Universidade Federal de Campina Grande
Centro de Graduação em Engenharia Elétrica
Departamento de Engenharia Elétrica

VINÍCIUS BARBOSA DE OLIVEIRA ARAÚJO

SÍNTESE LÓGICA E FÍSICA DE UM SOC EM FERRAMENTAS OPEN SOURCE E COMERCIAIS

CAMPINA GRANDE - PB

OUTUBRO DE 2021

SÍNTESE LÓGICA E FÍSICA DE UM SOC EM FERRAMENTAS OPEN-SOURCE E COMERCIAIS

VINÍCIUS BARBOSA DE OLIVEIRA ARAÚJO

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.

CAMPINA GRANDE/PB

OUTUBRO DE 2021

VINÍCIUS BARBOSA DE OLIVEIRA ARAÚJO

SÍNTESE LÓGICA E FÍSICA DE UM SOC EM FERRAMENTAS OPEN-SOURCE E COMERCIAIS

Área de Concentração: Eletrônica

Marcos Ricardo de Alcântara Morais

Orientador

Gutemberg Gonçalves dos Santos Júnior

Convidado

CAMPINA GRANDE - PB

OUTUBRO DE 2021

Sumário

Resumo

O projeto de um chip de circuito integrado leva diversas etapas. Esse processo envolve regras de manufatura elaboradas que devem ser passadas pelas fábricas. Até então só existiam processos muito caros e que não eram acessíveis para desenvolveres independentes. Há alguns anos, ferramentas e processos de código aberto vem surgindo e se tornando cada vez mais eficientes. Esse projeto irá descrever o fluxo de síntese lógica e física tanto com ferramentas open-source como em comerciais, após isso será elaborada uma comparação entre elas.

Palavras-chave: open-source, circuito integrado, síntese lógica, síntese física

Abstract

The design of an integrated circuit chip takes several steps. This process involves elaborate manufacturing rules that must be passed by factories. Until then, there are very expensive processes that are not accessible independent developer. For some years, open source tools and processes have been emerging and becoming more and more efficient. This project will describe the flow of logical and physical synthesis with both open source and commercial tools, after which a comparison will be made between them.

Keywords:: open source, tools, integrated circuit, logic synthesis, physical synthesis

1 Introdução

O projeto de um *chip* em circuito integrado é um processo dividido em várias etapas. Existe um fluxo padrão estabelecido, que trata de dividir o processamento e de automatizá-lo. Esse fluxo pode ser visto como duas grandes áreas, *frontend* e *backend*. Cada qual possui uma equipe especializada. O frontend é a área funcional do sistema. Onde é descrito, por meio de uma linguagem de hardware, o funcionamento do circuito que será fabricado à nível de registradores, chamado de *RTL* (*Register Transfer Level*).

Uma vez finalizado o *RTL*, a equipe de *frontend* implementa a síntese lógica. O objetivo é converter o projeto para um formato de portas lógicas chamada de *netlist*. Após isso, a equipe de *backend* implementa a síntese física. Nela são tratados problemas relacionados ao processo de fabricação.

A realização do fluxo é feito de forma automatizada. Para isso, existem ferramentas específicas. Elas ajudam a simular o comportamento do projeto, na tentativa de garantir que ele funcione no mundo real.

Comumente, é utilizado o termo *SoC* (*System on a Chip*) para projetos de *chip* que possuem vários blocos dentro dele.

Para se implementar a síntese lógica e física, somente existiam ferramentas de código fechado, e seus valores de licensa são bastante altos. Ou seja, o desenvolvimento de um *chip* era inviável para pessoas comuns, ficando restrito às empresas. No entanto, recentemente algumas ferramentas *open source* vem surgindo. Um projeto chamado *OpenLane*, tratou de agrupar algumas dessas ferramentas para criar um fluxo contínuo e automatizado de *backend*.

1.1 Objetivo

Este trabalho tem como objetivo realizar dois fluxos de síntese lógica e física de um *chip* digital. Um com com ferramentas do projeto *OpenLane* e outro com ferramentas comerciais. A tecnologia utilizada em ambos os projetos será referente ao arquivos disponibilizados pela *SkyWater* de 130nm de código aberto. Para isso, serão descritos os passos de implementação e as dificuldades ferramentais. Após a implementação, será comparados os resultados entre os dois fluxos.

