Centro de Graduação em Engenharia Elétrica

Departamento de Engenharia Elétrica

FERNANDO DA SILVA SOUSA

RECARACTERIZAÇÃO DE UMA BIBLIOTECA DE CÉLULAS

CAMPINA GRANDE – PARAÍBA

MAIO DE 2021

FERNANDO DA SILVA SOUSA

RECARACTERIZAÇÃO DE UMA BIBLIOTECA DE CÉLULAS

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

Área de Concentração: Eletrônica

Marcos Ricardo de Alcântara Morais

Orientador

Gutemberg Gonçalves dos Santos Júnior

Convidado

CAMPINA GRANDE – PARAÍBA

MAIO DE 2021

RESUMO

Uma biblioteca de células digitais contém informações básicas de *timing* e de consumo de todas as células para diferentes condições de operação. Essas informações são cruciais durante o projeto de um circuito integrado, já que estimam como essas células irão se comportar após o *tapeout*. Muitas vezes o projetista é limitado pelas condições de operação disponibilizadas pela *foundry*. Esse projeto irá descrever o processo de recaracterização de uma biblioteca de células digitais. Após isso, a biblioteca gerada será utilizada para implementar um microcontrolador para ajudar a analisar como uma ferramenta de *place & route* lida com condições de operação diferentes da nominal.

Palavras-chave: células digitais, biblioteca, recaracterização, circuito integrado, *place & route*

ABSTRACT

A standard cell library contains the basic timing and power information for all cells for different operating conditions. This information is crucial during an integrated circuit project, because it estimates how those cells will behave after tapeout. Sometimes the designer is limited by the operating conditions provided by the foundry. This project will describe the process of recharacterizing a standard cell library. After that, the generated library will be used to implement a microcontroller to help analyse how a place & route tool deals with operating conditions that differ from the nominal.

Keywords: standard cell, library, recharacterization, integrated circuit, place & route

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

No projeto de circuitos integrados, é necessário fazer estimativas de como o mesmo irá se comportar nas mais diversas condições de temperatura e de tensão, de modo a garantir o maior *yeld* possível durante a fabricação. Essas estimativas são feitas com base em dados obtidos através de simulações analógicas das células, que podem ser agrupados em bibliotecas nos mais diversos formatos, o mais comum deles sendo o *liberty*.

Esses arquivos *liberty* são o principal componente da síntese lógica de um circuito em RTL, e são também essenciais durante a implementação, já que contêm todas as informações sobre o comportamento do *timing* do circuito, como por exemplo o *delay* e o tempo de transição na saída de uma célula correspondente a cada uma de suas entradas.

Com as reduções nas dimensões dos transistores, são necessárias cada vez mais informações nas bibliotecas, e cada vez mais *corners* de funcionamento são analisados. Entretanto, muitas vezes as condições de funcionamento simuladas pela *foundry* não são suficientes, como por exemplo quando se faz necessário a utilização de células em domínios de tensão menores do que a nominal para reduzir o consumo, ou quando se deseja trabalhar em temperaturas intermediárias para reduzir o pessimismo durante o processo de síntese.

Para resolver esse problema, se faz necessária a geração de arquivos *liberty* personalizados. Isso é feito através da recaracterização da biblioteca provida, que consiste na simulação das células para novas condições e consequente geração da biblioteca no formato desejado.

1.1 Objetivos

O objetivo deste projeto é a recaracterização de uma biblioteca de células digitais de 28nm. Serão descritos todos os arquivos e passos necessários para esse processo, o que permitirá a criação de um fluxo de recaracterização para uma biblioteca qualquer.

Além disso, de modo a analisar como a ferramenta de *place & route* lida com pontos de operação diferentes dos nominais, serão feitas duas implementações de um microcontrolador, uma utilizando a biblioteca padrão e outra utilizando a recaracterizada, e serão comparados os resultados de *timing* e de *power* obtidos.

2 COMPONENTES DE UMA BIBLIOTECA DE CÉLULAS

Para que as diversas ferramentas de síntese e de *place & route* consigam simular corretamente o consumo e o *timing* de um bloco, elas precisam dispor de algumas informações básicas sobre cada célula.

Não seria viável para essas ferramentas simularem um bloco inteiro a nível de transistores, pois isso consumiria uma grande quantidade de recursos computacionais. Dessa forma, o que ocorre na prática é a simulação prévia de cada uma das células que compõe uma biblioteca, a nível de transistores, de modo a coletar informações de *timing* e de *power* referentes a cada um dos pinos que elas possuem nas mais diversas condições de funcionamento. Essas informações são então agrupadas, junto com a função lógica de cada célula, em um arquivo que pode vir em alguns formatos diferentes, o mais comum deles sendo o formato *liberty*, que usa a extensão *.lib*.

A imagem a seguir exemplifica o cabeçalho de um arquivo *.lib*.

Figura 1: Exemplo de cabeçalho de um arquivo *.lib*

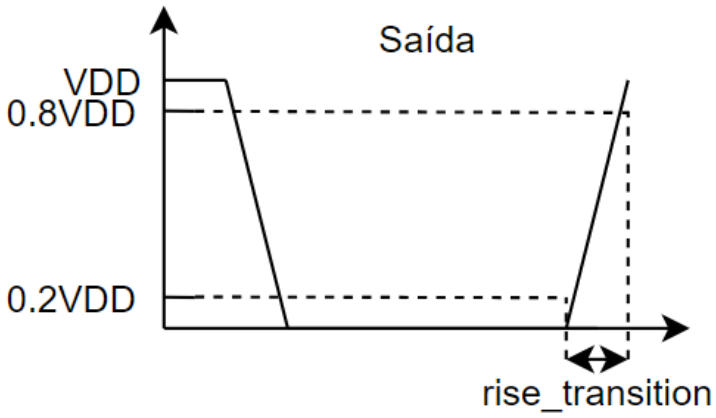
```
library(foo) {

  delay_model : table_lookup;
  in_place_swap_mode : match_footprint;

  /* unit attributes */
  time_unit : "1ns";
  voltage_unit : "1V";
  current_unit : "1uA";
  pulling_resistance_unit : "1kohm";
  leakage_power_unit : "1nW";
  capacitive_load_unit (1,pf);

  slew_upper_threshold_pct_rise : 80;
  slew_lower_threshold_pct_rise : 20;
  slew_upper_threshold_pct_fall : 80;
  slew_lower_threshold_pct_fall : 20;
  input_threshold_pct_rise : 30;
  input_threshold_pct_fall : 70;
  output_threshold_pct_rise : 70;
  output_threshold_pct_fall : 30;
  nom_process : 1;
  nom_voltage : 5;
  nom_temperature : 25;
  operating_conditions ( typical ) {
     process : 1;
     voltage : 5;
     temperature : 25;
  }
  default_operating_conditions : typical;

  lu_table_template(delay_template_5x5) {
    variable_1 : input_net_transition;
    variable_2 : total_output_net_capacitance;
    index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
    index_2 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
  }
  power_lut_template(energy_template_5x5) {
    variable_1 : input_transition_time;
    variable_2 : total_output_net_capacitance;
    index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
    index_2 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
  }
```

Fonte: BRUNVAND, 2010

Como pode ser visto na primeira linha, o *delay model* utilizado é o *lookup table*, que consiste basicamente em várias matrizes com os dados de caracterização, onde os eixos são definidos pelo *input net transition* (transição de entrada) e pelo *total output net capacitance* (capacitância total de saída). Esse tipo de modelo não é o único utilizado, mas é o mais comum. A partir dessas matrizes, as ferramentas podem fazer interpolações para obter o consumo/atraso de uma célula para condições de transição e capacitância não definidas nos índices.

As linhas a seguir definem as unidades elétricas padrão da *.lib*, de modo que somente números possam ser utilizados. Já os *thresholds* definem onde na forma de onda das entradas e saídas de cada célula são medidos os *delays* e os tempos de transição. Por exemplo, os dois primeiros atributos descrevem que, na medição da transição de subida de um sinal, o cálculo é feito utilizando os instantes nos quais o sinal está em 20% de VDD e em 80% de VDD. As duas linhas seguintes descrevem o processo equivalente para a transição de descida. A imagem a seguir ilustra esses instantes.

