PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE ENGENHARIA
FACULDADE DE INFORMÁTICA

EVOLUÇÃO DA FERRAMENTA LICHEN DE CARACTERIZAÇÃO DE CÉLULAS PARA PROJETO DE CIRCUITOS ASSÍNCRONOS

GUILHERME ESPINDOLA MEDEIROS

Monografia apresentada como requisito parcial à obtenção do grau de Engenheiro de Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Ney Laert Vilar Calazans
Co-Orientador: Prof. Msc. Matheus Trevisan Moreira

Porto Alegre
2014

AGRADECIMENTOS

Agradeço ao Prof. Dr. Ney Calazans, por me orientar no desenvolvimento deste trabalho. Agradeço também aos membros do grupo de pesquisa GAPH, aos colegas da faculdade e a todas as pessoas que de alguma maneira ajudaram no desenvolvimento do trabalho.

Agradecimentos especiais vão para o meu co-orientador Prof. Msc. Matheus Moreira, que além de tudo é um amigo. Reconheço todo o tempo por ele dedicado me ensinando conceitos e ajudando no desenvolvimento deste trabalho.

Finalmente, agradeço aos meus pais, por tudo que fizeram, a Marcella por me proporcionar a experiência única de ser pai, ao Arthur que está por vir e a todos os amigos. Sem vocês o desenvolvimento deste trabalho não seria possível.

EVOLUÇÃO DA FERRAMENTA LICHEN DE CARACTERIZAÇÃO DE CÉLULAS PARA PROJETO DE CIRCUITOS ASSÍNCRONOS

RESUMO

Este trabalho apresenta uma evolução da ferramenta *Library Characterization Environment* (LiChEn) destinada a caracterizar células padronizadas propostas para o projeto de circuitos assíncronos. LiChEn foi originalmente desenvolvida para automatizar o processo de caracterização de células padronizadas para circuitos assíncronos, mas em sua primeira versão ela possuía a limitação de somente poder caracterizar células com apenas uma saída. Através da proposta do presente trabalho de desenvolver uma nova heurística para a computação de Estados Estáticos (EE), dos Arcos de Transição Interna (ATI) e dos Arcos de Transição Dinâmica (ATD), supera-se a limitação citada e tem-se assim uma nova versão da LiChEn.

Palavras-Chave: bibliotecas de células, circuitos assíncronos, caracterização.

OPTIMIZATION OF LIBRARY CHARACTERIZATION ENVIRONMENT (LICHEN) TO CARACTEZING STANDARD CELLS TO ASSYNCHRONOUS CIRCUIT

ABSTRACT

This work presents the evolution the Library Characterization Environment (LiChEn), a tool for characterizing asynchronous standard cells. LiChEn was designed to automate the process of characterizing asynchronous standard cells. However, initially LiChEn had a limitation of only being able to characterize standard cells with a single output. Through the proposal of the new heuristic to compute Static States (SS), Internal Transition Arcs (ITA) and Dynamic Transition Arcs (DTA), target of this work, it was possible to overcome the above limitation and make available a new version of LiChEn.

Keywords: standard cell libraries, asynchronous circuits, characterization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ATD – Arcos de Transições Dinâmicas

ATI – Arcos de Transições Internas

CI – Circuito Integrado

CMOS – *Complementary Metal-Oxide-Semiconductor*

EDA – *Electronic Design Automation*

EE – Estados Estáticos

GALS – *Globally Asynchronous Locally Synchronous*

LTF – *Logic Threshold Function*

NCL – *Null Convention Logic*

PCHB – *Precharged Half Buffer*

QDI – *Quasi-delay-insensitive*

RTO – *Return-to-One*

RTZ – *Return-to-Zero*

SOC – *System-on-a chip*

SSTFB – *Static Single-Track Full Buffers*

VLSI – *Very Large Scale Integration*

SUMÁRIO

1. INTRODUÇÃO

Circuitos digitais são frequentemente projetados utilizando o paradigma síncrono, que implica uma noção de tempo discreta. Em circuitos digitais síncronos, um sinal externo, usualmente denominado relógio (do inglês, *clock*) é usado para sequenciar e sincronizar todos os eventos. Isto reduz consideravelmente a complexidade do projeto, permitindo aos projetistas ignorar o atraso de fios e portas lógicas de forma quase completa. Para tanto, apenas um conjunto de restrições relacionadas ao sinal de relógio devem ser respeitadas. No passado, estas restrições podiam ser satisfeitas com um esforço pequeno, o que justificou uma vasta adoção do paradigma. No entanto com a evolução da tecnologia utilizada para fabricar Circuitos Integrados (CIs), respeitar estas restrições tem se tornado uma tarefa cada vez mais difícil.

Nos projetos de CIs atuais, bilhões de transistores podem ser integrados em um único chip. Isto possibilitou um grande aumento da complexidade de aplicações implementáveis em um chip. Tais dispositivos são chamados de *sistemas-em-um-chip* (SoC). Embora existam técnicas e ferramentas para resolver os problemas de sincronização no desenvolvimento de SoCs, elas podem acrescentar novos problemas, relacionados a área e potência. O sinal global de relógio e seus circuitos de suporte dissipam uma potência média que equivale a 45% de toda a potência dissipada em chips de alto desempenho, e esta pode atingir até 75% em alguns casos [1]. Este problema de potência acaba tornando circuitos síncronos menos atraentes para aplicações onde potência e energia são as causas, como em aparelhos eletrônicos portáteis. Portanto, é necessário considerar novas estratégias de projeto de SoCs que sirvam a estas aplicações. De fato, de acordo com a *International Technology Roadmap for Semiconductors*, uma troca no paradigma é necessária para permitir futuras melhorias [8].

Neste contexto, as técnicas de projeto assíncrona vêm recebendo atenção crescente da comunidade de pesquisa em projetos de sistemas integrados (do inglês Very-Large-Scale-Integration, VLSI). O paradigma assíncrono [33] [28] [2], em contraste com o síncrono, não assume a verificação discreta do tempo. Esta característica faz com que o projeto tenha sensibilidade a fenômenos temporais [33]. Em circuitos assíncronos, sincronização, comunicação e sequenciamento de operações são realizados através de iterações localizadas entre componentes individuais do circuito, o que é denominado de *handshaking*. Assim, o circuito executa computação somente quando necessário [33]. Esta característica provê vantagens a circuitos assíncronos, como o potencial para baixo consumo de energia, alta velocidade de operação, melhor modularidade e alta robustez [33] [28] [14] [34] [2] [29].

Contudo, um dos maiores problemas para projetista de circuitos assíncronos é a baixa oferta de ferramentas que automatizam o desenvolvimento de circuito (do inglês, Electronic Design Automatiom, EDA) e bibliotecas de células padrão de suporte. Neste contexto,

a biblioteca de células ASCEnD [17] [23] [24] foi proposta para dar suporte a projetos assíncronos semi-dedicados. Contudo, muitos problemas relacionados a caracterização elétrica de células foram encontrados, como descrito em [21]. Para suprir parte das necessidades de métodos assíncronos de projeto, a ferramenta *Library Chacarterization Environment (LiChEn)* foi desenvolvida [21]. LiChEn permite uma caracterização elétrica precisa e automática de células assíncronas. LiChEn é baseada em simulações SPICE e dá suporte a qualquer tecnologia CMOS *bulk* em princípio. Contudo, uma das maiores limitações da versão atual da ferramenta é ser capaz de caracterizar apenas células com uma única saída.

O presente trabalho, consiste na extensão da ferramenta LiChEn para caracterizar células com múltiplas saídas. A importância desta extensão é que muitas células usadas em diversos tipos de projeto assíncronos possuem múltiplas saídas. Exemplos são os elementos de exclusão mútua essenciais em diversos estilos projeto assíncrono. Estes componentes possuem pelo menos duas saídas, células pchb que são uma boa alternativa quando se deseja trabalhar com o circuitos com pouco consumo de potência. Além disso, a ferramenta recebeu uma nova interface buscando otimizar a sua usabilidade.

2. MOTIVAÇÃO E OBJETIVOS

A evolução da tecnologia de semicondutores permite hoje a fabricação de circuitos com bilhões de transistores. Neste contexto, o fluxo de projeto semi-dedicado é um dos mais importantes no desenvolvimento de CIs. De fato, este fluxo, baseado no emprego de bibliotecas de células padronizadas é frequentemente associado ao rápido crescimento da indústria de semicondutores [11] [7] [9]. Uma célula é um componente capaz de realizar uma função específica, sendo utilizada por ferramentas de EDA para compor circuitos mais complexos. Desta maneira, a qualidade final destes circuitos está diretamente relacionada com a qualidade das células utilizadas no projeto.

Para a implementação de circuitos utilizando um fluxo semi-dedicado, ferramentas devem ser capazes de sintetizar descrições de *hardware* e mapeá-las utilizando uma biblioteca de células. Tais ferramentas exigem a disponibilidade de modelos elétricos para avaliar parâmetros como potência e atraso, sob diferentes condições. Tais condições são divididas em quatro classes:

- Condições funcionais: o estado lógico da célula, decodificando os valores de entrada e discriminando sua possível atividade de computação.
- Condições elétricas: especificação de rampa no(s) sinais de entrada e carga na(s) saída(s) da célula.
- Condições de fabricação: valores aceitáveis para variações do processo de fabricação de componentes eletrônicos.
- Condições operacionais: tensão e temperatura.

Estes modelos podem ser implementados de maneira o mais simplificada possível e possibilitar as melhores oportunidades de otimizações ao projeto. Eles devem refletir o comportamento da célula de forma precisa e tipicamente são implementados com base em resultados de simulações exaustivas previamente concebidas e validadas. Este processo é conhecido como caracterização elétrica. Isto claramente é trabalhoso e requer um nível de automação grande. Ainda que existam ferramentas de diferentes fabricantes para caracterizar células (como o ELC da Cadence [5] e o NCX da Synopsys [36]), a implementação de componentes e/ou de circuitos assíncronos geralmente não tem suporte nestas ferramentas. De fato conforme discutido no trabalho apresentado em [21], a caracterização elétrica de células assíncronas através dessas ferramentas é uma tarefa extremamente trabalhosa.

Durante o desenvolvimento da biblioteca ASCEnD-ST65, desenvolvida pelo grupo de pesquisa do autor deste trabalho, muitos desafios foram enfrentados na etapa de caracterização elétrica, como relatado em [17] [23] [24]. A ferramenta denominada *Library Characterization Environment (LiChEn)* foi proposta para auxiliar projetistas nesta tarefa [21].

LiChEn já foi utilizada com sucesso para a caracterização de *C-elements*, de portas *Null Convention Logic (NCL)* e NCL+ [25] [26] [19] e esses modelos foram utilizados em fluxos de projeto semi-dedicados como o que é apresentado em [16]. Entretanto, sua desvantagem atualmente é não dar suporte à caracterização elétrica para células com mais de uma saída, o que restringe sua utilização a células específicas e limita sua usabilidade e impede seu uso para dar suporte em fluxos como o apresentado em [3] que utiliza células de múltiplas saídas. O presente trabalho tem como principais objetivos:

1. Explorar conceitos básicos de projeto de circuitos assíncronos;
2. Estudar o estilo de projeto semi-dedicado baseado em células;
3. Explorar conceitos de caracterização elétrica;
4. Dominar modelos de potência e atraso para células;
5. Otimizar a ferramenta LiChEn para permitir a caracterização de células com múltiplas saídas;
6. Otimizar a interface da ferramenta LiChEn.

Finalmente, o interesse do autor na área de microeletrônica, em específico em caracterização cresceu devido ao fato de ferramentas comerciais não suportarem este paradigma. Além disso, a intenção em publicar o trabalho desenvolvido em veículos científicos também faz parte dos resultados esperados.