Os arquivos que serão entregues ao final do trabalho:

- Relatório de checagens de violações de *DRC*, *LVS*,*STA*.

- Última *netlist* modificada pela implementação física.
- Arquivo de *layout* no formato *GDSII* após o roteamento do *chip*.
- Comparações entre as ferramentas *open source* e comerciais.

2 Fundamentação Teórica

A primeira etapa do desenvolvimento de um circuito integrado é a especificação do sistema. A partir dela, é possível definir toda a estrutura funcional do *chip*. Geralmente são estabelecidos limites de frequência de trabalho do *clock*, portas de entrada e saída, entre outras características importantes a nível macro. Após definida essa estrutura, a equipe de *frontend* é encarregada de transcrever essas informações funcionais para uma linguagem de *hardware*, muitas vezes chamado de código *RTL* (*Register Transfer Level*). Durante o processo de desenvolvimento do *RTL*, o código é verificado várias vezes para checar e garantir a funcionalidade do sistema.

Dado o código *RTL*, é necessário transferir sua funcionalidade para o *layout* - uma estrutura de blocos de metais utilizada pelas fábricas em seus processos de fabricação.

Nas sessões seguintes serão discutidos os fundamentos necessários para se entender o fluxo completo do desenvolvimento do *layout* de um *chip*. Esse fluxo é dividido em três etapas: a síntese lógica, a síntese física e a sua verificação.

2.1 *Process Design Kit*

Existem grandes fábricas (*foundries*) pelo mundo destinadas a criar sistemas de manufatura voltado para fabricação de *chips* em circuitos integrados. É estabelecido um processo automatizado, que garante a entrega do circuito integrado dentro das suas especificações. Esse processo de manufatura é propriedade intelectual da empresa que o criou.

Quando se deseja fabricar um *chip*, é necessário que o projetista entre em contato com a fábrica. Deve ser solicitado o conjunto dos arquivos que compõem as regras que devem ser seguidas durante o projeto. Esses arquivos são compilados dentro de um *PDK*(*Process Design Kit*).

Em um *PDK* de *hardware digital*, deve-se haver células padronizadas (portas lógicas) e arquivos voltados para simulação delas em diversas condições de processo, temperatura e tensão. Vários outros arquivos também são disponibilizados, dentre eles: as regras de projeto, as informações físicas que agregam às simulações, entre outros.

2.1.1 Ferramentas *EDA*

O processamento de toda essa informação é passada para ferramentas de *EDA* (*Eletronic Design Automation*). Nelas, é possível montar a estrutura do *chip*, definir as

portas de entrada e saída, separar espaços para os blocos que compõem o *chip*, também é possível simular várias problemas que podem acontecer e muito mais.

2.2 Síntese Lógica

A primeira etapa do processo, é a síntese lógica. Sua função é converter o *RTL* em um arquivo com células padronizadas presentes no *PDK*. E que possuam, entre elas, caminhos que compõem a lógica do circuito. Esse novo arquivo gerado é chamado de *netlist*.

Uma série de arquivos do *PDK* devem ser carregados dentro da ferramenta de *EDA* que se está trabalhando. Os principais são:

- Código *RTL*.
- Arquivos das células padronizadas.
- Arquivos dos *IP's* (*Intelectual Property*).
- Bibliotecas de *timing*.
- Arquivo de tecnologia do processo.
- Limitações de entradas e saídas.

A estrutura do *RTL*, durante esse processo, é modificada. Inicialmente, a ferramenta *EDA* traduz o código em números binários. São utilizadas otimizações baseadas em mapas de *Karnaugh*, resultando em circuitos mais bem elaborados, compostos por portas lógicas. São atribuídas, a essas portas, células genéricas - que não se referem a nenhuma tecnologia específica. Após isso, é possível mapear as células genéricas em células da tecnologia presente no *PDK*.