Figura 2: Tempo de transição de subida de uma saída

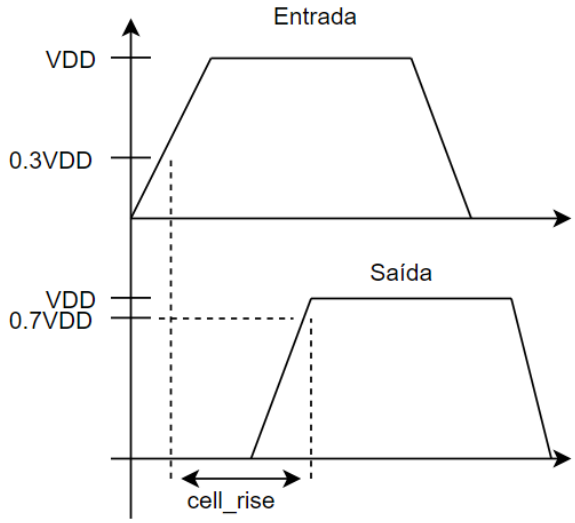


Fonte: Autor

Os próximos quatro parâmetros descrevem as informações necessárias para a medição do *delay* entre a entrada e a saída de uma célula. Os dois primeiros o indicam onde é feita a medição inicial quando a entrada transiciona de VDD para GND (*fall*) e onde é feita a medição inicial quando a entrada transiciona de GND para VDD (*rise*). Os outros dois parâmetros são utilizados da mesma forma, agora para a saída. Por exemplo, para medir o *delay* de subida da saída com relação a uma entrada que está também subindo, mede-se o tempo entre o momento

que o sinal de entrada atinge 30% de VDD e o momento que o sinal de saída atinge 70% de VDD. A imagem a seguir ilustra esse processo.

Figura 3: Delay de subida de uma saída



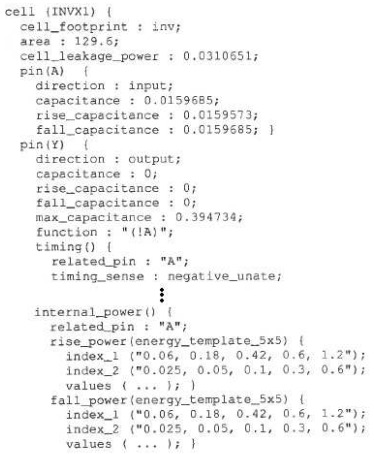
Fonte: Autor

As linhas a seguir descrevem as condições de operação da biblioteca, como temperatura e tensão. Já os *templates lu_table_template* e *power_lut_template* são utilizados como base para a definição dos índices das matrizes que computam os *delays* dos vários arcos de cada célula, assim como nas tabelas que computam os consumos. Esses índices geralmente variam de célula para célula e de arco para arco, fazendo com que boa parte do cabeçalho de um arquivo do tipo *liberty* seja composto por definições de *templates*.

2.1 Definição de Células

Para melhor descrever a definição de uma célula em um arquivo *liberty*, a figura a seguir foi tomada como exemplo.

Figura 4: Definição de célula inversora



```
cell (INVX1) {
  cell_footprint : inv;
  area : 129.6;
  cell_leakage_power : 0.0310651;
  pin(A)  {
    direction : input;
    capacitance : 0.0159685;
    rise_capacitance : 0.0159573;
    fall_capacitance : 0.0159685; }
  pin(Y)  {
    direction : output;
    capacitance : 0;
    rise_capacitance : 0;
    fall_capacitance : 0;
    max_capacitance : 0.394734;
    function : "(!A)";
    timing() {
      related_pin : "A";
      timing_sense : negative_unate;
                  ⋮
    internal_power() {
      related_pin : "A";
      rise_power(energy_template_5x5) {
        index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
        index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
        values ( ... ); }
      fall_power(energy_template_5x5) {
        index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
        index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
        values ( ... ); }
```

Fonte: BRUNVAND, 2010

Como pode ser visto na figura, as informações básicas da célula são a sua *footprint*, que é um código que agrupa um certo número de células que realizam a mesma função e podem ser trocadas livremente, sua área e seu consumo *leakage*, que é um consumo sempre presente, que independe de variações na entrada e na saída e que ocorre mesmo no estado *off*.

Para os pinos da célula são definidos a direção (entrada ou saída) e a capacitância, que vária a depender do tipo de transição naquele pino.

Especificamente para os pinos de saída, informações adicionais são necessárias, como a função descrita por aquele pino, que depende das entradas, e seções de *timing* e de consumo.

A seção de *timing* descreve os arcos existentes para a aquela saída, arcos esses que são computados com relação a cada uma das entradas da célula, já que *delays* e tempos de transição variam mesmo entre entradas do mesmo circuito.

O comportamento da saída com relação a cada entrada é também descrito, pelo atributo *timing_sense*, podendo assumir os seguintes valores:

- *negative_unate*: Indica que uma variação na entrada relacionada causa uma variação do tipo oposto na saída (subida causa descida e descida causa subida) ou nenhuma variação. Por exemplo, a relação entre a entrada e a saída de um inversor.
- *positive_unate:* Indica que uma variação na entrada relacionada causa uma variação do mesmo tipo na saída (subida causa subida e descida causa descida) ou nenhuma variação. Por exemplo, a relação entre a entrada e a saída de um *buffer*.

- *non_unate*: Indica que o tipo de variação da entrada relacionada não pode ser utilizado para determinar o tipo de variação na saída. Por exemplo, a relação entre uma das entradas de uma célula XOR e sua saída.

A seção *internal_power* descreve o consumo das transições de cada uma das saídas. Esse consumo é descrito também separadamente com relação a cada uma das entradas, e depende do sentido da transição (subida ou descida).

2.1.1 Células Combinacionais

Uma célula combinacional descreve um tipo de circuito lógico no qual a saída depende exclusivamente da combinação dos estados de suas entradas, independendo de valores de saída anteriores.

Para corretamente descrever o *timing* de uma célula deste tipo, são necessários os seguintes tipos de arcos:

- *cell_rise*: Indica o *delay* entre uma mudança na entrada e a subida do sinal de saída.
- *cell_fall*: Indica o *delay* entre uma mudança na entrada e a descida do sinal de saída.
- *rise_transition*: Indica o tempo de transição entre o início e o final da subida do sinal de saída.
- *fall_transition*: Indica o tempo de transição entre o início e o final da descida do sinal de saída.

Esses tipos de arcos são computados para cada uma das saídas, com relação a cada uma das entradas da célula. A imagem a seguir exemplifica um desses arcos.

Figura 5: Exemplo de arco de *timing*

```
cell_rise(delay_template_5x5) {
  index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
  index_2 ("0.025, 0.05, 0.1, 0.3, 0.6");
  values ( \
    "0.147955, 0.218038, 0.359898, 0.922746, 1.76604", \
    "0.224384, 0.292903, 0.430394, 0.991288, 1.83116", \
    "0.365378, 0.448722, 0.584275, 1.13597, 1.97017", \
    "0.462096, 0.551586, 0.70164, 1.24437, 2.08131", \
    "0.756459, 0.874246, 1.05713, 1.62898, 2.44989"); }
```

Fonte: BRUNVAND, 2010

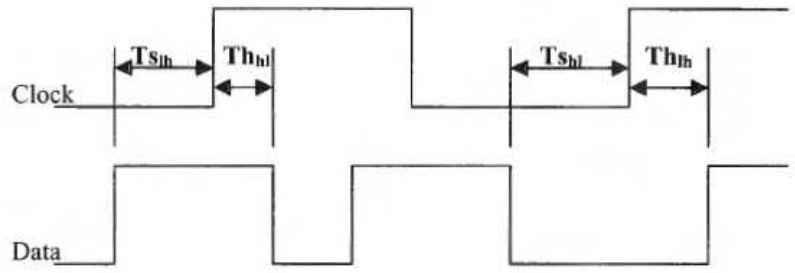
Como pode ser visto, o *template* dos índices é especificado logo após o tipo de arco (*cell_rise*). Pode-se notar também que os índices são novamente especificados. Isso pode ocorrer caso os índices utilizados não estejam presentes em nenhum dos *templates* definidos no cabeçalho ou pode ser somente uma escolha da *foundry* de explicitar melhor quais são os índices de cada arco. Por fim, a tabela *values* contém a matriz descrita anteriormente, com os possíveis valores de *delays* para aquele arco.

