Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Unidade Acadêmica de Engenharia Elétrica

Érico Ramalho de Freitas

Design e Caracterização de uma Biblioteca de Células Digitais

Campina Grande - Paraíba
Dezembro de 2018

Érico Ramalho de Freitas

Design e Caracterização de uma Biblioteca de Células Digitais

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica

Área de Concentração: Microeletrônica

Marcos Ricardo Alcântara Morais

Orientador

Gutemberg Gonçalvez dos Santos Júnior

Convidado

Campina Grande - Paraíba

Julho de 2018

Lista de Figuras

Lista de Códigos

Sumário

Resumo

Uma biblioteca de células digitais contém informação detalhadas de perfomance, consumo e área sobre diferentes condições de entrada e saída. Estas informações estão compiladas em um formato padronizado que as ferramentas de síntese possam interpretar e utilizar na síntese de módulos em RTL. Neste projeto será descrito as etapas de projeto e caracterização de uma biblioteca de células e sua validação por meio da síntese de um módulo de exemplo.

Palavras-Chave: células digitais, caracterização, síntese

Abstract

A standard cells library contains detailed informations about timing, power and area under different conditions of input and output. Those informations are compiled in a standardized format that synthesis tools can interpret and make use of it on RTL modules synthesis. In this project it will be described the phases of design and characterization of a standard cells library and its validation by synthesis of a example module.

Keywords: standard cells, characterization, synthesis

1 Introdução

O projeto de um circuito integrado digital passa por uma etapa de síntese lógica em que o *design* em RTL do circuito é traduzido, por meio de ferramenta de síntese, em portas lógicas e suas conexões. O resultado desta síntese é chamado de *netlist*. A ferramenta de síntese busca de um acervo de células digitais, chamadas de *standard cells*, disponíveis na biblioteca escolhida pelo projetista para montar um circuito que realize a função daquele projeto de forma eficiente e dentro das projeções de perfomance, consumo e área.

Dois arquivos principais são necessários para se compor uma biblioteca de células digitais: o **.lib** e o **.lef**. O primeiro arquivo deve possuir informações referentes ao comportamento a célula sob diferentes circustâncias de entrada e saída. Dentre estas informações estão os atrasos relativos entre sinal de entrada e saída e também o consumo da célula. O segundo possui informações físicas de produção da célula, como sua área, as camadas de metal que compõe os seus pinos as formas como estas camadas estão distribuídas.

Ambos esses arquivos são desenvolvidos através de um processo de *design* e caracterização de células. Este processo tem como base arquivos de tecnologia provenientes do *Process Design Kit*, ou PDK, da *foundry*. Modelos de dispositivos primitivos, como transistores, e regras de DRC servirão como parâmetros no *design* e caracterização da biblioteca de *standard cells*.

Com todas essas informações as ferramentas de síntese podem realizar cálculos de perfomance, consumo e área da *netlist* sintetisada, realizar otimizações e gerar relatórios de **QoR** (*Quality of Results*).

1.1 Objetivo

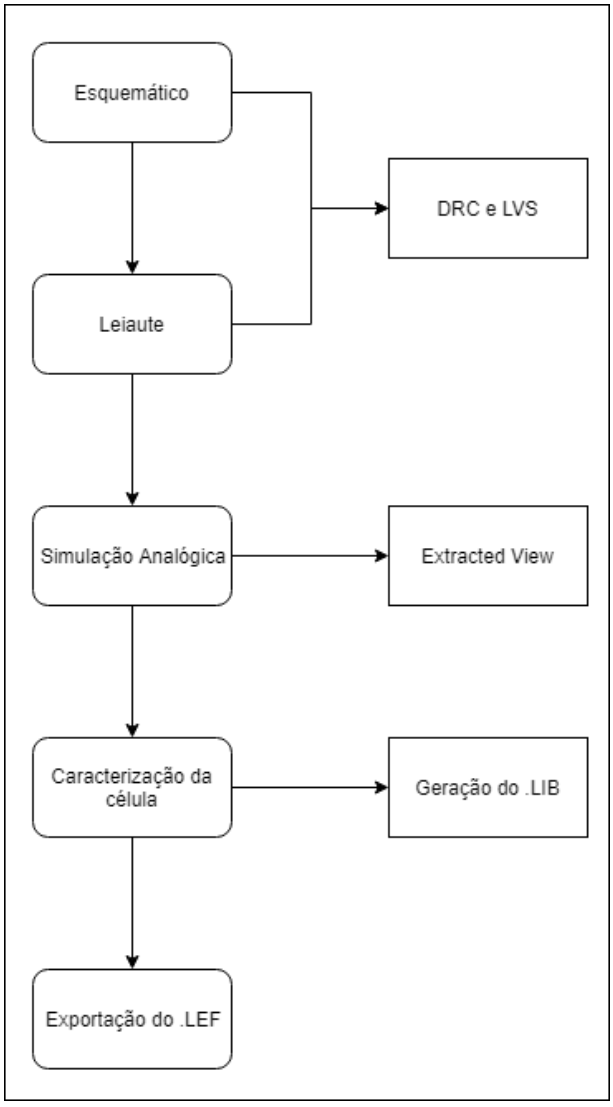
O objeto desse projeto é descrever passo à passo as etapas de desenvolvimento e caracterização de uma biblioteca de células digitais. Serão descritos desde a construção do esquemático e leiaute das células até suas caracterizações. Seguindo os passos desse trabalho será possível obter os arquivos que compõem uma biblioteca de *standard cells* funcional.

Para validar a funcionalidade desta biblioteca será executado a síntese de um módulo em RTL de exemplo e demonstrado os resultados dos relatórios de QoR.

2 Fluxo do Projeto

O projeto de uma biblioteca de células digitais é definido pelo fluxo descrito na Figura 1. No fim dessas etapas teremos obtido o **.lib** (Liberty file) e o **.lef** (Library Exchange Format file), que serão usados na síntese lógica e no backend de um IP.

Figura 1: Fluxo de Projeto



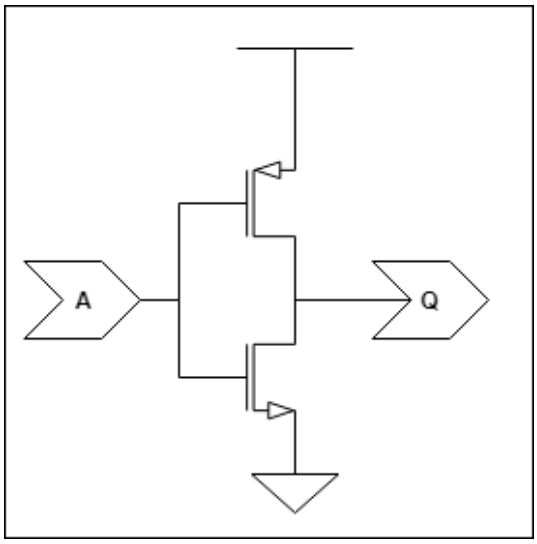
Fonte: Autor

As etapas de construação de esquemático e leiaute devem ser repitidas para cada uma das células, enquanto que a caracterização e exportação podem ser realizadas com todo o conjunto de células por meio de scripts.

2.1 Esquemático

A primeira etapa na criação de uma célula é a montagem do esquemático, composto de uma combinação de MOSFETs tipo p e n. A tecnologia mais utilizada é chamada de CMOS (Complementary Metal-Oxide Semicondutor). Ela define cade tipo de células digital como um conjunto de pares complementares de transistores pMOS e nMOS. O consumo das células em estado fixo é baixíssimo, só ocorrendo maior consumo durante a transição de estados ligado e desligado dos transistores. Na figura 2 vemos a definição de uma célula inversora em tecnologia CMOS.

Figura 2: Esquemático de um inversor.



Fonte: Autor

Uma maneira de construir o esquemático da céula é através da geração automática a partir de uma *netlist* estrutural sintetisada de uma descrição em *Verilog* por uma ferramente de síntese, como o **dc_shell** da Synopsys ou **Genus** da Cadence.

Outra maneira é construir o esquemático a mão, componente por componente. Para isso devemos instanciar os modelos de componentes primtivos nMOS e pMOS existentes no PDK e conectá-los para formar a célula sendo construida. As entradas e saídas da célula são definidas por meio de pinos. É necessário criar os pinos de entrada e saída e nomeá-los devidamente. Os pinos de alimentação também devem ser definidos e nomeados corretamente para que as ferramentas de *Place and Route* consigam fazer a conexão correta. É importante que a direção do sinal dos pinos de entrada e saída esteja definida corretamente, assim como os pinos de alimentação que devem ser definidos como *inout*.

Neste momento podem ser escolhidos alguns parâmetros dos transistores que influenciam na performance e no *fan-out* da célula. O comprimento L do canal é definido pela tecnologia usada (180nm, 130nm, 45nm, 28nm,...). O *fan-out* está diretamente relacionado à corrente que a célula consegue fornecer na sua saída. A corrente de dreno-fonte I_{ds} de um transistor nMOS saturado é dado por:

$$I_{ds} = \frac{\beta}{2} V_{GT}^2 \tag{1}$$

Onde:

$$\beta = \mu C_{ox} \frac{W}{L} \tag{2}$$

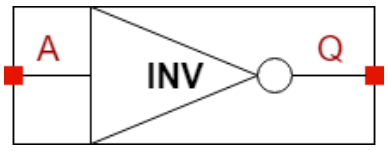
$$V_{GT} = V_{gs} - V_{th} \tag{3}$$

Sendo C_{ox} a capacitância por unidade de área do óxido do *gate*. O transistor pMOS se comporta de forma semelhante, apenas invertendo todas as tensões e correntes.

2.1.1 Vista de Símbolo

Uma vista que será útil para a manipulação da célula em momentos futuros é a vista de símbolo. Ela representa todo emaranhado de transistores apenas como uma caixa preta com entradas e saídas. O desenho desta caixa também pode ser editado para se paracer com os símbolos padrões de cada tipo de célula.

Figura 3: Símbolo de um Inversor



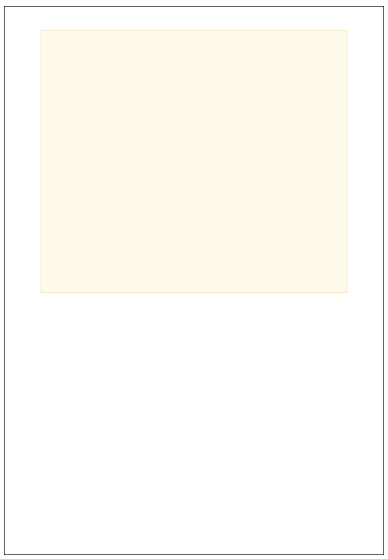
Fonte: Autor

2.2 Leiaute

O leiaute da célula se refere ao desenho das estruturas físicas que irão definir os transistores. O projetista utiliza de ferramentas de edição gráfica, como o Virtuoso, para desenhar formas geométricas representando o polisilício, semicondutores tipo N e P, e metais que compõem os MOSFETs. Cada camada desenhada aqui definirá a *photomask* utilizada no processo de fabricação na *foundry*. Por último será extraído do desenho um esquemático de transistores que será comparado com a vista de esquemático da célula definida anteriormente.

Normalmente o leiaute é feito sobre uma base de substrato tipo $\boldsymbol{p}$. Para construir os transistores pMOS é preciso desenhar um poço de substrato tipo $\boldsymbol{n}$. A Figura 4 mostra o retângulo que de define o poço dentro do substrato.

Figura 4: Poço NWELL

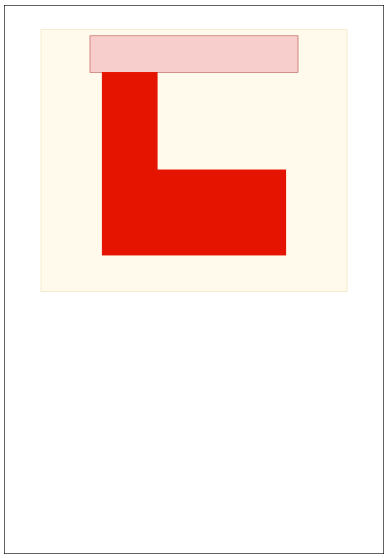


Fonte: Autor

Dentro do poço desenhamos o retângulo de difusão p_+. A forma deve ter a mesma largura do transistor pmos definido no esquemático. Ela deve também ter comprimento suficiente para comportar o canal de comprimento L definido pelo processo utilizado. Para evitar que a junção da difusão p_+ com o poço n polarize diretamente e haja fuga de corrente é preciso amarrar o poço à um potencial positivo. É necessário então adicionar uma difusão tipo n_+ fortemente dopada (chamada de *well tap*) ao poço que deverá ser conectado ao pino de V_{DD} da célula.

Na Figura 5 vemos a difusão p_+ em vermelho. Em rosa vemos a difusão n_+ que servirá para amarrar o poço n ao V_{DD}.

