo mais simples e consagrado NLDM (*Non Linear Delay Model*) aos modernos ECSM (KARIAT, 2010) e CCS (TRIHY, 2008). Na Figura 3.7, pode ser observada uma simplificação do fluxo para criação de uma biblioteca de células.

3.2.1 Relatórios da Etapa de Síntese Lógica

Para este trabalho, foi utilizada, como ferramenta principal para a realização da síntese lógica, a ferramenta da *Cadence – RTL Compiler* (CADENCE, 2012). O *RTL Compiler* é alimentado por três tipos centrais de entrada, o principal é o arquivo de descrição HDL do circuito, em seguida, a biblioteca de células com o arquivo LIBERTY – essencial para o mapeamento tecnológico, e por último o arquivo que define as restrições de projeto – SDC (*Synopsys Design Constraints*).

Após a realização da síntese lógica, é resultante um arquivo *netlist*, onde as células mapeadas são descritas de forma estrutural em um arquivo Verilog. Tal arquivo serve como entrada para as etapas de verificação formal e síntese física. Além do Verilog estrutural resultante, na ferramenta de síntese lógica, é possível gerar vários relatórios, detalhando aspectos importantes do circuito, que no escopo deste trabalho, são usados para nutrir o treinamento das Redes Neurais. Entre esses, é gerado o relatório do tipo *Worst Path*, o qual descreve o atraso de cada caminho do circuito, detalhando o atraso por célula, além do relatório do tipo *Instance Power Info*, onde são descritos todos os dados relativos à potência de cada célula instanciada no circuito.

O relatório do tipo *Worst Path* é apresentado em detalhe na Figura 3.8, em que temos, como exemplo, o caminho 434, sendo que o caminho 1 é o caminho crítico do circuito. Podemos perceber que, neste tipo de relatório, há tanto o número de células contidas em um caminho, como o tipo de cada célula (*Type*) e o nome da instância e o pino (*Pin*) com que a célula conecta-se ao caminho.

Além desses itens, é possível obter também valores de *fanout*, *load*, *slew*, além do atraso de cada célula, ainda, a célula ponto de partida do caminho, bem como a que designa o ponto final. Para cada instância pertencente ao caminho, é mostrado o atraso acumulado (*Arrival*) para tal caminho. Por último, na última coluna, é mostrado o tempo de folga (*Timing Slack*) do *path*, que indica uma violação às restrições de projeto (SDC), por ter um valor negativo.

A partir do console do *RTL Compiler*, esse relatório pode ser obtido através do seguinte comando abaixo:

```
> report timing –worst <número_de_caminhos>
```

```
path 434:

    Pin          Type     Fanout Load Slew Delay Arrival
                                 (fF) (ps)  (ps)    (ps)
-----------------------------------------------------------
(clock clock)    launch                                0 R
G38_reg/C                                   0           0 R
G38_reg/Q        DFC1          2  8.2  107  +811      811 F
g3675/A                                          +0     811
g3675/Q          BUF2          9 72.3  212  +386     1197 F
g37/A                                            +0    1197
g37/Q            INV1          2  8.7  223  +214     1411 R
g3339/A                                          +0    1411
g3339/Q          NAND20        1  4.2  110   +73     1484 F
g3090/B                                          +0    1484
g3090/Q          NOR20         1  4.2  167  +129     1612 R
g3061/B                                          +0    1612
g3061/Q          NOR20         1  5.0  142  +108     1720 F
g3051/A                                          +0    1720
g3051/Q          NOR20         1  6.1  209  +172     1892 R
G38_reg/D        DFC1                            +0    1892
G38_reg/C        setup                     0     +205    2098 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clock)    capture                               0 R
-----------------------------------------------------------
Timing slack :   -2098ps (TIMING VIOLATION)
Start-point  : G38_reg/C
End-point    : G38_reg/D
```

Figura 3.8: Relatório de atraso do tipo *Worst Path*.

O segundo tipo de relatório é o *Instance Power Info*, responsável por todos os dados relativos à potência de cada célula instanciada no circuito. Na Figura 3.9, pode-se observar que esse relatório possui dados sobre diversas componentes da potência consumida por uma célula - com um alto grau de relevância, sendo que esse relatório é dividido em três partes. Pode-se observar que o relatório informa tanto a instância da célula, quanto o tipo a que ela pertence, possibilitando uma maior facilidade referente à necessidade de agrupar informações do relatório de caminhos com o relatório de potência.

O primeiro grupo informa a potência estática da célula (*Leakage Power*), potência interna (*Internal Power*), potência consumida pela rede conectada aos pinos da célula (*Net Power*).

O segundo conjunto de dados apresenta o detalhamento do cálculo da *Net Power*, explicitando os pinos das células (*Pin*), juntamente com as redes (*Net*) conectadas, além de indicar a probabilidade (*Computed Probability*) e taxa de alternância (*Computed Toggle Rate*) computada, e, para cada um destes conjuntos, a potência consumida pela rede (*Net Power*).

O último conjunto de dados foca minuciosamente em como é efetuado o cálculo da potência interna das células. Nas duas primeiras colunas, são informados os arcos (*Arc From/To*) para o cálculo da potência, na terceira coluna, está o estado do arco (*When*) em questão, já a quarta consiste em informar a taxa de ativação do arco (*Arc Activity*), e, por fim, a quinta coluna relata o total absoluto de consumo para cada arco da célula (*Arc Power*).

O relatório de potência do *RTL Compiler* é obtido a partir do seguinte comando abaixo:

```
> report power -flat
```

```
      Instance Power Info
      -------------------

 Instance G42_reg of Libcell DFC1

  Leakage     Internal      Net
 Power(nW)    Power(nW)   Power(nW)
-----------------------------------
     0.224   644102.639    6648.345

                        Computed          Computed          Net
Pin         Net        Probability    Toggle Rate(/ns)    Power(nW)
-------------------------------------------------------------------
Q      G42                    0.2              0.0300       6648.345
RN     n_482                  0.5              0.0200          0.000
C      blif_clk_net           0.5              2.0000     353925.000
D      n_731                  0.3              0.0450       1494.652


 Arc  Arc              Arc            Arc
From   To    When   Activity(/ns)    Power(nW)
-----------------------------------------------
RN     RN                 0.0200   117786.000
C      C     D            0.5400   294752.000
C      C     !D           1.4600   313622.000
D      D     C            0.0225    25382.700
D      D     !C           0.0225   268557.000
C      Q                  0.0297   300670.000
RN     Q                  0.0003   325165.000
C      QN                 0.0297   304694.000
RN     QN                 0.0003   329518.000
```

Figura 3.9: Relatório de potência do tipo *Instance Power Info*.

3.2.2 Utilizando Redes Neurais Artificiais para Caracterização

Um dos desafios na integração de fluxos de projeto ASIC, como citado anteriormente, é a manutenção da consistência entre o cenário de análise e a síntese dos fluxos em questão. Isso significa que a utilização de células caracterizadas sob parâmetros distintos acarretará em resultados de análise com um erro em relação à realidade maior do que o aceito.

Tendo em vista a conservação da dita consistência entre fluxos, a utilização de Redes Neurais Artificiais para a caracterização de células torna-se uma proposta interessante. Além das características apresentadas neste capítulo, que mostram alguns benefícios, RNA também aponta uma solução de baixa complexidade computacional, por ocasionar uma insignificante carga de processamento, em virtude do fato do treinamento da RNA ser realizado *off-line*, e apenas é realizada a propagação da rede em tempo de computação, i.e., consiste em um número constante de operações de multiplicação. Comparado a outra possibilidade, que, no caso, seria a implementação de um interpretador de LIBERTY para a obtenção de dados de atraso e potência, o uso de RNAs como uma possibilidade de baixa complexidade se torna evidente.

As RNAs, como abordado anteriormente, são amplamente utilizadas para reconhecimento de padrões e aproximar funções complexas. Tanto no caso do atraso do circuito quanto na potência, ambos são definidos por equações matemáticas, de modo a preencher os requisitos de utilização de redes neurais, tal que se comportem como uma função complexa.

Utilizamos, para avaliação da potência, a sua parcela mais significativa, nomeada como *Internal Power*. Essa potência é definida pelo *RTL Compiler* pela seguinte equação abaixo:

$$\mathcal{P}_{internal} = \sum_{perarc} \left(TR_{arc_{ij}} \times \Phi(S_i, C_j) \right) + \sum_{perpin} \left(TR_i \times \Phi(S_i) \right)$$

Essa equação leva em consideração vários fatores, entre eles, os dados específicos de potência obtidos a partir da biblioteca de células, como o ambiente em que está inserida a célula, além da taxa de comutação dos arcos das células. Seguindo a análise dessa equação, foram definidos os parâmetros de entrada para o treinamento da rede neural capaz de estimar a potência da célula em sua saída. Na Figura 3.10, é possível observar a modelagem da RNA utilizada para a obtenção da potência.

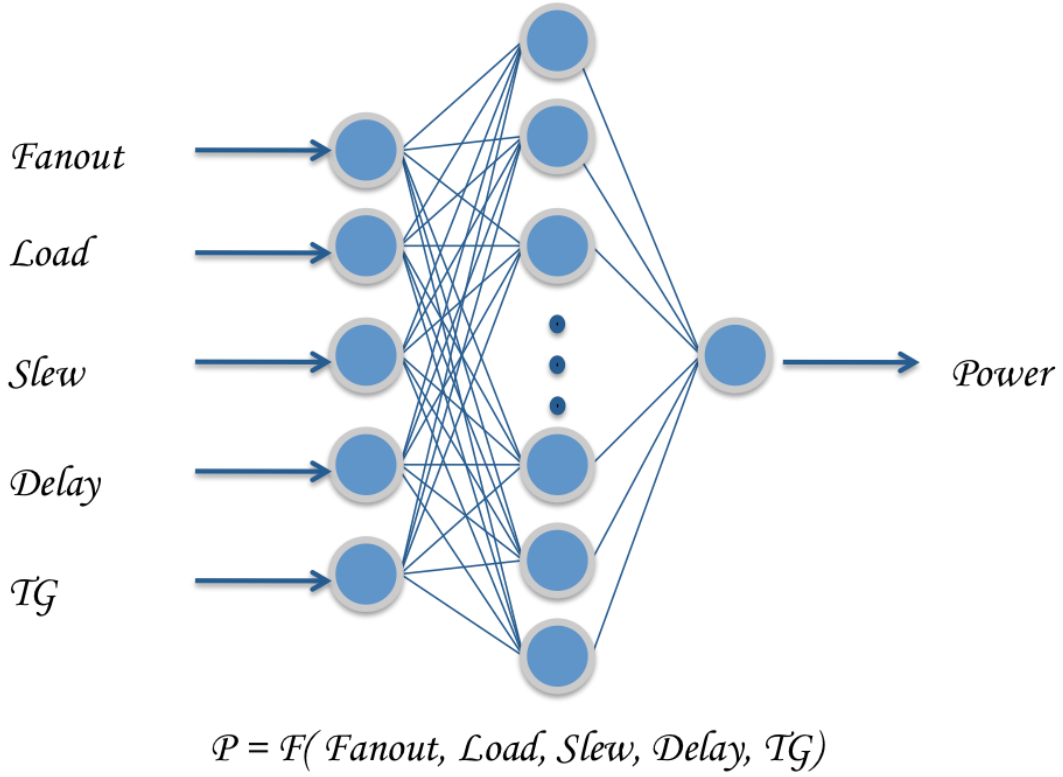


Figura 3.10: Modelo RNA para estimar Potência.

As três primeiras entradas - *Fanout, Load* e *Slew* remetem às condições do caminho em que a célula está contida; o *Delay* indica o atraso dessa célula relativo aos três primeiros parâmetros. Esses quatro primeiros parâmetros são obtidos através do relatório do *RTL Compiler* do tipo *Worst Path.* O último componente da entrada da rede neural é o *TG (Toggle Gate),* que considera a taxa de comutação dos arcos da célula em questão, sendo essa uma estimativa levantada pelo *RTL Compiler* e obtida a partir do relatório *Instance Power Info*.