Durante a síntese lógica, ainda não são considerados problemas físicos, como resistências ou capacitâncias, pois ainda não foram desenhadas no *layout* do projeto. Ainda assim, existem problemas de *timing* que devem ser corrigidos, além disso o consumo também pode ser levado em consideração. Geralmente, essas simulações são feitas para um tipo de variação de processo, temperatura e tensão, chamado de *corner* (melhor explicado na sessão de *Static Timing Analysis*), e que são baseados na leitura das bibliotecas.

Também podem ser inseridas estruturas de células que melhoram o desempenho do *chip*, como *clock gates* - portas lógicas que desligam o *clock* em momentos que ele não está sendo utilizado - para diminuir o consumo, ou células *multibit*, entre outras.

O resultado é a geração da *netlist*, um arquivo com portas lógicas e caminhos que representam toda a lógica do código *RTL*. Além disso, também é gerado um arquivo de

contraints, com definições do *clock* criado, transições máximas das entradas e saídas, entre outras informações que limitam o processamento da ferramenta.

2.3 Síntese Física

É necessário inserir informações físicas ao projeto para cada vez mais aproximar o modelo que está sendo projetado da realidade. Serão atribuídas mais estruturas dentro do *chip*, como as trilhas de metais que ligam todas as partes do circuito. Isso implicará em atrasos dos sinais, interferências e muitos outros problemas.

Alguns dos arquivos carregados durante essa etapa são:

- *Netlist*.
- Bibliotecas das células padronizadas.
- Arquivos dos *IP's* .
- Tabelas de capacitância das camadas de metais.
- Arquivo de *constraints*.
- Arquivo de tecnologia do processo.

2.3.1 *Floorplan*

Durante o *floorplan* são definidas a área do *core* e do *die*. Também são feitas as disposições dos *IP's*, caso existam. Os *IP's* são blocos com modelos prontos, que já foram simulados e verificados. Eles possuem um *layout* proprietário e reutilizável. A memória *RAM* é um exemplo de *IP*.

Dentro do *core* são alocadas as *macros*, muito semelhante aos *IP's*. São separados locais específicos com bloqueios para que nenhuma trilha de metal seja construida no local em que o bloco ou *IP* se encontra. Os blocos se diferem dos *IP's* por terem sido feitos e encapsulados pelo próprio projetista.

Além disso, deve-se ater aos pinos e *PAD's* do *chip*. Os *PAD's* também são partes fundamentais da estrutura, ele faz a comunicação com o mundo externo e possuem circuitos de proteção contra ruídos que possam danificar os componentes do circuito.

Existem formas de se alocar essas estruturas para melhorarem a *performance* da ferramenta. São duas abordagens principais de particionamento do *floorplan*, *flat* e *hierarquica*, cada qual possuem vantagens e desvantagens.

- *Flat*: Todas as instâncias dos blocos são dispostas dentro do *core* junto com o módulo original, sem que haja uma separação de espaço.
- *Hierarquica*: Os blocos são melhores estruturados, sendo alocado um espaço para cada estrutura. O nível de topo tratará de fazer as ligações de entrada e saída de cada bloco.

O objetivo do *floorplan* é minimizar a àrea, o *timing*, reduzir o tamanho do fio, facilitar o roteamento e reduzir a queda de tensão.

2.3.1.1 Malha de Alimentação

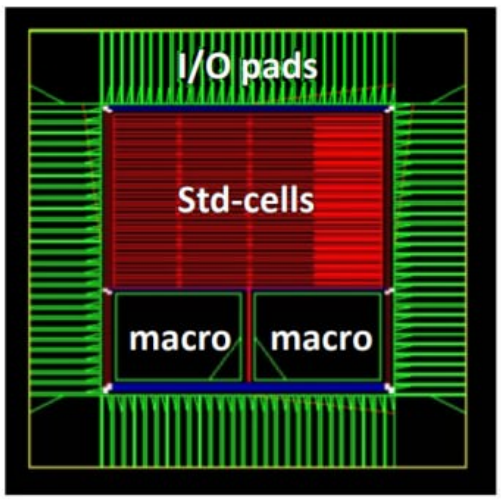
Ainda nessa etapa, é feita a malha de alimentação, o circuitos do *chip* necessitam ser alimentados por uma tensão específica, que depende da tecnologia em que está se trabalhando. Alguns cuidados devem ser tomados.