2.1.2 Células Sequenciais

Células sequenciais são geralmente operadas por um sinal de *clock*. Dessa forma, além dos arcos descritos anteriormente para as células combinacionais, esse tipo de célula requer informações adicionais. Essas informações são os tempos de *setup* e de *hold*.

O tempo de *setup* pode variar a depender do tipo de transição (subida ou descida) do sinal de entrada, e corresponde ao intervalo de tempo antes da borda do *clock* durante o qual o sinal de entrada deve estar estável para que possa ser capturado corretamente. O tempo de *hold* funciona de forma similar, com a única diferença sendo que o intervalo de tempo corresponde ao período adicional após a borda de *clock* durante o qual o sinal deve permanecer estável. A figura a seguir mostra duas formas de onda exemplificando essas informações.

Figura 6: *Setup* e *hold* para uma célula sequencial



Fonte: BRUNVAND, 2010

Essas informações de *setup* e de *hold* devem ser computadas para cada uma das entradas da célula, sempre com relação ao sinal de *clock*. Dessa forma, a seção de *timing* das entradas terá mais dados em uma célula sequencial do que em uma combinacional. A figura a seguir mostra a seção de *timing* da entrada de um *flip-flop*.

Figura 7: Seção de *timing* para entrada de célula sequencial

```
pin(D)    {
  direction : input;
  capacitance : 0.0225;

  timing() { /* hold time constraint for a rising transition on G */
    timing_type : hold_rising;
            rise_constraint(scalar) { values("-0.298"); }
            fall_constraint(scalar) { values("-0.298"); }
    related_pin : "G";
  }
  timing() { /* setup time constraint for a rising transition on G */
    timing_type : setup_rising;
            rise_constraint(scalar) { values("0.018"); }
            fall_constraint(scalar) { values("0.018"); }
    related_pin : "G";
  }
```

Fonte: BRUNVAND, 2010

O atributo *timing_type* é outro atributo que é exclusivo para células sequenciais, e basicamente serve para descrever o tipo de arco em questão. Os possíveis valores que ele pode assumir são:

- *hold_rising*: Designa o tempo de *hold* de uma entrada para uma célula sensível à borda de subida do *clock*.
- *setup_rising*: Designa o tempo de *setup* de uma entrada para uma célula sensível à borda de subida do *clock*.
- *hold_falling*: Designa o tempo de *hold* de uma entrada para uma célula sensível à borda de descida do *clock*.
- *setup_falling*: Designa o tempo de *setup* de uma entrada para uma célula sensível à borda de descida do *clock*.
- *preset*: Um arco desse tipo indica que o pino de saída será definido como 1 quando o pino relacionado é ativado.
- *clear*: Um arco desse tipo indica que o pino de saída será definido como 0 quando o pino relacionado é ativado.
- *rising_edge*: Identifica um arco onde o pino de saída é sensível à borda de subida do pino de entrada relacionado.
- *falling_edge*: Identifica um arco onde o pino de saída é sensível à borda de descida do pino de entrada relacionado.

No exemplo de célula escolhido, os tempos de *setup* e de *hold* são valores escalares, e não tabelas, ou seja, eles independem de outras variáveis. Entretanto, algumas bibliotecas computam esses valores em função de outras variáveis.

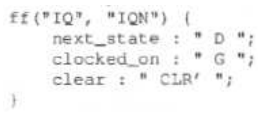
Por fim, o atributo *related_pin* nesse caso corresponde ao *clock* com relação ao qual esses tempos são medidos.

A funcionalidade de uma célula sequencial é descrita por um dos seguintes grupos:

- *ff*: Indica que a célula é um *flip-flop* (sensível à borda do sinal de controle).
- *latch*: Indica que a célula é um *latch* (sensível ao nível do sinal de controle).
- *statetable*: Descreve uma célula sequencial de lógica mais complexa, que tem o estado de sua saída definido por meio de uma *lookup table*.

A figura a seguir mostra a construção *ff* utilizada na definição de um *flip-flop* do tipo D.

Figura 8: Construção *ff* para *flip-flop* tipo D



```
ff("IQ", "IQN") {
    next_state : " D ";
    clocked_on : " G ";
    clear : " CLR' ";
}
```

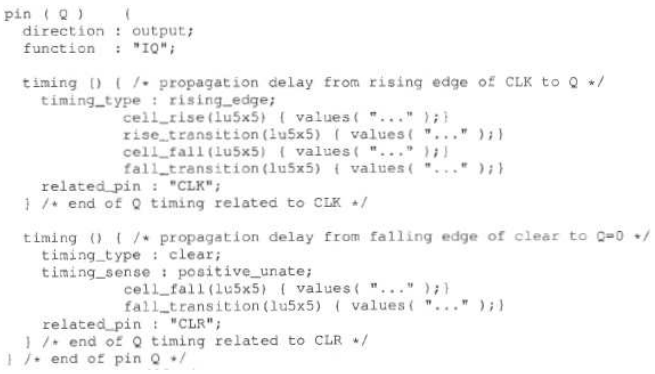
Fonte: BRUNVAND, 2010

As variáveis listadas em parêntesis são os valores internos para a saída e para a saída invertida do *flip-flop*, respectivamente. O atributo *next_stage* descreve o próximo estado do *flop* após a borda do *clock*. O atributo *clocked_on* indica qual o pino que serve como sinal de controle. Por fim, o atributo *clear* descreve o sinal que atua como *reset* nesse *flop*.

Além dos atributos descritos, o atributo *preset* pode estar presente, para indicar o sinal de *preset* da célula, bem como os atributos *clear_preset_var1* e *clear_preset_var2*, que descrevem o que ocorre com a saída e com a saída invertida, respectivamente, quando os sinais de *clear* e de *preset* são ativados simultaneamente.

A figura a seguir mostra a definição do pino de saída de um *flip-flop* do tipo D, que possui uma entrada do tipo *clear*.

Figura 9: Definição do pino de saída de um *flip-flop*



```
pin ( Q )    {
  direction : output;
  function  : "IQ";

  timing () { /* propagation delay from rising edge of CLK to Q */
    timing_type : rising_edge;
            cell_rise(lu5x5) { values( "..." );}
            rise_transition(lu5x5) { values( "..." );}
            cell_fall(lu5x5) { values( "..." );}
            fall_transition(lu5x5) { values( "..." );}
    related_pin : "CLK";
  } /* end of Q timing related to CLK */

  timing () { /* propagation delay from falling edge of clear to Q=0 */
    timing_type : clear;
    timing_sense : positive_unate;
            cell_fall(lu5x5) { values( "..." );}
            fall_transition(lu5x5) { values( "..." );}
    related_pin : "CLR";
  } /* end of Q timing related to CLR */
} /* end of pin Q */
```

Fonte: BRUNVAND, 2010

Com pode ser visto, a função do sinal de saída Q é definida com base no sinal interno IQ, que teve sua funcionalidade descrita anteriormente pelo bloco *ff*.

No primeiro bloco de *timing*, o atributo *timing_type* é do tipo *rising_edge* que, como foi explanado anteriormente, indica que esse *flop* é sensível à borda de subida. Já no segundo bloco, o *timing_type* é do tipo *clear*, indicando que o sinal CLR funciona como um *reset* para o *flop*, de forma assíncrona. Como a única transição possível causada por esse sinal é de descida, só existem dois arcos definidos (*cell_fall* e *fall_transition*).

É importante notar que, como trata-se de uma célula sequencial, os arcos combinacionais não são definidos para as entradas síncronas, já que modificações nelas não causam alterações na saída, que somente é modificada pelo sinal de *clock* ou por sinais de controle assíncronos (*clear* e *preset*).

3 RECARACTERIZAÇÃO

O processo de recaracterização de uma biblioteca consiste na simulação dos circuitos analógicos equivalentes de cada uma das células desejadas, para novas condições de temperatura e/ou tensão.

Esse processo pode ser feito de forma manual, simulando cada uma das células com a ferramenta desejada e analisando as formas de onda das entradas e das saídas, de modo a preencher as tabelas de *timing* para cada um dos arcos e de acordo com as possibilidades de variação na entrada e de capacitância na saída.

Entretanto, esse processo se torna inviável à medida que o número de células aumenta. Isso ocorre porque é necessário realizar uma simulação para gerar cada um dos valores que compõem as matrizes que definem todos os arcos de cada célula. O número de entradas na célula também contribui nessa inviabilidade, já que cada entrada possui seus arcos correspondentes que devem ser computados.