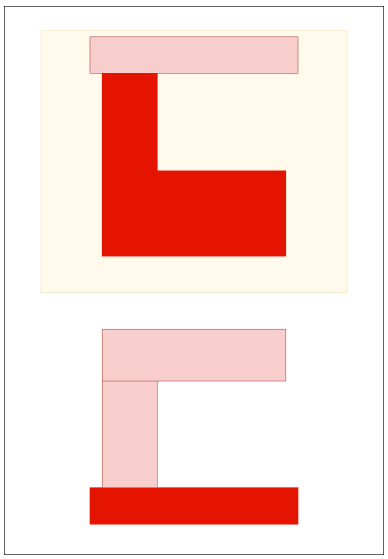
Figura 5: Difusão P_+ e N_+ Well Tap



Fonte: Autor

De forma semelhante os transistores nMOS são desenhados à baixo, fora do poço n e diretamente no substrato p. As medidas definidas no esquemático também devem ser respeitadas. Assim como nos transistores pMOS, para evitar polarização direta do substrato p com a difusão n_+, deve-se amarrar o substrato ao terra por meio de difusão p_+ fortemente dopada.

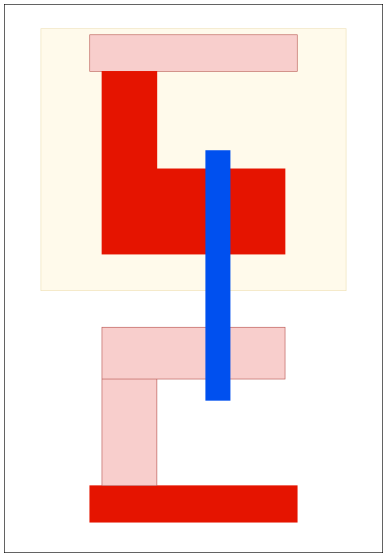
Figura 6: Difusões dos transistores pMOS e nMOS



Fonte: Autor

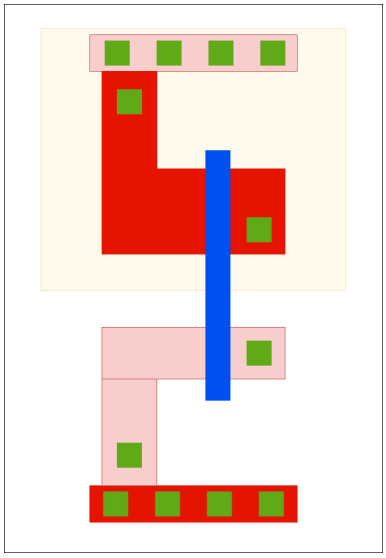
Em seguida é desenhado o polisilício que define o *gate* dos transistores. A camada de polisilício sobrepondo a difusão divide essa região entre o dreno e a fonte do transistor, delimitando o espaço onde se formará o canal. Consequemente a largura da região diretamente sobre a difusão deve ser igual a medida L do processo. Uma região maior de camada de polisilício entre os transistores é desenhada para a implantação dos contatos posteriormente. Na Figura 7 vemos a camada de polisilício definindo o transistores pMOS e nMOS, além de agir como conexão entre os dois *gates*.

Figura 7: Polisilício



Fonte: Autor

Figura 8: Contatos



Fonte: Autor

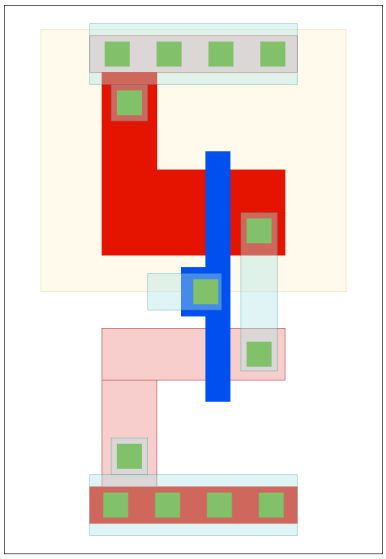
Os contatos na figura 8 servem para conectar as camadas de difusão e polisilício às camadas superiores de metal. São necessários colocar contatos nas seguintes regiões:

- *Taps* de poço n e substrato p.
- Polisilícios.
- Dreno do pMOS e fontes do nMOS que serão conectados ao V_{DD} e GND.
- Drenos e fontes que serão conectados por metal.

A próxima camada a ser inserida é a metal 1. As camadas de metal fazem as conexões de entrada e saída, os trilhos de V_{DD} e GND, e qualquer outra conexão interna da célula. Como o metal 1 está em uma camada superior as outras ele só faz conexão com as camadas de difusão e polisilício através dos contatos. As rotas de metal 1 devem então sobrepor os contatos aos quais elas irão se conectar.

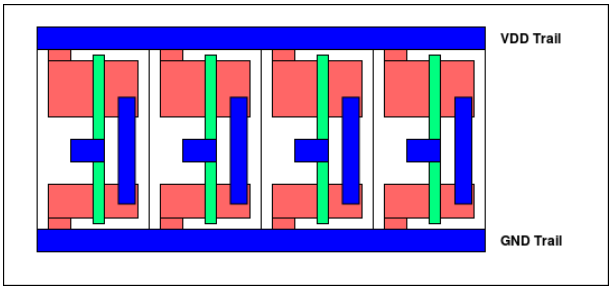
Na figura 9 pode-se ver o metal 1: sobre o contato do polisilício formando a entrada do inversor, conectando a fonte do pMOS ao dreno do nMOS e formando a saída, formando as vias de V_{DD} e GND. As vias de V_{DD} e GND devem especialmente serem localizadas no topo e na base das células, se estenderem de um lado ao outro e possuir a mesma largura. No momento do *Place and Route* as células são dispostas em fileiras e suas conexões de energia devem formar uma trilha uniforme, como pode ser visto na figura 10.

Figura 9: Metal 1



Fonte: Autor

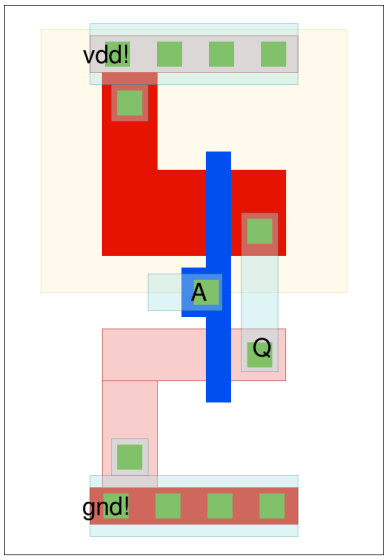
Figura 10: Trilhas de V_{DD} e GND



Fonte: Autor

As entradas e sáidas da célula, assim como os pontos de alimentação, precisam ser identificadas e etiquetadas. Isso é feito por meio da criação de pinos na mesma camada de metal em que estão localizados estes pontos. Os pinos devem ser etiquetados com os mesmos nomes dados aos pinos da célula no esquemático. Isto é necessário para comparação futura de funcionalidade com o esquemático.

Figura 11: Pinos



Fonte: Autor

2.2.1 DRC

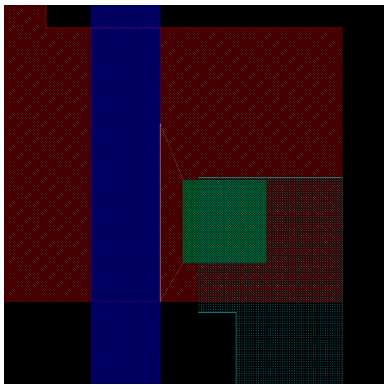
O DRC(*Design Rules Checking*) é um passo no projeto em que a ferramenta irá analizar o leiaute desenhado e verificar se as dimensões das formas condizem com as regras estabelecidas pelo PDK da *Foundry*. Estas regras de *design*, ou regras de leiaute, se referem a medidas de largura, separaração e sobreposição dos objetos de cada camada. Elas tem o intuito de gerar um circuito confiável no menor espaço possível.

As regras são *constraints*, ou restrições, geométricos para que o processo de frabricação mantenha a topologia e a geometria do desenho. Porém, essas regras não definem a correta ou incorreta fabricação do circuito. Elas são margens de tolerância que promovem uma grande probabilidade do circuito funcionar correntamente. Casos de circuito funcionais mesmo com algumas violações, e vice-versa, ainda podém ocorrer.

Apesar dessas regras poderem ser seguidas durante o processo de desenho do leiaute, sua quantidade torna difícil a visualização apenas por olho nu. Por isso torna-se necessário a utilização de ferramentas específicas de verificação, como **DIVA** ou **Assura**.

As ferramentas analizam todo o leiaute em busca de violação de *design*, descritas por arquivos no PDK da tecnologia, gerando marcadores para fácil visualização. Na figura 12 vemos um marcador indicando uma violação de mínima distância entre um contato sobre difusão e uma camada de polisilício.

Figura 12: Violação de DRC



Fonte: Autor

Após corrigir as violações e rodar novamente o DRC até que o leiaute esteja limpo, passamos agora para a verificação de funcionalidade.

2.2.2 Extração

A partir do vista leiaute da célula é essencial realizar a extração da *netlist* de transistores representado por aquele desenho. A ferramenta interpreta, por exemplo, uma camada de polisilício sobrepondo um retângulo de difusão tipo n_+ como um transistor nMOS. Enquanto que na vista de leiaute as camadas tem propósito e desenho, as camadas da vista extraída tem o propósito de *"net"*.

2.2.3 LVS

O processo de LVS(*Layout versus Schematic*) consiste em verificar se a *netlist* de transistores capturada pela vista extraída representa a mesma *netlist* da vista de esquemático. Ao rodar o LVS a ferramenta irá dizer se o circuito descrito pelas duas vistas é mesmo, mas não se o circuito está correto.

Um LVS bem sucedido é necessário para a continuação do fluxo. Qualquer incompatibilidade precisa ser corrigida, seja ela no leiaute ou no esquemático. Neste momento é necessário que os pinos da célula tenham sido nomeados do mesmo jeito em ambas as vistas. O parâmetro W dos transistores descritos no esquemático deve condizer com a

largura do retângulo de difusão desenhado no leiaute para que eles representem o mesmo transistor. Se o LVS não for bem sucedido, a ferramenta abrirá uma janela mostrando com os transistores no esquemático e leiaute qu estão incompatíveis. Há também uma opção para dar zoom sobre os transistores incompatíves em ambas as vistas para mais fácil correção.

2.2.4 *Analog-extracted view*

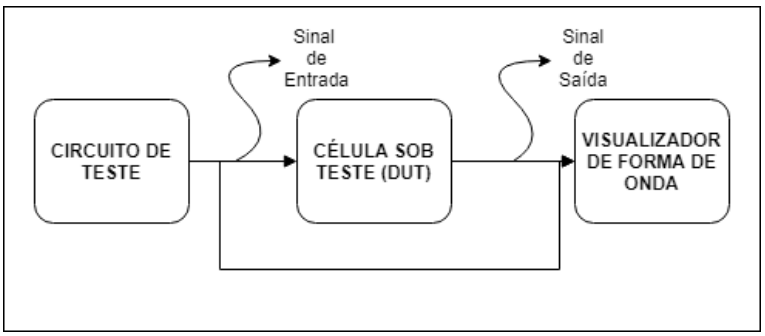
Após a verficação bem sucedida do LVS podemos agora extrair uma nova vista que irá incluir as capacitâncias parasitas extraidas pela ferramenta. Esta nova vista, chamada de *analog-extracted*, possui informações adicionais de fonte de alimentação permitindo uma simulação analógica da célula por ferramentas como **Spectre**.

2.2.5 Simulação Analógica

Uma análise fiél do comportamento real da célula pode ser feito usando ferramentas de simulação analógica, como *Spice* ou *Spectre*. Estas ferramentas modelam os transitores como dispositivos analógicos, não somente como chaves liga-desliga. Assim, será possível verificar características como tempo respota da saída à variações de entrada e a rampa de subida e descida durante a mudança desse sinal.