Ao contrário da potência, o *RTL Compiler* não explicita o método de cálculo para o atraso da célula, mas, através da bibliografia existente, podemos inferir quais componentes, obtidos através dos relatórios, têm peso na determinação do atraso de uma célula lógica.

Segundo Weste (1985), o atraso de uma única porta é dominado pelo tempo de descida (t_f) e subida (t_r) do sinal de saída. Essas duas medidas são definidas pelas equações abaixo, onde C_L é a carga da saída, V_{DD}, a tensão nominal, e β_p e β_n são constantes dos transistores do tipo P e N respectivamente.

$$t_r \cong 4 \cdot \frac{C_L}{\beta_p \cdot V_{DD}} \qquad t_f \cong 4 \cdot \frac{C_L}{\beta_n \cdot V_{DD}}$$

Considerando transistores do tipo P e N do mesmo tamanho, e lembrando que, devido às características elétricas dos dispositivos P e N, transistores do tipo N permitem uma maior mobilidade das cargas elétricas, sendo esses aproximadamente duas vezes mais rápidos, de maneira que podemos considerar que $\beta_n = 2\beta_p$, resultando na seguinte relação:

$$t_f = \frac{t_r}{2}$$

Resultando na equação final para a determinação do atraso médio de uma única célula, conforme a equação abaixo:

$$\tau_{av} = \frac{t_r + t_f}{4}$$

Através dessas equações, podemos inferir quais as variáveis de maior significância para a obtenção do atraso de uma célula. Na Figura 3.11, pode-se observar a modelagem da rede neural para estimar o atraso de uma célula, juntamente com a respectiva função de ativação. Os dados de *Fanout* e *Load* correspondem à capacitância da carga de saída (C_L), já o *Slew* indica os tempos de subida (t_r) e descida (t_f) do sinal de saída.

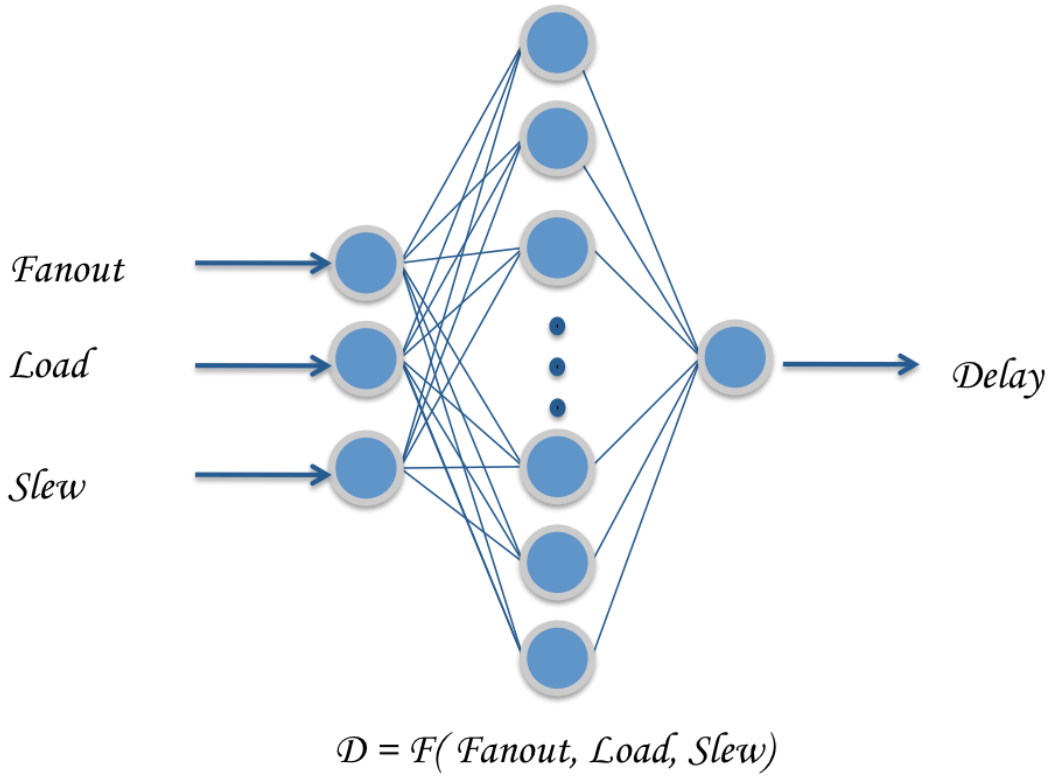


Figura 3.11: Modelo RNA para estimar atraso de uma célula.

3.2.3 Treinamento da RNA

Devido a características intrínsecas ao modelo de redes neurais artificiais, alguns cuidados devem ser tomados ao longo do processo de treinamento delas,, desde a escolha do algoritmo de treinamento e a quantidade de neurônios na camada escondida, até questões como o pré-processamento dos dados de entrada.

Tanto para a rede neural modelada para estimar atraso quanto para a responsável pela determinação da potência, foi utilizada a mesma estrutura, ou seja, mesmo número de neurônios na camada escondida – 55 neurônios, e mesmo algoritmo de treinamento – *Levenberg–Marquardt* (MATLAB, 2012). O pré-processamento dos dados de entrada pode ajudar substancialmente no desempenho preditivo das mesmas (HAYKIN, 1994). Em todos os experimentos, os dados de entrada para o treinamento foram normalizados de forma a propiciar condições plenas para que a rede neural pudesse obter o melhor desempenho.

Em nossos ensaios, os dados foram normalizados em magnitude, de forma a não saturar as ativações dos neurônios bruscamente. Também foram normalizados em sua distribuição espacial, onde a média é levada para zero, e o desvio padrão, para ***1***.

A normalização em magnitude respeita a fórmula abaixo:

$$y = \frac{(y_{max} - y_{min}) \times (x - x_{min})}{(x_{max} - x_{min})} + y_{min}$$

Já a normalização da distribuição espacial é definida conforme a seguinte fórmula:

$$y = (x - x_{mean}) \times \frac{y_{std}}{x_{std}} + y_{mean}$$

Aplicadas estas normalizações aos dados, a rede tem seu aprendizado facilitado, mas os resultados finais, após o treinamento e consequente propagação da rede neural, são submetidos a transformação reversa, tornando compatíveis com os dados originais. Este processo é realizado na ferramenta Substituidora.

3.2.4 Conclusões

Para obter resultados relevantes, as redes neurais foram treinadas com um conjunto expressivo de dados obtidos de diversas sínteses, dos mais variados tipos de circuitos digitais pertencente ao *benchmark ISCAS* (IWLS, 2009).

Nos testes realizados, foi sintetizado todo o conjunto de circuitos do *benchmark*, e, em seguida, foram gerados os relatórios citados acima para cada circuito. Após esta etapa, todos os relatórios foram analisados, e foi montado um conjunto de dados respectivos a cada célula. Este conglomerado de dados por célula foi utilizado para o treinamento da rede neural da célula. A diversidade de células testadas, representando distintas funções lógicas, colabora para uma melhor avaliação sobre a robustez do método empregado.

Os resultados obtidos em Guimarães Jr. (2010) foram satisfatórios. Após as redes neurais das células treinadas, foram realizados testes, selecionando amostras de dados dos relatórios de síntese, e eles foram comparados com a saída da rede neural artificial treinada anteriormente, possibilitando calcular o erro relativo entre os dois valores, e após este cálculo, podendo ser obtida a média e o desvio padrão deste erro.

Tanto para a rede neural responsável por obter a estimativa de atraso quanto para a RNA de potência, a média do desvio padrão e do erro relativo entre as diversas células testadas ficou em torno dos 5% para o desvio padrão, e 4% para o erro relativo (GUIMARÃES JR, 2010).

No geral, essa abordagem se mostrou robusta e flexível, possibilitando superar a dificuldade em integrar diversos fluxos de projeto (acadêmico e comercial), além de esta proposta apresentar uma baixa complexidade computacional, visto que todo o treinamento da rede é realizado *off-line*.

4 FERRAMENTA SUBSTITUIDORA DE STANDARD CELLS

Com o principal objetivo de explorar o espaço de projeto para a utilização de células geradas automaticamente, que, no escopo deste trabalho, consiste em utilizar células geradas pela ferramenta ASTRAN (Ziesemer, 2007), foi desenvolvida uma ferramenta capaz de, em tempo de projeto (entre as etapas de síntese lógica e síntese física), interferir no mapeamento tecnológico resultante e modificá-lo no sentido de substituir células comerciais por células ASTRAN, transformando o circuito resultante em uma combinação dos dois tipos de células.

A ferramenta desenvolvida tem, como restrição de projeto, o atraso do caminho crítico do circuito – atraso máximo encontrado no circuito, e visa a aperfeiçoar a área total do *chip* e a potência consumida por este.

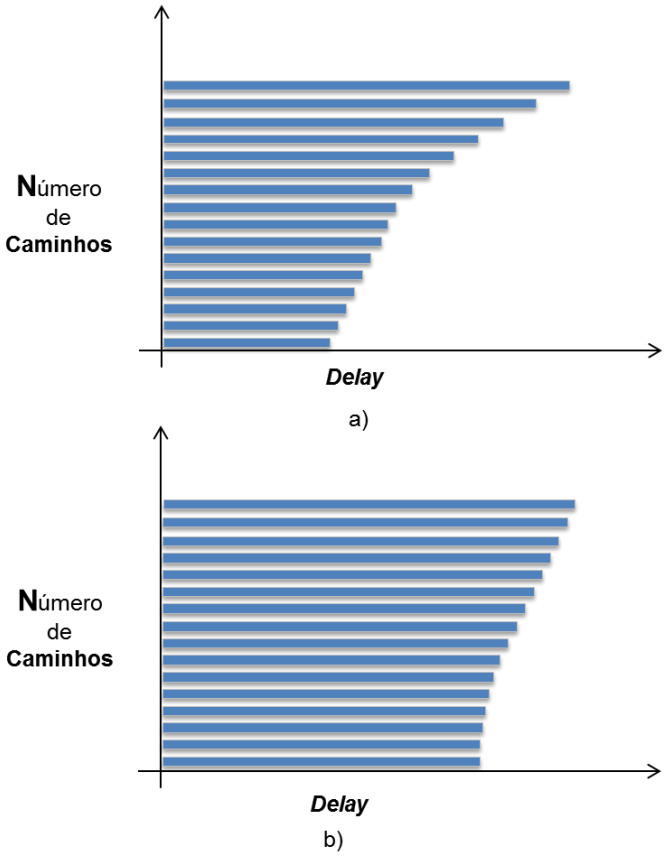


Figura 4.1: a) Distribuição do atraso do circuito anterior às substituições de células.

b) Distribuição esperada após substituições.

O efeito resultante à utilização do componente criado está exemplificado, de forma forma apenas ilustrativa, na

Figura 4.1, onde o primeiro gráfico ilustra conjuntos de caminhos com seus respectivos atrasos, já o segundo gráfico ilustra a possibilidade que, após as trocas

sugeridas pela ferramenta, o resultado esperado estará delimitado entre o atraso máximo do circuito e o restante dos caminhos menores em comparação com o circuito original.

Essa ferramenta foi desenvolvida utilizando a linguagem C++ (TUTORIAL, 2012), além de ter, incorporada em seu código, algoritmos e ferramentas importantes – *Simulated Annealing* (SECHEN, 1988; HENTSCHKE, 2002) e *Flex* (FLEX, 2012).

4.1 Estrutura do Programa

No programa desenvolvido, unificam-se componentes de programação orientada a objetos, e também código procedural estilo C utilizado no *scanner* responsável pelo analisador léxico implementado pela ferramenta *FLEX*. Abaixo, na Figura 4.2, é mostrado um diagrama de classe da ferramenta desenvolvida simplificado, de modo a tornar o modelo didático.