3. CONCEITOS FUNDAMENTAIS

No presente Capítulo conceitos necessários para a compreensão deste documento serão apresentados. Tais conceitos estão relacionados a estilos de projeto em microeletrônica, circuitos assíncronos, caracterização elétrica e modelos de atraso e potência.

3.1 Estilos de Projetos de Microeletrônica

A Figura 3.1 ilustra um visão geral das técnicas de projeto de circuitos integrados de acordo com Rabaey *et al.* [31], que podem ser divididas em dois grupos, projetos dedicados e semi-dedicados. O primeiro consiste em projetar o circuito de forma completamente manual, podendo ser otimizado para a sua função específica. Entretanto, isso torna o projeto mais complexo, e esta é a principal diferença para o projeto semi-dedicado que tem como principal objetivo reduzir a complexidade de projeto utilizando técnicas que permitem o reuso de e modularidade durante o projeto. Técnicas semi-dedicadas dividem-se em dois subgrupos: baseadas em células e baseados em organizações matriciais.

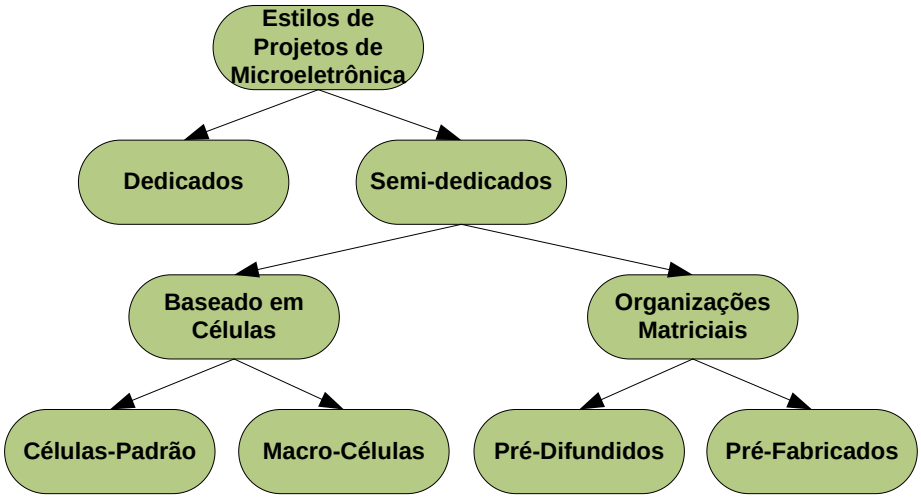

Figura 3.1 – Estilos de projetos em microeletrônica e a relação entre os mesmos [31].

Em projetos dedicados, cada transistor deve ser projetado manualmente. Desta maneira, pode-se ter um melhor controle da implementação do projeto, no qual se pode alcançar um alto nível de otimização em termos de consumo de potência, velocidade e outras características, de acordo com a funcionalidade desejada. Todo este trabalho manual tem como desvantagem o aumento do custo do projeto, do tempo de projeto e da complexidade do mesmo. Para o estado atual das tecnologias de fabricação de CIs, esta metodologia pode ser bastante difícil de ser justificada do ponto de vista econômico.

Técnicas baseadas em organizações matriciais podem ser divididas em dois tipos: circuitos pré-difundidos e pré-fabricados. Nos pré-difundidos a área do CI é pré-estabelecida, contendo um conjunto fixo de transistores e, para utilizar esses transistores, os projetistas necessitam implementar somente uma ou mais camadas de interconexão. Isto difere dos pré-fabricados onde o processo de fabricação é independente do circuito a ser projetado. Nesta metodologia um CI é fabricado e pode ser configurado posteriormente para criar a função desejada para o circuito. Projeto baseado em células pode ser também dividido em macro-células e células padrão. Projetos baseados em macro-células, fazem uso de programas de computadores para sintetizar módulos regulares do circuito. Estruturas como multiplicadores, caminhos dados e memórias são exemplos comuns de macro-células. Células padrão por outro lado, possuem funções básicas, tais como uma porta lógica ou um *flip-flop*. Estas células são os blocos construtivos básicos para criar circuitos de maior complexidade, que previamente projetada, validada e caracterizada.

3.2 Circuitos Assíncronos

Circuitos digitais podem ser classificados de acordo com o seu modo de operação. Circuitos síncronos são caracterizados por possuírem um sinal global de relógio para processar informação [15]. Circuitos assíncronos são fundamentalmente diferentes, e não implicam em uma noção discreta de tempo [33]. De acordo com Sparsø e Furber [33], circuitos assíncronos podem levar a baixo consumo de potência, alta velocidade de operação e melhor modularidade [38] [4], porém para possuir estas características é necessário a implementação de componentes específicos, componentes estes que não existe em bibliotecas de células padronizadas convencionais. As subseções 3.2.1, 3.2.2, 3.2.3, 3.2.4 contém alguns dos principais componentes utilizados em circuitos assíncronos.

3.2.1 O C-element de Muller-Bartky

O C-element de Muller [27] é um dos componentes fundamentais e é extensivamente utilizado em circuitos assíncronos [33] [6]. Isto ocorre devido ao componente poder operar como um sincronizador de eventos. Na versão mais básica, o *C-element* possui uma única saída e esta saída só irá alterar de valor quando todos os valores da entrada possuírem o mesmo valor lógico. A tabela verdade desse componente está representada na Tabela 3.1. Quando as entradas assumem o mesmo valor, a saída passa a assumir este mesmo valor. Entretanto, quando um valor diferente ocorre entre as entradas, a saída mantém o mesmo valor lógico previamente armazenado. Existem diferentes implementações deste componente, porém as mais populares são as propostas por: Sutherland [35], van

Berkel [37] e Martin [13]. A escolha de um específico *C-element* a ser utilizado é discutido em [22].

Tabela 3.1 – Tabela verdade de um C-element típico de duas entradas [35] [37] [13]

A	B	Q_i
0	0	0
0	1	Q_{i-1}
1	0	Q_{i-1}
1	1	1

3.2.2 NCL

Null Convention Logic (NCL) é um estilo de projeto assíncrono que utiliza células específicas, cujo símbolo é ilustrado na Figura 3.2. Muitas vezes estas células são chamadas de *células de threshold*, pois utilizam função lógica de *theshold* (LTF) para definir o valor de saída. Esta função opera de acordo com a Equação 3.1, onde Q é o valor lógico de saída, N_i é o número do pino de entrada, w_i é um peso específico associado a cada entrada, quando este peso não está descrito no símbolo do componente, w_i assume o valor 1 e M é o valor de limiar ou *threshold*. A saída Q só irá para o nível lógico 0 quando todas as entradas estiverem no nível 0 e só atinge o valor 1 quando a soma dos pesos das entradas em 1 é igual ou maior que o valor de M. Para outras combinações, a célula mantém o valor previamente armazenado.

$$Q = \begin{cases} 1, & \sum_{i=1}^{n} N_i w_i \geq M \\ 0, & \text{se todas as entradas estiveram em 0} \end{cases} \tag{3.1}$$

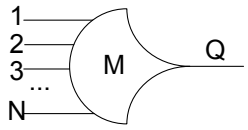

Figura 3.2 – Símbolo de uma porta NCL com *threshold* (M) e (N) entradas.

A Figura 3.3 apresenta um exemplo de uma porta NCL com três entradas (N=3), peso (w_i=1) por isso não representado na figura e um valor de *threshold* igual a dois (M=2). A tabela verdade para esta porta está representada na Tabela 3.2. Podemos observar que a saída só comutará para o valor lógico 1, quando as condições da Equação 3.1 forem atingidas, no exemplo este valor de condição equivale ao mesmo valor de M, isto ocorre pois o peso (w_i) é 1. A saída só comutará para 0 quando todas as entradas estiverem em

0. Desta maneira, podemos observar que o C-element é um caso específico de NCL, onde o seu valor de *threshold* é equivalente ao seu número de entradas (N=M) e contém $w_i = 1$.

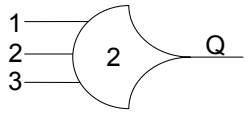

Figura 3.3 – Símbolo representando uma porta NCL com 3 entradas (N=3) e *threshold* 2 (M=2).

Tabela 3.2 – Tabela verdade da porta NCL da Figura 3.3.

A	B	C	Q_i
0	0	0	0
0	0	1	Q_{i-1}
0	1	0	Q_{i-1}
0	1	1	1
1	0	0	Q_{i-1}
1	0	1	1
1	1	0	1
1	1	1	1

3.2.3 NCL+

NCL+ é uma família de componentes similar a NCL. A principal diferença é que estes componentes trabalham fazendo uso do protocolo *Return-to-One* (RTO) [20] ao invés do *Return-to-Zero* (RTZ). Dessa forma o comportamento deles é o inverso do observado para componentes NCL. A saída só comutará para 1 quando todas as entradas estiverem em 1 e só irá para 0 quando pelo menos M de suas entradas estivem em 0 ou se a soma dos pesos w_i com as entradas N forem maior ou igual ao valor de *threshold*. Para outras combinações a saída mantém o estado anterior. Figura 3.4 mostra um exemplo do símbolo de uma NCL+ com três entradas (N=3) e *threshold* dois (M=2), Tabela 3.3 contém a tabela verdade do exemplo da Figura 3.4.

3.2.4 Mutex

Um mutex é um elemento de exclusão mútua utilizado para sincronização de eventos em projetos não síncronos. Esse elemento permite arbitrar entre duas requisições to-

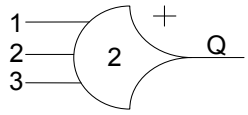

Figura 3.4 – Símbolo representando uma porta NCL+ com 3 entradas (N=3) e *threshold* 2 (M=2).

Tabela 3.3 – Tabela verdade para a porta NCL+ da Figura 3.4.

A	B	C	Q_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	Q_{i-1}
1	0	0	0
1	0	1	Q_{i-1}
1	1	0	Q_{i-1}
1	1	1	1

talmente assíncronas garantindo atender uma requisição por vez, mesmo que essas sejam solicitadas exatamente ao mesmo instante, sem introduzir sinais metaestáveis no sistema. De fato, caso as duas requisições aconteçam exatamente no mesmo instante, o mutex deverá atender primeiramente uma delas, escolhida de forma aleatória, e após a primeira ser atendida a requisição seguinte será atendida. A Equação 3.2 se refere ao filtro que é o principal elemento no desenvolvimento de um mutex, como podemos observar na Tabela 3.4 quando ocorre duas requisições ao mesmo tempo, o componente atende estas requisições de maneira aleatória, isto ocorre devido a fenômenos elétricos.

$$\begin{aligned} AA &= \overline{RB}.RA \\ AB &= \overline{RA}.RB \end{aligned} \tag{3.2}$$

Tabela 3.4 – Tabela verdade de um filtro, elemento principal no projeto de um componente mutex.

RA	RB	AA	AB
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	1
		1	0

3.3 Caracterização

Quando se utiliza o estilo de construção semi-dedicados baseado em células para CIs é necessário avaliar e conhecer as características elétricas e comportamentais de cada célula, para permitir otimizar adequadamente o projeto. O processo de caracterização de uma célula consiste normalmente em utilizar ferramentas de EDA especializadas para extrair estas informações a partir do projeto detalhado de cada célula. As principais características extraídas neste processo são: capacitância dos pinos de entrada, modelos de dissipação de potência e atraso da célula. As medidas de potência basicamente são divididas em porções estática e dinâmica. Modelos de atraso são divididos em atrasos de propagação e de transição. Note-se que não se considera aqui restrições de tempo tais como tempo de *setup* e *hold*, dado ao fato de que estas métricas não são relevantes para o projeto de circuitos assíncronos. As subseções 3.3.1 e 3.3.2, contêm os conceitos relacionados a modelos de tempo e potência relevantes para este trabalho.