- Trilhas mais largas diminuem os problemas de queda de tensão.
- Geralmente, a malha inicia nos metais mais altos e descem até o metal mais baixo, numa estrutura chamada de *rails* onde serão colocadas as células padronizadas.

São criadas trilhas de metais, *VDD* e *VSS*, semelhante a aneis de alimentação ao redor do *core* e dos blocos. Após isso, são colocadas *stripes* - trilhas robustas - que atravessam o *core* de um lado ao outro. Geralmente a quantidade de trilhas fica a critério do projetista.

Para exemplificar a estrutura do floorplan e da malha de alimentação:

Figura 1 – Estrutura de um floorplan, junto com a malha de alimentação.



Visto em: <https://www.ques10.com/p/26321/explain-floor-planning-and-routing-1/?>

2.3.2 Static Timing Analysis

A análise de *timing* ocorre ao longo da maioria das etapas do fluxo de síntese física. Com ela, é possível encontrar os caminhos no projeto que não estão atendendo o tempo requerido. É uma forma de garantir que o *clock* irá chegar nos tempos corretos, sem que haja perda de informação.

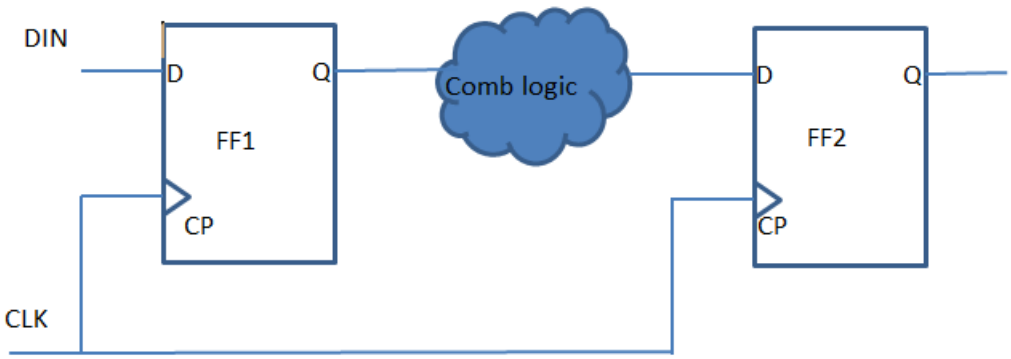
Existem vários problemas que o sinal pode sofrer e alterar a suas características, podendo acelerar, atrasar ou mesmo ocasionar um erro lógico. Dentro do *PDK* possuem informações como capacitâncias, indutâncias e resistências e que serão utilizadas para simular o comportamento do *chip*.

São chamados de *corner* os tipos de processo, tensão e temperatura que serão simulados durante as análises de *timing*. Eles podem ser *slow*, *typical* ou *fast*.

- *Corners typical*: Modo de operação com temperaturas e tensões normais.
- *Corners slow*: Modo de operação comm temperaturas mais altas e tensões mais baixas.
- *Corners fast*: Modo de operação com temperaturas mais baixas e tensões mais altas.

Geralmente, antes de fazer a análise de timing, são extraídos os parasitas das trilhas, pois acarretam em atrasos no sinal. O relatório é baseado nos caminho que são violados, existem dois tipos de violações de tempo, os de setup e os de *hold*. Para exemplificar, observe a ilustração abaixo. Existem dois *flipflops* (*FF1* e *FF2*) que trocam a informação de um dado.

Figura 2 – Representação do caminho do dado e do *clock*.



Visto em: <https://vlsi.pro/sta-setup-and-hold-time-analysis/>

- Tempo de *Setup*: tempo em que a chegada da borda do *clock* na porta CP, em *FF2*, se dá antes da chegada do dado na porta D do *FF2*.

- Tempo de *Hold*: tempo em que a chegada do dado na entrada D do *FF2*, se dá antes da chegada da borda do *clock* na porta CP do *FF2*,

Uma das soluções para corrigir as violações do tmepo de setup é colocar células de atraso no caminho do dado. A solução para corrigir violações de tempo de hold é inserir células de *buffers* no caminho do *clock*.