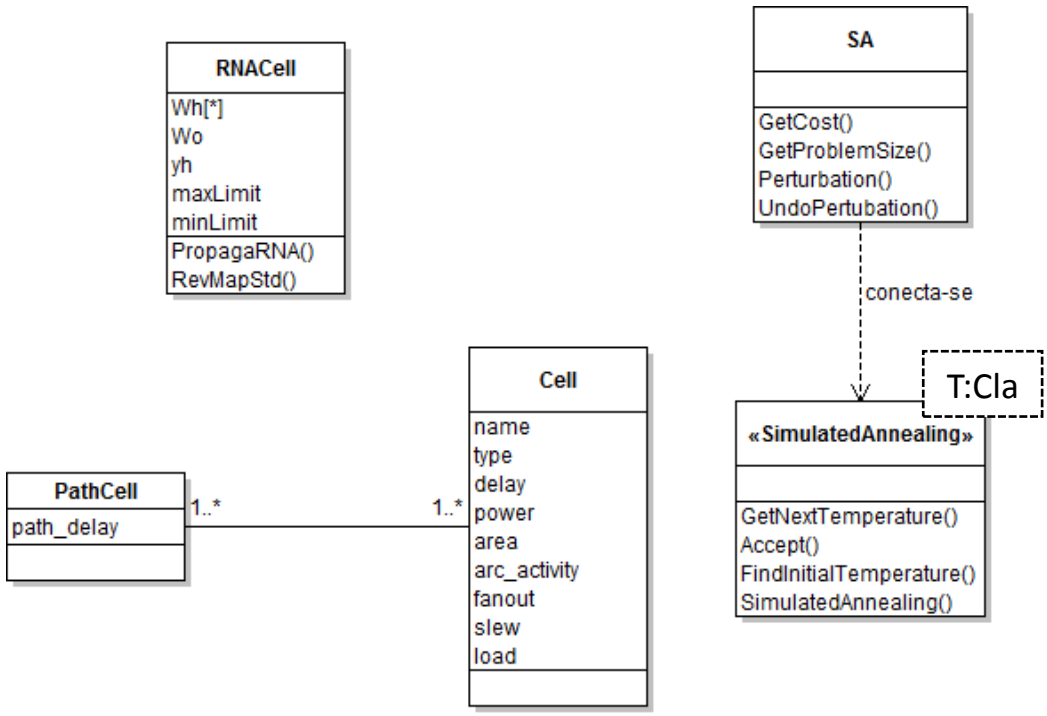


Figura 4.2: Diagrama de Classes simplificado.

O diagrama simplificado apresenta quatro classes básicas mais uma classe *template*, desenvolvida por Hentschke (2002), com o objetivo de controlar a execução do algoritmo do *Simulated Annealing*. Essa classe *template* exige o desenvolvimento de uma classe específica – SA, capaz de fornecer as características do circuito ao *template*, para uma adequada execução do algoritmo. Os pontos particulares dessas duas últimas classes citadas serão detalhados em uma seção posterior.

A classe RNACell tem, como função principal, fornecer ao algoritmo de decisão, i.e. *Simulated Annealing*, os valores de atraso e potência das células geradas automaticamente, valores esses gerados através da propagação das duas RNAs. Outra funcionalidade atribuída a essa classe é a execução da fórmula reversa da normalização em magnitude, e também da normalização da distribuição espacial.

As últimas duas classes restantes - *PathCell* e *Cell*, são responsáveis por compor o modelo do circuito. Cada objeto da classe *Cell* concebe uma instância de uma célula utilizada no circuito, portanto, tem, como atributos, grandezas que identificam o

comportamento dessa instância de célula no circuito. Por esse motivo, tornam-se necessárias as informações relativas à área, atraso, *load*, *slew*, *fanout*, potência e taxa de ativação.

Já um objeto da classe *PathCell* representa um caminho combinacional do circuito sintetizado composto de distintas instâncias de células. O único atributo dessa classe indica o atraso do caminho do circuito, sendo que o atraso do caminho crítico é uma restrição, i.e. não pode ser violado, para a execução do algoritmo de meta-heurística. No relacionamento dessas duas classes, um caminho (*PathCell*) pode conter diversas instâncias de células, sendo essa relação igual para a classe *Cell*, logo, uma instância de célula pode pertencer a vários caminhos no circuito.

Além da representação das células do circuito original, a classe *Cell* também armazena as características das células geradas automaticamente, para posterior comparação com as células comerciais.

4.1.1 Flex

O *Flex – Fast Lexical Analyzer,* em resumo, consiste em obter uma sequência de caracteres e, posteriormente, dividi-los em *tokens*. Para isso, é necessário definir uma lista de expressões regulares responsáveis por gerar os *tokens* a serem analisados posteriormente pelo programa que esta o utilizando.

No contexto do atual trabalho, o *Flex* é responsável por traduzir, em *tokens,* os dados de interesse obtidos a partir dos relatórios de síntese da ferramenta *RTL Compiler*. Neste caso é descrito um arquivo – *scanner.l*, contendo todas as expressões regulares utilizadas.

Na Figura 4.3, é apresentado um trecho desse arquivo; a ferramenta *Flex* utiliza-se do *scanner.l* para gerar um código fonte para ser integrado ao projeto. O código gerado é chamado de *lex.scanner.c*, o qual, principalmente, contém uma função chamada *yylex( ),* que retorna o próximo *token* para o programa chamador.

Alguns desses *tokens* remetem, por exemplo, à potência consumida por uma instância de célula, ao ambiente ao qual está inserida (*slew*, *load, fanout*), bem como dados dos caminhos do circuito, e.g. atraso total do caminho.

```
BFLOAT          {BLOAD}
PINS            {CHAR}+
EXPR            ("!"|{PINS}|"&")+

%%

path+{BRANCOS}+{INTEGER}                                                    {return KEYWORD_PATH;}
Timing+{BRANCO}+slack+{BRANCOS}+":"+{BRANCOS}+{SINAL}+{INTEGER}             {return KEYWORD_TIMING; }
Start-point+{SEPARADOR}+{CELL}                                              {return KEYWORD_START; }
End-point+{SEPARADOR}+{CELL}                                                {return KEYWORD_END;  }
{CELL}+("/Q"|"/QN"|"/S")+{BRANCOS}+{CELL_T}+{BFANOUT}+{BLOAD}+{BSLEW}+{BDELAY}+{BARRIVAL}   {return TOKEN_LINE; }
```

Figura 4.3: Trecho do arquivo *scanner.l*

A partir dos *tokens* retornados pela função *yylex( ),* é possível montar a estrutura de dados principal da ferramenta substituidora de células, utilizando as classes *PathCell* e *Cell*. Após esse processo, todo o circuito original sintetizado – apenas possuindo células comerciais, está pronto para ser analisado, utilizando o algoritmo do *Simulated Annealing,* pelo programa desenvolvido.

4.1.2 Simulated Annealing

O *Simulated Annealing*, ou Simulação de Têmpera, remete ao processo de cristalização dos metais, procedimento que, primeiro, proporciona um aquecimento do metal para altas temperaturas, facilitando a sua maleabilidade; no momento que o formato desejado é alcançado, o metal é submetido a um resfriamento, para que, assim, adquira rigidez e resistência. Essas propriedades mecânicas dos metais são possíveis devido às características singulares de suas moléculas, que, ao encontrar-se em altas temperaturas, ficam em um estado de alta energia, agitadas, sem nenhum tipo de organização. Aos poucos, com o resfriamento do metal, as moléculas se arranjam de maneira organizada, compacta. Esse processo é exemplificado na Figura 4.4.

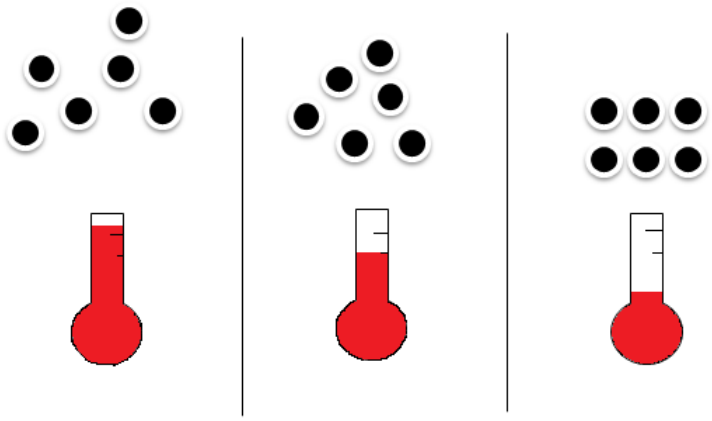


Figura 4.4: Organização das Moléculas na Têmpera.

O algoritmo *Simulated Annealing* funciona seguindo a mesma regra da organização das moléculas em um processo de têmpera, ou seja, no início da simulação, a temperatura é alta, logo é permitida uma maior aceitação nas perturbações gerada pelo algoritmo. Na sequência, com a diminuição da temperatura, apenas perturbações julgadas superiores ou boas serão ace itas.

O algoritmo tem modelo de procedimento bem definido. No primeiro momento, é estabelecido qual o custo inicial do problema, e a temperatura no início do processo é marcada como alta, assim, após cada perturbação, é avaliado, novamente, o custo do problema, e, dependendo da temperatura, essa perturbação é aceita ou não. No início do processo, algumas perturbações que resultam em um estado de custo maior que o inicial serão aceitos, pois a temperatura alta, influenciando diretamente na probabilidade de aceitação, permite tal comportamento. Já no final do processo, a temperatura é baixa, resultando apenas na aceitação de estados resultantes com custo menor que o original.

Essa probabilidade, além de depender da temperatura, também leva em consideração a variação do custo, ou seja, a diferença do custo da solução atual pelo custo da solução modificada. A equação abaixo mostra esse cálculo.

$$prob = e^{\frac{-delta}{temperatura}}$$

O *Simulated Annealing* pode ser considerado um algoritmo simples, isso pode ser verificado a partir da Figura 4.5, que representa o fluxograma do algoritmo. No início, são atribuídos a temperatura inicial e o custo do estado inicial do sistema, em seguida, é gerada uma perturbação nesse sistema e avaliado o seu custo, caso respeite as condições

da diferença de custo ou da probabilidade de aceitação, conforme a equação acima, e o programa vai para próxima iteração. No caso de não satisfazer às condições de controle, a perturbação é desfeita. Para ambos os casos, uma nova temperatura é calculada para a próxima iteração. O bloco condicional "Continua Executando" consiste em uma generalização dos controles limítrofes, podendo ser uma condição de execução até determinada temperatura, ou que tenha alcançado um número máximo de iterações.