Dessa forma, para o processo de recaracterização foi utilizada a ferramenta da *Cadence Liberate* que, por meio de *scripts*, faz esse processo de forma automática para cada uma das células especificadas, utilizando o simulador analógico *Spectre* como base.

3.1 PDK Utilizado

O PDK escolhido para realizar o processo de recaracterização foi o PDK de 28nm FD-SOI da *STMicroeletronics*, que utiliza essa tecnologia FD-SOI com o intuito de permitir *designs* de *low power* e alta performance.

As células dessa tecnologia foram projetadas para operar em tensões entre 0,6 e 1,1 V. Além disso, a tecnologia permite técnicas de *body-biasing* para atingir modos de operação que balanceiam *leakage* e performance de acordo com as necessidades do *designer*.

Essa técnica de *body-biasing* consiste em aplicar uma tensão no corpo do transistor diferente da tensão na fonte, o que faz com que a tensão de *threshold* (V_T) do transistor seja alterada. Alterando essa tensão, é possível melhorar a performance das células em troca de maior consumo, ou vice-versa.

A biblioteca escolhida para recaracterização foi a de células de *low power*, que por padrão utiliza essa técnica de *body-biasing* de modo a reduzir o consumo. A tensão escolhida

para a recaracterização foi de 0,6 V, de modo a reduzir o consumo ainda mais. A temperatura foi mantida em seu valor nominal, 25ºC.

3.2 Arquivos Necessários

Para que o processo de recaracterização possa ser executado, alguns arquivos do PDK devem ser localizados. Seguem eles:

- Arquivo *liberty*: É necessário a presença de um arquivo desse tipo para ser utilizado como base durante as simulações, para que a ferramenta conheça todos os arcos e parâmetros de cada célula e da biblioteca.
- Modelos *spice*: Para a simulação das células, é necessário possuir os modelos dos dispositivos básicos do PDK, como transistores, resistores, capacitores e diodos. Esses modelos geralmente estão agrupados de acordo com o processo escolhido.
- Circuitos *spice* das células: Para que a tradução dos circuitos equivalentes das células e consequente simulações possam ser feitos, é necessário possuir cada uma das células descritas utilizando os dispositivos básicos do PDK.

3.3 Configurações Iniciais

3.3.1 *Template*

O primeiro passo para o processo de recaracterização é a criação de um *template*, que conterá as definições dos pinos de alimentação utilizados, dos parâmetros para a medição de *delays* e de tempos de transição, dos índices para as tabelas de *timing* e das células em si.

Para um número reduzido de células, a criação manual desse arquivo é simples. Entretanto, para a recaracterização de uma biblioteca inteira, o *Liberate* permite que o *template* seja gerado de forma automática. Isso é feito a partir da leitura de um arquivo *liberty* do PDK. O código a seguir mostra os comandos necessários para a geração do *template* de forma automática.

Código 1: Geração automática do *template*

```
read_library ${LIBFILE}
write_template -define_index -define_max_trans -pvt_filename ${PVTFILE}.tcl ${TEMPLATE}.tcl
```

Fonte: Autor

A opção *-define_index* do comando *write_template* é utilizada pois, por padrão, o *Liberate* assume que todos os arcos de *timing* de uma célula utilizam os mesmos índices. Entretanto, isso nem sempre é o caso, e a opção faz com que a ferramenta defina o índice para os arcos que possuam valores diferentes.

Já a opção *-define_max_trans* adiciona o parâmetro *max_transition* ao cabeçalho da biblioteca, de acordo com a maior transição permitida em todas as células.

Por fim, a opção *-pvt_filename* salva as informações de *power*, como os pinos de alimentação e seus tipos, bem como os pinos de alimentação utilizados por cada uma das células em um arquivo separado, que pode ser carregado durante o processo de caracterização.

Os códigos a seguir mostram o arquivo de PVT gerado e parte do *template* gerado pelo *Liberate* para o PDK de 28nm.

Código 2: Arquivo de PVT gerado

```
set_gnd -type primary -attributes {related_bias_pin "gnds"} gnd $gnd
set_vdd -type primary -attributes {related_bias_pin "vdds"} vdd $vdd
set_gnd -type pwell -attributes {physical_connection device_layer} gnds $gnds
set_gnd -type nwell -attributes {physical_connection device_layer} vdds $vdds

# set_pin_vdd/gnd extracted from library C28SOI_SC_12_CORE_LL
set_pin_vdd -supply_name vdd C12T28SOIDV_LLBR0P6_NAND3X18_P0 A $vdd
set_pin_gnd -supply_name gnd C12T28SOIDV_LLBR0P6_NAND3X18_P0 A $gnd
…
```

Fonte: Autor

No arquivo de PVT, pode-se ver os quatro pinos de *power* que cada célula possui, com os principais sendo *vdd* e *gnd*, e os pinos utilizados para realizar *body-biasing* sendo *vdds* e *gnds*.

Além disso, a ferramenta define a conexão dos pinos de alimentação de cada uma das células da biblioteca com os pinos de *power* definidos. Isso é feito para cada um dos pinos de cada célula, o que não é necessário para a biblioteca em questão, mas poderia ser útil em outros casos.

Código 3: Parte do *template* gerado

```
set_var slew_lower_rise 0.3
set_var slew_lower_fall 0.3
set_var slew_upper_rise 0.7
set_var slew_upper_fall 0.7

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 1
set_var max_transition 1.1e-09
set_var min_transition 2e-12
set_var min_output_cap 2e-16

set cells { \
  C12T28SOIDV_LLBR0P6_NAND3X18_P0 \
  C12T28SOI_LLBR0D8_NAND2X14_P0 \
  C12T28SOI_LLBR0D8_NAND2X7_P0 \
  C12T28SOI_LLBR0P6_NAND3X12_P0 \
  C12T28SOI_LLBR0P6_NAND3X18_P0 \
  C12T28SOI_LLBR0P6_NAND3X24_P0 \
…
}
…
define_template -type delay \
         -index_1 {0.002 0.011 0.022 0.043 0.085 0.17 0.34 0.675 } \
         -index_2 {0.0002 0.0019 0.0037 0.0075 0.015 0.0447 } \
         delay_template_8x6_8

define_template -type constraint \
         -index_1 {0.002 0.042 0.084 0.34 0.675 } \
         -index_2 {0.002 0.012 0.023 0.094 0.1875 } \
         constraint_template_5x5

define_template -type power \
         -index_1 {0.002 0.011 0.022 0.043 0.085 0.17 0.34 0.675 } \
         -index_2 {0.0002 0.0019 0.0037 0.0075 0.015 0.0447 } \
         power_template_8x6_8

if {[ALAPI_active_cell "C12T28SOI_LLHF_SDFPHRQNX4_P0"]} {
define_cell \
       -clock { CP } \
       -async { RN } \
       -input { D E TE TI } \
       -output { QN } \
       -pinlist { CP D E RN TE TI QN } \
       -delay delay_template_8x6_8 \
       -power power_template_8x6_8 \
       -constraint constraint_template_5x5 \
       C12T28SOI_LLHF_SDFPHRQNX4_P0
```

Fonte: Autor

A partir das quatro primeiro linhas, pode-se ver que a medição do tempo de transição nessa tecnologia é feita entre 30% de VDD e 70% de VDD. Nas quatro linhas seguintes, pode-se ver que a medição do *delay* entre a variação da entrada e consequente variação da saída para essa tecnologia é entre 50% de VDD na entrada e 50% de VDD na saída.

A variável *cells* contém a lista de células a serem caracterizadas.

O comando *define_template* é utilizado para definir os índices utilizados na computação dos arcos, que podem ser de *delay* ou tempo de transição (*-type delay*), de consumo (*-type power*) ou de *constraints* para células sequenciais (*-type constraint*).

Já o comando *define_cell* é utilizado para criação de cada uma das células a serem caracterizadas. As seguintes opções desse comando são relevantes:

- *-input*: Lista as entradas da célula, não incluindo sinais de controle como *clocks* e sinais assíncronos como *clear/preset*.
- *-output*: Lista todas as saídas da célula.
- *-async*: Lista os sinais assíncronos da célula, como *clear/preset*.
- *-clock*: Lista os sinais de controle de células sequenciais.
- *-pinList*: Lista todos os pinos da célula, incluindo todos os agrupados nas diferentes categorias descritas anteriormente.
- *-delay*: Define o *template* para a computação dos arcos de *delay* e de tempos de transição.
- *-power*: Define o *template* para a computação do consumo da célula.
- *-constraint*: Define o *template* para a computação dos tempos de *setup* e de *hold* das entradas de uma célula sequencial.