Alguns conceitos presentes na análise de *timing*:

- *Skew*: Diferença do tempo de chegada do *clock* entre dois *flipflops*.
- *Slew*: Relacionado a subida e descida do *clock*, também chamado de *transition time*.
- *Slack*: Relacionado a diferença entre o tempo requerido e o tempo de chegada do dado.

Também existem violações de *DRV* (*Design Rule Verification*). Elas se referem às limitações que a tecnologia impõe. Algumas das *DRVs* são:

- *Max Transition* (ou *Max Slew*): São referentes ao tempo de subida e descida do sinal, existe um limite máximo de tempo para cada pino de entrada.
- *Max Capacitance*: Existe um limite máximo de capacitância que um pino de saída pode ter.
- *Max Fanout*: Existe um limite de pinos de entrada que um pino de saída pode conectar.
- *Max Length*: Cada fio deve ter um comprimento máximo em cada camada.

2.3.3 *Placement*

O próximo passo é colocar todas as células que foram mapeadas na síntese lógica dentro da estrutura do *core*, essa etapa é chamada de *placement*. Além das células, em alguns tipos de processos, também são inseridas as *well taps*, células especiais para prevenção de *latch-ups*, e *end caps*, células que previnem violações de regras de projeto.

Existem dois métodos tradicionais de realização de *placement*, o *timing driven* e o *congestion driven*:

- *Timing Driven Placement*: Baseado em otimizações de *timing* e consumo.
- *Congestion Driven Placement*: Aumenta o distanciamento das células, o que ocasiona maior roteamento entre elas.

A escolha entre os métodos é baseada na experiência do projetista em análise de resultados posteriores ao *placement*. O fluxo de síntese física é bastante iterativo. Problemas que são diagnosticados nas etapas futuras podem ter influências de como foram colocadas as células no projeto. Ou seja, caso seja entendido que deve ser modificado o *placement*, o fluxo deve retornar para corrigir os problemas.

2.3.4 *Clock Tree Synthesis*

A árvore de clock ou *clock tree synthesis*(*CTS*) é responsável pela distribuição e balanceamento do clock desde a fonte até as células sequenciais em todo o circuito. Essa distribuição é feita por meio das trilhas de metais que ligam os componentes.

A ferramenta utiliza a análise de *timing* para checar se os tempos do *clock* estão corretos, diminuindo os caminhos e inserindo inversores ou *buffers*. Assim, a ferramenta consegue diminuir o *skew* e a latência, deixando o circuito mais rápido.

Existem buffers e inversores especiais, feitos para a árvore de *clock*. Eles são projetados para serem balanceados, isso significa que possuem um *transition time*, tanto de subida, quando de descida parecidos. Isso acarreta em menores problemas de *skew* ao longo do circuito.

Alguns efeitos ruins que a árvore pode causar no projeto são:

- Caso o projeto necessite de muitos *buffers* e inversores, pode ocasionar problemas de congestionamento.
- Como existe um chaveamento maior do *clock*, isso pode ocasionar problemas de *crosstalk* (interferência de sinais entre trilhas de metais vizinhas).

Para corrigir os problemas de *crosstalk* são utilizados fios mais largos, o que diminui a densidade de corrente e consequentemente menos problemas eletromagnéticos podem ocorrer.

Para evitar que sejam colocados muitos *buffers* e inversores no circuito, a ferramenta pode tentar encurtar os caminhos de metais dentro do circuito, acelerando dessa forma a chegada do sinal.

2.3.5 Roteamento

O roteamento acontece após a *CTS*, nela os caminhos de metais são consolidades de forma precisa, isso inclui as conexões entre acélulas, com os pinos dos *IPs*, os pinos dos *PADs* e dos blocos presentes no *core*.

São inseridas vias para fazer as interconexões entre os metais afim de utilizar o máximo da pastilha do substrato. Deve ser informado, a ferramenta, as regras de

design, existe uma parte do fluxo de verificação física destinada a uma checagem mais elaborada dessas regras, onde serão carregados arquivos específicos. A priori, durante a implementação essa checagem é eventual, mas que devem ser corrigidas caso exista alguma violação.