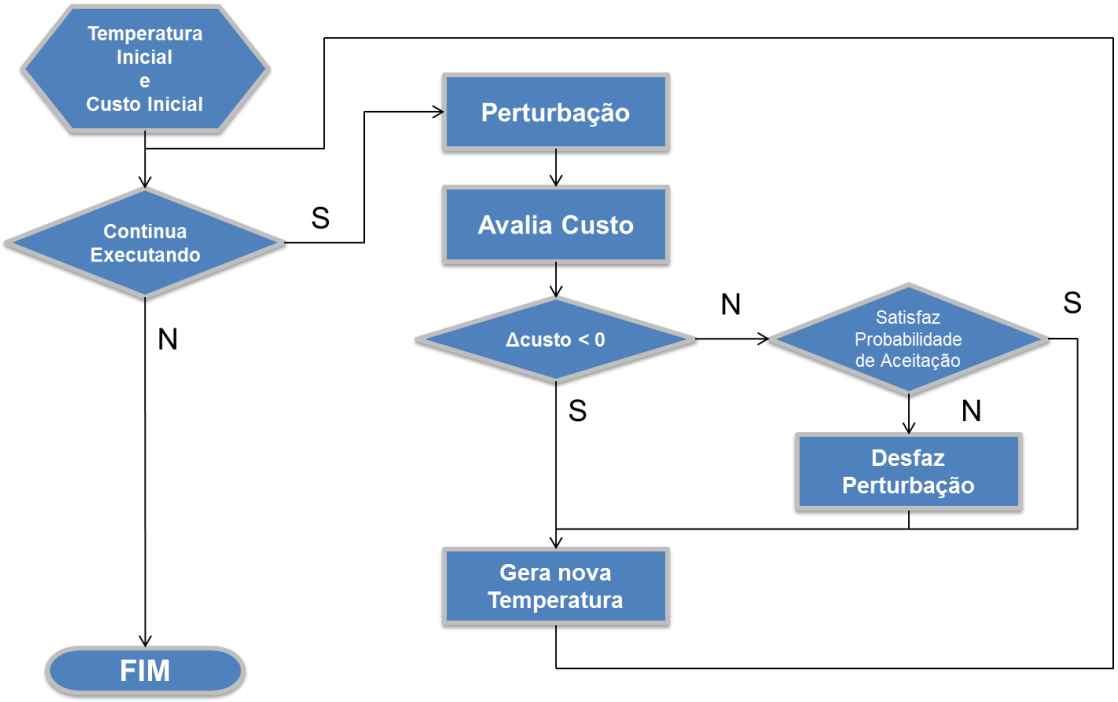


Figura 4.5: Fluxograma do *Simulated Annealing*.

Como citado previamente, a execução deste algoritmo é sobre duas classes – SA e *SimulatedAnnealing*. A classe SA representa a descrição dos métodos inerentes às características do circuito, como a obtenção do custo do estado atual do circuito, gerar a perturbação no circuito e desfazer esta em caso de não aceite. A classe *template SimulatedAnnealing* é responsável pelo controle de execução do algoritmo – vide Figura 4.5 –, ou seja, inclui o cálculo de uma nova temperatura e também da probabilidade de aceitação de um novo estado para o circuito.

4.1.2.1 Função de Perturbação

Segundo Hentschke (2002), uma função de perturbação deve possuir três propriedades fundamentais: Simplicidade, Aleatoriedade e Consistência.

Detalhando essas características básicas, percebe-se que a simplicidade dá-se ao fato de necessitar de perturbações com pequena granularidade – para possibilitar alcançar qualquer estado, e também de baixa complexidade computacional -, visto que grande parte do tempo de processamento é utilizada para realizar perturbações, e, no caso dessas não serem aceitas, é necessário desfazê-las.

Como o espaço de soluções é muito amplo, diversas possibilidades devem ser pesquisadas sem a necessidade de uma relação entre si, fato devido ao qual uma função de perturbação deve ser aleatória, assim evitando que as perturbações realizadas no circuito adquiram uma tendência, deturpando a eficiência do algoritmo.

Apesar de ser óbvia, é de extrema relevância a última propriedade, o fato de ser consistente a função remete em não alterar as características iniciais do circuito. Durante todo o processamento, o circuito deve-se manter consistente com o estado inicial, e.g. não deverá ser trocada uma célula de um tipo por uma que possua outra função lógica.

A função de perturbação desenvolvida tem, como objetivo principal, gerar novos estados do circuito em análise, a partir da tentativa de trocas de células comerciais por células geradas pela ferramenta ASTRAN.

Em um primeiro momento, é selecionada uma célula do circuito, obedecendo à premissa de aleatoriedade. Neste instante, é verificado se existe, no *pool* de células, geradas automaticamente, uma célula de mesmo tipo, caso exista é apurado a possibilidade trocar esta por uma célula ASTRAN com um fator de escala diferente - Figura 4.6, desde que não viole a restrição de *timing* imposta, significando que, caso ocorra a trocam essa célula ASTRAN não irá transformar nenhum caminho ao qual pertence em um novo caminho crítico, resultando em um acréscimo de atraso.

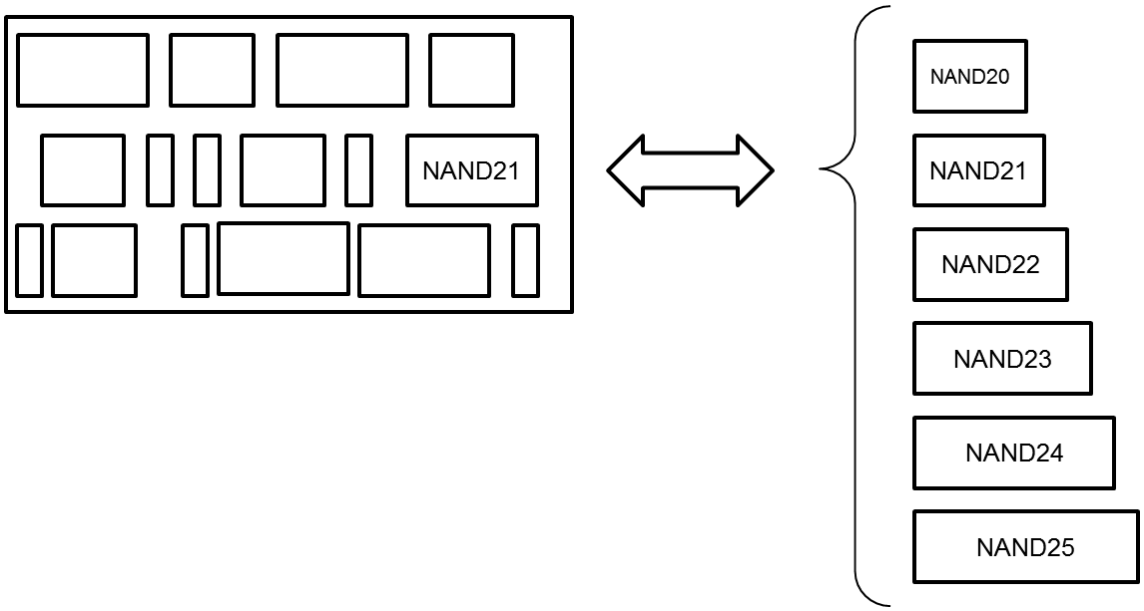


Figura 4.6: Funcionamento da Função de Perturbação.

No momento em que a restrição de atraso é obedecida, e a nova célula com o fator de escala, selecionada, é calculada a potência que essa irá consumir quando inserida naquele ambiente do circuito, ou seja, são levados em consideração todos os caminhos nos quais essa instância de célula foi inserida. Com o mesmo intuito, também é atualizado o novo valor da área ocupada. Com a atualização destas duas grandezas, potência consumida e área utilizada, é possível avaliar o quão bom é o novo estado do circuito, pois esses dois novo valores influenciam diretamente na função de custo do algoritmo.

Toda essa sequência de passos para a execução da função de perturbação é mostrada na Figura 4.7, onde é apresentado o fluxograma do processo. Vale ressaltar que, para a função de desfazer a perturbação para o caso de não aceite, esse processo não é utilizado, visto que a célula original excluída do circuito é armazenada temporariamente para ser recolocada no caso de falha.

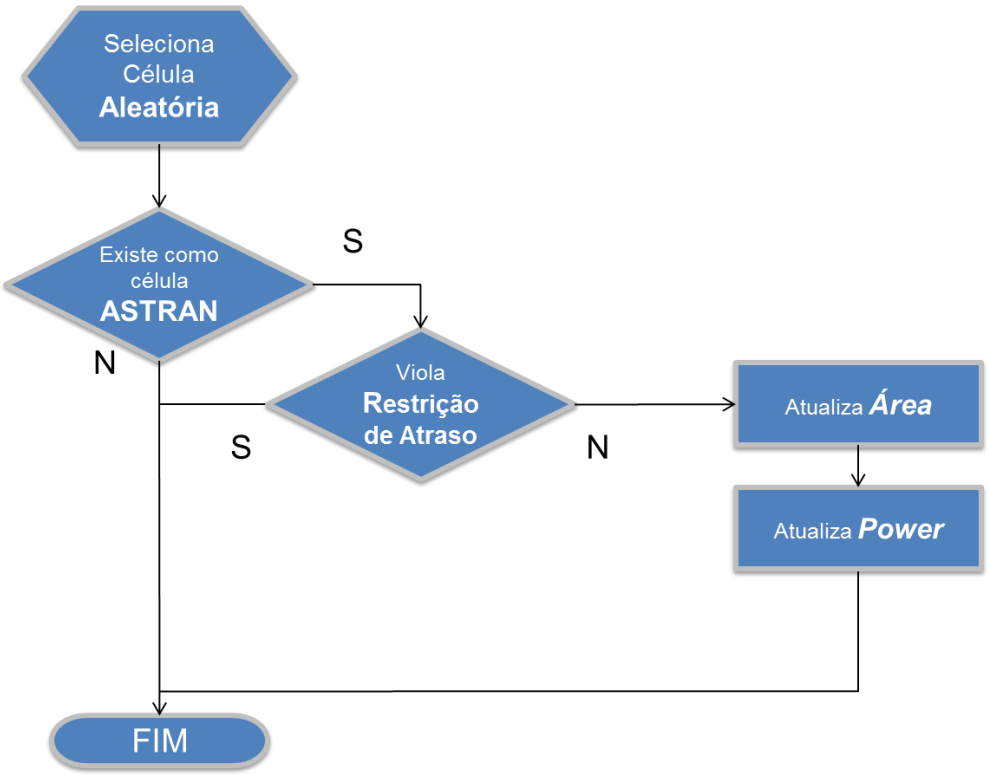


Figura 4.7: Fluxograma da Função de Perturbação.

4.1.2.2 Função de Custo

A função de custo consiste em indicar, através de uma única grandeza, a situação do estado do sistema, ou seja, deve conciliar todos os aspectos importantes a serem analisados em um único número inteiro, além de acompanhar a execução do algoritmo, indicando o sucesso ou falha das trocas de células sugestionadas, i.e. quanto maior o custo, pior se torna o circuito, e, no momento em que o custo começa a diminuir, indicar o melhoramento do circuito.

As grandezas selecionadas para serem incorporadas à função de custo são os dados de potência e área, sendo necessário combiná-los, a fim de resultar em apenas um único número inteiro. Potência total do circuito e área total das células foram selecionadas, pois representam diretamente o objetivo da ferramenta Substituidora desenvolvida, visto que o atraso é uma restrição com apenas o objetivo de mantê-lo idêntico ao valor do circuito original.

É evidente que um grande problema da função de custo é a composição de um único número a partir de duas grandezas distintas, com unidades de medida distintas. Para solucionar tal impasse, o valor de retorno da função de custo é um acompanhamento do acréscimo ou decréscimo percentual dos valores de área e potência.

Em um primeiro momento, ao custo do circuito inicial, é atribuído um valor inicial. Após uma perturbação realizada no circuito, supõe-se que ocorreu um acréscimo de área igual a *3%* e um decréscimo de potência equivalente a *8%*. Estes dois valores são convertidos para um número decimal, e sua diferença é somada ao custo inicial estipulado.

É exemplificado, na Figura 4.8 o funcionamento da função de custo. Nessa figura, é apresentada uma perturbação que resulta na troca de duas células (NAND21 trocada pela

NAND20), com os valores de ganho e perda percentuais para área e potência. Vale salientar que todos os valores são hipotéticos, bem como o comportamento das células.