Para verificar o comportamento analógico da célula primeiramente precisamos montar uma ambiente de simulação. Criamos um novo esquemático onde será construido um circuito de teste para excitar as entradas da célula e verificamos a forma de onda da entrada e saída como visto na figura 13

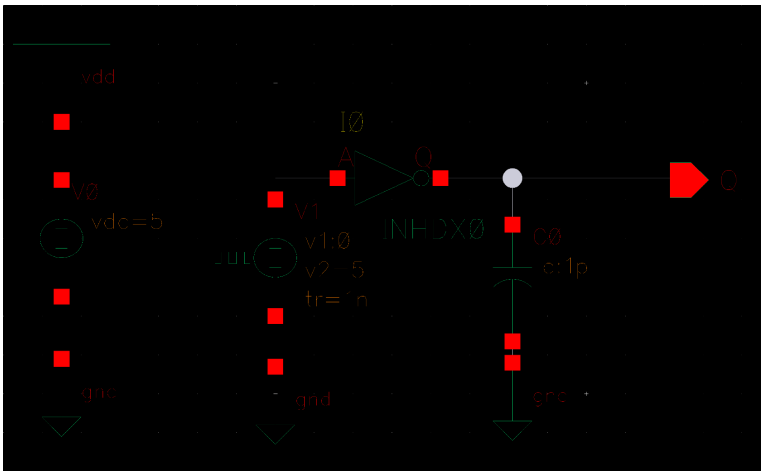
Figura 13: Ambiente de simulação



Fonte: Autor

Primeiramente instanciamos a célula sob teste no esquemático. Esta irá vir como vista de símbolo gerada anteriormente. Para a montagem do circuito de teste deverá haver uma biblioteca de componentes analógicos como fonte de tensão, gerador de pulsos, capacitores, etc. É essencial a instanciação de um fonte de tensão CC conectada em uma de suas extremidades à um V_{DD} e a outra à um GND. Definindo um valor para tensão da fonte neste conjunto irá definir a tensão de referência entre os pinos de alimentação da célula. Para variar as entradas da célula colocamos geradores pulsos, variando de 0 até o valor da fonte e em diferentes *duty-cicles* para assim conseguirmos todas as combinações de entrada. Algumas outros parâmetros do gerador, como tempo e subida e descida, podem ser ajustados para tornar o pulso menos ideal e mais próximo da realidade. Uma sugestão para a simulação é a adição de um capacitor, ou até outra célula, à saída da célula sob teste para simular o efeito de uma carga capacitiva. Na figura 14 vemos o esquemático de teste de uma célula inversora.

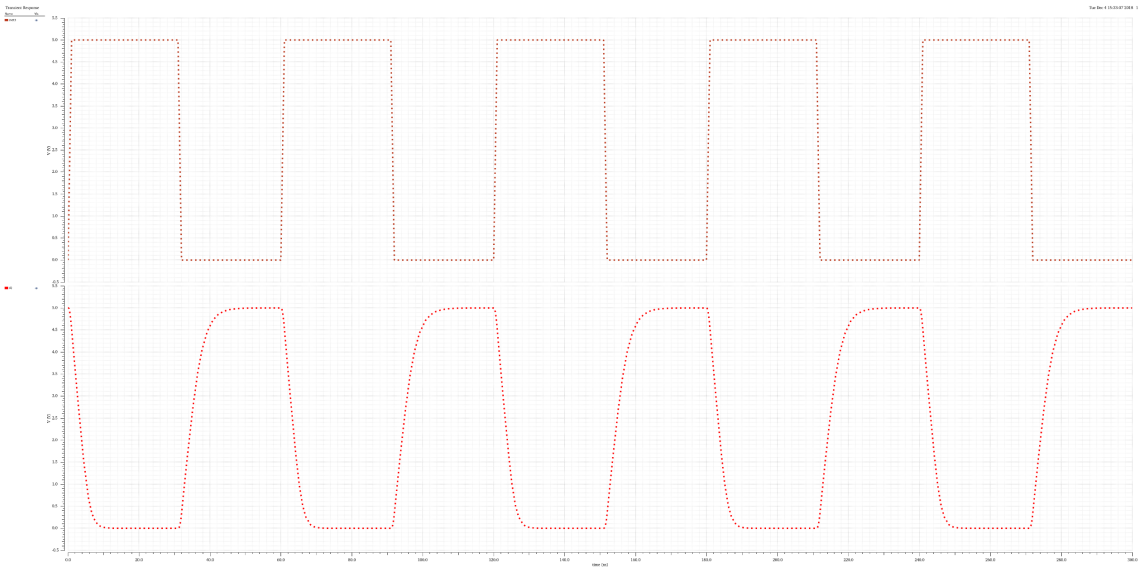
Figura 14: Testbench do inversor



Fonte: Autor

Para simular e visualizar as formas de onda usamos a ferramenta ADE (*Analog Design Enviroment*). Nas configurações de ambiente definimos o tipo de análise como transitório e escolhemos o tempo de parada para simulação, por exemplo 300 nanosegundos. Definimos também as *nets* que queremos visualizar. Podemos escolher todas as *nets* conectadas as entradas e saídas da célula para analisar seu comportamento em paralelo. Depois configurado o ambiente podemos então rodar a simulação. Na figura 15 podemos os pulsos de entrada do inversor e comportamento da saída em dois gráficos.

Figura 15: Forma de onda de um inversor



Fonte: Autor

2.3 Caracterização

A etapa de caracterização é uma das mais importantes no desenvolvimento de uma célula digital. É ela que nos dará informações que ainda não temos, como a carga de entrada, performance e consumo da célula. Até o momento, com as vista de esquemático e leiaute, nós obtemos a *netlist* de transistores representada pela célula. Porém, as ferramentas de síntese precisam de muito mais informações para que elas possam sintetisar uma descrição comportamental em uma conjunto de células digitais. Informações como a performance da célula sobre diferentes condições de carga de saída e diferentes rampas e entrada, assim como o consumo da célula, são pontos levados em consideração pelas ferramentas na hora de síntetisar uma *netlist* e na hora do *place and route* e suas otimizações. É neste momento que entra a caracterização da célula.

O processo de caracterização consiste na simulação das células digitais em um ambiente analógico e então extrair informações de uma forma que outras ferramentas possam entender e fazer uso delas. Neste trabalho foi usado a ferramenta de caracterização Liberate que utiliza Spectre ou HSpice como ferramentas de simulação.

2.3.1 *Liberty File Format*(.lib)

As informações extraída na caracterização precisam ser codificadas em um formato padrão. Este é formato é o *liberty*, que geralmente usa a extensão **.lib**. A listagem 1 mostra

a estrutura geral de uma arquivo *liberty*. Cada uma das sessões desse arquivo possuem inúmeros detalhes que podem ser adicionados a mão ou podem ser automaticamente gerados pela ferramenta de caracterização.

```
/* General Syntax of a Technology Library */
library (nameoflibrary) {
... /* Library level simple and complex attributes */
... /* Library level group statements */
... /* Default attributes */
... /* Scaling Factors for delay calculation */
/* Cell definitions */
cell (cell_name) {
... /* cell level simple attributes */
/* pin groups within the cell */
pin(pin_name) {
... /* pin level simple attributes */
/* timing group within the pin level */
timing(){
... /* timing level simple attributes */
} /* end of timing */
... /* additional timing groups */
} /* end of pin */
... /* more pin descriptions */
} /* end of cell */
... /* more cells */
} /* end of library */
```

Código 1: Estrutura de Liberty File Format

A listagem 2 mostra o exemplo de um *header* do arquivo. Todas essas informações estão acima da definição da primeira célula. Na primeira linha vemos a definição do **delay_model** como **table_lookup**. Este não é único modelo, mas é o mais largamente utilizado. Ele define os atrasos como uma tabela de atrasos que foram simulados com diferentes condições de entrada e saída. Neste caso, um eixo da tabela é definido pele rampa de transição da entrada e o outor eixo pela carga capacitiva da saída. A ferramenta de síntese pode então estimar com uma precisão razoável (desde que os dados da tabela sejam o suficiente e estejam precisos) o atraso da célula, conhecendo informações sobre as conexões da mesma.

Logo abaixo há as definições das unidades padrões dos diversos parâmetros elétricos utilizados na descrição da célula. As definições de **threshold** indicam em que posição da forma de onda os atrasos e *slew* são computados. Isto implica que, segundo consta no *header*, o *slew* de subida e descida será calculado entre os pontos de 20% e 80% das ondas de saída e entrada. Então, o tempo será medido do ponto de 20% da rampa de subida ou descida até o ponto de 80% da rampa. Estes pontos são porcentagens do V_{DD}. O atraso do sinal será medido do ponto de 50% da rampa de subida ou descida da entrada até o ponto de 50% da rampa de subida ou descida da saída.

```
delay_model : table_lookup;
capacitive_load_unit (1,pf);
current_unit : "1mA";
leakage_power_unit : "1nW";
pulling_resistance_unit : "1kohm";
time_unit : "1ns";
voltage_unit : "1V";
voltage_map (VDD, 1.8);
voltage_map (GND, 0);
voltage_map (VSS, 0);
default_cell_leakage_power : 0;
default_fanout_load : 1;
default_max_transition : 4.1;
default_output_pin_cap : 0;
in_place_swap_mode : match_footprint;
input_threshold_pct_fall : 50;
input_threshold_pct_rise : 50;
nom_process : 1;
nom_temperature : 25;
nom_voltage : 1.8;
output_threshold_pct_fall : 50;
output_threshold_pct_rise : 50;
slew_derate_from_library : 1;
slew_lower_threshold_pct_fall : 20;
slew_lower_threshold_pct_rise : 20;
slew_upper_threshold_pct_fall : 80;
slew_upper_threshold_pct_rise : 80;
operating_conditions (PVT_1P8V_25C) {
  process : 1;
  temperature : 25;
  voltage : 1.8;
}
default_operating_conditions : PVT_1P8V_25C;
lu_table_template (constraint_template_3x3) {
  variable_1 : constrained_pin_transition;
  variable_2 : related_pin_transition;
  index_1 ("0.006, 0.156844, 4.1");
  index_2 ("0.006, 0.156844, 4.1");
}
lu_table_template (delay_template_5x5) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  index_1 ("0.006, 0.0306768, 0.156844, 0.80191, 4.1");
  index_2 ("0.01, 0.0271101, 0.0734956, 0.199247, 0.54016");
}
lu_table_template (mpw_constraint_template_3x3) {
  variable_1 : constrained_pin_transition;
  index_1 ("0.006, 0.156844, 4.1");
}
power_lut_template (passive_power_template_5x1) {
  variable_1 : input_transition_time;
  index_1 ("0.006, 0.0306768, 0.156844, 0.80191, 4.1");
}
power_lut_template (power_template_5x5) {
  variable_1 : input_transition_time;
  variable_2 : total_output_net_capacitance;
  index_1 ("0.006, 0.0306768, 0.156844, 0.80191, 4.1");
  index_2 ("0.01, 0.0271101, 0.0734956, 0.199247, 0.54016");
}
```

Código 2: Exemplo de Liberty Format header

Ao final do *header*, antes das definições de células, temos os ***lu_table_template*** e ***power_lut_template***. Estas são as *look up tables* que irão definir ambas as computações de atraso e consumo. No exemplo temos definido os eixos de uma tabela 5x5 em que o primeiro eixo é o ***input_net_transition*** e o segundo é o ***total_output_net_capacitance***. Os valores reais da tabela estarão definidos na sessão de definição de cada célula.

2.3.2 *Template File*

Um dos arquivos mais importantes para a ferramenta Liberate na caracterização das células é o arquivo de modelo (*Template File*) da biblioteca. Esse arquivo possui informações sobre as células e seus pinos, e sobre índices das tabelas de caracterização (*look up table*). O Liberate utilizará as informações nesse modelo para definir os parâmetros de simulação e construir o arquivo Liberty.

Existem duas formas de se obter um arquivo modelo:

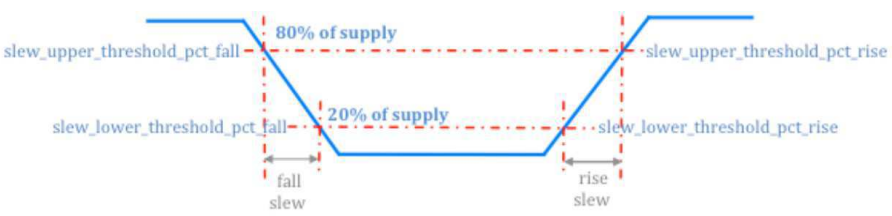
- Gerar o modelo a partir de um arquivo Liberty existente.
- Criar manualmente o arquivo modelo.

A geração a partir de uma biblioteca ocorre simplesmente usando os comandos de leitura do *.lib* e escrita do *template*. Porém ela exige um arquivo Liberty pronto, que será difícil de ter se essa for a primeira rodada de caracterização. A próxima opção é a criação manual do arquivo.

Primeiramente devemos criar o arquivo e definir algumas variáveis do processo. Algumas dessas variáveis são os ***slew thresholds***. Devemos então definir:

- *slew_lower_rise* e *measure_slew_lower_rise*
- *slew_upper_rise* e *measure_slew_upper_rise*
- *slew_lower_fall* e *measure_slew_lower_fall*
- *slew_upper_fall* e *measure_slew_upper_fall*

Figura 16: Pontos de medida do *slew* de entrada



Todas as células que serão caracterizadas precisam estar definidas no modelo com o comando **define_cell**. Este comando define os pinos de entrada e saída da célula e os valores dos índices das tabelas de atraso e consumo. Esses valores podem ser definidos a mão. Porém, isso se tornaria um trabalho tediante se tivesse que ser feito para todas as células da biblioteca. Uma maneira de minimizar o trabalho é definindo modelos de tabelas e deixando que a ferramente defina os valores a partir desses modelos e de algumas outras variáveis. Para isso usamos o comando **define_template** e definimos o tamanho que queremos da tabela colocando a quantidade de números por índice, seja ela 3x3, 5x5, etc. Os números colocados nesse template não importam. Eles serão substituídos posteriormente.