Alguns dos objetivos do roteamento podem ser descritos como:

- Estabelecer a conectividade com o mínimo de vias possíveis e fios com caminhos mais curtos.
- Atingir as *constraints* de timing.
- Não existir violações de regra de projeto
- Completar o roteamento dentro da área do *core*.

Na maioria das ferramentas o roteamento é dividido da seguinte forma:

- *Global Routing*: as regiões que serão roteadas são divididas em setores em formato de retangulos chamados de *global cells*. A partir disso, são feitas otimizações de espaço afim de diminuir o tamanho dos fios nas próximas etapas.
- *Detail Routing*: utilizando o roteamento feito pelo global routing, são feitas várias iterações apara checar o *DRC* e corrigir as violações que foram ocasionadas.

Ainda nessa etapa, são inseridas células especiais para correção de regras de projeto estabelecidos durante o processo de manufatura do *chip*. Existem as *decap cell*, as *fillers cells* e os *metal fills*.

- *Fillers cells*: Utilizados para dar continuidade e preencher os *gaps* entre as *standard cells* nas colunas. Elas são *physical only*, dado que não possuem finalidade alguma na lógica do projeto. Para inserí-las basta ligá-las à malha de alimentação.
- *De-cap cells*: É possível inserir as *de-caps* afim de diminuir a queda de tensão ao longo do circuito. Elas guardam energia para abastecer o circuito caso seja necessário.
- *Metal fills*: Também conhecidos como *dummy-fills*. São colocados por último, afim de uniformizar a densidade de metal. Sua inserção acontece pois a *foundry* informa que deve existir uma densidade específica de metal dentro do *chip* para evitar problemas durante o processo de fabricação.

Após roteado o sinal e corrigidas as violações, é gerada uma nova *netlist*. São comparados pontos dessa nova *netlist* com pontos da *netlist* gerada após a síntese lógica, a

isso é dado o nome de Checagem de Equivalência Lógica (*LEC*). O resultado da comparação deve ser sem erros, caso ocorra algum é dado como um erro lógico entre as *netlists* e deve ser checado o ponto onde foi ocasionado.

Além da *netlist*, também são gerados os arquivos:

- *.def* (*Design Exchange Format*): Representação compactrada do projeto, internamente possui a sua representação física e algumas informações adicionais, como as *contraints*.
- *.gds* (*Graphic Design System*): É a representação física final do layout. Também pode ser gerado no formato *OASIS* (*Open Artwork System Interchange Standard*). Esse é o arquivo requisitado pela *foundry* para a fabricação do *chip*.

2.4 Verificação Física

Após obter os arquivos gerados na implementação do *chip*, se dá início à verificação física. Ela consiste em analisar o que foi implementado com um maior detalhamento.

2.4.1 Design Rule Check

Nessa etapa da verificação de *DRC* é feita a checagem com todas as regras que a tecnologia do *PDK* fornece. Ela pode ocorrer em qualquer etapa da implementação, no entanto, em alguns fluxos, também é checado após a inserção do *metal fill*, pois além deve ser checada as regras de densidade de metais.

As principais regras analisadas são:

- Mínimo espaçamento e largura dos metais e vias.
- Vias desalinhadas.
- Curtos circuitos e circuitos abertos.
- Densidade de metais.
- Área mínima dos metais.

Após a checagem, são gerados relatórios de qual regras foram violadas e em quais locais dentro do circuito elas estão. Basta, após a análise, voltar ao fluxo da implementação e corrigí-las manualmente. Também é possível guiar a ferramenta para rerotear o local violado.

2.4.2 *Layout vs Schematic* (*LVS*)

O *layout vs schematic* é utilizado para checar se a lógica presente no layout gerado é a mesma lógica da *netlist* final do circuito. Para isso, deve ser extraído do layout, um formato de *netlist* que será comparada com a outra *netlist*.