3.3.1 Atraso de Propagação

O atraso de propagação é o intervalo de tempo necessário para que uma alteração em um pino de entrada se propague até um pino de saída [31]. Para medir este atraso de propagação é necessário conhecer os tempos de subida e/ou descida do nível lógico do pino de entrada que afeta o a saída e o tempo para que ocorra alteração da saída. A Figura 3.5 contém um exemplo deste conceito para um *C-element* com duas entradas, onde se pode observar como é possível medir o atraso de propagação do nível lógico baixo para o alto (D_{Plth}), provocado pela comutação do valor lógico do pino de entrada B que afeta a saída Q. Também se observa como medir o atraso gerado pelo pino de entrada A, que modifica o seu valor lógico afetando a a saída Q e gerando o atraso de propagação do nível lógico alto para o baixo(D_{Plth}).

3.3.2 Atraso de Transição

Atraso de transição é o tempo que a saída leva para responder a uma comutação de nível lógico, indo de um valor de tensão associado a um nível lógico válido, a um valor de tensão associado ao nível lógico oposto. A Figura 3.6 ilustra como medir o atraso de transição de um pino de saída Q de um dado componente. O atraso do nível de tensão baixo para alto (D_{Tlth}) é medido através do tempo que a saída leva para alcançar o nível mínimo de tensão alto (V_{HIGH}) depois de deixar o valor lógico baixo máximo (V_{LOW}). A

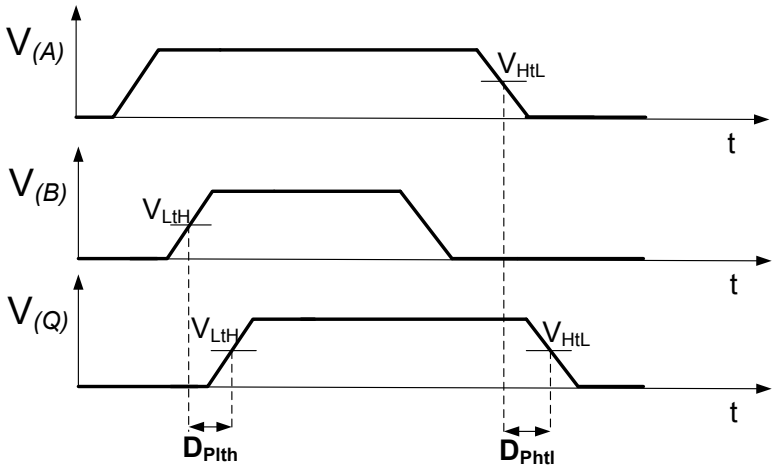

Figura 3.5 – Definição de atrasos de propagação para um *C-element* com 2 entradas.

transição do nível lógico de alto para baixo (D_{Thtl}) é definida similarmente, porém avaliando a passagem do nível alto para o baixo.

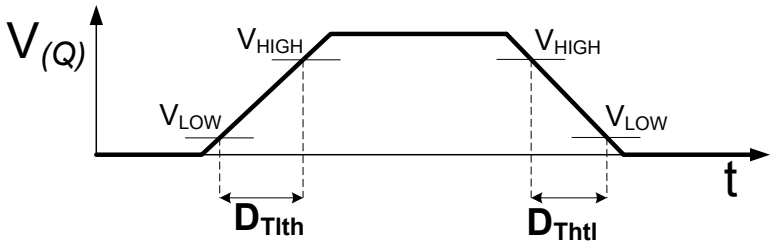

Figura 3.6 – Definição dos atrasos de transição para um pino de saída de um componente Q, de acordo com os valores de tensão máximo para o nível lógico 0 e o mínimo para o nível lógico 1.

3.4 Potência

A potência de uma célula pode ser dividida em duas classes: estática e dinâmica, cada uma explorado em uma subseção a seguir.

3.4.1 Potência Estática

Potência estática ($P_{Estática}$) ou de fuga (do inglês *leakage*) é o consumo que ocorre quando o circuito está em um estado quiescente e estável. É medido através da média da corrente de fuga ($I_{leakage}$) drenada da fonte de alimentação, multiplicada pela tensão de operação ($V_{(vdd)}$).

$$P_{Estática} = V_{(\text{vdd})} * \frac{\int_0^T I_{\text{vdd}} \mathrm{d}t}{T} \tag{3.3}$$

3.4.2 Potência Dinâmica

A potência dinâmica é classicamente dividida em duas classes distintas: potência interna (P_I) e de chaveamento (P_C). A primeira leva em conta somente a carga gerada pela fonte de alimentação que foi causada por um chaveamento em um pino de entrada e consiste na multiplicação da tensão da fonte de alimentação com todo o consumo Estático medido através da Equação 3.4, onde (I_{antes}*$t_{inicial}$) é a carga gerada enquanto o circuito está em estado quiescente. Esta carga é somada com a carga gerada do próximo estado quiescente multiplicado pelo delta de variação de tempo, o resultado destas operações é multiplicado pela tensão da fonte de alimentação gerando o consumo estático total ($Estatico_{\text{tot}}$) [31].

$$Estatico_{\text{tot}} = V_{(\text{vdd})} * (I_{\text{antes}} * t_{\text{inicial}} + I_{\text{depois}} * (t - t_{\text{inicial}})) \tag{3.4}$$

$$P_I = \left(V_{(\text{vdd})} * \int_0^T I_{(\text{t})} \mathrm{d}t \right) - Estatico_{\text{tot}} \tag{3.5}$$

A potência de chaveamento (P_C) é calculada de maneira similar porém desconsidera a corrente necessária para carregar o capacitor de saída ($C_{\text{saída}}$), medindo somente o chaveamento dos pinos, a Equação 3.6 demonstra a fórmula para este cálculo [31].

$$P_C = \left(V_{(\text{vdd})} * \int_0^T I_{(\text{t})} \mathrm{d}t \right) - Estatico_{\text{tot}} - \left(\frac{1}{2} * C_{\text{saída}} * {V_{(\text{vdd})}}^2 \right) \tag{3.6}$$

4. ESTADO DA ARTE

O uso de células no projeto de CIs é um estilo projeto já consolidado há várias décadas. De fato, existe uma vasta variedade de bibliotecas de células disponível para diferentes tecnologias, com diferentes propósitos e associadas a diferentes processos de fabricação. Neste contexto, a caracterização elétrica tem um papel importante no projeto de um CI, utilizando ferramentas comerciais de EDA que são desenvolvidas com suporte ao paradigma síncrono. Entretanto, se desejamos caracterizar células assíncronas, isto pode implicar em um trabalho manual em pequena ou grande escala [21], visto que ferramentas convencionais geralmente não são compatíveis com estas células. Este Capítulo apresenta uma visão de alguns dos esforços recentes para a implementação de ferramentas que suportem o paradigma assíncrono.

4.1 Machado e Schivitz: Um Fluxo Automático de Caracterização [12]

Machado e Schivitz apresentam um fluxo automático para caracterização de atrasos e potência estática consumida por células de uma biblioteca. O fluxo conta com uma ferramenta desenvolvida em C++, que permite modificar diferentes parâmetros do ambiente a ser caracterizado. A ferramenta determina os atrasos de transição e propagação através da tabela verdade e outros parâmetros como vetor de rampa de entrada, degrau de transição e capacitância de saída. Todas as simulações elétricas são executadas com a ferramenta NGSPICE [12], um simulador de circuitos baseado em SPICE. Embora a ferramenta dê suporte à caracterização de células assíncronas, ela possui a limitação de não medir o consumo de potência dinâmica da célula. Isto limita a qualidade final do projeto de células assíncronas e na avaliação de potência de forma mais detalhada.

4.2 Prakash:Caracterização e Avaliação Estática de Atraso de Circuitos Assíncronos [30]

Em [30], Prakash apresenta a caracterização e a análise estática de temporização para uma biblioteca de componentes assíncronos. A biblioteca contém circuitos assíncronos baseados em *Static-Single-Track Full Buffers* (SSTFB) [2]. O fluxo de caracterização é dividido em três etapas principais: a primeira consiste em determinar os arcos de tempo para o circuito. Após isto, é necessário criar um ambiente para simulações no SPICE e avaliações. Por último, potência e capacitância dos pinos são medidos.

A análise estática de tempo proposta por Prakash é composta em três etapas: *(i)* capturar as restrições de tempo utilizando a ferramenta comercial *Prime Time*, da *Synopsys* [36]; *(ii)* quebrar laços, desabilitando caminhos específicos; *(iii)* verificação de desempenho. Todos estes passos são gerados através de *scripts*. Este fluxo foi utilizado para validar modelos gerados pelo autor.

A desvantagem desta abordagem para avaliação de potência, é não levar em consideração efeitos de carga e descarga das capacitâncias no consumo interno de potência. Isto acaba gerando uma avaliação de potência imprecisa, que limita a usabilidade da proposta.

4.3 Interface GALS para Integração de Sistemas Digitais Complexos [10]

A Infineon [10] publicou um relatório técnico sobre um conjunto de células assíncronas fabricadas em uma tecnologia CMOS de 40nm. Neste relatório mostra-se o resultado da análise de temporização e potência para estas células, que consistem em *C-elements* e elementos de exclusão mútua (mutexes). Estas células foram desenvolvidas para dar suporte a uma método de projeto GALS. Heer et al. demonstram a importância de caracterizar células assíncronas e sua aplicabilidade em um ambiente industrial, para um projeto de chip desenvolvido na Infineon. Este trabalho concluiu que a caracterização desenvolvida foi bem correlacionada com os resultados finais do *hardware*. Contudo, os autores não explicam como o processo de caracterização elétrica é realizado, nem qual a análise é aplicada para obtenção dos resultados. Também, não há menção sobre qual ferramenta para caracterização elétrica foi utilizada.

4.4 Moreira et al.: Um Ambiente de Caracterização para Células de Bibliotecas Padrão [21]

Em [18], Moreira (et al.) propuseram LiChEn, uma ferramenta automatizada baseada no simulador elétrico *Spectre* da Cadence, desenvolvida para dar suporte a qualquer tecnologia de CMOS. A ferramenta cria modelos de potência interna e dinâmica, bem como modelos de atrasos de propagação e transição para células. Todos os modelos baseiam-se em tabelas não-lineares, obtidos via simulações SPICE. Embora LiChEn forneça uma caracterização elétrica precisa, a ferramenta é limitada devido ao fato de só dar suporte à caracterização de células com apenas uma saída, impedindo a sua utilização para muitos modelos de células assíncronas que possuem mais de uma saída, tais como somadores, mutexes, células PCHB etc.

5. LICHEN 2.0

O presente trabalho consiste na implementação de uma nova versão da ferramenta de EDA de código aberto LiChEn, primeiramente proposta em [21] e desenvolvida em C/C++. LiChEn fornece caracterização para células especificamente utilizadas em circuitos assíncronos. As principais contribuições do trabalho são: (i) prover suporte para caracterização de múltiplas saídas; (ii) e desenvolver uma nova interface para otimizar a usabilidade da ferramenta.

Inicialmente, o fluxo de projeto semi-dedicado foi explorado, para compreender a sua importância e contextualizar a necessidade da ferramenta. Também foi explorado o paradigma assíncrono e a primeira versão do LiChEn. A parte prática do trabalho esta dividida em três etapas, organizadas linearmente como ilustra a Figura 5.1. Primeiramente, um novo conjunto de comandos foi especificado e a gramática deste conjunto desenvolvida. O primeiro passo teve como principal objetivo aumentar a usabilidade através de uma nova interface. Segundo, uma heurística para encontrar Arcos de Transições Dinâmicas (ATD), Arcos de Transições Internas (ATI) e Estados Estáticos (EE) foi desenvolvida para dar suporte a células com múltiplas saídas. Um arco é um conjunto de combinações de entradas que produz uma transição em um pino e é dividido em dois grupos: dinâmicos e internos. ATD é a denominação de um arco que caracteriza a alteração de um valor lógico em um pino de entrada alterando o valor da saída, diferentemente dos ATIs que é o termo para a comutação dos valores da entrada que não produzem qualquer alteração na saída. EEs são os mesmos estados que se pode observar na tabela verdade de uma função lógica. A última atividade consiste na validação da ferramenta LiChEn 2.0, conduzida através da caracterização de componentes com múltiplas saídas. Os estudos de caso aqui apresentados são uma célula *C-element* e uma porta OR PCHB.