Com esses modelos de tabela definidos e atribuídos a definição de cada célula é necessário uma rodada preliminar de tentativas ou *trial* para então gerar o arquivo de *template* que será usado na caracterização final. Essa caracterização preliminar deve ser feita com os parâmetros **-trial** e **-auto_index**. Isto implica que a ferramenta não irá realizar a caracterização das células, mas somente irá definir os índices das **look up tables**. Para fazer uso da indexação automática é pre ciso definir as seguintes variáveis:

- *min_transition* - Tempo de transição mínima de entrada
- *min_output_cap* - Carga capacitiva mínima de saída
- *max_transition* - Tempo de transição máxima de entrada

A carga capacitiva máxima de saída será determinada através de simulações com SPICE. Na listagem 3 vemos ums exemplo de definição de modelos.

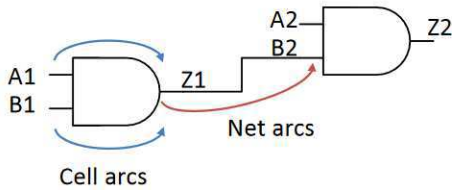
```
1  define_template -type delay \
2      -index_1 {1 2 3 4 5} \
3      -index_2 {1 2 3 4 5} \
4      delay_template_5x5
5  define_template -type power \
6      -index_1 {1 2 3 4 5} \
7      -index_2 {1 2 3 4 5} \
8      power_template_5x5
9  define_template -type constraint \
10     -index_1 {1 2 3} \
11     -index_2 {1 2 3} \
12     const_template_3x3
```

Código 3: Definição de modelos de tabela

Outro conceito que pode ser necessário na definição das células é o de arcos de temporização ou *timing arcs*. Os arcos são componentes de um caminho temporizado (*timing path*) que definem a propagação do sinal por aquela seção do caminho. Eles podem ser:

- Arcos de célula - Entre a entrada e a saída de uma célula
  - São definidos no arquivo Liberty (.lib)
- Arcos de *net* - Entre o pino de saída de uma célula e o pino de entrada de outra
  - São calculados internamente na ferramenta de *backend*.

Figura 17: Arcos de temporização



Os arcos de célula definem a interação entre dois pinos. Além disso, esses arcos podem ser de atraso ou de *contraints*. O arco de atraso, como já diz o nome, tem o intúito de calcular os parâmetros de atraso da célula. Já os arcos de *contraints* são usados no cálculo do tempo de *setup* e *hold*.

Os arcos podem ser definidos no arquivo modelo para cada uma das células com o comando **define_arc**. Porém essas definições só são necessárias se a ferramente não reconhecer o tipo de célula. Em qualquer outro momento os arcos são definidos automaticamente pela ferramenta.

O Liberate precisa de uma *netlist* de todas as células que serão caracterizadas. Essa *netlist* precisa ser definida como um sub-circuito em formato de **Spectre** ou **Spice**. Isto pode ser feito de forma similar a simulação analógica da célula. Para que essas definições sejam válidas as células devem ter passado tanto por **DRC** quanto por **LVS**. Além disso elas devem possuir a vista *analog extracted*. A vista *analog extracted* é importante para as simulações por conter informações extras dos tamanhos dos transistores, assim como todas as capacitâncias parasitas.

Primeiro abrimos uma nova vista de esquemático e instanciamos todas as células que queremos simular. Não é necessário conectar as pinos das células a nada. Podemos

ignorar as advertências de conexões abertas desta vez. Definido as células, abrimos o *ADE* e seguimos alguns passos de configuração de parâmetros:

- Definir o simulador. Este podendo ser Spectre ou HSpice.
- Definir o ambiente. Nas opções devemos adicionar a vista **analog_extracted** a frente da lista de vistas. Isto faz prioriza esta vista nos lugares onde o simulador irá procurar a *netlist*.
- Criar a *netlist*.

Como resultado do último passo, uma janela irá se abrir com um arquivo de texto contendo a definição de *netlist* de todas as células. Porém, esse arquivo contém mais coisas que o necessário e será preciso editar o arquivo tirando somente as informações necessárias de sub-circuito de cada célula.

Podemos salvar o arquivo como **.scs** ou **.sp**, dependendo do simulador usado. Do texto iremos tirar somente as definições de sub-circuitos de cada célula. Essas definições podem ser identificadas pelo termo **subckt**. Cada definição começa por uma linha **.subckt** <**cell_name**> e é terminada pela linha **.ends** <**cell_name**>. Qualquer outra informação fora desses escopos são comentários ou definições de controle.

Dentro da definição de sub-circuito podemos notar as informações de capacitância, assim como as instâncias dos transitores pertencentes a *netlist* daquela célula. Algumas ajustes ainda precisam ser feitos dentro do sub-circuito devido a limitações da ferramenta. Todas as instância de **vdd!** devem ser mudadas para **vdd** porque o Liberate não lida muito bem com caracteres que não sejam alfabéticos. O simulador define o nó de terra como **0**. A ferramenta também não gosta disso, sendo necessário mudar todas as suas aparições por **gnd**. No código 4 vemos uns exemplo de definição de sub-circuito.

```
1 .subckt INHDX0 a q VDD GND
2 c1 VDD GND 204.098e-18
3 c2 a GND 434.872e-18
4 c3 q GND 296.678e-18
5 m0 q a VDD VDD pch l=180e-9 w=720e-9 m=1 nf=1 sd=540e-9 ad=345.6e-15 as=1.1776e-12 pd=2.4e
      -6 ps=5e-6 nrd=666.667e-3 nrs=2.2716 sa=740e-9 sb=480e-9 sca=3.4875 scb=1.01194e-3 scc
      =6.13706e-6
6 m1 q a GND GND nch l=180e-9 w=420e-9 m=1 nf=1 sd=540e-9 ad=201.6e-15 as=835.6e-15 pd=1.8e
      -6 ps=3.96e-6 nrd=1.14286 nrs=4.73696 sa=740e-9 sb=480e-9 sca=4.33868 scb=1.68052e-3
      scc=10.5161e-6
7 .ends INHDX0
```

Código 4: Definição de Sub-circuito de uma inversora

2.4 LEF

O arquivo LEF (*Library Exchange Format*) faz parte do conjunto de arquivos que as ferramentas *place and route* irão utilizar na suas escolhas de decisão. O arquivo possui informações abstratas das células em um formato que a ferramenta possa interpretar. Dentro há informações básicas como o perímetro da célula, assim como o formato e posicionamento dos pinos de de entrada e saída e trilhos de alimentação, como na Figura 18. Os códigos 5 e 6 mostram um trecho de um arquivo LEF.

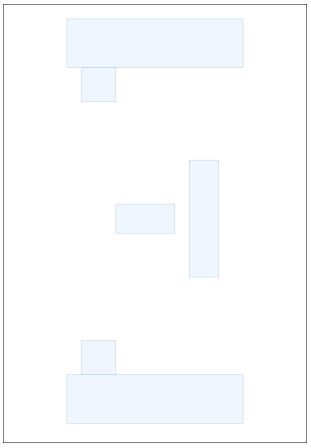
```
1  MACRO INHDX0
2    CLASS CORE ;
3    ORIGIN -2.32 28.79 ;
4    FOREIGN INHDX0 2.32 -28.79 ;
5    SIZE 0.145 BY 3.12 ;
6    SYMMETRY X Y ;
7    SITE core ;
8    PIN a
9      DIRECTION INPUT ;
10     USE SIGNAL ;
11     PORT
12       LAYER METAL1 ;
13         RECT 1.73 -27.82 2.53 -27.44 ;
14     END
15   END a
16   PIN GND
17     DIRECTION INOUT ;
18     USE GROUND ;
19     PORT
20       LAYER METAL1 ;
21         RECT 1.55 -29.99 3.23 -29.19 ;
22         RECT 1.73 -29.99 2.02 -28.61 ;
```

Código 5: Exemplo de arquivo LEF

```
22     END
23   END GND
24   PIN q
25   DIRECTION OUTPUT ;
26   USE SIGNAL ;
27   PORT
28     LAYER METAL1 ;
29       RECT 2.71 -27.21 3.09 -26.855 ;
30       RECT 2.81 -28.38 3.09 -26.855 ;
31       RECT 2.71 -28.38 3.09 -28.05 ;
32   END
33   END q
34   PIN VDD
35     DIRECTION INOUT ;
36     USE POWER ;
37     PORT
38       LAYER METAL1 ;
39         RECT 1.55 -25.51 3.23 -24.71 ;
40         RECT 1.73 -26.09 2.02 -24.71 ;
41     END
42   END VDD
43 END INHDX0
```

Código 6: Exemplo de arquivo LEF

Figura 18: Abstrato do inversor



Fonte: Autor

É possível exportar o arquivo **LEF** apenas com a vista de leiaute de cada célula. Porém, esse arquivo não teria todas as informações físicas disponíveis. Para obter uma biblioteca física completa é necessário a extração mais detalhada da célula usando a ferramenta **Abstract**. Seguindo uma sequência de passos na ferramenta iremos obter uma vista de abstrato.

A geração do abstrato começa com o carregamento da biblioteca de células em desenvolvimento na ferramenta. O primeiro passo é a localição dos pinos da células. Se os pinos foram criados normalmente durante o leiaute usando a camada de metal do tipo pino, exemplo *Metal1(pin)* em vez de *Metal1(drw)*, esse passo é simples e pode ser rodado sem muitas modificação. Porém, existe a possibilidade do projetista ter usado de rótulos sobre os metais para representar os pinos. Neste caso é necessário dizer a ferramenta que intercessão de camadas deve ser considerado um pino, como por exemplo camadas de metal que possuam camadas de texto ou rótulo dentro de suas formas. Durante a localização dos pinos também é definido o *offset* da caixa que delimita a área da célula. Este deslocamento fará com que a caixa que define a célula seja menor que a geometria real e consequemente fará com as células fiquem levemente sobreposta durante sua implantação no *Place and Route*.

O próximo passo é rodar a extração das conexões e obstruções internas da célula. Em seguida podemos realmente gerar a vista de *abstract*. Antes dessa etapa devemos verificar nas configurações que todas as camadas estão definidas como **bloqueio detalhado**(*detailed blockage*). Se todas estas etapas foram feitas corretamente iremos obter a vista de abstrato com informações de bloqueios das células (onde a ferramenta de *backend* não deve rotear). Por fim, completameos o processo exportando o arquivo **LEF** a partir dos abstratos criados.

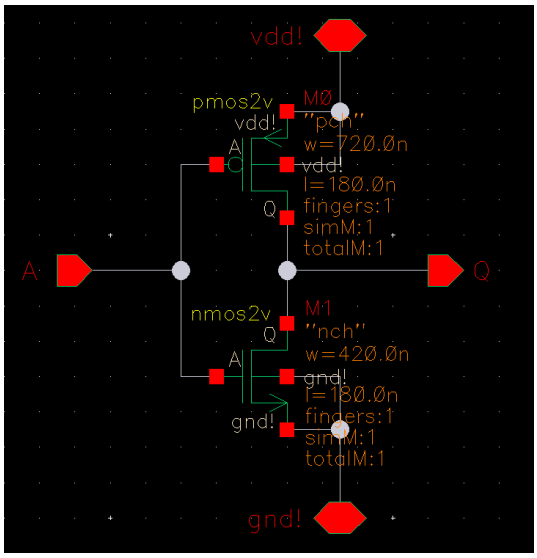
3 Resultados

A verificação da teoria demonstrada nesse trabalho foi aplicada utilizando o PDK da TSMC de 180 nanômetros por ser o disponível no laboratório e este não possuir uma biblioteca de células digitais. Primeiramente foram descidos um conjunto de células a serem desenvolvida para dar início a biblioteca. Foram elas a célula inversora, a NAND, o buffer e o flip-flop tipo D.

3.1 Esquemático

A ferramenta utilizada na etapa de construção do esquemático e também na etapa de leiaute foi o Virtuoso da Cadence. Primeiramente foram construídos os esquemáticos em tecnologia CMOS de cada célula. Foram utilizadas topografias comuns já conhecidas de cada célula. Foi escolhido o **pmos2v** e **nmos2v** como modelo fundamental de transistor a ser instanciado. Estes modelos podem trabalhar em até 2V. Os pinos da células foram adicionados e as devidas conexões de fios foram feitas. É válido notar que os pinos foram definidos de acordo com as suas funções de entrada, saída e alimentação. Os pinos de alimentação foram nomeados como **vdd!** e **gnd!**. A exclamação indica que esta é uma *net* global o que implica que ela está implicitamente conectada a todas as outras instâncias de **vdd!**, fazendo com que não seja necessário conectar todos os pinos de alimentação de todas as células aos V_{DD} quando for simulado. Nas figuras 19 à 22 vemos os esquemáticos finalizados.