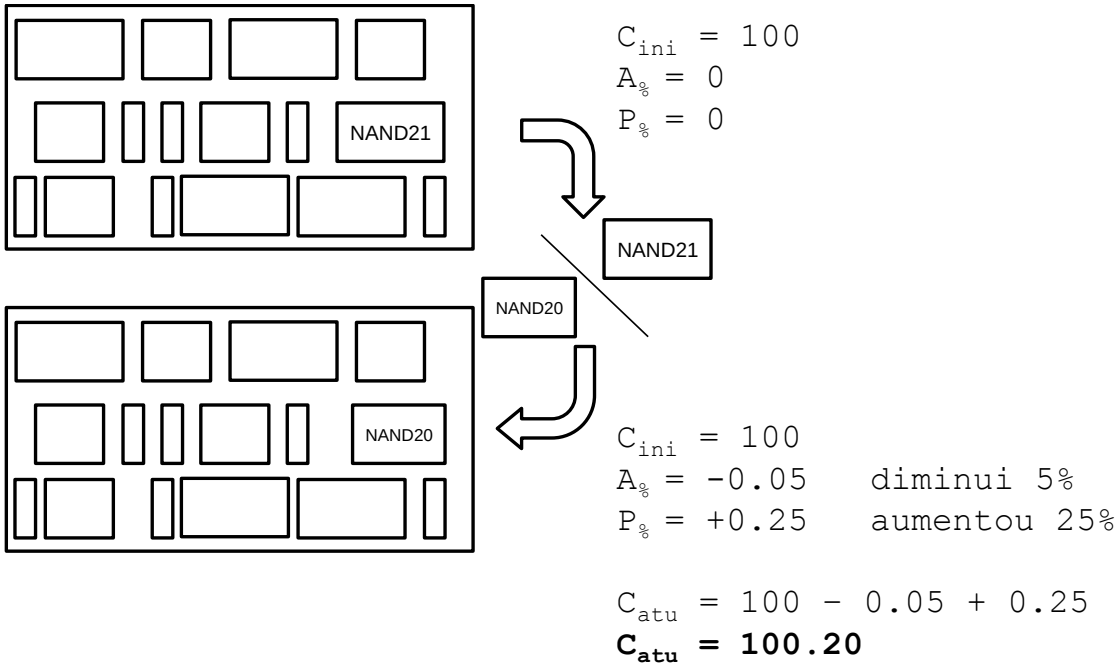


Figura 4.8: Exemplo de cálculo da Função de Custo.

Podemos observar que a troca de células na Figura 4.8 resulta em um acréscimo de *25%* na potência total consumida pelo circuito e um decréscimo de *5%* na área total das células empregadas no circuito. Conforme mostrado na figura, o custo inicial do circuito, que antes era *100*, com a troca, aumentou para *100.20*, indicando uma troca malsucedida.

4.1.2.3 Função para determinação da Temperatura - Schedule

A função de *Schedule* computa o valor da temperatura, determinando seu valor desde o início do processo até o fim da execução do algoritmo, além de, como mencionado anteriormente, fazer parte da probabilidade de aceitação dos estados intermediários gerados pelas perturbações.

Para Lam (1988), uma boa função de *Schedule* deve alcançar o compromisso entre qualidade da solução final e o tempo de computação. Neste trabalho, foi utilizada uma implementação para o cálculo de temperatura feita por Hentschke (2002), a qual foi baseada no trabalho de Lam (1988).

De maneira simples, o funcionamento desta função de *Schedule* leva em consideração a temperatura anterior, a taxa de aceitação de perturbações, ou seja, a relação entre o número de iterações e a quantidade de perturbações aceitas, por último, o desvio padrão da variação do custo ao decorrer das iterações.

5 UM NOVO FLUXO DE PROJETO PARA CIRCUITOS INTEGRADOS

Através do esforço de integrar dois fluxos de síntese de circuitos digitais distintos – fluxo comercial e fluxo acadêmico –, utilizaram-se diversas ferramentas, e outras foram desenvolvidas. O objetivo, que era de que todas pertencessem ao mesmo fluxo de projeto, resultou na criação de um novo fluxo de projeto de circuitos integrados.

O dito novo fluxo de projeto de circuitos integrados consiste no principal fluxo desenvolvido neste trabalho, o qual se refere, principalmente, à inserção, no fluxo comercial, da ferramenta Substituidora desenvolvida. No entanto, outros fluxos, chamados preliminares, também foram desenvolvidos, com o intuito de servir como base preparatória para o fluxo principal.

5.1 Fluxos Preliminares

Como foi amplamente citado anteriormente, este trabalho tem, como objetivo, a exploração do espaço de projeto envolvendo células geradas automaticamente pela ferramenta acadêmica ASTRAN, contudo, existe uma grande distância entre um desenho de uma célula gerada através de uma ferramenta e uma biblioteca de células comerciais pronta para ser utilizada pelas ferramentas de síntese lógica e física. Com isso, foi necessário o desenvolvimento de um processo para possibilitar a inclusão de células ASTRAN em um fluxo de síntese utilizando ferramentas comerciais, o que já foi apresentado neste trabalho, mas será detalhado nas seções a seguir.

5.1.1 Caracterização de Células Geradas pelo ASTRAN

Este processo visa a colocar os dois tipos de células abordados neste trabalho no mesmo patamar, ou seja, como as células comerciais são utilizadas, na maioria das vezes, no formato de uma biblioteca (arquivo com estrutura LIBERTY), no caso das células ASTRAN geradas automaticamente, fez-se necessário desenvolver uma sequência de passos, utilizando-se diversas ferramentas para ser possível, partindo de um desenho, elaborar uma biblioteca de células contendo apenas células ASTRAN.

Anterior a este capítulo, foi apresentada a Figura 3.7, mostrando uma simplificação do processo de caracterização de células ASTRAN, explicitando, em detalhes, a funcionalidade de cada bloco.

Na Figura 5.1, mostram-se quais ferramentas são utilizadas por cada bloco funcional em analogia à Figura 3.7; em um primeiro momento, as células são geradas pelo *software* ASTRAN – conforme fluxo apresentado na Figura 2.6, logo, a partir dos arquivos de saída gerados pelo ASTRAN, serão fornecidos, à entrada da ferramenta *Cadence Virtuoso*® , dados para verificação da geometria da célula e outras validações, e, por fim, será gerado o leiaute extraído, que consiste em uma descrição do subcircuito da célula, em consonância com a tecnologia empregada no projeto. Um exemplo de arquivo do leiaute extraído da célula pode ser observado na Figura 5.2, consistindo em um arquivo texto no formato *SPICE*, detalhando a área dos transistores, conexões, além de pinos de entrada, saída e alimentação.

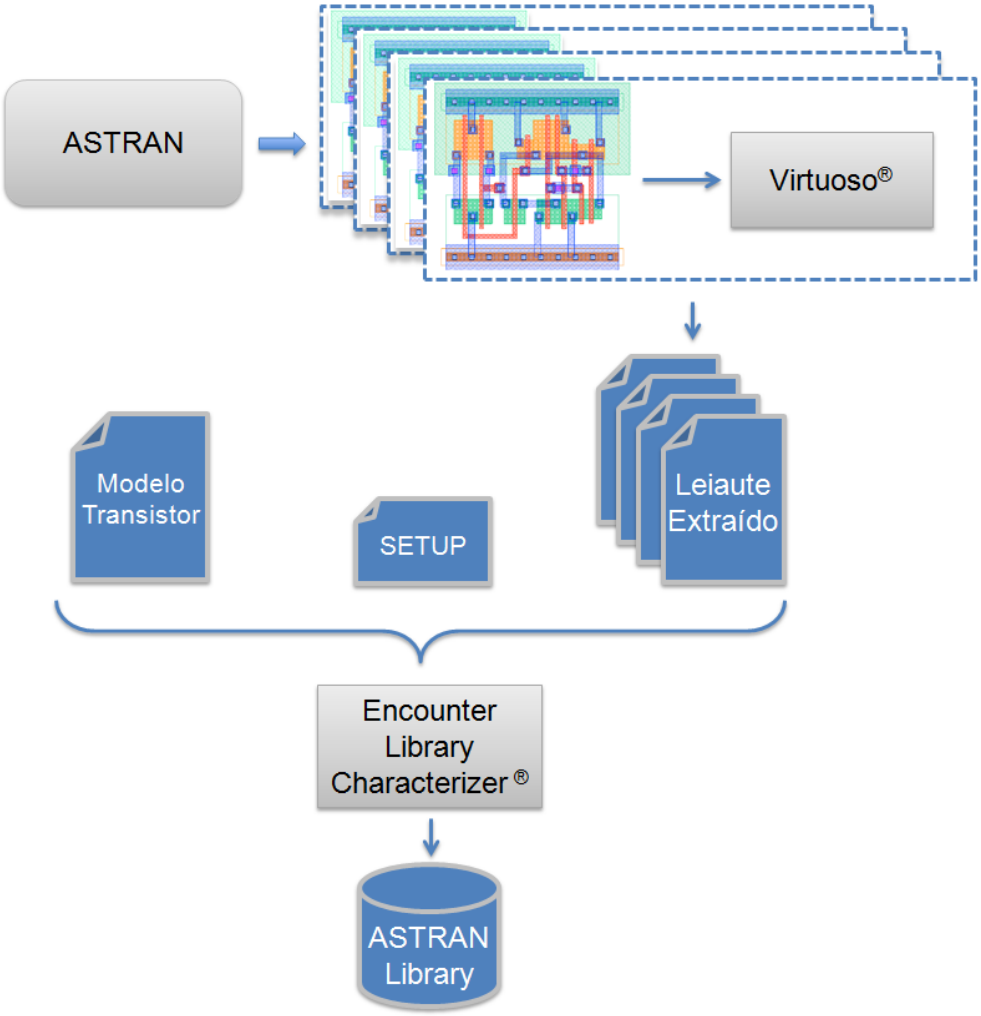


Figura 5.1: Fluxo detalhado de Caracterização de Células

Como é observado no fluxo detalhado de caracterização de células, além do conjunto de arquivos descrevendo o leiaute extraído das células, o *Cadence Encounter Library Characterizer*® também necessita de dois outros arquivos de entrada, o arquivo de *SETUP* e o arquivo em formato *SPICE*, que descreve o modelo elétrico do transistor para a tecnologia adotada.

Esses três arquivos, que servem como entrada para a execução do ELC - *Encounter Library Characterizer*, são apresentados nas figuras seguintes. O leiaute extraído, Figura 5.2, contém informações sobre uma célula de função lógica referente a um inversor (INV1) e fator de escala igual a 1, informações são apresentadas na primeira linha, juntamente com os pinos pertencentes à célula. No bloco seguinte, é mostrada a relação entre esses pinos, bem como suas características, e, por último, são detalhados dados dos transistores que compõem a célula.

```
.SUBCKT INV1 A Q VDD GND
C0 A GND  625.88125E-18 M=1.0
C1 A Q  292.906226685796E-18 M=1.0
C2 VDD Q  448.547175324676E-18 M=1.0
C3 VDD A  1.42635E-15 M=1.0
X4 GND VDD  NWD  AREA=38.180000827559E-12 PJ=25.8000000030734E-6
M=1.0
M5 Q A VDD VDD  MODP  L=349.999993431993E-9 W=800.000009348878E-9
+AD=779.999979536039E-15 AS=759.999973110742E-15
PD=2.75000002147863E-6
+PS=2.70000009550131E-6 NRD=625E-3 NRS=625E-3 M=1.0
M6 GND A Q GND  MODN  L=349.999993431993E-9 W=499.999998737621E-9
+AD=602.499983497901E-15 AS=602.499983497901E-15
PD=2.75000002147863E-6
+PS=2.75000002147863E-6 NRD=1 NRS=1 M=1.0
.ENDS INV1
```

Figura 5.2: Exemplo arquivo do Leiaute Extraído.