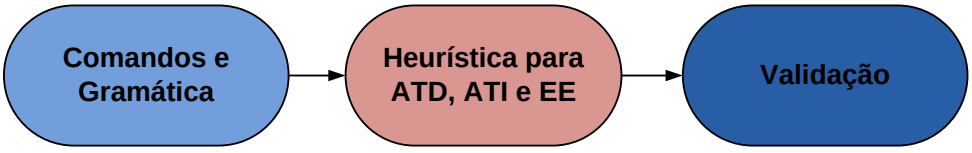

Figura 5.1 – Relação entre as atividades práticas.

5.1 Comandos e Gramática

Para a implementar uma nova gramática o conjunto de ferramentas *Command Line Interface* (CLI) desenvolvido por Royer [32], foi utilizado. Este conjunto de ferramentas permite ao usuário criar sua interface de linha de comando utilizando descrições em XML dos comandos desejados. CLI permite uma integração automatizada com a linguagem de programação C++. Este fator é importante para manter a estrutura de código da ferramenta LiChEn. A Figura 5.2 ilustra um exemplo do XML utilizado no CLI. Primeiro um identificador indica o começo e o nome do novo comando, neste caso "library_name". Após, existe uma mensagem opcional para descrever a função do comando, no exemplo a mensagem serve para exibir o nome da biblioteca que está sendo utilizada pela LiChEn. Na próxima linha o identificador "cpp" demonstra qual linguagem de programação que será utilizada, no caso C++. O segundo comando possui praticamente a mesma estrutura do primeiro, porém um novo identificador chamado de "param" indica se o comando possui um parâmetro. Os parâmetros que podem ser utilizados são os mesmos da linguagem de programação subjacente. Todos os comandos disponíveis na LiChEn estão descritos no Anexo A.

```
32 <!-- COMMAND library_name() - Display library name -->
33 <keyword string="library_name"><help lang="en">Display library name.</help>
34   <endl>
35     <cpp>co.print_library_name(<out/>);</cpp>
36   </endl>
37 </keyword>
38
39 <!-- COMMAND set_library_name(<name>) -->
40 <keyword string="set_library_name"><help lang="en">Set library name.</help>
41   <param type="string" id="lib_name">
42     <endl>
43       <cpp>co.set_library_name((const char* const)<value-of param="lib_name"/>);</cpp>
44     </endl>
45   </param>
46 </keyword>
47
48 <!-- COMMAND - print_library() - Print all cells of the library-->
49 <keyword string="print_library"><help lang="en">Print all cells of the library.</help>
50 <endl>
51   <cpp>co.print_library(<out/>);</cpp>
52 </endl>
53 </keyword>
```

Figura 5.2 – Exemplo de uma descrição XML utilizada para criar uma interface de comandos com CLI.

O desenvolvimento da nova gramática pode ser dividido em três em etapas: (i) a primeira etapa consiste na avaliação do conjunto de comandos da primeira versão da ferramenta, descartando comandos que possuam a mesma característica ou que estavam mal definidos. Após esta análise definiu-se um novo conjunto de comandos; (ii) a segunda etapa consistiu da elaboração de mensagens de ajuda, para quem não está familiarizado com a ferramenta obter informações da maneira mais sucinta possível; (iii) última etapa é a descrição em XML do novo conjunto de comandos. Com os parâmetros definidos e as mensagens de ajuda utilizando o CLI, a descrição em XML é compilada junto com a LiChEn, gerando uma classe em C++ que contém somente o nome do comando e seus parâmetros,

não influenciando a estrutura de classes da LiChEn. Isto torna a nova versão da ferramenta mais modular.

CLI fornece alguns recursos como a verificação da sintaxe de comandos e a verificação se todos os parâmetros de um determinado comando foram fornecidos com os dados corretos referentes ao tipo do parâmetro. Outro recurso é a função de completar automaticamente o comando, evitando que algum erro de digitação ocorra. Estes recursos são fundamentais para facilitar o uso da ferramenta, permitindo que ela seja empregada para diferentes finalidades relacionadas à pesquisa, desenvolvimento e educação.

5.2 Heurísticas para ATD, ATI e EE

Nesta etapa, uma heurística para busca de ATD, ATI e EE visando múltiplas saídas foi desenvolvida. Em sua versão original, LiChEn emprega um algoritmo de busca em profundidade para encontrar todos os arcos de transições e estados estáticos. Um exemplo da computação dos arcos de transição e estados estáticos para um *C-element* com duas entradas é ilustrado na Figura 5.3 [18]. Nesta "R" indica a ocorrência uma transição do nível lógico baixo para o alto (do inglês, *rise*) e "F" indica uma transição do nível lógico alto para o baixo (do inglês, *fall*) [21].

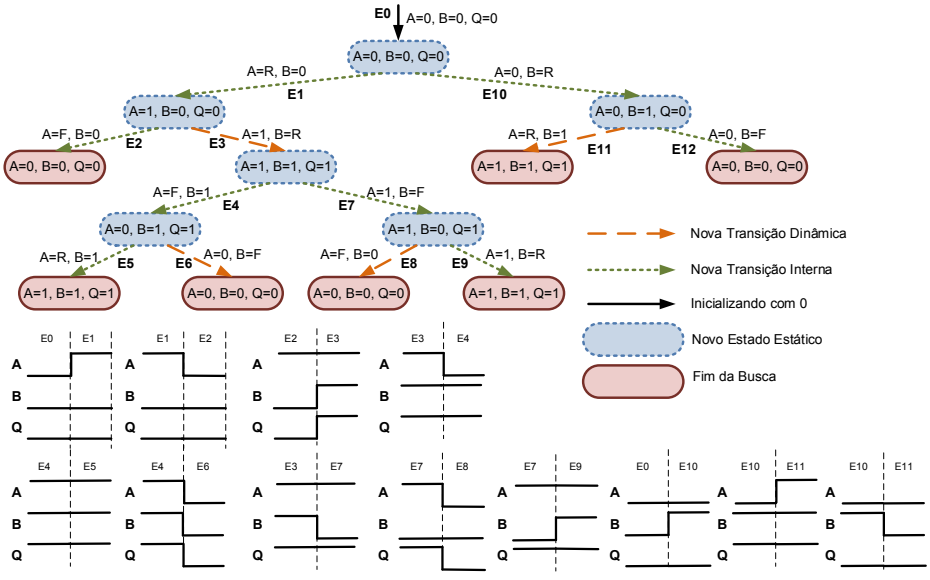

Figura 5.3 – *(a)* Computação dos arcos de transição e estados estáticos para um *C-element* de duas entradas [18] *(b)* Forma de onda das transições.

A função de um *C-element* de duas entradas clássico é descrita em 5.1. Primeiramente, a ferramenta inicializa todas as entradas com o valor lógico 0 e computa o resultado do valor da saída, de acordo com a função lógica. Após, chaveia-se o valor lógico de uma

entrada por vez e avalia-se o valor da saída de acordo com esta alteração. No exemplo, A é alterado para o valor lógico 1 (E1). O resultado da função é manter Q=0, então temos um ATI (A=R, B=0, Q=0). Depois, o valor de A é alterado mais uma vez, agora de 1 para 0, (E2), resultando no EE (A=0, B=0, Q=0). Como esta combinação já foi previamente computada, a busca neste ramo é encerrada. A pesquisa retorna ao nodo (A=1, B=0, Q=0), e altera o valor lógico da entrada B (E3). Desta vez a alteração gera uma mudança na saída criando um novo EE (A=1, B=1, Q=1) e um novo ATD (A=1, B=R, Q=R). A partir deste, o próximo passo é modificar o valor lógico A (E4), gerando um novo ramo na árvore de pesquisa, como ocorre em E5 e E6. Uma vez que a pesquisa nestes ramos é encerrada, a entrada B tem o seu valor lógico modificado novamente (E7). Esta pesquisa continua até se encerrar em todos os ramos, neste caso, com o evento (E12) [21], todos ATD, ATI e EE para a Figura 5.3 estão representados na Tabela 5.1. A heurística só encontra ATD, ATI e EE para células com uma saída.

$$Q = (A * B) + (A * Q) + (B * Q) \tag{5.1}$$

Tabela 5.1 – Tabela com EEs, ATDs, ATIs da Figura 5.3

EEs	**ATDs**	**ATIs**
0,0,0	1,R,R	R,0,0
1,0,0	0,F,F	F,0,0
1,1,1	F,0,F	F,1,1
0,1,1	R,1,R	R,1,1
1,0,1		1,F,1
0,1,0		0,R,0
		0,F,0

Na Figura 5.4 tem-se o diagrama de transistores de um componente PCHB OR, que será utilizado como estudo de caso para ilustrar o resultado da nova heurística para calcular ATDs, ATIs e EEs. O diagrama apresentado utiliza a codificação *dual-rail*, onde se emprega um pino para indicar o nível lógico 0 e outro para indicar o nível lógico 1, isto para cada entrada e saída do componente, e lógica dominó para computação dos valores lógicos. A Equação 5.2 contém as equações Booleanas do modelo. A Figura 5.5 ilustra uma parte da computação dos arcos de transição e estados estáticos para o componente OR PCHB. Os demais estados e arcos foram omitidos, pois a árvore completa possui ao todo 257 nodos. De forma similar ao algoritmo original, primeiramente inicializa-se todas as entradas com o valor lógico 0 e computa-se o resultado do valor de saída. Note-se que, no caso da computação para o OR PCHB, alguns estados não ocorrem simultaneamente, como é o caso de A.0 e A.1 ou B.0 e B.1 possuírem o mesmo valor lógico. Isto se deve à codificação utilizada para o projeto do componente. Entretanto, para o algoritmo adotado,

todos os casos foram cobertos, mesmos os que não ocorrem jamais. Após a primeira comutação, altera-se o valor lógico da mesma entrada e calcula-se novos valores para a saída, até se encontrar um estado que já ocorreu na árvore. Quando isso acontece, volta-se para o nodo anterior e altera-se o valor lógico do próximo pino. Após isso, avalia-se o valor de saída de acordo com esta alteração, após o que o algoritmo segue alterando valores lógicos dos pinos de entrada, até que um chaveamento na última variável da raiz de entrada da árvore gere um EE já computado, o que indica o fim da expansão da árvore.

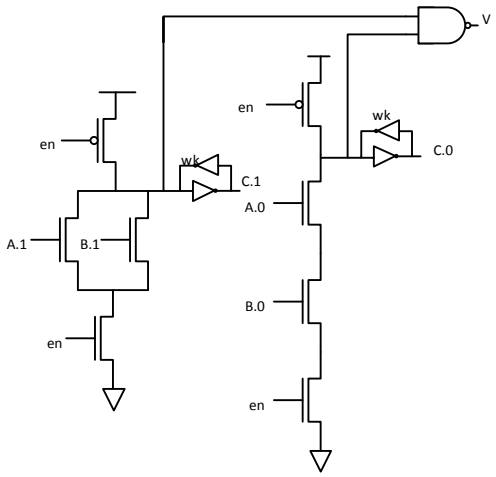

Figura 5.4 – Diagrama de transistores de um componente OR PCHB [3].

$$\begin{aligned} C.1 &= En * (A.1 + B.1 + C.1) \\ C.0 &= En * ((A.0 * B.0) + C.0) \\ V &= En * (A.1 + B.1 + (A.0 * B.0) + C.1 + C.0) \end{aligned} \tag{5.2}$$

A principal diferença da nova heurística para a heurística original reside no tratamento dos pinos de entrada e saída e as funções lógicas da célula como listas presentes na classe da célula. A Figura contém o pseudo-código da nova heurística desenvolvida. Após a definição destas listas, inicia-se a computação dos arcos de forma similar à heurística original de chavear somente um valor lógico de um pino de entrada por vez. A diferença é que a cada chaveamento soluciona-se cada função lógica, referente a cada pino de saída, de forma independente. Dessa forma, a nova heurística acaba computando em alguns casos, estados que não podem ocorrer, pois se ocorrerem significa que a célula esta operando inadequadamente.