3.3.2 *Userdata*

Para que o arquivo *liberty* seja gerado corretamente algumas informações adicionais que não são coletadas durante a criação do *template* devem ser fornecidas à ferramenta, como área, *footprint*, tipo de célula (*filler, decap, welltap*) e algumas outras *flags* utilizadas no formato (*dont_use*, *dont_touch*).

Para a obtenção dessas informações, que estão presentes nos arquivos *liberty* do PDK, foi criado um *script* em *Python* para a leitura e escrita delas no formato adequado. O *script* pode ser encontrado no APÊNDICE A.

Durante a escrita do arquivo *liberty*, essas informações devem ser fornecidas à ferramenta no formato *liberty*. O resultado parcial obtido com o *script* criado pode ser visto no código que segue.

Código 4: *Userdata* parcial

```
library (C12T28SOI_userdata) {
  cell (C12T28SOIDV_LLBR0P6_NAND3X18_P0) {
    area : 2.6112;
    cell_footprint : 3BAA5927A22E66B0AE1214A806440F12;
  }

  cell (C12T28SOI_LLBR0D8_NAND2X14_P0) {
    area : 1.4688;
    cell_footprint : 00D3E6753A12F3BDAE93CE03120A93FD;
  }

  cell (C12T28SOI_LLBR0D8_NAND2X7_P0) {
    area : 1.1424;
    cell_footprint : ABC1A221DE8F46AA85BD602498CEF70F;
  }
…
}
```

Fonte: Autor

3.3.3 Escolha dos modelos *spice*

Os modelos *spice* dos dispositivos básicos do PDK geralmente são organizados através de seções, que permitem que o usuário escolha o tipo de processo que será utilizado durante a caracterização. Os processos são geralmente divididos em *typical, slow* e *fast*.

- *typical*: O processo típico utiliza valores de parasitas (resistências e capacitâncias indesejadas no circuito) comuns, que são os mais esperados.
- *slow*: O processo lento utiliza valores de parasitas altos, constituindo o pior caso para *setup* em implementações de blocos.
- *fast*: O processo rápido utiliza valores de parasitas baixos, constituindo o pior caso para *hold* em implementações de blocos.

Durante os trabalhos desenvolvidos, foi utilizado somente o processo típico, de modo a simplificar as análises. Dessa forma, como os modelos do PDK utilizam a seção típica por padrão, não foi necessário a configuração delas.

3.4 Execução

De modo a facilitar o processo, foi utilizado como base para o *script* de caracterização do *Liberate* o *RAK* fornecido pela *Cadence*, onde foram feitas as devidas alterações. Os *scripts* utilizados podem ser vistos no APÊNDICE B.

O processo de caracterização pode ser resumido aos seguintes comandos:

- *set_operating_condition*: Permite a definição da tensão (*-voltage*) e da temperatura de operação (*-temp*) da simulação com o *Spectre*.
- *set_var extsim_model_include*: Informa o caminho dos modelos *spice* dos dispositivos.
- *define_leafcell*: É utilizado para informar ao simulador quais são os nomes dos transistores e diodos utilizados nos circuitos.
- *read_spice*: Informa à ferramenta o caminho dos arquivos contendo os circuitos das células a serem caracterizadas no formato *spice*.
- *char_library*: Realiza o processo de caracterização em si.
- *write_library*: Cria o arquivo *liberty* com os dados coletados durante a simulação. É nesse comando que, com a opção *-user_data*, os dados adicionais coletados manualmente são fornecidos à ferramenta.

Com isso, o processo de recaracterização foi executado para uma tensão de 0,6 V e uma temperatura de 25ºC. Como planejava-se executar o fluxo de *place & route* completo, foi feita também a recaracterização da biblioteca de células *filler*, *decap* e *welltap* da tecnologia para as mesmas condições, e utilizando os mesmos *scripts*.

4 APLICAÇÃO DOS RESULTADOS

De modo a validar os resultados obtidos após a recaracterização da biblioteca, foi feita síntese lógica seguida do fluxo de *place & route* do *core* do microcontrolador *PULPino*, utilizando as ferramentas *Genus* e *Innovus* da *Cadence*.

O intuito foi o de realizar o fluxo completo em 0,6 V utilizando tanto a biblioteca recaracterizada quanto as opções de interpolação da ferramenta de *place & route* a partir dos arquivos *liberty* providos pela *foundry*, para que os resultados pudessem ser validados e as aproximações da ferramenta analisadas.

Dessa forma, o principal objetivo dessa implementação foi o de analisar os possíveis erros que seriam obtidos ao tentar aproximar a condição de operação de 0,6 V ao invés de recaracterizar a biblioteca, utilizando somente os arquivos fornecidos pela *foundry*.

4.1 Microcontrolador *PULPino*

O *PULPino* é um microcontrolador de código aberto de um *core*, baseado em processadores de 32 bits desenvolvidos pela *ETH Zurich*.

Figura 10: Logo do PULPino

Fonte: github.com/pulp-platform/pulpino

O *core RISCY* possui quatro estágios de *pipeline* e tem um IPC próximo de 1. Ele possui suporte para instruções do tipo inteiro RV32I, instruções comprimidas RV32C e instruções de multiplicação RV32M.

4.2 Fluxo de Implementação

4.2.1 Síntese Lógica

Inicialmente, utilizando a ferramenta *Genus* foram feitas duas sínteses lógicas do *core*, uma utilizando a biblioteca recaracterizada para o ponto de operação de [0,6 V, 25ºC, típico] e outra utilizando o arquivo *liberty* fornecido pela *foundry* para o ponto de operação [0,8 V, 25ºC, típico]. A tensão de 0,8 V foi escolhida por ser o menor valor de tensão fornecido pela *foundry* no *corner* típico.

A frequência do *clock* escolhida foi de 50 MHz.

A tabela abaixo resume os resultados ao final da síntese com relação ao *timing* e ao consumo de ambos os blocos.

Tabela 1: Comparativo pós síntese

	0,6 V, 25ºC, *typ*	**0,8 V, 25ºC, *typ***	**Variação**	
			Absoluta	**Percentual**
***WNS (Worst Negative Slack)* (ns)**	8.28	15.15	6.87	82.97%
Consumo (mW)	0.66	1.23	0.57	86.36%

Fonte: Autor

A partir da tabela, pode-se notar que o consumo foi reduzido com a utilização da biblioteca recaracterizada, mas que o *timing* foi um pouco degradado. Ambos os resultados eram esperados, já que o consumo é proporcional à tensão de alimentação, e que a velocidade das células é inversamente proporcional a essa tensão.

4.2.2 Implementação

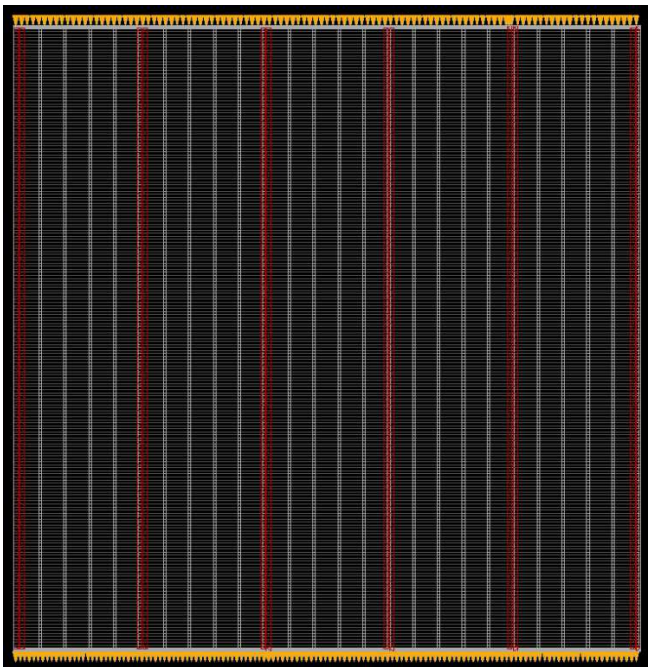
De posse das duas *netlists* geradas pela síntese, iniciou-se o processo de implementação do bloco.