A Figura 5.3 apresenta as características de um transistor tipo N, são descritos diversos parâmetros que respeitam o modelo elétrico do transistor. Por ser um arquivo muito extenso, apenas alguns parâmetros do transistor N são mostrados, mas o arquivo completo também descreve os dados do transistor P.

```
.MODEL MODN NMOS LEVEL=49
+MOBMOD =1.000e+00 CAPMOD =2.000e+00
+NOIMOD =3.000e+00
+VERSION=3.11
+K1     =5.0296e-01
+K2     =3.3985e-02    K3     =-1.136e+00  K3B    =-4.399e-01
+NCH    =2.611e+17     VTH0   =4.979e-01
+VOFF   =-8.925e-02    DVT0   =5.000e+01   DVT1   =1.039e+00
+DVT2   =-8.375e-03    KETA   =2.032e-02
+PSCBE1 =3.518e+08     PSCBE2 =7.491e-05
+DVT0W  =1.089e-01     DVT1W  =6.671e+04   DVT2W  =-1.352e-02
+UA     =4.705e-12     UB     =2.137e-18   UC     =1.000e-20
+U0     =4.758e+02
```

Figura 5.3: Exemplo do arquivo com o Modelo Elétrico do Transistor.

Para controlar a forma de execução do ELC e como ele irá utilizar os dados fornecidos pelos dois outros arquivos, é necessário o arquivo de *SETUP*. Na Figura 5.4, é mostrado um trecho inicial desse arquivo, em cujo primeiro bloco, são descritas características do processo, como algumas tensões de referência e temperatura. Já no segundo bloco, são apresentados alguns pontos de medição para a realização das simulações responsáveis pela caracterização das células. No restante do arquivo, constam dados relativos a alguns cenários de simulação, ou seja, à manipulação do comportamento das formas de onda de entrada das células, os chamados vetores de simulação.

```
Process typical {
    voltage = 3.3;          // as voltage
    temp    = 25 ;          // as temperature
    Vtn     = 0.498 ;       // nmos Vt
    Vtp     = 0.691 ;       // pmos Vt
} ;
Signal std_cell {
    unit    = REL ;         // relative value
    Vh      = 1.0 1.0 ;     // 100% rise/fall
    Vl      = 0.0 0.0 ;
    Vth     = 0.5 0.5 ;     // 50% rise/fall
    Vsh     = 0.9 0.9 ;
    Vsl     = 0.1 0.1 ;
    tsmax   = 2.0n ;        // maximum output slew rate
} ;
```

Figura 5.4: Exemplo do arquivo de SETUP.

5.1.2 Fluxo de Treino das Redes Neurais Artificiais

Na Figura 3.6, foi apresentado um diagrama esquemático, explicitando os passos que compõem o fluxo para o treinamento das redes neurais. Já na Figura 5.5, é o mesmo fluxo da Figura 3.6, utilizando o nome das ferramentas utilizadas pelo processo. O treino de RNA é necessário para mapear o comportamento das células geradas automaticamente pelo ASTRAN, conforme explicado no capítulo 3. Tendo em vista isso, a ferramenta utilizada para realizar a síntese lógica, na qual serão gerados os relatórios de atraso e potência, é a *Cadence RTL Compiler*®.

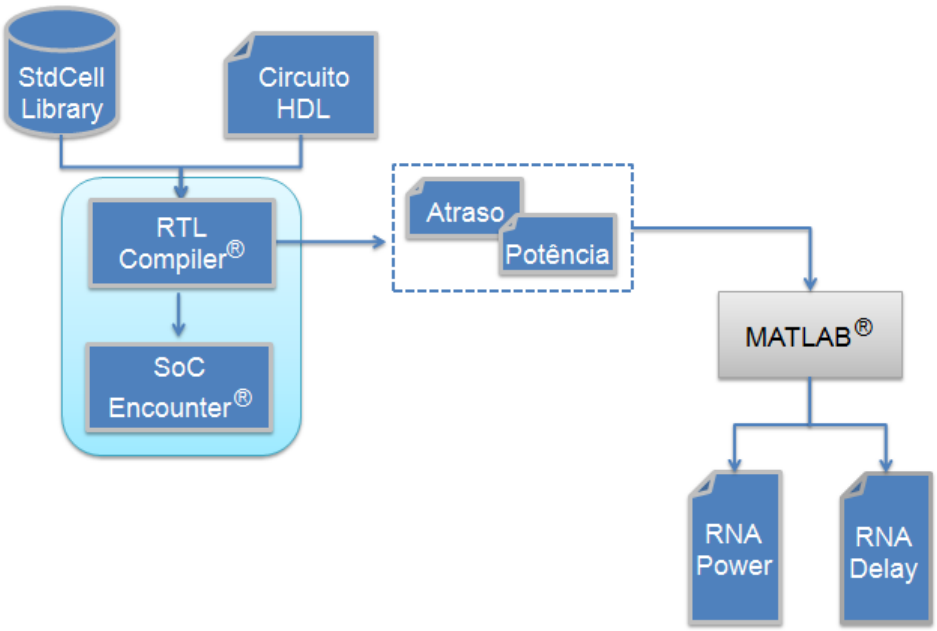


Figura 5.5: Fluxo para Treinamento de RNA.

Após os relatórios oriundos do *RTL Compiler serem gerados*, eles servem de entrada para o *MathWorks MATLAB®*, onde é utilizado um *toolkit* referente ao trabalho com redes neurais artificiais. Após obter os dados dos relatórios da síntese lógica, estes são tratados e fornecidos como entrada para o *MATLAB®*. Na Figura 5.6, é possível observar a tela do *toolkit* para o acompanhamento da execução do treinamento das redes neurais.

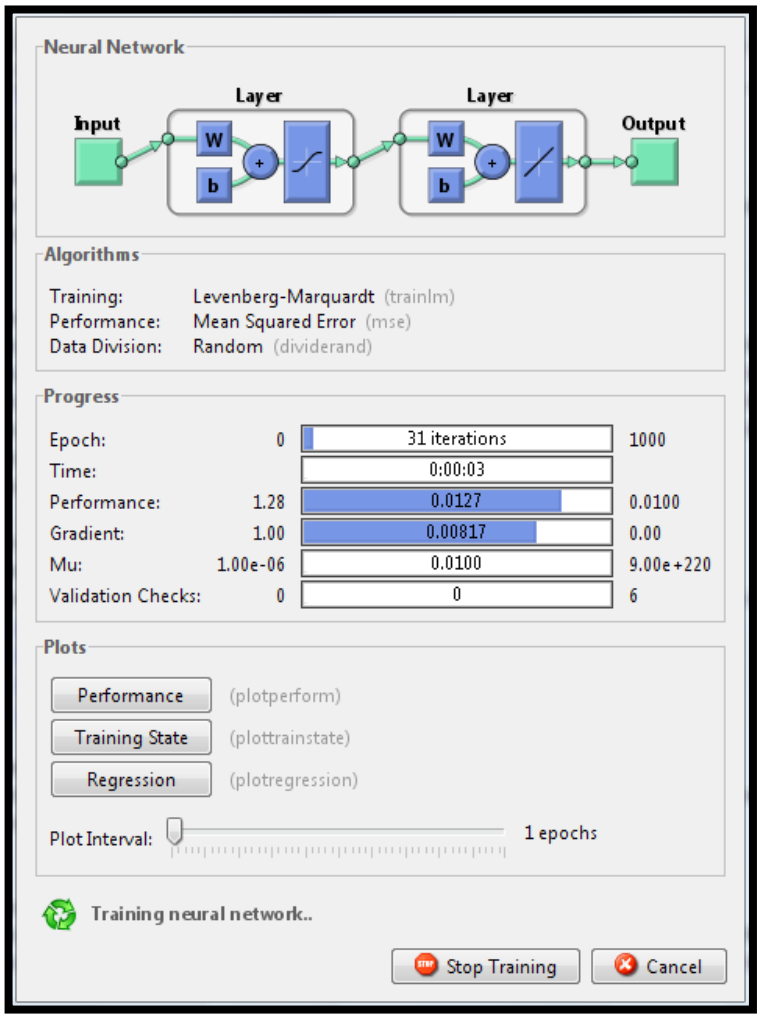


Figura 5.6: Treinamento da RNA pelo MATLAB®.

Após a execução do algoritmo de treinamento das redes neurais, são gerados dois arquivos em formato texto, um indicando os pesos dos neurônios para a RNA responsável por prover a potência da célula, e outro arquivo análogo, com o intuito de fornecer o atraso das células.

Esse fluxo para o treinamento das redes neurais artificiais é realizado após a realização da caracterização das células geradas pelo ASTRAN descrita na seção anterior - Caracterização de Células Geradas pelo ASTRAN.

5.2 Fluxo Principal

Para propiciar a utilização de células geradas automaticamente por ferramentas comerciais, é necessário percorrer uma sequência de passos, descritos anteriormente neste trabalho. No primeiro momento, após a geração dos leiautes pelo ASTRAN, é necessário fazer a união do desenho do leiaute com a tecnologia a ser utilizada e, em seguida, a composição de uma biblioteca de células conforme o modelo comercial adotado; esses

passos correspondem ao fluxo de caracterização de células. No seguimento, para ser possível utilizar a ferramenta Substituidora desenvolvida, é necessária a realização do segundo fluxo responsável pelo treinamento das redes neurais artificiais.

Um ponto importante a ser salientado é o fato que a biblioteca comercial utilizada no fluxo principal deverá ser recaracterizada utilizando-se dos mesmos parâmetros empregados na criação da biblioteca de células geradas pelo ASTRAN. Isso se faz necessário, pois não se tem como comparar duas bibliotecas de células e observar possíveis diferenças nos parâmetros de caracterização.

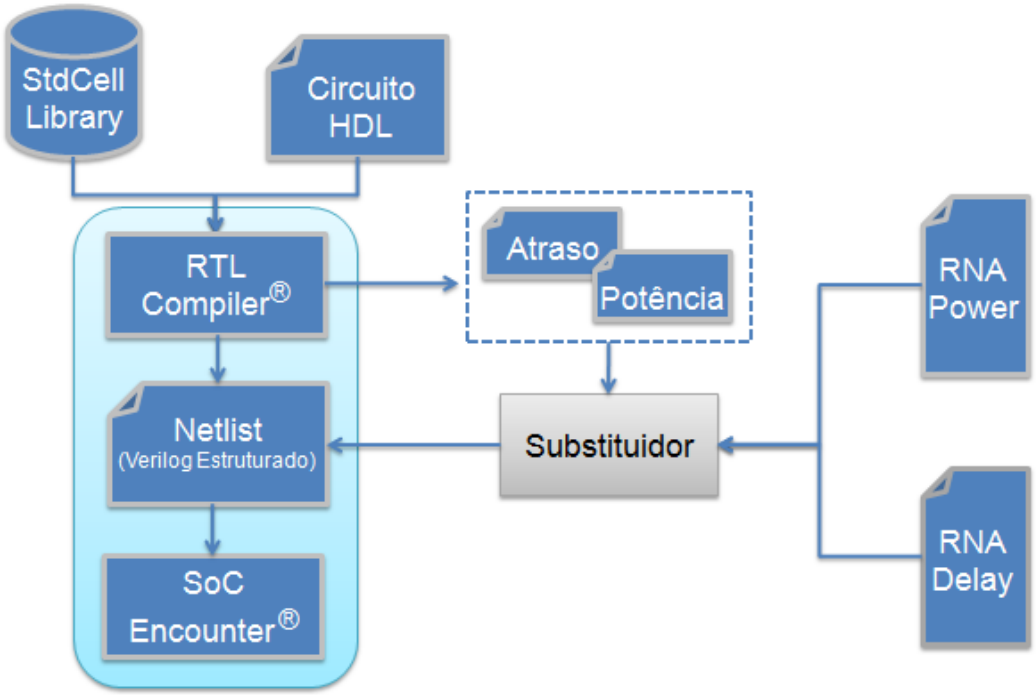


Figura 5.7: Fluxo Principal - Utilização do Substituidor de células.

Na Figura 5.7, é apresentado o fluxo principal desenvolvido, que compreende a realização da síntese lógica através da ferramenta *Cadence RTL Compiler®*, que é responsável por produzir os relatórios de síntese para a ferramenta desenvolvida, bem como fornecer o *netlist* para a próxima etapa: a síntese física.