Como no caso da célula OR PCHB apresentada, quando os pinos de entrada A.0 e A.1 estiverem com o nível lógico 1, por exemplo, se indica que o circuito está operando de forma incorreta, pois o modelo apresentado utiliza codificação *dual-rail* para apresentar os dados, o que significa que para cada entrada existe um pino indicando o nível lógico

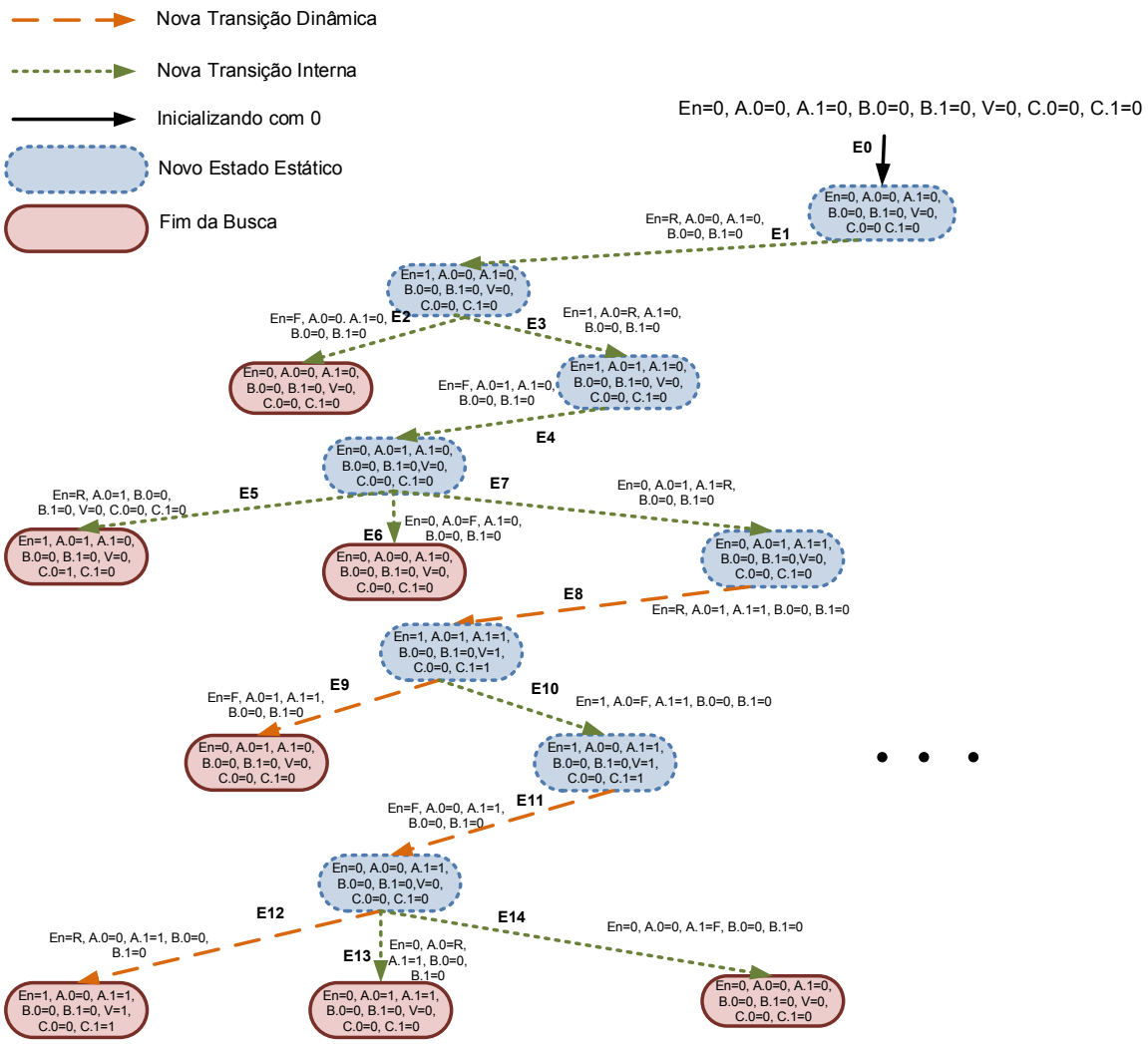

Figura 5.5 – Exemplo da árvore de computação para a Figura 5.4.

alto e outro pino indicando o nível lógico baixo. A nova heurística não verifica estes problemas, porém a computação destes casos gera dados que não influenciam na avaliação final da caracterização da célula. A nova heurística desenvolvida dá suporte à geração da árvore de computação para células com qualquer número de saídas, o único requisito para a computação é possuir uma equação Booleana para cada saída. Com esta modificação foi possível organizar uma nova estrutura de classes, ilustrada na Figura 5.7, onde se tem uma classe contendo todos os comandos disponíveis. O principal desafio encontrado foi a criação da árvore de computação, verificando o chaveamento correto das entradas, a inserção na árvore e retorno para o próximo chaveamento de maneira correta.

```
#define nro_folhas_max <nro_pinos_de_entrada>

init_pinos(lista_pinos, 0);
for lista_funcao != lista_funcao.fim(){
    nodo = resolve_funcao(lista_funcao, lista_pinos);
    inserir_arvore(arv, nodo)
    if(lista_pinos[0] == 0)
        lista_pinos[0] = 1;
    else if(lista_pinos[0] == 1)
            lista_pinos[0] = 0;

    get_arcos_recursivo(celula, nodo);
}

get_arcos_recursivo(celula, nodo){
    nro_folhas = nodo->nro_folhas
    while (nro_folhas != nro_folhas_max){
        nodo = resolve_funcao(lista_funcao, lista_pinos);
        if (!alterou_saida)
            inserir_arvore(ATI, nodo);
        else if(alterou_saida)
            inserir_arvore(ATD, nodo);
        if (nodo != compara_nodo_arvore(arv, nodo))
            inserir_arvore(EE, nodo);
        inserir_arvore(arv, nodo);
        if(lista_pinos[0] == 0)
            lista_pinos[0] = 1;
        else if(lista_pinos[0] == 1)
            lista_pinos[0] = 0;
        get_arcos_recursivo(celula, nodo);
    }
}

```

Figura 5.6 – Pseudo-código da nova heurística para cálculo de EEs, ATDs e ATIs.

5.3 Ambiente de Simulação

O ambiente de simulação da LiChEn utiliza o fluxo de caracterização elétrica ilustrado na Figura 5.8, Este fluxo leva em consideração os processos de diferentes *corners*, parâmetros elétricos, condições operacionais, rampas de entrada e carga de saída, para realizar simulações em SPICE, onde para cada EE, ATI e ATD se gera uma descrição SPICE que considera os vetores de carga de saída, rampa de entrada e tempo inicial de simulação. Isto é realizado para avaliar cada arco sob diferentes condições.

A criação de cada descrição SPICE e as simulações dos mesmos, são processados de forma automatizada pela ferramenta LiChEn, levando em consideração os EE, ATD e ATI calculados. A avaliação do consumo estático de potência do circuito é obtida através da criação de uma descrição SPICE para cada EE calculado na árvore de computação, com a inserção de instruções para avaliar o consumo da fonte de alimentação até o *ground* do circuito. O consumo interno de potência é calculado de maneira similar ao consumo estático, contudo as descrições são criadas a partir dos ATI. Além de se criar instruções para realizar as medições referentes a consumo interno, adiciona-se à descrição condições de capacitância da carga de saída e de rampa de entrada, para realizar medidas em diferentes casos. Por fim, os modelos de atraso de propagação e transição são obtidos após a criação de uma descrição SPICE para cada ATD encontrado, medindo os atrasos de propagação e de transição para a saída que teve o seu valor alterado, e contendo as condições de capacitância na carga de saída e rampa de entrada.

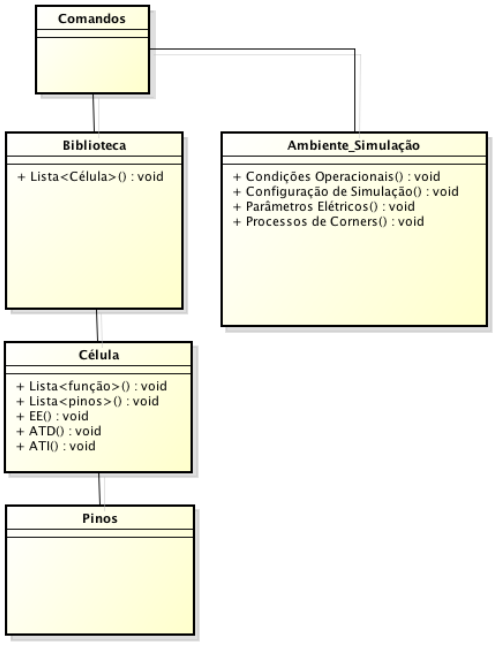

Figura 5.7 – Diagrama de classes da LiChEn 2.0.

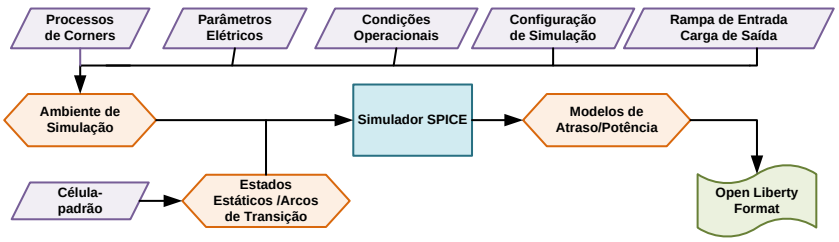

Figura 5.8 – Fluxo de caracterização elétrica utilizado pela LiChEn.

6. EXPERIMENTOS E RESULTADOS

Para validar o ambiente gerado, um conjunto mínimo de células foi caracterizado através de simulações SPICE. O conjunto consiste em um C-element da biblioteca ASCEnD e uma célula OR PCHB com características já conhecidas. Todas as células foram simuladas com as seguintes condições:

- Transição Máxima: 0,3ns.
- Tensão Operacional: 1V.
- Temperatura Operacional: 25°C.
- Processo de simulação: típico.
- Valor de tensão para o nível lógico 1: 0,8V.
- Valor de tensão para o nível lógico 0: 0,1V.
- Valor de tensão para o uma transição de *low-to-high*: 0,6V.
- Valor de tensão para o uma transição de *high-to-low*: 0,4V.
- Tempo de início de simulação: 0,5ns.
- Tempo de simulacão: 20ns.
- Passo de simulação: 0,001.
- Vetor de carga de saída: 0,001pF, 0,003pF, 0,008pF, 0,012pF, 0,018pF, 0,035pF, 0,07pF.
- Vetor de rampa de entrada: 0,001, 0,003, 0,01, 0,03, 0,08, 0,15, 0,3.

Estas condições são as mesmas que foram utilizadas na caracterização da biblioteca ASCEnD [17].