Para se implementar a checagem de *LVS* é necessário que a ferramenta leia alguns arquivos:

- *Layout* do projeto: geralmente é o *GDS*, nele será feita a extração.
- Um *deck* de regras: um arquivo com instruções para guiar a ferramenta. Também contém as definições das camadas de metais utilizadas para a extração.
- Arquivo de equivalência: consiste de pares de células, uma da *netlist* do *layout* e outra da *netlist* do esquemático.

Problemas que ocasionam erros de comparação durante o *LVS*:

- Circuitos Aberto: quando existem geometrias do mesmo caminho em aberto.
- Curto circuitos: quando o caminho está interceptado por uma geometria de outra net.
- Componentes faltando.
- *Global nets* não definidas: quando não é indicada à ferramenta qual o pino de alimentação e a que *net* ele está conectado na malha de alimentação do circuito.

2.5 *PDK*: *SkyWater130nm*

A *Google* junto com a empresa *SkyWater* decidiram desenvolver um *PDK open source* em um processo de 130nm. A *SkyWater* é uma *foundry* localizada em Minesota, nos Estados Unidos.

Em julho de 2020 foi lançado uma das versões do *PDK*. A *foundry* indica que, no momento do lançamento da versão, o projeto ainda não está consolidado e que deve ser utilizado somente para *chips* de teste.

Algumas características do *PDK*:

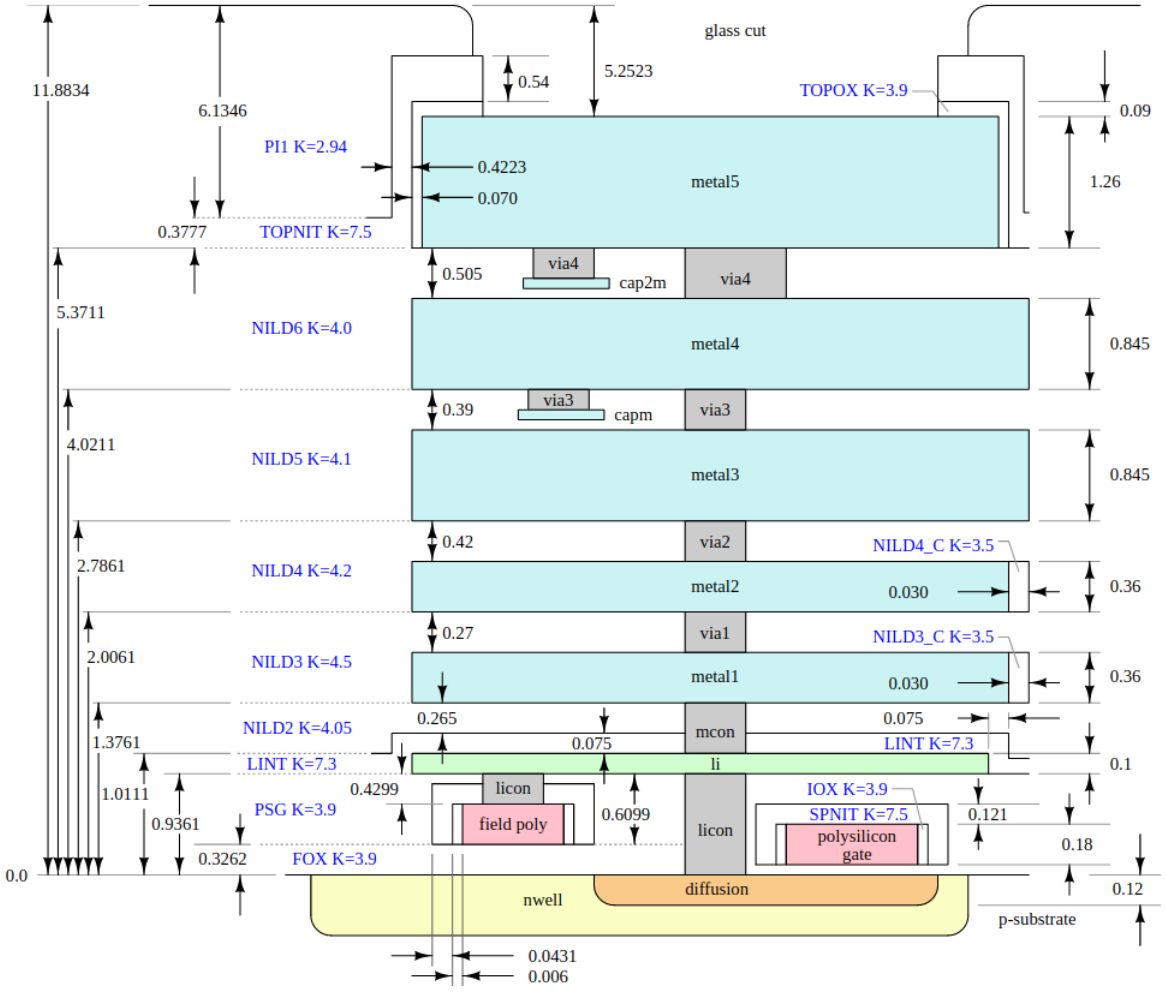
- São utilizados pela tecnologia 5 camadas de metais interconectadas.
- Exites várias bibliotecas de células: *high density*, *high speed*, células de *IO*.