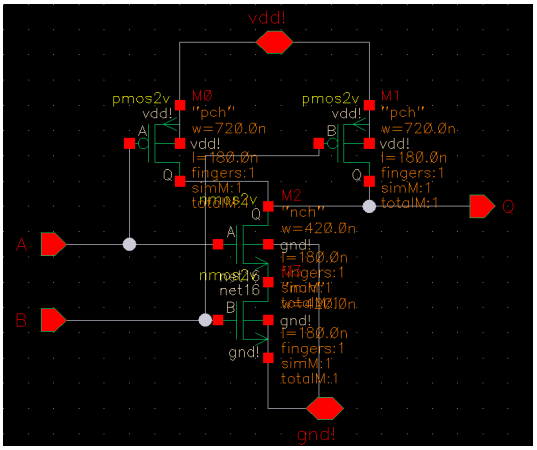
Figura 19: Esquemático do Inversor



Fonte: Autor

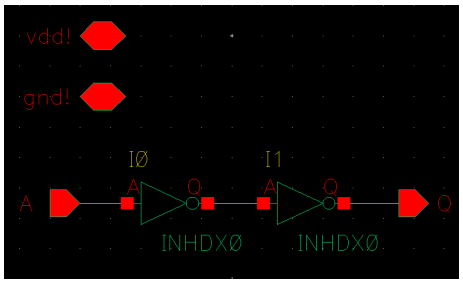
Na Figura 21 podemos ver que o esquemático da célula de buffer foi montado conectando duas instâncias da célula inversora, representadas pelas suas vistas simbólicas. Os pinos de **vdd!** e **gnd!** estão implicitamente alimentando as duas células, sem precisar serem conectadas diretamente.

Figura 20: Esquemático da NAND



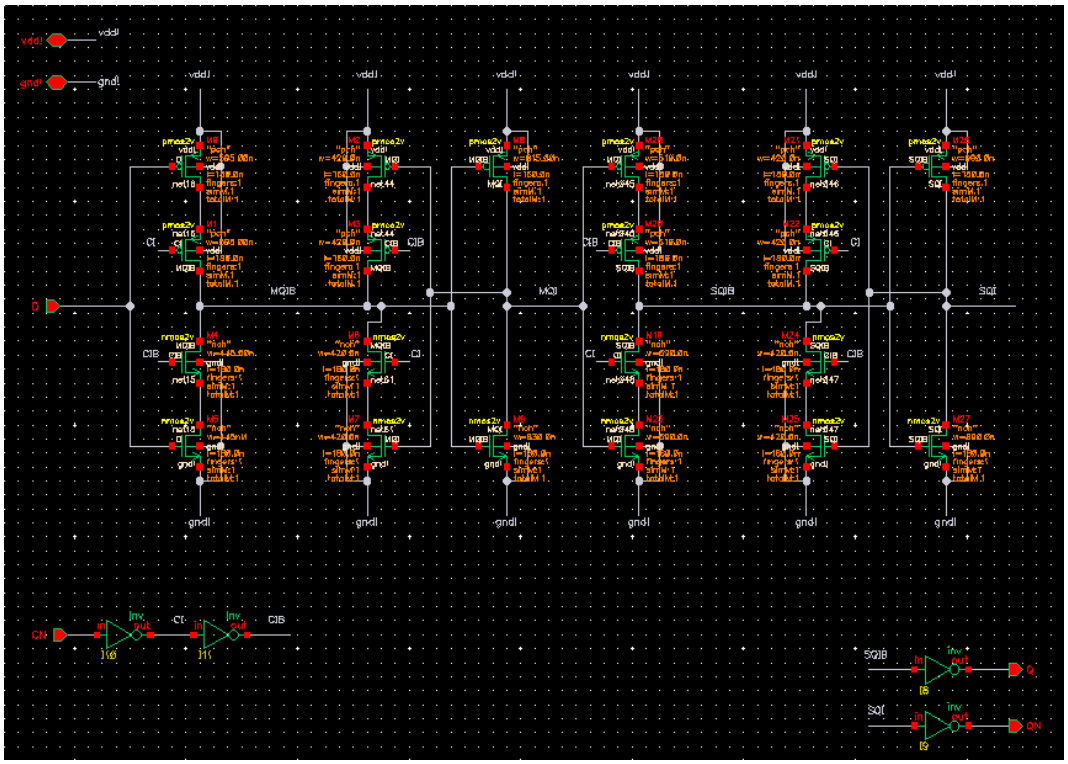
Fonte: Autor

Figura 21: Esquemático do Buffer



Fonte: Autor

Figura 22: Esquemático do Flip-flop D



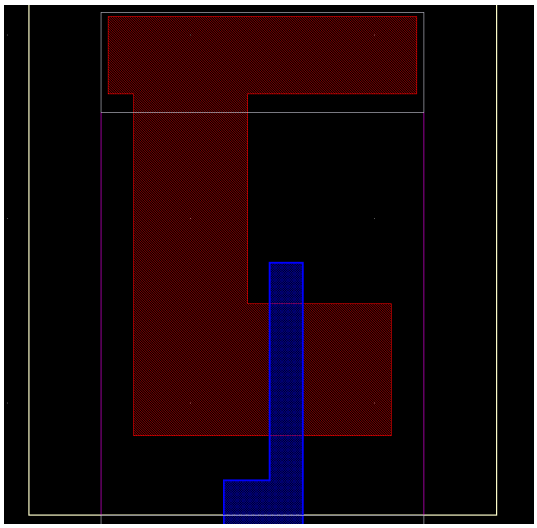
Fonte: Autor

3.2 Leiaute

Após salvar a vista de esquemático e criar o símbolo da célula o próximo passo foi a criação do leiaute. O desenho do leiaute foi feito com base em formas padrões comumente conhecidas. Sobre essa base houveram algumas tentativas de otimizar a célula com relação à área, seguindo os limites impostos pelas regras de DRC do PDK da TSMC.

No PDK da TSMC para definir as difusões tipo n_+ e p_+ usa-se a mesma camada **DIFF**. O que define se essa difusão será de um tipo ou de outro são as camadas **PIMP** e **NIMP**. Na figura 23 vemos a camada **DIFF** em vermelho. O retângulo em magenta define a área do **PIMP** e o retãngulo branco a área do **NIMP**.

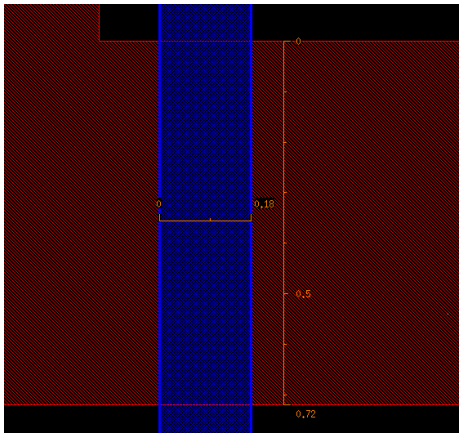
Figura 23: Caption



Fonte: Autor

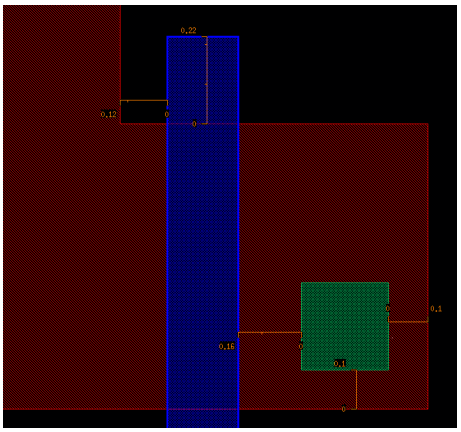
Na figura 24 vemos que a camada de polisilício em azul. Ela obrigatoriamente deve ter a largura de 180nm definido pela tecnologia. A largura da difusão é de 720nm como foi definido no esquemático. Algumas regras de DRC do PDK podem ser notadas na figura 25. Por exemplo, a camada de polisilício deve extrapolar a camada de difusão por pelo menos 22nm e essa parte deve estar a também à no mínimo 1nm de qualquer outra camada de difusão. Os contatos sobre a difusão devem estar à 1nm da borda da camada e à uma distância mínima de 16nm das camadas de polisilício.

Figura 24: Medidas do transistor PMOS



Fonte: Autor

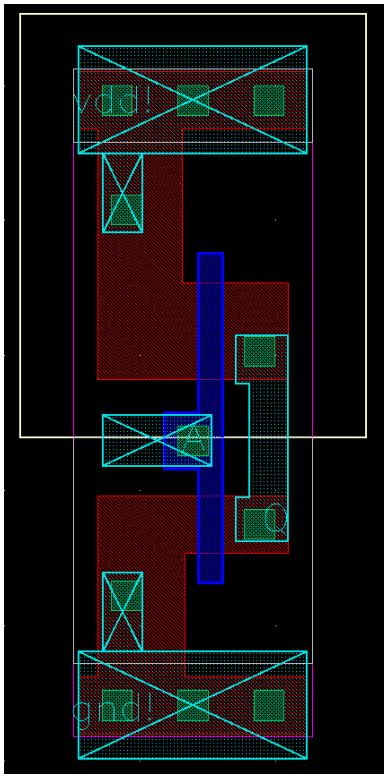
Figura 25: Exemplos de regra de DRC



Fonte: Autor

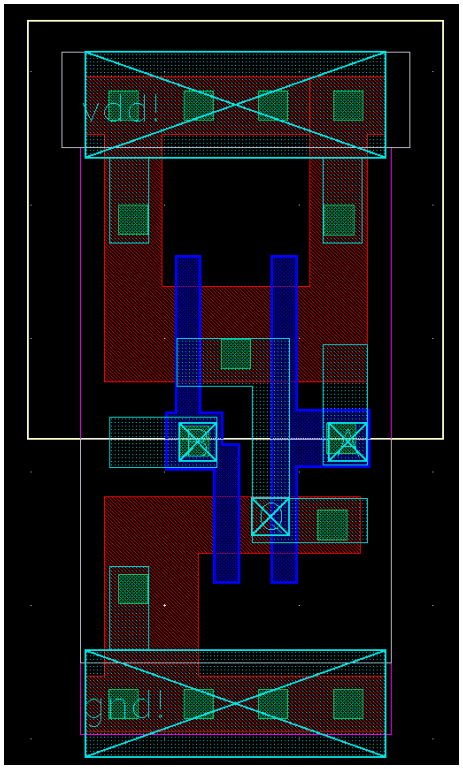
Nas figuras 28 à 29 vemos o leiaute completo das quatro células.