A ferramenta de *place & route* utilizada, *Innovus*¸ permite que um ponto de operação diferente do nominal seja utilizado. Isso é feito a partir da interpolação de dois arquivos *liberty* diferentes. Dessa forma, durante a implementação da *netlist* obtida através da síntese com a

biblioteca de 0,8 V, foram carregadas duas bibliotecas, nos seguintes pontos de operação [0,8 V, 25ºC, *typ*] e [0,9 V, 25ºC, *typ*] e a ferramenta foi instruída a tentar aproximar o ponto de operação [0,6 V, 25ºC, *typ*].

A primeira etapa foi o *floorplanning,* que consiste na definição da área do bloco a ser implementado, posicionamento dos pinos de entrada e saída, *placement* das células de *WELLTAP* e de *ENDCAP* e disposição das *Stripes* para alimentação das células. O *floorplan* de ambos os fluxos foi idêntico, utilizando uma área de 250 x 250 um, e posicionando as entradas no topo do bloco e as saídas na parte inferior. A figura abaixo mostra o resultado final dessa etapa.

Figura 11: *Floorplan*

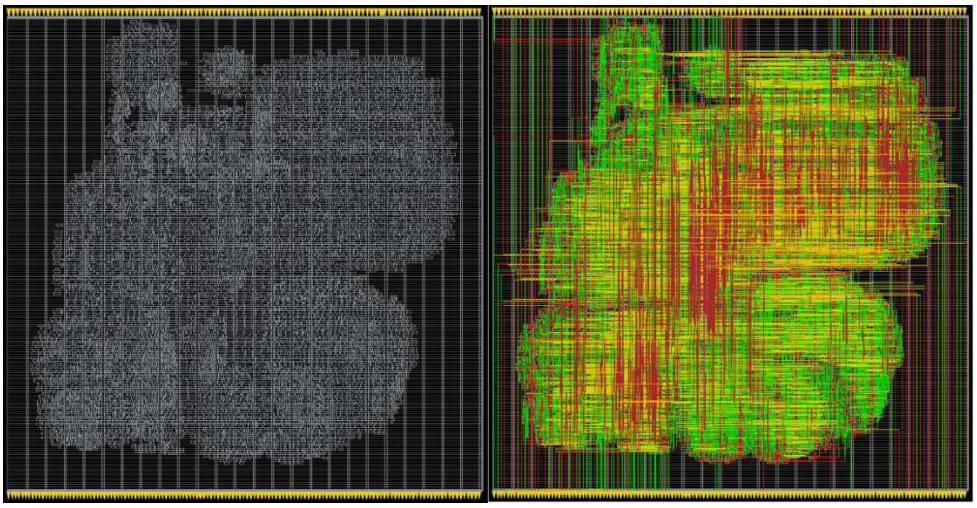


Fonte: Autor

Após isso, foi feito o *placement* das células na região definida. Durante essa etapa, a ferramenta faz um roteamento preliminar, de modo a estimar como se comportarão os sinais após o roteamento final do bloco, e tenta otimizar a localização das células de modo a minimizar violações de *setup*. Durante esse processo, o *clock* ainda é considerado ideal, chegando ao mesmo tempo em pontos.

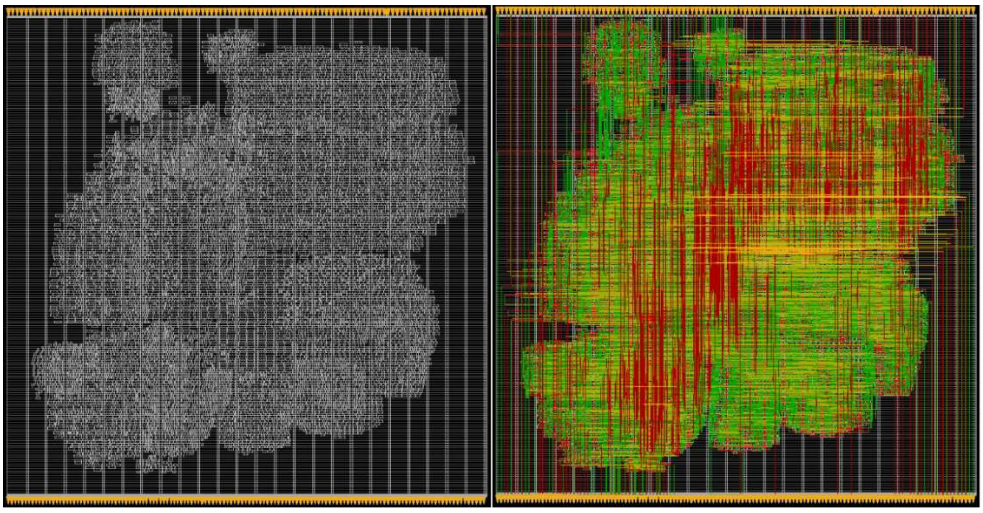
As imagens abaixo mostram os resultados para ambos os fluxos após essa etapa.

Figura 12: *Placement* (lib recaracterizada)



Fonte: Autor

Figura 13: *Placement* (interpolação)



Fonte: Autor

A tabela abaixo lista os resultados de *timing* e consumo ao final desta etapa.

Tabela 2: Comparativo pós *placement*

	0,6 V, 25ºC, *typ*	**0,8 V, 25ºC, *typ***	**Variação**	
			Absoluta	**Percentual**
***WNS* (ns)**	8.26	12.69	4.43	53.63%
Consumo (mW)	0.63	1.07	0.44	69.84%

Fonte: Autor

Com base nesses resultados preliminares, percebe-se que o erro diminuiu com relação aos resultados obtidos pelo *Genus*, uma vez que o *Innovus* tentou aproximar o ponto de operação do bloco para 0,6 V. Como era esperado, o consumo continua menor para a biblioteca recaracterizada e o *timing* pior.

A próxima etapa do fluxo de implementação é a *CTS* (*Clock Tree Synthesis*), que consiste em considerar a propagação do sinal de *clock* ao longo do bloco e na inserção de *buffers* e inversores para garantir que o sinal de controle chegará às células em momentos similares, já que o *clock* é essencial para o *timing*.

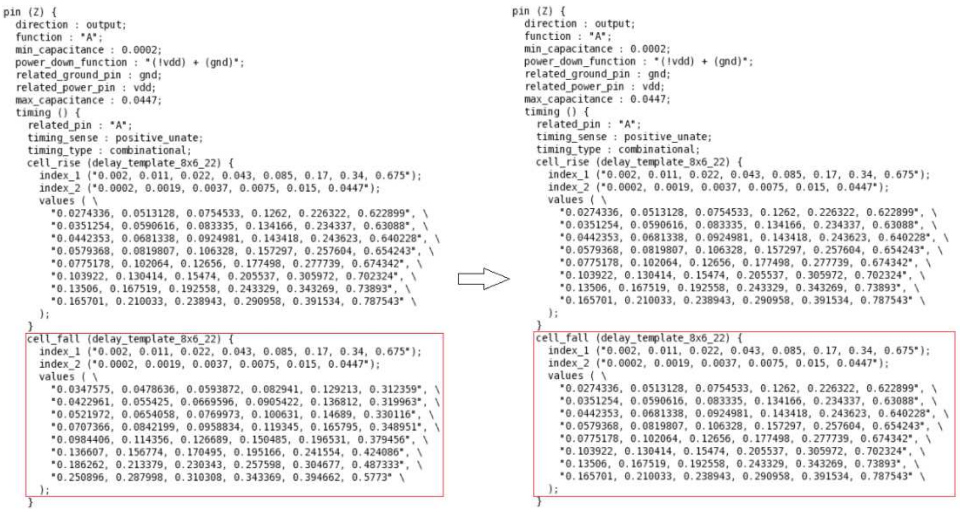
A tecnologia possui células especificamente projetadas para construir essa árvore, agrupadas em uma biblioteca a parte. Isso significa que as células da biblioteca padrão não são adequadas para propagar o sinal de *clock*. Durante a execução do fluxo não foi possível obter acesso a essa biblioteca adicional.

Como consequência disso, não foi possível fazer a construção da *clock tree* utilizando somente a biblioteca padrão, e o *Innovus* informou que havia um desbalanceamento dos valores de *cell_rise* e de *cell_fall* para todos os *buffers* e inversores disponíveis.