A síntese lógica tem, como entrada, o circuito em uma descrição HDL, a biblioteca de células lógicas e também a especificação das restrições do projeto. Na Figura 5.8, é possível observar um trecho do *script* responsável por controlar a execução do *RTL Compiler®*, em destaque, a parte em que se definem as restrições para a realização da síntese lógica.

Ainda no *script* da síntese lógica, constam a atribuição da biblioteca de células a ser utilizada, além dos comandos tratamentos para a geração dos relatórios. Por último, é adicionado o comando para o fornecimento dos arquivos de entrada para a próxima etapa do processo, que consiste na realização da síntese física.

```
set inv_out INV10/Q

load ../../rtl/${DESIGN}.v
elaborate

redirect /dev/null {
  set clock [define_clock -p 0 -n clock [clock_ports]]
  external_delay -i 0 -c $clock /designs/*/ports_in/*
  external_delay -o 0 -c $clock /designs/*/ports_out/*
  if {[info exists std_load]} {
    set_attr external_wire_cap [expr 6 * $std_load] /des*/*/ports_out/*
  }

  set allins [find / -port ports_in/*]
  set clks [lsearch -glob $allins [clock_ports]]
  set noclksin [lreplace $allins $clks $clks]
  set_attr external_driver $inv_out $noclksin
}

set errorInfo ""

ungroup -flatten -all

synthesize -to_mapped -effort low
synthesize -incremental -effort high
```

Figura 5.8: Exemplo do *script* para Síntese Lógica.

A síntese física é realizada pela ferramenta *Cadence SoC Encounter*®, que compreende em executar as etapas de posicionamento de células, roteamento de conexões, projeto da distribuição da rede de relógio e outros processos para otimização do circuito e viabilização de sua fabricação.

O *SoC Encounter*® em resumo consiste em uma aglutinação de algoritmos que visam a otimização do circuito de maneira automática nas mais diversas etapas do processo de síntese física, como também possui um ambiente gráfico que possibilita a visualização do circuito em desenvolvimento e consequente interferência no projeto por um projetista de circuitos experiente.

Um exemplo dessa interface da ferramenta de síntese física pode ser observado na Figura 5.9, em que notamos, ao redor do circuito, os anéis de alimentação e, oriundos deles, as bandas de alimentação, cruzando a área do circuito. No interior dos anéis, percebem-se diversas caixas retangulares divididas em bandas, que são as células empregadas no projeto do circuito. As células são conectadas em diversas redes: dados, relógio e controle. Ainda ressaltamos redes que se aportam nas extremidades do circuito, que têm o objetivo de capturar estímulos externos através dos pinos de entrada e saída.

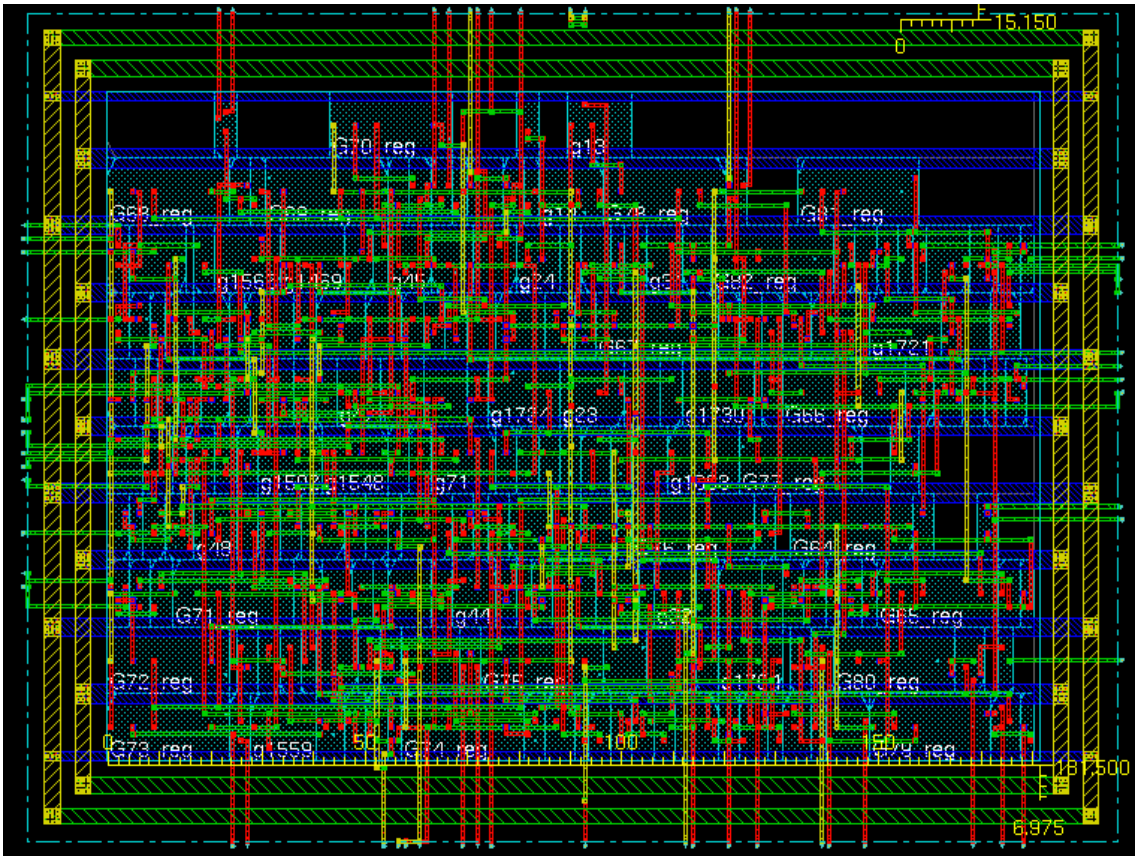


Figura 5.9: Exemplo de uma Síntese Física no *SoC Encounter*®.

6 RESULTADOS

Esta seção apresenta os resultados obtidos utilizando o fluxo de projeto desenvolvido neste trabalho. Vale ressaltar que os valores apresentados estão altamente relacionados com a geração das células realizada pela ferramenta ASTRAN, e visa a uma análise sobre o comportamento dessas células inseridas em um circuito misto, ou seja, junto a células *Standard Cells* de uma biblioteca comercial.

Outro ponto importante na análise dos resultados é relativo à concepção da ferramenta Substituidora (Capítulo 4), no caso, construída de tal forma que se propõe a manter o *delay* total na troca de células, e diminuir a área total e potência consumida.

Uma questão que também se deve levar em consideração é que a ferramenta Substituidora utiliza-se de valores obtidos a partir dos relatórios da síntese lógica, no entanto, os resultados alcançados são adquiridos na síntese física. A decisão por obter os resultados na etapa de síntese física é devido ao fato de observar as células geradas pelo ASTRAN em um comportamento mais próximo à realidade, é claro que dentro das limitações de uma ferramenta de CAD.

Todos os circuitos de *benchmarks* utilizados pertencem ao conjunto ISCAS/89 (IWLS, 2005) e foram sintetizados na tecnologia de 350*nm*.

6.1 Comparação entre Circuitos Comerciais e Circuitos Mistos

Primeiramente, os circuitos do *benchmark* foram sintetizados utilizando apenas células comerciais, com a ferramenta *RTL Compiler*. Logo após esta etapa realizada e gerados os relatórios da síntese lógica, foi utilizada a ferramenta Substituidora desenvolvida, e, após a utilização dela, dois circuitos são obtidos, um contendo apenas células comerciais, e outro, que possui algumas células geradas pelo ASTRAN. Com esses dois circuitos, é realizada a síntese física, com a ferramenta *SoC Encounter*, e os resultados obtidos de área total do *chip*, potência consumida e atraso do caminho crítico são comparados entre os dois circuitos.

Para a realização de uma síntese física aceitável, é imprescindível deter atenção a vários pontos, tais como:

- O projeto dos *Power rings*, que são responsáveis pela distribuição de energia por todo o *chip*;
- O método adotado para distribuição do relógio de todo o circuito, observando-o, para não ocorrer grandes atrasos entre todas as ramificações;
- Algoritmo utilizado para o posicionamento de células dentro do circuito integrado e
- A fase de roteamento, que também pode ser realizada por diversos algoritmos dentro da ferramenta *SoC Encounter*.

Levando em consideração todos os pontos apresentados acima, para os resultados obtidos neste trabalho, utilizaram-se os mesmos parâmetros para todos os itens enumerados, visando assim a um cenário o mais aproximado possível entre a síntese física dos circuitos com apenas células comerciais e os circuitos mistos.

Abaixo, na Tabela 6.1, podem ser observados os resultados de *delay* obtidos para um conjunto de circuitos ISCAS/89 em duas situações diferentes, a primeira, denominada ‘Original’, pois possui apenas células comerciais, sem intervenção da ferramenta Substituidora desenvolvida. Os circuitos chamados ‘Misto’ são compostos por células comerciais e também por células geradas automaticamente pelo ASTRAN.

Previamente à análise dos resultados, vale ressaltar que o algoritmo de otimização da ferramenta Substituidora tem, como restrição, a manutenção do atraso máximo do circuito e visa a minimizar a potência total consumida e a área total utilizada. Devido a este requisito estabelecido, pode-se observar que, para alguns circuitos, o circuito ‘Misto’ teve um pior resultado em relação ao ‘Original’. No entanto, a média do ganho obtido sobre os circuitos analisados é -0,64%.

Tabela 6.1: Resultados relativo ao Atraso Máximo dos Circuitos.

Circuito	Fluxo de Síntese		Ganho
	Original (*ns*)	Misto (*ns*)	(*%*)
s386	2,112	2,096	0,75
s820	2,627	2,811	-7,00
s832	2,822	2,921	-3,50
s1196	7,281	7,409	-1,75
s13207	130,589	130,328	0,19
s1238	7,418	7,326	1,24
s38417	6,341	5,808	8,40
s15850	61,591	63,745	-3,49

A perda de desempenho em alguns caso é em consequência do erro intrínseco da etapa de síntese lógica comparado à maior proximidade da fase de síntese física com o circuito real. Na maioria dos casos, o caminho crítico do circuito na fase de síntese física não é o mesmo na etapa de síntese lógica, não obstante, o caminho crítico no circuito ‘Original’ e no circuito ‘Misto’ pós síntese física também não são iguais.

Como exemplo dessa discrepância entre o cenário de síntese lógica e síntese física, é tomado o circuito ‘s38417’ como amostra, que obteve decrésimo no atraso geral, para a configuração ‘Misto’, o caminho crítico (1º em atraso) na fase de síntese física não era o mesmo na fase de síntese lógica, e sim apenas o 1483º pior caminho em atraso. Já para configuração ‘Original’, o pior caminho (caminho crítico, 1º em atraso) na síntese física, na síntese lógica, não estava dentro dos 45.000 piores caminhos analisados pela ferramenta Substituidora.

Já na Tabela 6.2, são apresentados os resultados finais referentes à potência consumida pelos circuitos analisados. É importante ressalvar que a potência de um circuito é encarada como uma variável de minimização pelo algoritmo da ferramenta Substituidora, isto é, a ferramenta visa a melhorar – diminuir, o consumo de potência pelo circuito avaliado.

Tabela 6.2: Relativo ao consumo de Potência dos Circuitos.

Circuito	Fluxo de Síntese		Ganho
	Original (μW)	Misto (μW)	(%)
s386	0,5558	0,5225	5,99
s820	1,6800	1,6000	4,76
s832	1,5840	1,5250	3,72
s1196	2,8910	2,8010	3,11
s13207	1,9620	1,8570	5,35
s1238	3,1410	2,9500	6,08
s38417	33,0600	32,4500	1,84
s15850	1,0460	1,0530	-0,66

Como consequência do algoritmo de otimização da ferramenta Substituidora, é possível observar que, em apenas um dos circuitos amostrados, não ocorreu diminuição da potência consumida pelo circuito. Na média, os circuitos da Tabela 6.2 diminuíram em 3,77% o consumo de potência. O circuito ‘s1238’ obteve a maior redução (6,08%), e o circuito ‘s15850’ teve o pior desempenho, com o aumento de 0,66% da potência consumida.