Figura 26: Leiaute da Inversora



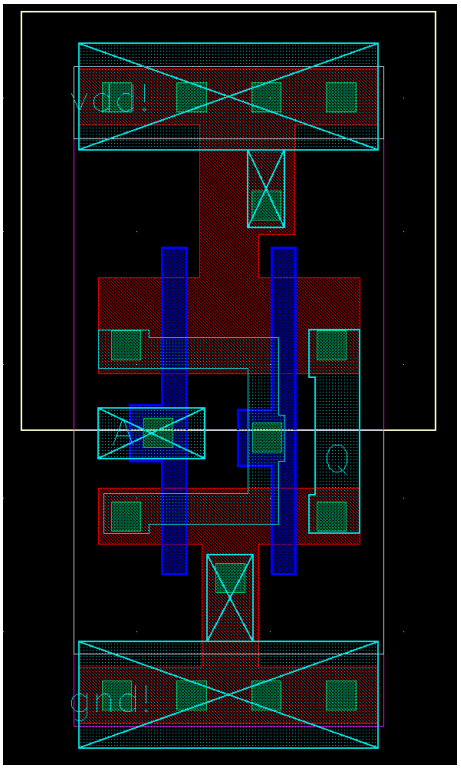
Fonte: Autor

Figura 27: Leiaute da NAND



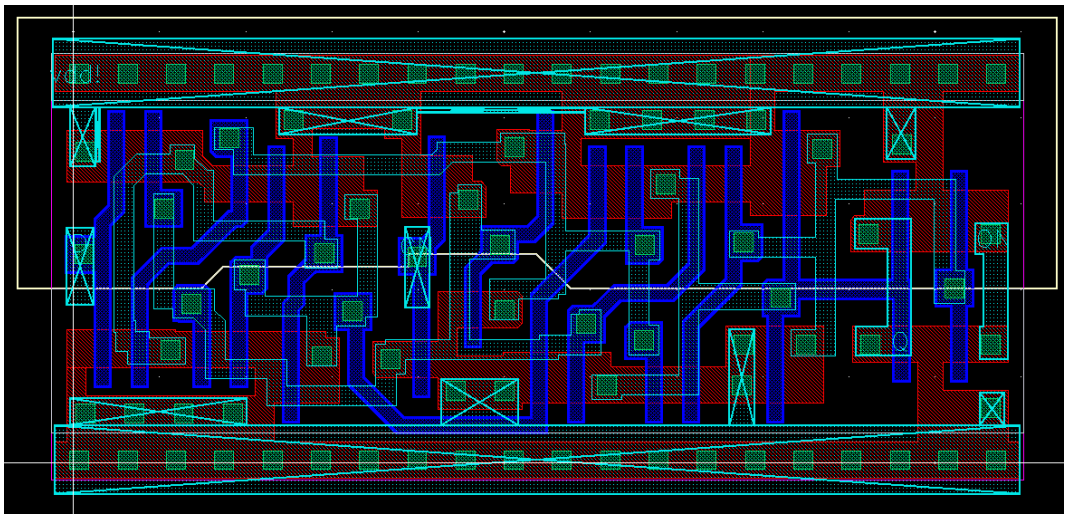
Fonte: Autor

Figura 28: Leiaute do Buffer



Fonte: Autor

Figura 29: Leiaute do Flip-flop D



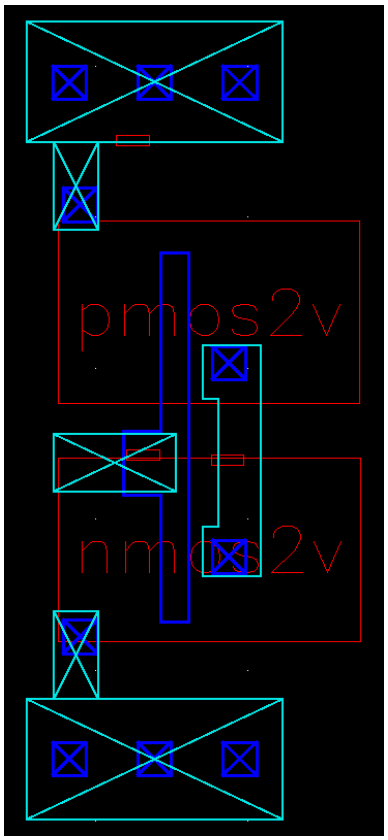
Fonte: Autor

Após o término do leiaute foi rodado a o teste de DRC para verifcar qualquer regra que tenha sido deixada passar. Depois de corrigir as violações e rodar novamente o DRC até que ele saísse limpo, pôde-se então rodar o LVS.

A rodada de LVS irá comparar a *netlist* de transistores definida pelo esquemático e a *netlist* definida pelo leiaute. Esse processo irá checar transistor por transistor nos mínimos detalhes. Ou seja, a geometria do leiaute deve estar definindo o mesmo transistor. Se o LVS falhar será preciso verificar que transistores que não estão correspondendo e consertá-los. O conserto pode ser feito no leiaute ou até mesmo no esquemático se o desenho correto do leiaute não consiga ser alcançado. De uma forma ou de outra deve-se rodar novamente o DRC e LVS até que ambos passem com sucesso.

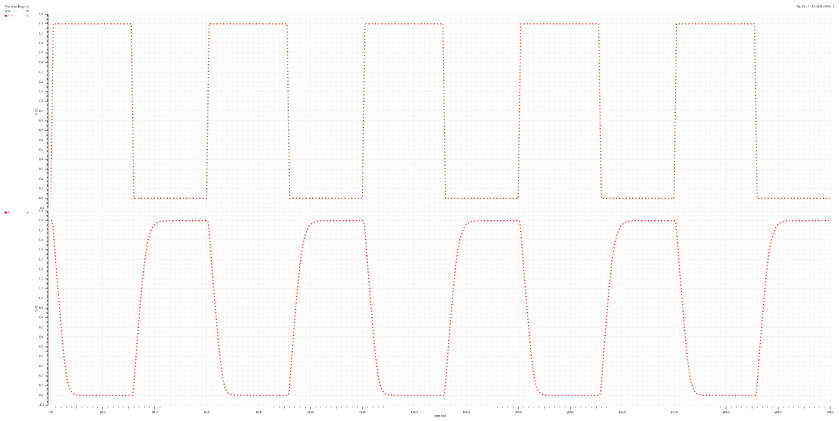
Após o LVS e DRC retornarem com 0 violações, foi possível extrair a vista de ***analog_ extracted***. Apesar dos teste de DRC e LVS terem passado, isto não garante que a célula esteja correta. O próximo passo é simular as células a partir das vistas ***analog_ extracted*** que contém informações de capacitâncias parasitas como visto na figura 30. Nas figuras 31 à 34 vemos as formas de onda da simulação analógica.

Figura 30: Vista ***analog_ extracted***



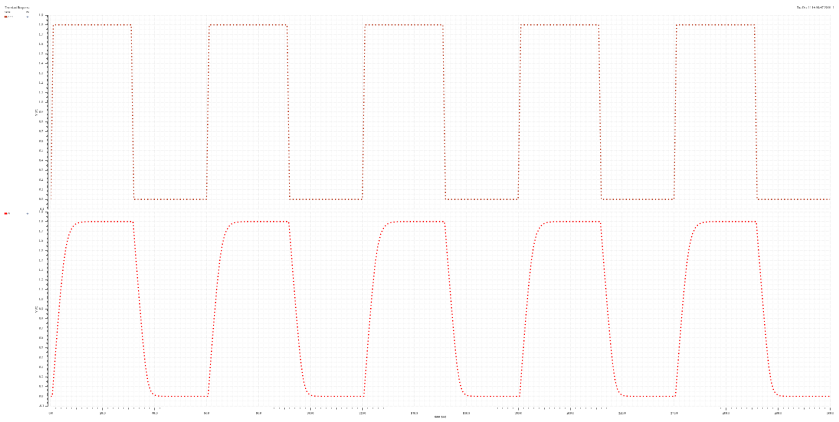
Fonte: Autor

Figura 31: Simulação analógica de uma inversora



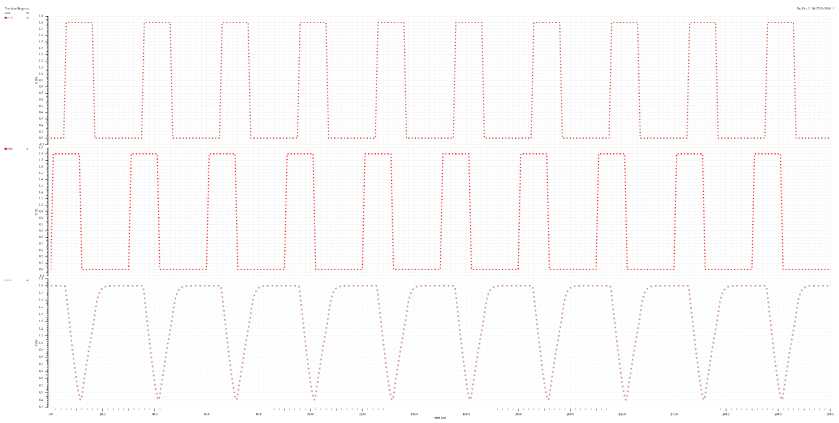
Fonte: Autor

Figura 32: Simulação analógica de um buffer



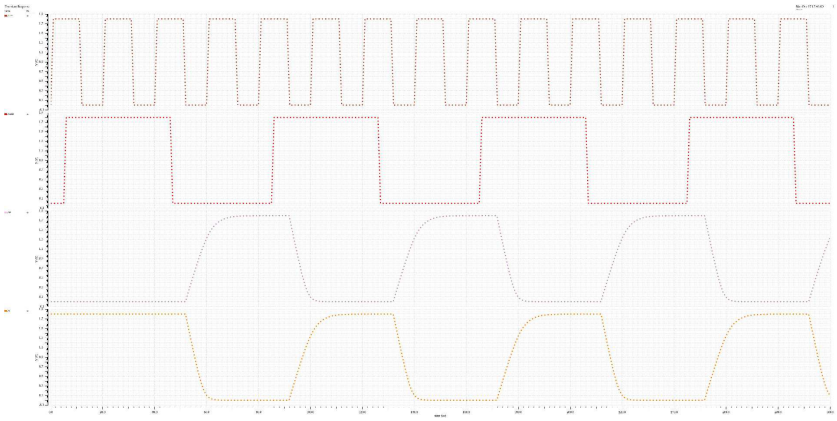
Fonte: Autor

Figura 33: Simulação analógica de uma NAND



Fonte: Autor

Figura 34: Simulação analógica de um flip-flop D



Fonte: Autor

3.3 Caracterização

O passo seguinte foi a etapa de caracterização das células. Primeira inicou-se um esquemático onde todas as células foram instanciadas. Na ferramenta de **ADE** foi definido que simulador analógico seria o HSPICE. Após configurar o ambiente e adicionar a **analog_extracted view** como prioridade na busca pela *netlist* foi iniciada a simulação para gerar as definições de sub-circuito de cada célula em formato HSpice. Cada definição de sub-circuito foi copiada para um arquivo **.sp** com o nome da célula.

A ferramenta de caracterização utilizada foi o Liberate da Cadence. Para agilizar o processo de elaboração dos *scripts* de caracterização foram usados como referência os *scripts* do *Rapid Adoption Kit*(RAK) do Liberate para geração de arquivos modelo.

No arquivo de template (código 10) foram definidos as ***thresholds*** para medição do ***slew*** e atrasos em 20% e 80%. Foram também definidos os pontos de cálculo de atraso como 50% da rampa de entrada e saída. Foram definidos os modelos de tabela 5x5 para a geração das ***look up tables*** de atraso, e consumo. A tabela de constraints, específica para células senquenciais, foram definidas como 3x3. Os valores dos índices definidos aqui serão substituídos pelos valores reais após a primeira rodada *trial* de caracterização.

Para que o Liberate possa calcular os valores das índices na rodada preliminar, as variáveis mostrada no trecho de código 7 precisam ser definidas.

```
1 set_var max_transition 4.1e-9
2 set_var min_transition 6e-12
3 set_var min_output_cap 1e-14
```

Código 7: Variáveis usadas para rodada de **-auto_index**

Após a definição das células no *template* foi realizado a primeira rodada de caracterização. Essa rodada foi feita com o comando **char_library -auto_index -trial**, seguida do comando **write_template** para escrever o arquivo modelo definitivo a partir do modelo escrito atual. O arquivo *template* gerado agora possui todos os valores de índices das tabelas definidos. Os dois templates podem ser vistos nos códigos 10 e 11 em anexo.

Agora com o arquivo modelo definitivo em mãos podemos dispensar os parâmetros de **-auto_index** e **-trial**. A segunda rodada irá realmente simular o comportamento das células para os diferentes valores de entrada de saída definidos nas ***look up tables***. A tabela de caracterização será escrita no arquivo Liberty como visto no trecho de código 8.