Para contornar esse problema, foram editados todos os arquivos *liberty* utilizados, especificamente para as células BFX4 (*buffer*) e IVX4 (inversor), de modo que esses de *delay* se tornassem idênticos. Como essa alteração afeta ambos os fluxos, o erro inserido foi considerado aceitável.

A imagem abaixo ilustra esse processo para um dos arquivos.

Figura 14: Edição do arquivo liberty (BFX4)



Fonte: Autor

Essa alteração permitiu a construção da *clock tree* utilizando somente essas duas células. Para eliminar as violações de *hold* geradas, foram feitas otimizações em ambos os fluxos. A tabela abaixo lista os resultados de *timing* e de consumo ao final desta etapa.

Tabela 3: Comparativo pós *CTS*

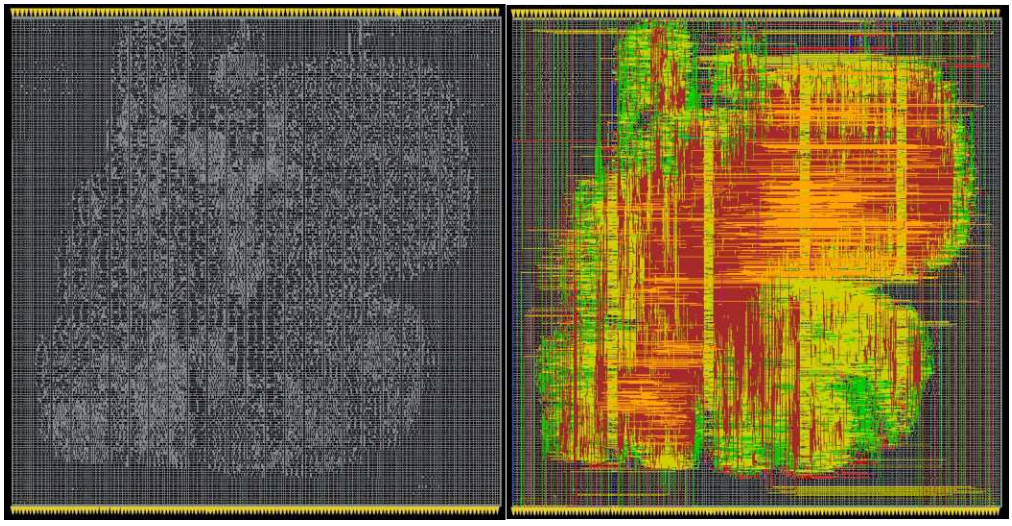
	0,6 V, 25ºC, *typ*	**0,8 V, 25ºC, *typ***	**Variação**	
			Absoluta	**Percentual**
***Setup WNS* (ns)**	8.05	12.65	4.60	57.14%
***Hold WNS* (ns)**	0.00	0.00	0.00	-
Consumo (mW)	0.70	1.34	0.64	91.43%

Fonte: Autor

Por fim, foi feito então o roteamento de ambos os blocos, que é a etapa onde, como o nome sugere, os pinos de dados das células são efetivamente roteados, já que a etapa anterior roteou os pinos de *clock*. Após isso foi feita a inserção de *filler cells*, que tem como objetivo garantir a continuidade das trilhas de alimentação das células.

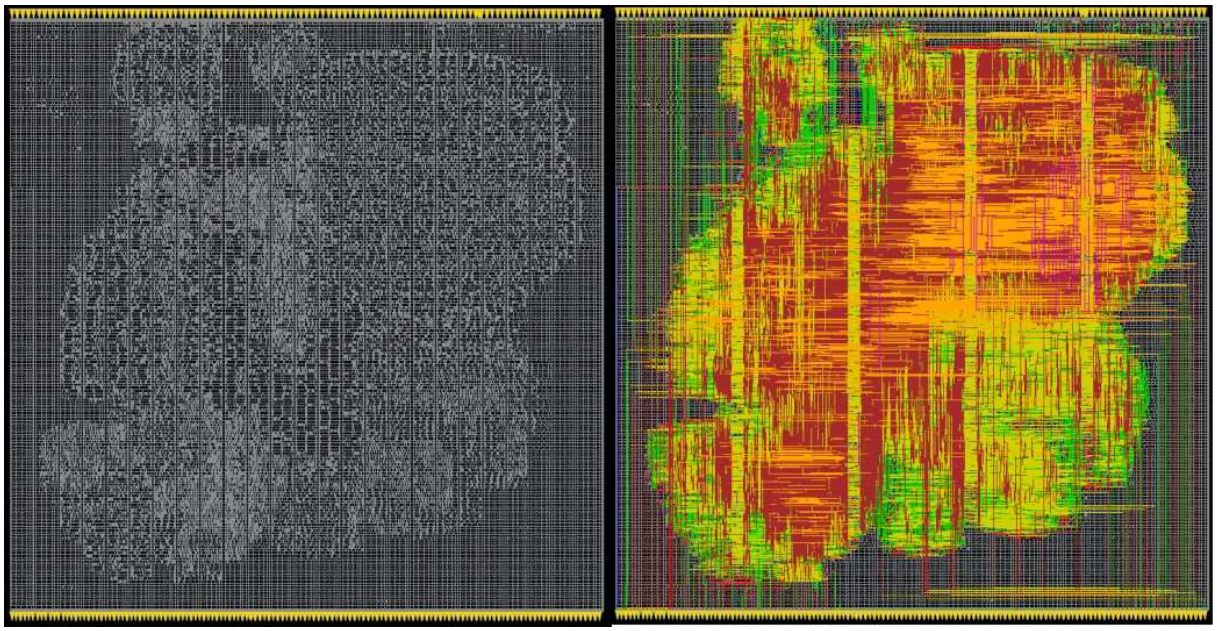
As imagens a seguir mostram o resultado final da implementação.

Figura 15: Resultados finais da implementação (lib recaracterizada)



Fonte: Autor

Figura 16: Resultados finais da implementação (interpolação)



Fonte: Autor

A tabela a seguir contém a comparação dos resultados finais dos fluxos.

Tabela 4: Comparativo final

	0,6 V, 25ºC, *typ*	**0,8 V, 25ºC, *typ***	**Variação**	
			Absoluta	**Percentual**
***Setup WNS* (ns)**	7.96	12.51	4.55	57.16%
***Hold WNS* (ns)**	0.063	0.048	0.015	23.81%
Consumo (mW)	0.73	1.45	0.72	98.63%

Fonte: Autor

Os resultados finais de consumo demonstram um erro elevado da ferramenta durante a interpolação. Aliando esses resultados com o fato do *slack* de *hold* ter se mostrado melhor na biblioteca recaracterizada e do *slack* de *setup* ter se mostrado pior, é possível inferir que a interpolação feita pela ferramenta gerou resultados equivalentes ao de uma biblioteca com tensão maior que 0,6 V.

5 CONCLUSÕES

O trabalho executado possibilitou a recaracterização da biblioteca de células de *low power* provida pela *STMicroeletronics,* bem como a criação de um fluxo de recaracterização de uma biblioteca de células, fluxo esse que pode ser facilmente adaptado para outras tecnologias.

Os testes realizados com a ferramenta de implementação mostraram que o processo de interpolação utilizado durante o trabalho em condições de operação diferentes da nominal não são precisos o suficiente, principalmente quando o ponto de operação desejado não está próximo dos pré-existentes. Como o fluxo de implementação de um bloco já consiste em uma estimativa do comportamento de dispositivos reais, a adição de mais uma camada de aproximações não é viável.

Aliando o fato de aplicações de *low power* estarem se tornando cada vez mais necessárias com a imprecisão das técnicas de interpolação utilizadas por ferramentas de implementação, é possível perceber a importância do trabalho com recaracterização de células. Além disso, o processo de recaracterização permite uma maior granularidade durante o fluxo de um projeto, dando liberdade ao *designer* para decidir a quais condições de operação seu bloco estará realmente submetido, não estando somente limitado ao que foi fornecido pela *foundry*.

REFERÊNCIAS

Best-in-Class Standard-Cell Libraries for HighPerformance, Low-Power and High-Density SoC Design in 28nm FD-SOI Technology. Disponível em: < https://www.st.com/content/ccc/resource/sales_and_marketing/presentation/technology_presentation/ff/ea/08/a2/a8/6d/41/46/Standard_Cell_White_Paper_20150928>. Acesso em: 30 de Abril de 2021.