6.1 Caracterização do C-element

Um *C-element* foi caracterizado nas duas versões do LiChEn, com o objetivo de verificar erros causados pelas alterações realizadas na heurística para geração de EE, ATI e ATD. Os resultados encontrados foram exatamente os mesmos em ambas as versões demonstrando que a nova heurística não alterou a caracterização da célula. Os resultados estão descritos em [18].

6.2 Caracterização do OR PCHB

Através da nova heurística foi possível gerar todos os EE, ATI e ATD para a célula OR PCHB apresentada na Figura 5.4, que possui o comportamento descrito pela Equação 5.2. A árvore de computação possui ao todo 256 nodos, sendo 51 deles EE, 186 ATI e 69 ATD. A Figura 6.1 apresenta o consumo estático de potência para cada EE. Como a célula possuí apenas o pino de entrada En acionando um transistor PMOS, pode-se observar que quando este transistor está com sua entrada no nível lógico 0, tem-se um EE em que o consumo estático é muito inferior em comparação com um EE em que En esteja no nível lógico 1.

Por apresentar sua lógica composta sobretudo por transistores NMOS, transistores estes que possuem uma corrente de fuga muito maior comparada com transistores PMOS, o consumo interno de potência é menor quando os transistores PMOS encontram-se desligados, como ilustra a Figura 6.2.

A avaliação do atraso de transição e de propagação para a célula foi realizada verificando os atrasos obtidos através das simulações SPICE somente quando o pino alterava o seu valor lógico. As medições de atraso de transição são realizadas verificando o tempo que a saída leva para ir do nível lógico *low-to-high* ou *high-to-low*.

A Figura 6.3 mostra o atraso de transição para a saída V, onde se pode observar que com o aumento da carga de saída e da rampa e entrada, a transição de um estado para o outro fica mais lenta. Este comportamento é o mesmo para as demais saídas, o que se ilustra na Figura 6.4, que corresponde à saída C.0, porém representada como X no gráfico e para a saída C.1, representada pela letra W na Figura 6.5.

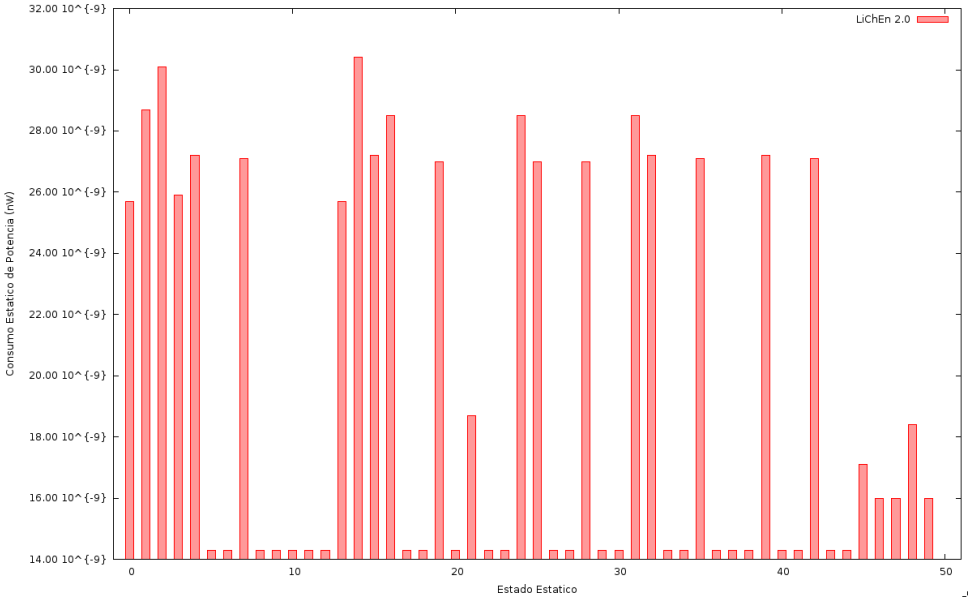

Figura 6.1 – Consumo Estático de Potência.

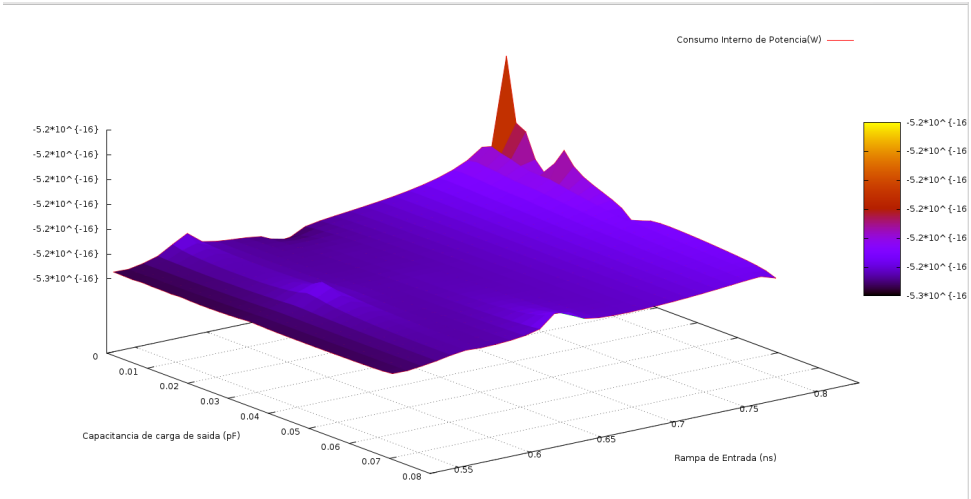

Figura 6.2 – Consumo Interno de Potência.

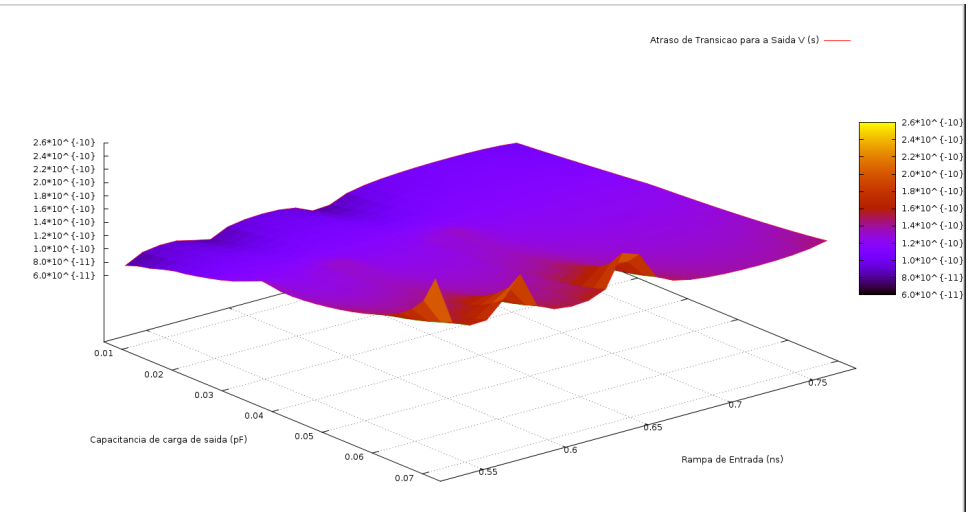

Figura 6.3 – Atraso de Transição para a saída V.

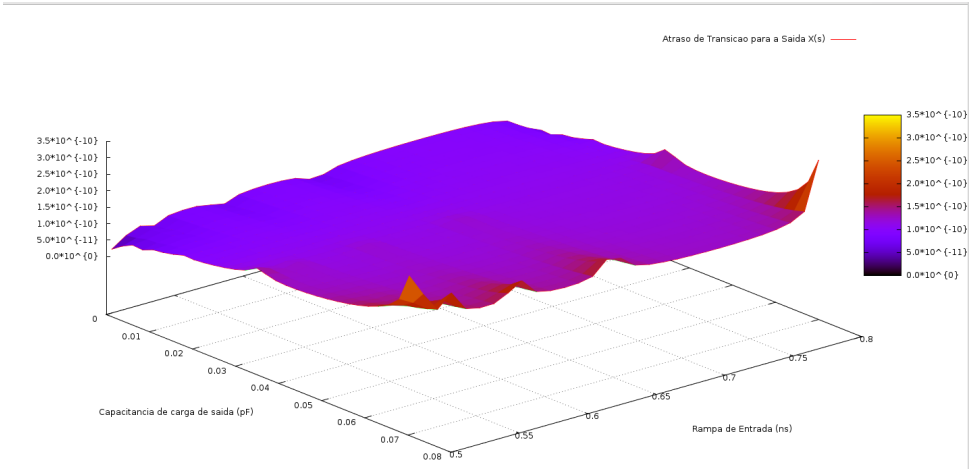

Figura 6.4 – Atraso de Transição para a saída X.

O atraso de propagação para cada saída V, C.0 (representado pela letra X) e C.1 (representado pela letra W) está ilustrado nas Figuras 6.6, 6.7 e 6.8,

Ele consiste no intervalo de tempo necessário para que uma alteração em um pino de entrada se propague até a saída. Pode-se observar que com o aumento da capacitância da carga de saída e da rampa de entrada, o tempo de propagação de uma saída aumenta, chegando no pior caso a 70ns para a saída V, quando a capacitância na carga de saída é 0,08pF e a rampa de entrada é 0,8ns.

Os resultados apresentados nesta Seção são o conteúdo que irá ser escrito no arquivo .lib, formato utilizado pela indústria e servem para o projetista escolher um célula que melhor se adapte às características desejadas de seu projeto. Como se pode observar, os atrasos de propagação e transição estão diretamente relacionados à capacitância da carga de saída e à rampa de entrada, onde a capacitância da carga de saída tem um comportamento predominante sobre a resposta do circuito. Isto ocorre porque com o aumento da

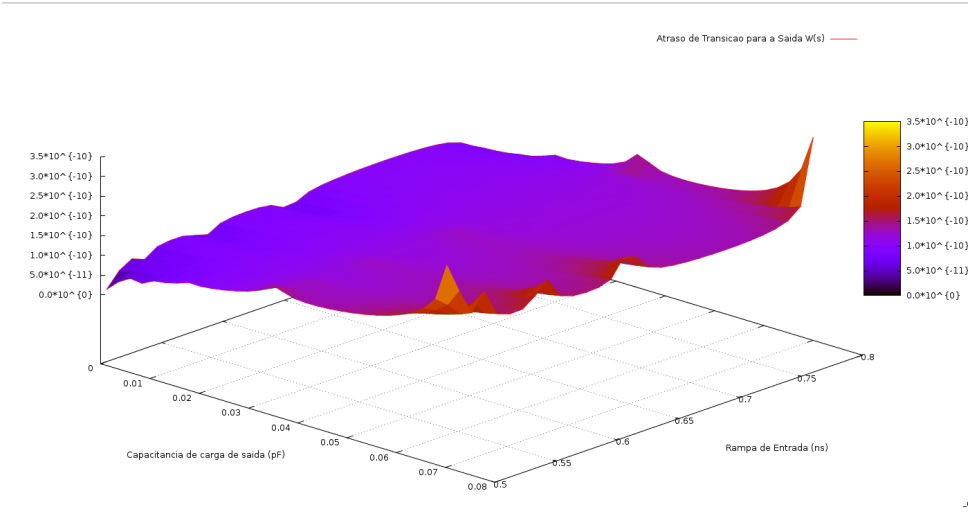

Figura 6.5 – Atraso de Transição para a saída W.