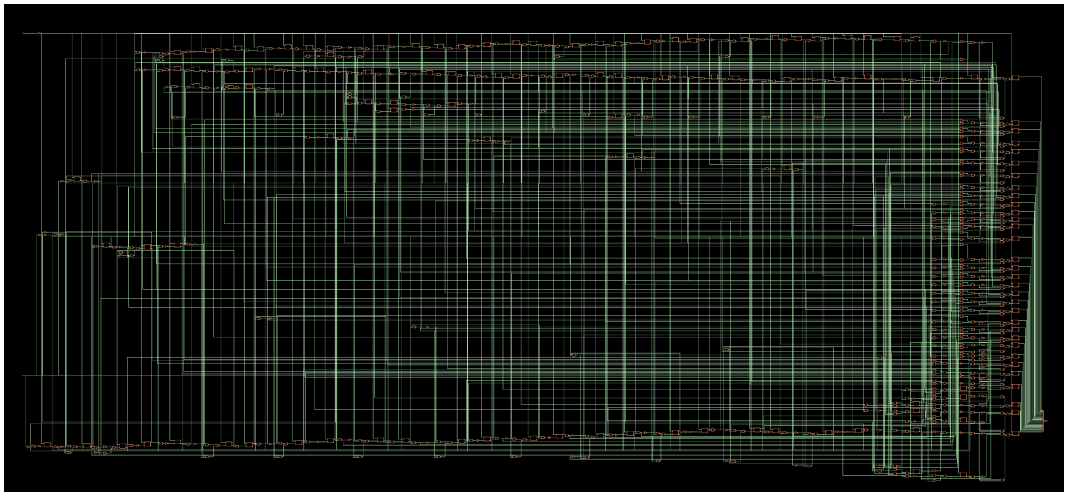
```
1         rise_transition (delay_template_5x5) {
2           index_1 ("0.006, 0.0306768, 0.156844, 0.80191, 4.1");
3           index_2 ("0.01, 0.0271101, 0.0734956, 0.199247, 0.54016");
4           values ( \
5             "0.090131, 0.217995, 0.564812, 1.50478, 4.05233", \
6             "0.0901356, 0.217995, 0.564744, 1.50478, 4.05231", \
7             "0.0906698, 0.218094, 0.564656, 1.50477, 4.05234", \
8             "0.0965337, 0.220413, 0.565586, 1.50518, 4.05242", \
9             "0.133257, 0.248304, 0.592785, 1.5235, 4.05457" \
10          );
11        }
```

Código 8: Tabela de transição de subida

3.4 Síntese Lógica

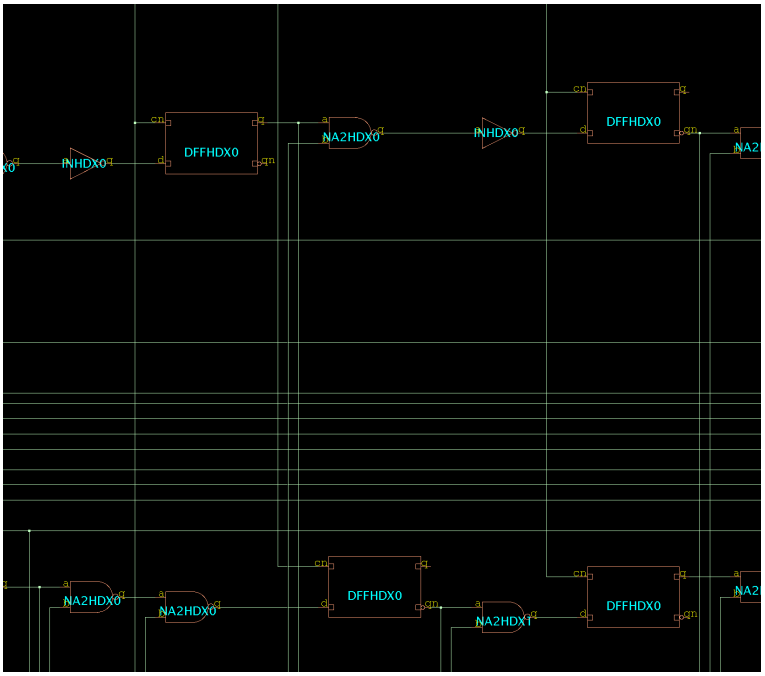
Para validação da funcionalidade da biblioteca gerada usou-se um módulo de exemplo e aplicou-se uma síntese lógica usando a ferramenta Genus da Cadence. O script de síntese pode ser analizado no código 12 em anexo. O *script* de síntese realiza um mapeamento preliminar do módulo com células genéricas. Em seguida ele realiza o mapeamento usando as células da biblioteca desenvolvida e por último é feito uma tentativa de otimização. Nas Figuras 35 e ressintzoom vemos o esquemático de células resultante da síntese. Os relatórios de QoR podem ser vistos nas Figuras 37 à 39.

Figura 35: Esquemático da Síntese



Fonte: Autor

Figura 36



Fonte: Autor

Figura 37: QoR genérico

```
Timing
--------

 Clock Period
--------------
sysclk 2000.0


  Cost    Critical         Violating
 Group   Path Slack  TNS     Paths
-------------------------------------
C2C          1409.8   0.0          0
C2O          1980.0   0.0          0
default    No paths   0.0
I2C          1468.6   0.0          0
I2O        No paths   0.0
sysclk     No paths   0.0
-------------------------------------
Total                 0.0          0

Instance Count
--------------
Leaf Instance Count             539
Physical Instance count           0
Sequential Instance Count       128
Combinational Instance Count    411
Hierarchical Instance Count       0

Area
----
Cell Area                          2253.379
Physical Cell Area                 0.000
Total Cell Area (Cell+Physical)    2253.379
Net Area                           159.950
Total Area (Cell+Physical+Net)     2413.329
```

Fonte: Autor

Figura 38: QoR mapeado

```
Timing
--------

 Clock Period
--------------
sysclk 2000.0


  Cost    Critical         Violating
 Group   Path Slack  TNS     Paths
-------------------------------------
C2C           827.6   0.0          0
C2O          1666.9   0.0          0
default    No paths   0.0
I2C           852.2   0.0          0
I2O        No paths   0.0
sysclk     No paths   0.0
-------------------------------------
Total                 0.0          0

Instance Count
--------------
Leaf Instance Count             829
Physical Instance count           0
Sequential Instance Count       128
Combinational Instance Count    701
Hierarchical Instance Count       0

Area
----
Cell Area                          5046.066
Physical Cell Area                 0.000
Total Cell Area (Cell+Physical)    5046.066
Net Area                           5344.364
Total Area (Cell+Physical+Net)     10390.430
```

Fonte: Autor

Figura 39: QoR otimizado

```
Timing
--------

 Clock Period
--------------
sysclk 2000.0


  Cost    Critical         Violating
 Group   Path Slack  TNS     Paths
-------------------------------------
C2C           793.3   0.0          0
C2O          1666.9   0.0          0
default    No paths   0.0
I2C            55.1   0.0          0
I2O        No paths   0.0
sysclk     No paths   0.0
-------------------------------------
Total                 0.0          0

Instance Count
--------------
Leaf Instance Count             744
Physical Instance count           0
Sequential Instance Count       128
Combinational Instance Count    616
Hierarchical Instance Count       0

Area
----
Cell Area                          4931.426
Physical Cell Area                 0.000
Total Cell Area (Cell+Physical)    4931.426
Net Area                           4947.820
Total Area (Cell+Physical+Net)     9879.246
```

Fonte: Autor

4 Conclusão e Trabalhos Futuros

Análises de perfomance, consumo e área no desenvolvimento de IPs são tarefas muito complexas, visto que antes da fabricação do chip só é possível usar de suposições e simulações que modelam o comportamento real do componentes físicos. Com isso torna-se necessário uma grande precisão nessas simulações para que os resultados tornem-se os mais próximos possível da realidade, quando o IP é fabricado. A biblioteca de *standard cells* tem um passo importante nessa análise e uma caracterização bem sucedida fará com o que as as expectativas para o projeto após sua fabricação sejam alcançadas.

Seguindo os passos descritos nesse trabaho foi desenvolvido uma biblioteca com quatro tipo de células. Os resultados da síntese lógica mostram que com essas poucas células foi possível sintetisar um módulo em *verilog* e relatórios de perfomance, consumo e área puderam ser gerados. A pouca variedade de células e o fato delas possuírem *fan-out* relativamente baixo resultou no aumento de células usadas e consequentemente na área utilizada.

Uma sugestão para trabalhos futuros é a alocação de esforço para crescimento da biblioteca no quesito número de células e a sua variedade. Uma análise mais profunda sobre parâmetros das ferramentas é necessário para os resultados e poder guiar a ferramenta para uma caracterização mais precisa. Também é necessário entender as diferenças na caracterização de células combinacionais e células sequenciais. Por último, é sugerido comparar a biblioteca com outra que tenha sido gerada por ferramentas automáticas de geração de leiaute.

Referências

BRUNVAND, Erik. **Digital VLSI chip design with cadence and synopsys CAD tools.** Pearson, 2006.

WESTE, Neil H. E. and HARRIS, David Money. **CMOS VLSI design: a circuits and system perspective.** Pearson, 2015.

X, Llopart. **Characterization of digital cells.** 2015. Disponível em: http://www.in2p3.fr/actions/formation/microelectronique15/LibCharctSeminar_short.pdf

WICHERN, Don. **Characterizing cells and writing a technology library file.** 2015. Disponível em: http://ece451web.groups.et.byu.net/cadence-help/src/tlf.pdf

VANGALA, Manoj. **FinFET cell library design and characterization.** 2017. Disponível em: https://repository.asu.edu/attachments/191168/content/Vangala_asu_0010N_17268.pdf

Liberate - Template file generation. 2017.

ANEXO

```
##### char.tcl
######-----------------------------------------
set SRC_DIR               [pwd]    ;# directory where all source data (netlist, models, etc ...) are stored
set RUN_DIR               [pwd]    ;# directory where all generated data (ldb, liberty, etc ...) are saved
set PROCESS               tt    ;# [ff|tt|ss]
set VDD_VALUE             1.80
set TEMP                  25
set LIBNAME               tsmc_${PROCESS}_${VDD_VALUE}_${TEMP}
#set LIBNAME               example
set SETTINGS_FILE         ${SRC_DIR}/tcl/settings.tcl
set TEMPLATE_FILE         ${SRC_DIR}/template/gen_template.tcl
set CELLS_FILE            ${RUN_DIR}/cells.tcl
set MODEL_INCLUDE_FILE    ${SRC_DIR}/models/spectre/include_${PROCESS}
set NETLIST_DIR           ${SRC_DIR}/netlists
set USERDATA              ${SRC_DIR}/userdata/userdata.lib

######-----------------------------------------
##### Set operating condition
######-----------------------------------------
puts "INFO: Set Operating Condition"
if {$TEMP == "m40"} {
    set_operating_condition -voltage ${VDD_VALUE} -temp -40
  } else {
    set_operating_condition -voltage ${VDD_VALUE} -temp ${TEMP}
}
######-----------------------------------------
##### Read template
######-----------------------------------------
puts "INFO: Read template file ${TEMPLATE_FILE}"
source ${TEMPLATE_FILE}
######-----------------------------------------
##### define device models
######-----------------------------------------
puts "INFO: Define device models (spectre, define_leafcell)."
set_var extsim_model_include ${MODEL_INCLUDE_FILE}
define_leafcell -type nmos -pin_position {0 1 2 3} { nmos_rf }
define_leafcell -type pmos -pin_position {0 1 2 3} { pmos_rf }
#define_leafcell -type diode -pin_position {0 1} {  }

## read netlist
set spicefiles ${MODEL_INCLUDE_FILE}
foreach cell $cells {
  lappend spicefiles ${NETLIST_DIR}/${cell}.sp
}
read_spice $spicefiles

######-----------------------------------------
##### Run characterization
######-----------------------------------------
##  Simple command
puts "INFO: Run Characterization"
# char_library -extsim spectre -cells $cells -auto_index -trial
char_library -extsim spectre -cells $cells -ecsm -ccs

######-----------------------------------------
##### Write output
######-----------------------------------------
### Write ldb ###
puts "INFO: Write ldb"
```

```
# if {![file exists ${RUN_DIR}/ldb]} { file mkdir ${RUN_DIR}/ldb }
# In packet_arc mode, by default, existing ldb does not get overwritten. User should use -overwrite option
write_ldb -overwrite ${RUN_DIR}/ldb/${LIBNAME}.ldb

### Write Liberty ###
puts "INFO: Write Liberty"
if {![file exists ${RUN_DIR}/lib]} { file mkdir ${RUN_DIR}/lib }
write_library -overwrite -ccs -filename ${RUN_DIR}/lib/${LIBNAME}_ccs.lib ${LIBNAME}
write_library -overwrite -ecsm -filename ${RUN_DIR}/lib/${LIBNAME}_ecsm.lib ${LIBNAME}
write_library -overwrite -filename ${RUN_DIR}/lib/${LIBNAME}_nldm.lib ${LIBNAME}
```

Código 9: Script de Caracterização

```
##### template.tcl
######-------------------------------------
set_vdd -type primary VDD $VDD_VALUE
set_gnd -type primary VSS 0
set_gnd -no_model     GND 0

set_var slew_lower_rise 0.2
set_var slew_lower_fall 0.2
set_var slew_upper_rise 0.8
set_var slew_upper_fall 0.8

set_var measure_slew_lower_rise 0.2
set_var measure_slew_lower_fall 0.2
set_var measure_slew_upper_rise 0.8
set_var measure_slew_upper_fall 0.8

set_var delay_inp_rise 0.5
set_var delay_inp_fall 0.5
set_var delay_out_rise 0.5
set_var delay_out_fall 0.5

#set_var def_arc_msg_level 0
set_var process_match_pins_to_ports 0  ;# disable check for exact match of subckt ports and define_cell command
set_var max_transition 4.1e-9
set_var min_transition 6e-12
set_var min_output_cap 1e-14  ;# 0.1fF

set cells {
    NA2HDX0
    BUHDX0
    INHDX0
    DFFHDX0
}

define_template -type delay \
    -index_1 {1 2 3 4 5} \
    -index_2 {1 2 3 4 5} \
    delay_template_5x5
define_template -type power \
    -index_1 {1 2 3 4 5} \
    -index_2 {1 2 3 4 5} \
    power_template_5x5
define_template -type constraint \
    -index_1 {1 2 3} \
    -index_2 {1 2 3} \
    const_template_3x3

set cell NA2HDX0
```

```
49 if {[ALAPI_active_cell $cell]} {
50     define_cell \
51        -input { A B } \
52        -output { Q } \
53        -pinlist { A B Q } \
54        -delay delay_template_5x5 \
55        -power power_template_5x5 \
56        $cell
57 }
58
59 set cell BUHDX0
60 if {[ALAPI_active_cell $cell]} {
61 define_cell \
62        -input { A } \
63        -output { Q } \
64        -pinlist { A Q } \
65        -delay delay_template_5x5 \
66        -power power_template_5x5 \
67        $cell
68 }
69
70 set cell INHDX0
71 if {[ALAPI_active_cell $cell]} {
72     define_cell \
73        -input { A } \
74        -output { Q } \
75        -pinlist { A Q } \
76        -delay delay_template_5x5 \
77        -power power_template_5x5 \
78        $cell
79 }
80
81 set cell DFFHDX0
82 if {[ALAPI_active_cell $cell]} {
83     define_cell \
84        -clock { CN } \
85        -input { D } \
86        -output { Q QN } \
87        -pinlist { CN D Q QN } \
88        -delay delay_template_5x5 \
89        -power power_template_5x5 \
90        -constraint const_template_3x3 \
91        $cell
92 }
```