BRUNVAND, E. ***Digital VLSI Chip Design with Cadence and Synopsys CAD Tools***. Upper Saddle River, NJ: Pearson, 2010.

Cirit, M. A. ***Characterizing a VLSI standard cell library***. *Proceedings of the IEEE 1991 Custom Integrated Circuits Conference.*

GRANDHI, S. K. ***Library Characterization:*** *Its Impact on Semiconductor Industry & the flow*. Disponível em: <https://www.slideshare.net/sgrandhi/library-characterization-flow>. Acesso em: 10 de Abril de 2021.

Liberate - *Template file generation*. 2017.

PULPino. Disponível em: < https://github.com/pulp-platform/pulpino>. Acesso em: 30 de Abril de 2021.

WICHERN, Don. ***Characterizing cells and writing a technology library file***. Disponível em: <http://ece451web.groups.et.byu.net/cadence-help/src/tlf.pdf>. Acesso em: 30 de Abril de 2021.

APÊNDICE A – CÓDIGOS PYTHON

Código 5: Leitura e escrita de dados adicionais para o *liberty*

```
import re
import sys
import gzip
import os
import subprocess
import argparse
import csv
import shutil
import datetime
import traceback
from os import walk
from pathlib import Path

#libType = 'pr'
libType = 'clk'

if libType == 'core':
    libPath = '/mnt/hdd1tera/cmos28fdsoi_29/C28SOI_SC_12_CORE_LL/5.1-05/libs/'
    libFile = 'C28SOI_SC_12_CORE_LL_tt28_1.00V_0.00V_0.00V_0.00V_25C.lib'
else:
    libPath = '/mnt/hdd1tera/cmos28fdsoi_29/C28SOI_SC_12_PR_LL/5.1-05/libs/'
    libFile = 'C28SOI_SC_12_PR_LL_tt28_1.00V_0.00V_0.00V_0.00V_25C.lib'
file = libPath + libFile

lib = open(file,'rt')
lines = []
cells = {}

read_next = True
# Read for inputs of all cells
while True:
    if read_next:
        s = lib.readline()

    if not s:
        break

    if ' cell(' in s:
        area = ''
        footprint = ''

        read_next = True
        cell = s.partition('(')[2].partition(')')[0]
```

```
        while True:

            s = lib.readline()

            if not s or ' cell(' in s or 'test_cell()' in s:
                read_next = False
                if 'test_cell()' in s:
                    read_next = True

                combCells[cell] = {}

                cells[cell]['area'] = area
                cells[cell]['footprint'] = footprint
                if libType == 'pr':
                    cells[cell]['special_def'] = special_def

                break

            if 'area' in s:
                area = s.partition(': ')[2].replace(';','').strip()

            if 'cell_footprint' in s:
                footprint = s.partition(': ')[2].replace(';','').strip()
            if libType == 'pr':
                if 'is_tap_cell' in s:
                    special_def = 'is_tap_cell'
                if 'is_filler_cell' in s:
                    special_def = 'is_filler_cell'
                if 'is_decap_cell' in s:
                    special_def = 'is_decap_cell'


lib.close()


with open('userdata_' + libType + '.lib', 'w') as file:
    file.write('library (C12T28SOI_userdata) {\n')
    for cell in cells.keys():
        file.write('  cell (' + cell + ') {\n')
        file.write('    area : ' + cells[cell]['area'] + ';\n')
        file.write('    cell_footprint : ' + cells[cell]['footprint'] + ';\n')
        if libType == 'pr':
            file.write('    ' + cells[cell]['special_def'] + ' : true;\n')
        file.write('  }\n\n')
```

Fonte: Autor

APÊNDICE B – SCRIPTS TCL

Código 6: *Script* de configurações

```
set SRC_DIR             [pwd]
set RUN_DIR             [pwd]

set LIBTYPE             core

if { $LIBTYPE == “core” } {
    set LIB                 C28SOI_SC_12_CORE_LL
} else {
    set LIB                 C28SOI_SC_12_PR_LL
}

set PROCESS             tt    ;# [ff|tt|ss]

set vdd                 0.60
set gnd                 0.00
set vdds                0.00
set gnds                0.00

set TEMP                25
set TYPE                nldm  ; #nldm, ccs or ecsm

set TEMPLATE_FILE       ${SRC_DIR}/template/liberate_templates_${LIBTYPE}.tcl
set CELLS_FILE          ${RUN_DIR}/cells.tcl
set NETLIST_DIR         /mnt/hdd1tera/cmos28fdsoi_29/${LIB}/5.1-05/physical
set USERDATA            ${SRC_DIR}/userdata/userdata_${LIBTYPE}.lib

#####----------------------------------------
##### Set dependent variables
#####----------------------------------------
set TEXT_TEMP ${TEMP}
if { ${TEMP} < 0 } {
    set TEXT_TEMP [expr ${TEMP} * -1]
    set TEXT_TEMP m${TEXT_TEMP}
}
set TEXT_VDDS ${vdds}
if { ${vdds} < 0 } {
    set TEXT_VDDS [expr ${vdds} * -1]
    set TEXT_VDDS m${TEXT_VDDS}
}
set TEXT_GNDS ${gnds}
if { ${TEMP} < 0 } {
    set TEXT_GNDS [expr ${gnds} * -1]
    set TEXT_GNDS m${TEXT_gnds}
}
```

```
set PVT                ${PROCESS}_${vdd}_${TEXT_TEMP}
set LIBNAME            ${LIB}_${PROCESS}28_${vdd}V_${gnd}V_${TEXT_VDDS}V_${TEXT_GNDS}V_${TEXT_TEMP}C

if { ${PROCESS} == “tt” } {
    set MODEL_INCLUDE_FILE   ${SRC_DIR}/models/spectre/corners.scs
} elseif { ${PROCESS} == “ss” } {
    set MODEL_INCLUDE_FILE   ${SRC_DIR}/models/spectre/corners_max.scs
} else {
    set MODEL_INCLUDE_FILE   ${SRC_DIR}/models/spectre/corners_min.scs
}
set THREAD    0
set CLIENTS   0
```

Fonte: Autor

Código 7: *Script* de caracterização

```
#####----------------------------------------
##### Set and print user define variables
#####----------------------------------------
source tcl/setup.tcl

#####----------------------------------------
##### Set operating condition
#####----------------------------------------
puts "INFO: Set Operating Condition"
set_operating_condition -name ${PVT} -voltage ${vdd} -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} { lvtnfet eglvtvnfet }
define_leafcell -type pmos -pin_position {0 1 2 3} { lvtpfet eglvtvpfet }
define_leafcell -type diode -pin_position {0 1} { diodepwtw diodetwx tdndsx tdpdnw }

#####----------------------------------------
##### read cell netlists
#####----------------------------------------
## read netlist
set spicefiles "${NETLIST_DIR}/${LIB}.spi"
read_spice -format spectre "$spicefiles"
```

```
#####----------------------------------------
##### Run characterization
#####----------------------------------------
char_library -extsim spectre -cells $cells -thread $THREAD -ccs -ccsn -ccsp



#####----------------------------------------
##### Write output
#####----------------------------------------
### Write ldb ###
file mkdir ${RUN_DIR}/ldb
write_ldb -overwrite ${RUN_DIR}/ldb/${LIBNAME}.ldb

### Write Liberty ###
file mkdir ${RUN_DIR}/lib
if {${TYPE} == "nldm" } {
    write_library -driver_waveform -unique_pin_data -bus_syntax {[]} -user_data ${USERDATA} -
overwrite -filename ${RUN_DIR}/lib/${LIBNAME}_nldm.lib ${LIBNAME}
} elseif {${TYPE} == "ccs"} {
    write_library -driver_waveform -unique_pin_data -bus_syntax {[]} -user_data ${USERDATA} -ccs -
ccsn -ccsp -overwrite -filename ${RUN_DIR}/lib/${LIBNAME}_ccs.lib ${LIBNAME}
} else {
    write_library -driver_waveform -unique_pin_data -bus_syntax {[]} -user_data ${USERDATA} -ecsm -
ecsmn -overwrite -filename ${RUN_DIR}/lib/${LIBNAME}_ecsm.lib ${LIBNAME}
}
```

Fonte: *Cadence* (alterado)