Na Tabela 6.3, a potência consumida pelos circuitos com maior e menor ganho é detalhada, apontando o consumo interno, consumo por chaveamento e por corrente de fuga da célula. Das três parcelas, pode-se notar que o consumo interno é melhorado nos dois casos, refletindo diretamente na troca de células entre os circuitos. Já a potência consumida pelo chaveamento das células mostra uma consequência secundária, visto que ela foi o ponto discrepante entre os dois circuitos. Adotada como hipótese inicial para a confecção da ferramenta Substituidora, o intuito de troca de células visa não só a um ganho de cada célula individualmente, mas a um ganho geral do circuito, viabilizando um melhor roteamento e posicionamento, dentre outras fases da síntese física.

Tabela 6.3: Composição da Potência Consumida pelo Circuitos.

Circuito	Fluxo de Síntese						Ganho
	Original (μW)			Misto (μW)			(%)
	Interno	Chaveamento	Fuga	Interno	Chaveamento	Fuga	
s1238	1,656	1,485	1,70e-5	1,543	1,406	1,72e-5	6,08
s15850	0,442	0,603	1,80e-5	0,433	0,619	1,81e-5	-0,66

Por último, a Tabela 6.4 apresenta a área total ocupada pelo circuito, lembrando que a ferramenta Substituidora também julga a área do circuito como uma variável de minimização. O circuito ‘s820’ obteve o maior ganho, com uma redução de 2,51% da área total do circuito. Já o circuito ‘s15850’ se manteve com a mesma área, independente das trocas ocasionadas pela ferramenta Substituidora. Na média, os circuitos analisados obtiveram um ganho de 1,25%.

Tabela 6.4: Área Total ocupada pelos Circuitos.

Circuito	Fluxo de Síntese		Ganho
	Original (μm^2)	Misto (μm^2)	(%)
s386	23.595,00	23.171,00	1,79
s820	47.899,00	46.696,00	2,51
s832	46.996,00	46.397,00	1,27
s1196	65.818,00	64.398,00	2,15
s13207	278.890,00	277.520,00	0,49
s1238	78.716,00	77.376,00	1,70
s38417	1.614.900,00	1.613.100,00	0,11
s15850	145.820,00	145.820,00	0,00

7 CONCLUSÃO

Este trabalho apresentou um novo fluxo de projeto para confecção de circuitos integrados digitais, utilizando células geradas automaticamente. Para tal, foram desenvolvidos um novo método de comparação entre células utilizando Redes Neurais Artificiais e uma ferramenta capaz de avaliar o circuito atual e decidir sobre a troca de uma célula comercial por uma gerada automaticamente pelo ASTRAN.

O novo fluxo de projeto previamente descrito visa, além de ampliar a utilização de células acadêmicas geradas pelo ASTRAN, inserindo estas em fluxo comercial, a mostrar, de maneira mais ampla, os impactos gerados sobre o circuito em consequência da troca de células, isto é, avaliando os resultados nas últimas fases inclusas na síntese física.

Este texto está inserido no contexto da síntese de circuitos digitais, partindo da geração de células de forma automática (ASTRAN), além da caracterização de células para montagem de bibliotecas de células, até o desenvolvimento de um novo tipo de ferramenta de CAD utilizada entre as etapas de síntese lógica e física. Para isso, foi desenvolvida uma nova maneira de comparação entre células, utilizando Redes Neurais Artificiais, e também, uma nova ferramenta de CAD (Substituidora) para a troca de células comerciais por células geradas automaticamente pelo ASTRAN.

Como visto na seção de resultados, o novo fluxo de síntese resultou em circuitos, em média, com 3,77% menor consumo de potência e 1,25% menos área consumida, tendo, apenas, um encargo de 0,64% por parte do atraso total do circuito (GUIMARÃES JR, 2015). Esses resultados demonstram um caminho promissor para o uso de células geradas automaticamente em um fluxo comercial, tendo em vista que a base para um futuro processo incremental utilizado para o desenvolvimento de novas e melhores células geradas automaticamente foi estabelecido.

7.1 Trabalhos Futuros

Como citado anteriormente ao longo dos capítulos, foi descrito um novo fluxo de projeto, um fluxo de base para o aprimoramento das células automaticamente geradas, então, como trabalho futuro, propõe-se o teste de novos tipos de células lógicas, bem como células geradas em outras tecnologias, tais como 180nm, 90nm entre outras.

REFERÊNCIAS

ALPAYDIN, E. **Introduction to Machine Learning.** 2nd ed. Cambridge: MIT Press, 2004.

CADENCE. Cadence Design Systems. Disponível em: <http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx>. Acesso em Jan. 2012.

DAS, B.P.; JANAKIRAMAN, V.; AMRUTUR, B.; JAMADAGNI, H.S.; ARVIND, N.V. Voltage and Temperature Scalable Gate Delay and Slew Models Including Intra-Gate Variations. In: *VLSI Design, 2008. VLSID 2008. 21st International Conference on* , vol., no., pp.685,691, 4-8 Jan. 2008

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

FLEX. Flex: The Fast Lexical Analyzer. Disponível em: <http://flex.sourceforge.net/>. Acesso em Jan. 2012.

FUNABIKIY, N.; TAKEFUJI, Y. A parallel algorithm for channel routing problems [VLSI]. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.11, no.4, pp.464,474, Apr 1992.

GÜNTZEL, J. **Geração de Circuitos Utilizando Matrizes de Células Pré-difundidas.** 1993. 174f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

GUIMARÃES JR, D. S. et al. Estimador de Potência e Atraso em Standard-Cells Utilizando Redes Neurais Artificiais. **XVI Workshop Iberchip.** Foz do Iguaçu, Brasil, 2010.

GUIMARÃES JR, D. S.; PUGET J.; REIS R. A. A mixed cells physical design approach. In: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, Lisbon, 2015, pp.1446-1449. Doi: 10.1109/ISCAS.2015.7168916.

HAYKIN, S. **Neural Networks – A Comprehensive Foundation.** 2nd ed. New York: Macmillian, 1994.

HAYKIN, S. **Neural Networks and Machine Learning.** 3nd ed. New Jersey: Person, 2008.

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

HOSSEINI, S.B.; SHAHABI, A; SOHOFI, H.; NAVABI, Z. A reconfigurable online BIST for combinational hardware using digital neural networks. In: *Test Symposium (ETS), 2010 15th IEEE European* , vol., no., pp.139,144, 24-28 May 2010.

IWLS 2005 benchmarks. Disponível em: < http://iwls.org/iwls2005/benchmarks.html.>. Acesso em Dez. 2009.

LAM, J.; DELOSME, J. Performance of a new annealing schedule. In: DESIGN AUTOMATION CONFERENCE, DAC, 25., 1988. **Proceedings...** New York: IEEE, 1988.

JANAKIRAMAN, V.; BHARADWAJ, A; VISVANATHAN, V. Voltage and Temperature Aware Statistical Leakage Analysis Framework Using Artificial Neural Networks. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.29, no.7, pp.1056,1069, July 2010.

KARIAT, V. ECSM 2.0 Static Modeling Format Standard. Disponível em: <http://www.si2.org/events_dir/2006/oaconfspring2006/cadenceecsm.pdf>.Acesso em Nov. 2010.

KINDEL, M. **Síntese Automática de Células CMOS.** 1997. 79f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

LAZZARI, C. **Automatic Layout Generation of Static CMOS Circuits Targeting Delay and Power Reduction**. 2003. 113f. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

LIMA, F. **Projeto com Matrizes de Células Lógicas Programáveis.** 1999. 124f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

LPSOLVE. Disponível em: <http://sourceforge.net/projects/lpsolve>. Acesso em: 27 set. 2010.

LUBASZEWSKI, M. **Geração Automática de Lógica Aleatória Utilizando a Metodologia TRANCA.** 1990. 232f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MATLAB. Levenberg-Marquardt backpropagation. Disponível em: <http://www.mathworks.com/help/toolbox/nnet/ref/trainlm.html>. Acesso em Jan. 2012.

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

MEINHARDT, C. **Geração de Leiautes Regulares Baseados em Matrizes de Células**. 2006. 131f. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

MENEZES, C. C. **Geração Automática de Leiaute através de uma Matriz de Células NAND – MARTELO**. 2004. 53f. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

MORAES, F. **Synthese Topologique de Macro-Cellules em Technologie CMOS.** 1994. 180f. Tese (Doutorado em Ciência da Computação) Universidade de MontPellier II, Glenoble.

MORAES, F. **TRAGO – Síntese Automática de Leiaute para Circuitos em Lógica Aleatória.** 1990. 199f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

OHTA, M. An algorithm for multi-layer channel routing problem using chaotic neural networks. In: *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on* , vol.5, no., pp.149,152 vol.5, 2000.

POSSER, G. et al. Automatic Cell Layouts Generation Using the ASTRAN Tool. **XXV SIM – South Symposium on Microelectronics,** Porto Alegre, Brasil, 2010.

RABAEY, J. M. **Digital Integrated Circuits: A Design Perspective**. 2nd ed. Upper Sanddle River: Prentice Hall, 1996.

REIS, R. A. **Geração de Células Transparentes.** 1993. 128p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

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

SAIT, S. M.; et al. Fuzzy simulated evolution for Power and performance optimization of VLSI placement. In: Int. Joint Conf. Neural Networks IJCNN, USA, 738-743, 2001.

SARRAFZADEH, M.; WONG, C. K. **An Introduction to VLSI Physical Design .** 1st ed. San Francisco: Mc Graw-Hill, 1996.

SECHEN C. **VLSI Placement and Global Routing Using Simulated Annealing.** Boston: Kluwer Acad. Publishers, 1988.

SMANIOTTO, G. H.; et al. Optimizing cell area by applying an alternative transistor folding technique in an open source physical synthesis CAD tool. In: *2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS)*, Florianópolis, 2016, pp. 355-358.

SRIRAM, M.; KANG, S. M. A parallel min-cut technique for standard cell placement using a modified Hopfield neural network. In: Conf. Int. Circuits and Systems, China, 894-897, 1991.

TRIHY, R. 2008. Addressing library creation challenges from recent Liberty extensions. In Proceedings of the 45th Annual Design Automation Conference (Anaheim, California, June 08 - 13, 2008). DAC '08. ACM, New York, NY, 474-479.

TUTORIAL. C++ Language Tutorial. Disponível em: <http://www.cplusplus.com/doc/tutorial/>. Acesso em Jan. 2012.

WESTE, Neil H. E., ESHRAGHIAN, Kamran, Principles of CMOS VLSI design: a systems perspective, Addison - Wesley Longman Publishing Co., Inc., Boston, MA, 1985.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

YANG, L. et al. An evaluation of confidence bound estimation methods for neural networks. In: ESIT 2000.

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

ZIESEMER, A.; LAZZARI, C.; REIS R. Transistor Level Automatic Layout Generator for non-Complementary CMOS Cells. **IFIP/CEDA VLSI-SoC2007, International Conference on Very Large Scale Integration,** Atlanta, USA, p. 116-121, out. 2007.

ZIESEMER, A.; et al. Automatic layout synthesis with ASTRAN applied to asynchronous cells. In: *Circuits and Systems (LASCAS), 2014 IEEE 5th Latin American Symposium on*, Santiago, 2014, pp. 1-4.

ZIESEMER, A; REIS R. A. Simultaneous Two-Dimensional Cell Layout Compaction Using MILP with ASTRAN. In: *2014 IEEE Computer Society Annual Symposium on VLSI*, Tampa, FL, 2014, pp. 350-355.