Código 10: Script de *template* antes da geração automática de índices

```
1 ##### gen_template.tcl
2 ######-----------------------------------
3 set_var slew_lower_rise 0.2
4 set_var slew_lower_fall 0.2
5 set_var slew_upper_rise 0.8
6 set_var slew_upper_fall 0.8
7
8 set_var measure_slew_lower_rise 0.2
9 set_var measure_slew_lower_fall 0.2
10 set_var measure_slew_upper_rise 0.8
11 set_var measure_slew_upper_fall 0.8
12
13 set_var delay_inp_rise 0.5
14 set_var delay_inp_fall 0.5
15 set_var delay_out_rise 0.5
16 set_var delay_out_fall 0.5
```

```
17
18 set_var def_arc_msg_level 0
19 set_var process_match_pins_to_ports 1
20 set_var max_transition 4.1e-09
21 set_var min_transition 6e-12
22 set_var min_output_cap 1e-14
23
24 set cells { \
25   BUHDX0 \
26   DFFHDX0 \
27   INHDX0 \
28   NA2HDX0
29 }
30
31 define_template -type delay \
32          -index_1 {0.006 0.0306768 0.156844 0.80191 4.1 } \
33          -index_2 {0.01 0.0271101 0.0734956 0.199247 0.54016 } \
34          delay_template_5x5
35
36 define_template -type power \
37          -index_1 {0.006 0.0306768 0.156844 0.80191 4.1 } \
38          -index_2 {0.01 0.0271101 0.0734956 0.199247 0.54016 } \
39          power_template_5x5
40
41 if {[ALAPI_active_cell "BUHDX0"]} {
42 define_cell \
43        -input { a } \
44        -output { q } \
45        -pinlist { a q } \
46        -delay delay_template_5x5 \
47        -power power_template_5x5 \
48        BUHDX0
49
50 }
51
52 define_template -type delay \
53          -index_1 {0.006 0.0306768 0.156844 0.80191 4.1 } \
54          -index_2 {0.01 0.0272042 0.074007 0.20133 0.547704 } \
55          delay_template_5x5_2
56
57 define_template -type constraint \
58          -index_1 {0.006 0.156844 4.1 } \
59          -index_2 {0.006 0.156844 4.1 } \
60          constraint_template_3x3
61
62 define_template -type power \
63          -index_1 {0.006 0.0306768 0.156844 0.80191 4.1 } \
64          -index_2 {0.01 0.0272042 0.074007 0.20133 0.547704 } \
65          power_template_5x5_2
66
67 if {[ALAPI_active_cell "DFFHDX0"]} {
68 define_cell \
69        -clock { cn } \
70        -input { d } \
71        -output { q qn } \
72        -pinlist { d cn q qn } \
73        -delay delay_template_5x5_2 \
74        -power power_template_5x5_2 \
75        -constraint constraint_template_3x3 \
76        DFFHDX0
77
78 }
79 define_template -type delay \
80          -index_1 {0.006 0.0306768 0.156844 0.80191 4.1 } \
```

```
81          -index_2 {0.01 0.0269232 0.0724856 0.195154 0.525416 } \
82          delay_template_5x5_3
83
84 define_template -type power \
85          -index_1 {0.006 0.0306768 0.156844 0.80191 4.1 } \
86          -index_2 {0.01 0.0269232 0.0724856 0.195154 0.525416 } \
87          power_template_5x5_3
88
89 if {[ALAPI_active_cell "INHDX0"]} {
90 define_cell \
91        -input { a } \
92        -output { q } \
93        -pinlist { a q } \
94        -delay delay_template_5x5_3 \
95        -power power_template_5x5_3 \
96        INHDX0
97
98 }
99
100 define_template -type delay \
101          -index_1 {0.006 0.0306768 0.156844 0.80191 4.1 } \
102          -index_2 {0.01 0.0268455 0.0720683 0.193471 0.519385 } \
103          delay_template_5x5_4
104
105 define_template -type power \
106          -index_1 {0.006 0.0306768 0.156844 0.80191 4.1 } \
107          -index_2 {0.01 0.0268455 0.0720683 0.193471 0.519385 } \
108          power_template_5x5_4
109
110 if {[ALAPI_active_cell "NA2HDX0"]} {
111 define_cell \
112        -input { a b } \
113        -output { q } \
114        -pinlist { a b q } \
115        -delay delay_template_5x5_4 \
116        -power power_template_5x5_4 \
117        NA2HDX0
118
119 }
```

Código 11: *Template* gerado pelo rodada de **-auto_index**

```
1  set DESIGN urngTaus
2  set GEN_EFF low
3  set MAP_OPT_EFF medium
4  set SYNDIR sintese;
5  set _OUTPUTS_PATH $SYNDIR/outputs
6  set _REPORTS_PATH $SYNDIR/reports
7  set _LOG_PATH $SYNDIR/logs
8
9  set_db / .init_hdl_search_path {sintese/rtl}
10 set rtl_list_file [open "sintese/rtl_list.txt" r]
11 set rtl_list [read $rtl_list_file]
12 close $rtl_list_file
13
14 set_db / .information_level 5
15 set_db / .hdl_error_on_latch false
16 set_db / .hdl_report_case_info true
17 set_db / .hdl_array_naming_style %s_%d
18 set_db / .hdl_create_label_for_unlabeled_generate false
19 set_db / .auto_ungroup none
20 set_db / .tns_opto true
21
```

```
22 ##################################################################
23 ## Library setup
24 ##################################################################
25 set TECHLIBBASE "/mnt/hdd1tera/Liberate/"
26
27 set TECHLIB_S_PATH "$TECHLIBBASE/lib \
28           $TECHLIBBASE/LEF"
29
30 set LEF_LIST "{tsmc_180n_techfile.lef \
31          tsmc_180n_abs_dig_cell.lef}"
32
33 set TIME_LIST "tsmc_tt_1.80_25_nldm.lib" ;# \
34          tsmc_ss_1.62_125_nldm.lib \
35          tsmc_ff_1.98_m40_nldm.lib"
36
37 set_db / .init_lib_search_path $TECHLIB_S_PATH
38 set_db / .library $TIME_LIST
39
40 ## PLE
41 set_db / .lef_library $LEF_LIST
42
43 ##################################################################
44 ## Load Design
45 ##################################################################
46 read_hdl -sv $rtl_list
47
48 set_db / .lp_insert_clock_gating true
49
50 elaborate $DESIGN
51
52 edit_netlist uniquify $DESIGN -verbose
53
54 time_info Elaboration
55
56 check_design -unresolved
57
58 ##################################################################
59 ## Constraints Setup
60 ##################################################################
61 read_sdc -stop_on_errors ./$SYNDIR/constraints.sdc
62 check_design -unresolved
63 report_clocks
64 report_clocks -generated
65
66 if {![file exists ${_LOG_PATH}]} {
67   file mkdir ${_LOG_PATH}
68   puts "Creating directory ${_LOG_PATH}"
69 }
70
71 if {![file exists ${_OUTPUTS_PATH}]} {
72   file mkdir ${_OUTPUTS_PATH}
73   puts "Creating directory ${_OUTPUTS_PATH}"
74 }
75
76 if {![file exists ${_REPORTS_PATH}]} {
77   file mkdir ${_REPORTS_PATH}
78   puts "Creating directory ${_REPORTS_PATH}"
79 }
80 report_timing -lint
81
82 ##################################################################
83 ## Clock gating Setup
84 ##################################################################
85 set_db / .library_sets.libraries.lib_cells.dont_use false
```

```
set_db / .lp_power_analysis_effort high
set_db / .lp_power_unit mW
set_db / .lp_insert_clock_gating false
#set_db / .libraries.lib_cells.clock_gating_integrated_cell.name ""
#set_db / .lp_insert_discrete_clock_gating_logic true

################################################################################
## Optimize Netlist
################################################################################
set_db / .dp_perform_sharing_operations true
set_db / .dp_area_mode true
set_db / .dp_postmap_downsize true
set_db / .optimize_merge_flops true
set_db / .lp_insert_operand_isolation false
set_db / .optimize_merge_latches false
set_db / .remove_assigns true
set_db / .use_tiehilo_for_const duplicate
set_db / .hdl_preserve_unused_registers true
set_db / .hdl_latch_keep_feedback true

################################################################################
## Define cost groups (clock-clock, clock-output, input-clock, input-output)
################################################################################

if {[llength [all::all_seqs]] > 0} {
  define_cost_group -name I2C -design $DESIGN
  define_cost_group -name C2O -design $DESIGN
  define_cost_group -name C2C -design $DESIGN
  path_group -from [all::all_seqs] -to [all::all_seqs] -group C2C -name C2C
  path_group -from [all::all_seqs] -to [all::all_outs] -group C2O -name C2O
  path_group -from [all::all_inps]  -to [all::all_seqs] -group I2C -name I2C
}

define_cost_group -name I2O -design $DESIGN
path_group -from [all::all_inps]  -to [all::all_outs] -group I2O -name I2O
foreach cg [vfind / -cost_group *] {
  report_timing -cost_group [list $cg] >> $_REPORTS_PATH/${DESIGN}_pretim.rpt
}

################################################################################
## Synthesizing to generic
################################################################################
set_db / .syn_generic_effort $GEN_EFF
syn_generic

report_dp > $_REPORTS_PATH/generic/${DESIGN}_datapath.rpt
write_snapshot -outdir $_REPORTS_PATH -tag generic
report_summary -directory $_REPORTS_PATH

################################################################################
## Synthesizing to gates
################################################################################
set_db / .syn_map_effort $MAP_OPT_EFF
syn_map

write_snapshot -outdir $_REPORTS_PATH -tag map
report_summary -directory $_REPORTS_PATH
report_dp > $_REPORTS_PATH/map/${DESIGN}_datapath.rpt

foreach cg [vfind / -cost_group *] {
  report_timing -cost_group [list $cg] > $_REPORTS_PATH/${DESIGN}_[vbasename $cg]_post_map
     .rpt
}

```

```
149 ################################################################################
150 ## Optimize Netlist
151 ################################################################################
152 set_db / .syn_opt_effort $MAP_OPT_EFF
153 syn_opt
154
155 write_snapshot -outdir $_REPORTS_PATH -tag syn_opt
156 report_summary -directory $_REPORTS_PATH
157
158 foreach cg [vfind / -cost_group *] {
159   report_timing -cost_group [list $cg] > $_REPORTS_PATH/${DESIGN}_[vbasename $cg]_post_opt
      .rpt
160 }
161
162 ################################################################################
163 ## write backend file set (verilog, SDC, config, etc.)
164 ################################################################################
165 report_dp > $_REPORTS_PATH/${DESIGN}_datapath_incr.rpt
166 report_messages > $_REPORTS_PATH/${DESIGN}_messages.rpt
167 write_snapshot -outdir $_REPORTS_PATH -tag final
168 report_summary -directory $_REPORTS_PATH
169 report_power > $_REPORTS_PATH/${DESIGN}_power_final.rpt
170
171 report_clock_gating -detail > $_REPORTS_PATH/${DESIGN}_cg_final.rpt
172 report_clock_gating -gated_ff > $_REPORTS_PATH/${DESIGN}_cg_gated_final.rpt
173 report_clock_gating -ungated_ff > $_REPORTS_PATH/${DESIGN}_cg_ungated_final.rpt
174
175 write_hdl  > ${_OUTPUTS_PATH}/${DESIGN}_map.v
176 write_sdc > ${_OUTPUTS_PATH}/${DESIGN}_map.sdc
177 write_sdf -edges check_edge > ${_OUTPUTS_PATH}/${DESIGN}_map.sdf
178
179 file copy [get_db / .stdout_log] ${_LOG_PATH}/.
```

Código 12: Script de síntese lógica