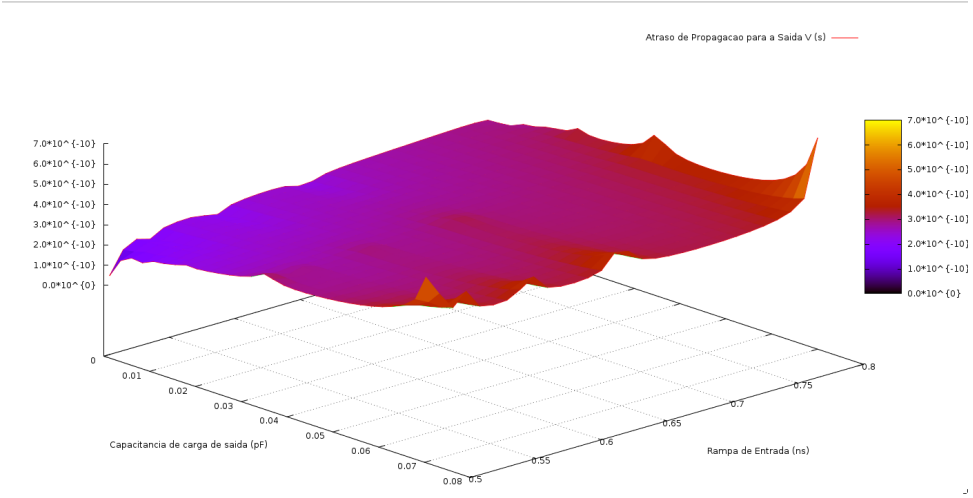

Figura 6.6 – Atraso de Propagação para a saída V.

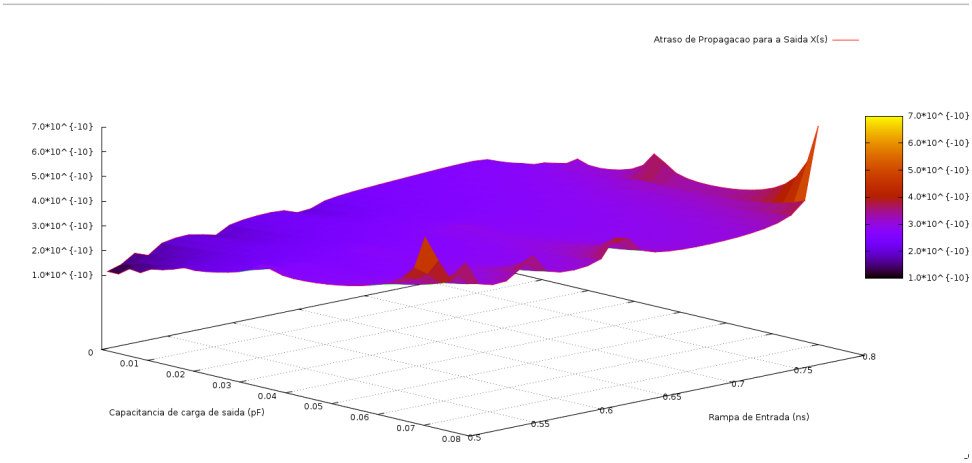

Figura 6.7 – Atraso de Propagação para a saída X.

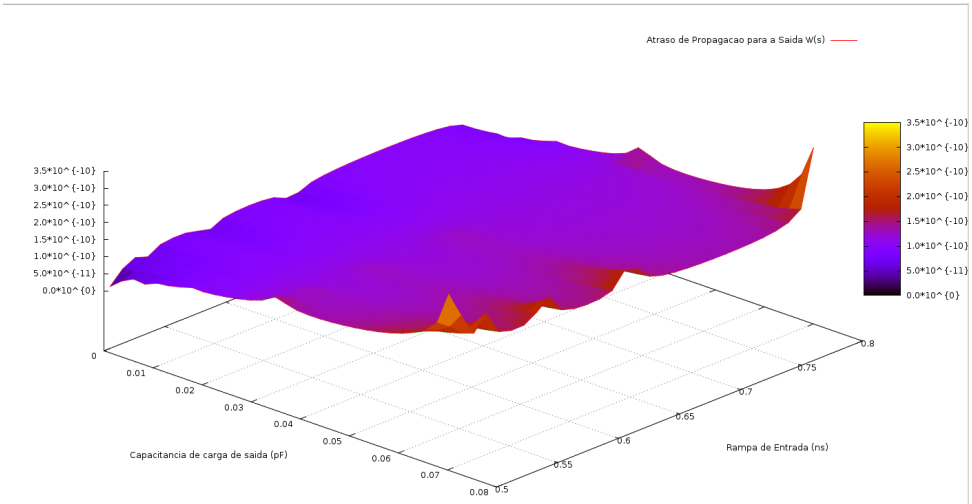

Figura 6.8 – Atraso de Propagação para a saída W.

capacitância de saída é necessário um tempo maior até atingir esta carga para carregar e um tempo maior para que esta carga se descarregue.

7. CONCLUSÃO E TRABALHOS FUTUROS

O trabalho descrito neste documento teve como principal objetivo o desenvolvimento de uma nova heurística para geração de EE, ATI e ATD para células padronizadas com múltiplas saídas. Essa heurística foi aplicada para a avaliação de consumo de potência estático, interno e modelos de atraso de transição e propagação. Isso permitiu o aumento do conjunto de células que podem ser caracterizadas usando a ferramenta LiChEn. Outra contribuição deste trabalho, foi o desenvolvimento de uma nova interface de linha de comando, aumentando a usabilidade da ferramenta.

Os conhecimentos obtidos nas disciplinas de microeletrônica e VLSI foram de essencial importância para possibilitar o desenvolvimento deste trabalho. Tais conhecimentos são referentes a estilos de projetos, modelos de atraso e potência e caracterização elétrica. Além disso, o trabalho apresentou diversos tópicos relacionados a pesquisa, como o estudo do paradigma assíncrono, o entendimento de componentes essenciais para este paradigma como *C-element* e células NCL.

Um conjunto maior de células com múltiplas saídas deve ser caracterizado, células tais como uma por AND PCHB, um meio somador PCHB e um somador completo PCHB. Também é necessário exportar os resultados obtidos para o formato *Liberty* para facilitar a verificação dos dados em ferramentas de EDA, possibilitando uma validação plena da nova LiChEn. Estes são trabalhos futuros para o desenvolvimento da ferramenta. Finalmente, cabe citar que o método aqui proposto foi validado com a caracterização de uma célula de múltiplas saídas.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] M. Amde, T. Felicijan, A. Efthymiou, D. Edwards, and L. Lavagno, “Asynchronous on-chip networks,” *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 2, pp. 273–283, Mar 2005.

[2] P. A. Beerel, R. O. Ozdag, and M. Ferretti, *A Designer’s Guide to Asynchronous VLSI*. Cambridge, MA: Cambridge University Press, 2010.

[3] P. Beerel, G. Dimou, and A. Lines, “Proteus: An asic flow for ghz asynchronous designs,” *Design Test of Computers, IEEE*, vol. 28, no. 5, pp. 36–51, Sept 2011.

[4] F. Bouesse, N. Ninon, G. Sicard, M. Renaudin, A. Boyer, and E. Sicard, “Asynchronous Logic vs Synchronous Logic: Concrete Results on Electromagnetic Emissions and Conducted Susceptibility,” in *6th International Workshop on Electromagnetic Compatibility of Integrated Circuits*, 2007, p. 5.

[5] Cadence, “Cadence design systems inc.” 2014. [Online]. Available: www.cadence.com

[6] K.-S. Chong, B.-H. Gwee, and J. S. Chang, “Energy-Efficient Synchronous-Logic and Asynchronous-Logic FFT/IFFT Processors,” *Solid-State Circuits, IEEE Journal of*, vol. 42, no. 9, pp. 2034–2045, Sept 2007.

[7] H. Eriksson, P. Larsson-Edefors, T. Henriksson, and C. Svensson, “Full-custom vs standard-cell design flow - an adder case study,” in *Asia and South Pacific Design Automation Conference*, 2003, pp. 507–510.

[8] I. T. R. for Semiconductors, “Itrs 2011 edition,” 2011. [Online]. Available: www.irts.net

[9] M. Hashimoto, K. Fujimori, and H. Onodera, “Standard cell libraries with various driving strength cells for 0.13, 0.18 and 0.35 µm technologies,” in *Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific*, Jan 2003, pp. 589–590.

[10] C. Heer, J. Dienstuhl, M. Krstic, and S. Salisbury, “Specification of characterization for the additional asynchronous standard cells for infineon 40 nm cmos process,” GALAXY, Tech. Rep., 2010. [Online]. Available: www.galaxy-project.org/files/D5_INFINEON_R022.pdf

[11] A. Jambek, A. NoorBeg, and M. Ahmad, “Standard cell library development,” in *Microelectronics, 1999. ICM ’99. The Eleventh International Conference on*, Nov 2000, pp. 161–163.

[12] I. C. Machado, R. B. Schivitz, C. Meinhardt, and P. F. Butzen, “An automatic flow for timing and static power cell characterization,” in *Microelectronics Students Forum 2013*, 2013.

[13] A. J. Martin, “Formal program transformations for vlsi circuit synthesis,” in *Formal Development of Programs and Proofs*, E. W. Dijkstra, Ed. Addison-Wesley, 1989, pp. 59–80.

[14] A. Martin and M. Nystrom, “Asynchronous Techniques for System-on-Chip Design,” *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1089–1120, June 2006.

[15] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Science, 1994.

[16] M. Moreira, A. Neutzling, M. Martins, A. Reis, R. Ribas, and N. Calazans, “Semi-custom ncl design with commercial eda frameworks: Is it possible?” in *Asynchronous Circuits and Systems (ASYNC), 2014 20th IEEE International Symposium on*, 2014, pp. 53–60.

[17] M. Moreira, “Design and implementation of a standard cell library for building asynchronous ASICs,” in *End of Term Work, Computer Engineering – PUCRS*, 2010, p. 139.

[18] M. Moreira and N. Calazans, “Electrical Characterization of a C-Element with LiChEn,” in *IEEE International Conference on Electronics, Circuits and Systems*, 2012, pp. 583–585.

[19] ——, “Design of standard-cell libraries for asynchronous circuits with the ASCEnD flow,” in *VLSI (ISVLSI), 2013 IEEE Computer Society Annual Symposium on*, Aug 2013, pp. 217–218.

[20] M. Moreira, R. Guazzelli, and N. Calazans, “Return-to-one Protocol for Reducing Static Power in C-elements of QDI Circuits Employing m-of-n Codes,” in *Integrated Circuits and Systems Design (SBCCI), 2012 25th Symposium on*, Aug 2012, pp. 1–6.

[21] M. Moreira, C. Menezes Oliveira, N. Calazans, and L. Ost, “LiChEn: Automated electrical characterization of asynchronous standard cell libraries,” in *Digital System Design (DSD), 2013 Euromicro Conference on*, Sept 2013, pp. 933–940.

[22] M. Moreira, B. Oliveira, F. Moraes, and N. Calazans, “Impact of C-elements in Asynchronous Circuits,” in *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, March 2012, pp. 437–343.

[23] M. Moreira, B. Oliveira, J. Pontes, and N. Calazans, “A 65nm standard cell set and flow dedicated to automated asynchronous circuits design,” in *SOC Conference (SOCC), 2011 IEEE International*, Sept 2011, pp. 99–104.

[24] M. Moreira, B. Oliveira, J. Pontes, F. Moraes, and N. Calazans, “Adapting a C-element design flow for low power,” in *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*, Dec 2011, pp. 45–48.

[25] M. Moreira, C. Oliveira, R. Porto, and N. Calazans, “Design of NCL gates with the ASCEnD flow,” in *Circuits and Systems (LASCAS), 2013 IEEE Fourth Latin American Symposium on*, Feb 2013, pp. 1–4.

[26] ——, “NCL+: Return-to-one null convention logic,” in *Circuits and Systems (MWSCAS), 2013 IEEE 56th International Midwest Symposium on*, Aug 2013, pp. 836–839.

[27] D. E. Muller and W. S. Bartky, *A Theory of Asynchronous Circuits*. International Symposium on the Theory of Switching, 1957.

[28] C. J. Myers, *Asynchronous circuit design.* Wiley-Interscience, 2001.

[29] S. Nowick and M. Singh, “High-performance asynchronous pipelines: An overview,” *Design Test of Computers, IEEE*, vol. 28, no. 5, pp. 8–22, Sept 2011.

[30] M. Prakash, “Library characterization and static timing analysis of asynchronous circuits,” Master’s thesis, Faculty of the USC Viterby School of Engineering, University of Southern California, 2007.

[31] J. M. Rabaey, A. Chandraksan, and B. Nikolic, *Digital Integrated Circuits a Design Perspective*. Prentice Hall, 2003.

[32] A. Royer, “Cli toolkit,” 2014. [Online]. Available: http://alexis.royer.free.fr/CLI

[33] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design - A Systems Perspective*. Springer, 2001.

[34] K. S. Stevens, D. Gebhardt, J. You, Y. Xu, V. Vij, S. Das, and K. Desai, “The future of formal methods and gals design,” *Electronic Notes in Theoretical Computer Science*, vol. 245, pp. 261–273, 2009.

[35] I. Sutherland, “Micropipelines,” *Communications of the ACM*, pp. 720–238, 2010.

[36] Synopsys, “Synopsys,” 2014. [Online]. Available: www.synopsys.com

[37] C. K. van Berkel, “Beware the isochronic fork,” *Integration, the VLSI Journal*, vol. 13, no. 2, pp. 103–128, Jun. 1992.

[38] C. K. van Berkel, M. B. Josephs, and S. M. Nowick, “Scanning the technology: Applications of asynchronous circuits,” in *Proceedings of the IEEE*, 1999, pp. 223–233.

ANEXO A – Comandos Disponíveis na Ferramenta LiChEn

Este anexo contém a lista de todos os comandos disponíveis e implementados na nova versão da LiChEn.

LiChen - Command Line Interface documentation

1. Command Line Interface LiChen (general presentation)

LiChen is composed of the following menus:

- LiChen (main menu) *No help available*

2. Menu LiChen (main menu)

LiChen is composed of the following commands:

print_all	Print All characterization environment.
devices	Display devices configuration.
library_name	Display library name.
set_library_name <lib_name>	Set library name.
print_library	Print all cells of the library.
config_devices [...]	config
add_cell -n <cell_name> -i <input_pins> -o <output_pins> -f <function> -a <area>	Area
add_schematics <path_sch>	Add schematic file(s).These file(s) must contain the schematics of the cell(s) to be characterized.
print_schematics	Print all schematics files.
add_slope <vector_name> <vector_values>	Add input slope - Values are separate with ,
remove_cell <cell_name>	Remove cell(s) from library. Cells listed separated by ,
print_cell <cell_name>	Print library cell(s) name(s).
print_slopes	Print environment slope vector(s). (Default is to print all vectors)
remove_slopes <vector_name>	Remove slope vector(s) from the environment. Vector name are separated by ,
add_load <vector_name> <vector_values>	Add output vector [vector_name] [vector_values]. Vector values are separated by ,
print_loads	Print environment load vector(s). (Default is to print all vectors)
remove_loads <vector_name>	Remove load vector(s) from the environment. Vector names are separated by ,
add_model_table <cell_name> <slope_vector_name> <load_vector_name>	Add model table [cell_name] [slope_vector_name] [load_vector_name].
set_voltage <voltage>	Sets operational voltage.
set_temp <temp>	Sets operational temperature.
set_process <process>	Sets simulation process.
set_models <models_file>	Sets models file.
set_vh <vh>	Sets logic 1 voltage value.
set_vl <vl>	Sets logic 0 voltage value.
set_vhtl <vhtl>	Sets high-to-low voltage value.
set_vlth <vlth>	Sets low-to-high voltage value.
set_start_time <start_time>	Sets simulation start time.
set_sim_time <sim_time>	Sets simulation total time.
set_sim_step <sim_step>	Sets simulation step.
set_load_unit <load_unit>	Sets global load unit.
set_time_unit <time_unit>	Sets global time unit.
set_resistance_unit <resistance_unit>	Sets global resistance unit.
set_vdd <vdd_label>	Sets vdd label.
set_gnd <gnd_label>	Sets gnd label.
set_max_tran <max_tran>	Sets maximum transition of the library.
print_configuration	Print environment settings.
characterize_library	Characterize Library.

2.1. print_all

Synopsis:
```
print_all
```

Description:
Print All characterization environment.

2.2. devices

Synopsis:
```
devices
```

Description:
Display devices configuration.

2.3. library_name

Synopsis:
```
library_name
```

Description:
Display library name.

2.4. set_library_name <lib_name>

Synopsis:
```
set_library_name <lib_name>
```

Description:
Set library name.

Parameters:
lib_name *(string)* *string value*

2.5. print_library

Synopsis:
```
print_library
```

Description:
Print all cells of the library.

2.6. config_devices [...]

Synopsis:
```
config_devices [-d <pos_drain>] [-g <pos_gate>] [-s <pos_source>] [-vdd <power_suply>] [-gnd <ground>] [-n
<nmos_model_name>] [-p <pmos_model_name>]
```

Description:
config

Options:

-d <pos_drain>	Drain
-g <pos_gate>	Gate
-s <pos_source>	Source
-vdd <power_suply>	Power Suply
-gnd <ground>	Ground
-n <nmos_model_name>	NMOS model name
-p <pmos_model_name>	PMOS model name

Parameters:

pos_drain	*(string)* *string value*
pos_gate	*(string)* *string value*
pos_source	*(string)* *string value*
power_suply	*(string)* *string value*
ground	*(string)* *string value*
nmos_model_name	*(string)* *string value*
pmos_model_name	*(string)* *string value*

2.7. add_cell -n <cell_name> -i <input_pins> -o <output_pins> -f <function> -a <area>

Synopsis:
```
add_cell -n <cell_name> -i <input_pins> -o <output_pins> -f <function> -a <area>
```

Description:
Area

Parameters:

cell_name *(string) string value*
input_pins *(string) string value*
output_pins *(string) string value*
function *(string) string value*
area *(string) string value*

2.8. add_schematics <path_sch>

Synopsis:
```
add_schematics <path_sch>
```

Description:
Add schematic file(s).These file(s) must contain the schematics of the cell(s) to be characterized.

Parameters:

path_sch *(string) string value*

2.9. print_schematics

Synopsis:
```
print_schematics
```

Description:
Print all schematics files.

2.10. add_slope <vector_name> <vector_values>

Synopsis:
```
add_slope <vector_name> <vector_values>
```

Description:
Add input slope - Values are separate with ,

Parameters:

vector_name *(string)* Vector name.
vector_values *(string)* Values are separate with , .

2.11. remove_cell <cell_name>

Synopsis:
```
remove_cell <cell_name>
```

Description:
Remove cell(s) from library. Cells listed separated by ,

Parameters:

cell_name *(string)* Vector name.

2.12. print_cell <cell_name>

Synopsis:
```
print_cell <cell_name>
```

Description:
Print library cell(s) name(s).

Parameters:

cell_name *(string)* Cell(s) name(s).

2.13. print_slopes

Synopsis:
```
print_slopes
```

Description:
Print environment slope vector(s). (Default is to print all vectors)

2.14. remove_slopes <vector_name>

Synopsis:
```
remove_slopes <vector_name>
```

Description:
Remove slope vector(s) from the environment. Vector name are separated by ,

Parameters:
vector_name *(string)* Vector name are separated by ,

2.15. add_load <vector_name> <vector_values>

Synopsis:
```
add_load <vector_name> <vector_values>
```

Description:
Add output vector [vector_name] [vector_values]. Vector values are separated by ,

Parameters:
vector_name *(string)* Vector name
vector_values *(string)* Vector values are separated by ,

2.16. print_loads

Synopsis:
```
print_loads
```

Description:
Print environment load vector(s). (Default is to print all vectors)

2.17. remove_loads <vector_name>

Synopsis:
```
remove_loads <vector_name>
```

Description:
Remove load vector(s) from the environment. Vector names are separated by ,

Parameters:
vector_name *(string)* Vector names are separated by ,

2.18. add_model_table <cell_name> <slope_vector_name> <load_vector_name>

Synopsis:
```
add_model_table <cell_name> <slope_vector_name> <load_vector_name>
```

Description:
Add model table [cell_name] [slope_vector_name] [load_vector_name].

Parameters:
cell_name *(string)* Cell name.
slope_vector_name *(string)* Slope vector name.
load_vector_name *(string)* Load vector name.

2.19. set_voltage <voltage>

Synopsis:
```
set_voltage <voltage>
```

Description:
Sets operational voltage.

Parameters:
voltage *(string)* Voltage operational value.

2.20. set_temp <temp>

Synopsis:
```
set_temp <temp>
```

Description:
Sets operational temperature.

Parameters:
temp *(string)* Operational temperature value.

2.21. set_process <process>

Synopsis:
```
set_process <process>
```

Description:
Sets simulation process.

Parameters:
process *(string)* Simulation process.

2.22. set_models <models_file>

Synopsis:
```
set_models <models_file>
```

Description:
Sets models file.

Parameters:
models_file *(string)* Simulation process.

2.23. set_vh <vh>

Synopsis:
```
set_vh <vh>
```

Description:
Sets logic 1 voltage value.

Parameters:
vh *(string)* Logic 1 voltage value.

2.24. set_vl <vl>

Synopsis:
```
set_vl <vl>
```

Description:
Sets logic 0 voltage value.

Parameters:
vl *(string)* Logic 0 voltage value.

2.25. set_vhtl <vhtl>

Synopsis:
```
set_vhtl <vhtl>
```

Description:
Sets high-to-low voltage value.

Parameters:
vhtl *(string)* High-to-low voltage value.

2.26. set_vlth <vlth>

Synopsis:
```
set_vlth <vlth>
```

Description:
Sets low-to-high voltage value.

Parameters:
vlth *(string)* Low-to-high voltage value.

2.27. set_start_time <start_time>

Synopsis:
```
set_start_time <start_time>
```

Description:
Sets simulation start time.

Parameters:
start_time *(string)* Simulation start time.

2.28. set_sim_time <sim_time>

Synopsis:
```
set_sim_time <sim_time>
```

Description:
Sets simulation total time.

Parameters:
sim_time *(string)* Simulation total time.

2.29. set_sim_step <sim_step>

Synopsis:
```
set_sim_step <sim_step>
```

Description:
Sets simulation step.

Parameters:
sim_step *(string)* Simulation step.

2.30. set_load_unit <load_unit>

Synopsis:
```
set_load_unit <load_unit>
```

Description:
Sets global load unit.

Parameters:
load_unit *(string)* Global load unit.

2.31. set_time_unit <time_unit>

Synopsis:
```
set_time_unit <time_unit>
```

Description:
Sets global time unit.

Parameters:
time_unit *(string)* Global time unit.

2.32. set_resistance_unit <resistance_unit>

Synopsis:
```
set_resistance_unit <resistance_unit>
```

Description:
Sets global resistance unit.

Parameters:
resistance_unit *(string)* Global resistance unit.

2.33. set_vdd <vdd_label>

Synopsis:
```
set_vdd <vdd_label>
```

Description:
Sets vdd label.

Parameters:
vdd_label *(string)* Vdd label.(separated with ,)

2.34. set_gnd <gnd_label>

Synopsis:
```
set_gnd <gnd_label>
```

Description:
Sets gnd label.

Parameters:
gnd_label *(string)* gnd label. (separated with ,)

2.35. set_max_tran <max_tran>

Synopsis:
```
set_max_tran <max_tran>
```

Description:
Sets maximum transition of the library.

Parameters:
max_tran *(string)* Maximum transition of the library.

2.36. print_configuration

Synopsis:
```
print_configuration
```

Description:
Print environment settings.

2.37. characterize_library

Synopsis:
```
characterize_library
```

Description:
Characterize Library.