- Existem células especiais para projetar memórias.
- Documentação bem elaborada.

Na imagem abaixo é possível ver o processo de manufatura do *chip*, camadas de *BEOL* (*Back-End of Line*) e *FEOL* (*Front-End of Line*).

- *BEOL*: são as camadas de metais disponíveis para o projetista. Geralmente é a maior parte do *chip*. No caso do *PDK SkyWater*, existem 5 camadas de metais e uma de interconexão chamada *li1*.
- *FEOL*: é encapsulada e não fica disponível para o projetista alterar. Está relacionada ao processo de manufatura. Representada na Figura 3 pelas camadas de difusão na parte mais baixa.

Figura 3 – Disposição dos metais e vias do *PDK SkyWater* e suas regras de tecnologia.



Fonte: Documentação do PDK SkyWater 130nm

2.6 *XMCPROCv0*

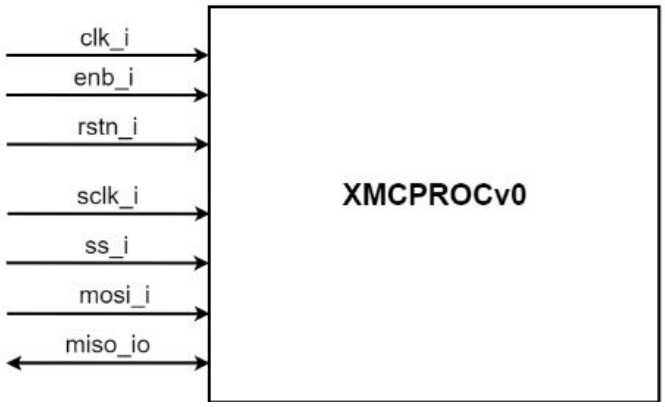
O *XMCPROCv0* é um *SoC* (*System on Chip*), ele é um coprocessador criptográfico *AES* (*Advanced Encryption Standard*). Seu circuito foi dividido em 4 blocos o *xaes*, *xmemc*, *xspis* e *xcrc_prl*, além deles também existe uma memória *SRAM*.

O *RTL* do *chip* for desenvolvido e verificado pela equipe de *frontend* do Laboratório de Excelência em Microeletrônica do Nordeste.

Nele existem 6 entradas e 1 entrada-saída, duas delas são de *clocks* assíncronos, o referente ao *SPI* e outra referente ao restante do *chip*, suas frequências são de 50 Mhz e 25 Mhz respectivamente.

O *chip* foi encapsulado em um módulo *wrapper*. Esse módulo deve integrar o *core* do *chip*, junto com os *PADs* (células especiais de proteção utilizadas para conectar o *chip* com o mundo externo).

Figura 4 – Estrutura de bloco do *XMCPROCv0*.



Fonte: *Blockguide* do *XMCPROCv0*

3 Desenvolvimento em Ferramentas *Open Source*

Existem diversas ferramenta *EDA* no mercado. Assim como o *PDK* da tecnologia, a maioria dessas ferramentas são pagas e seus valores são bastante altos, de forma que projetistas independente quase nunca podem arcar com os seus custos.

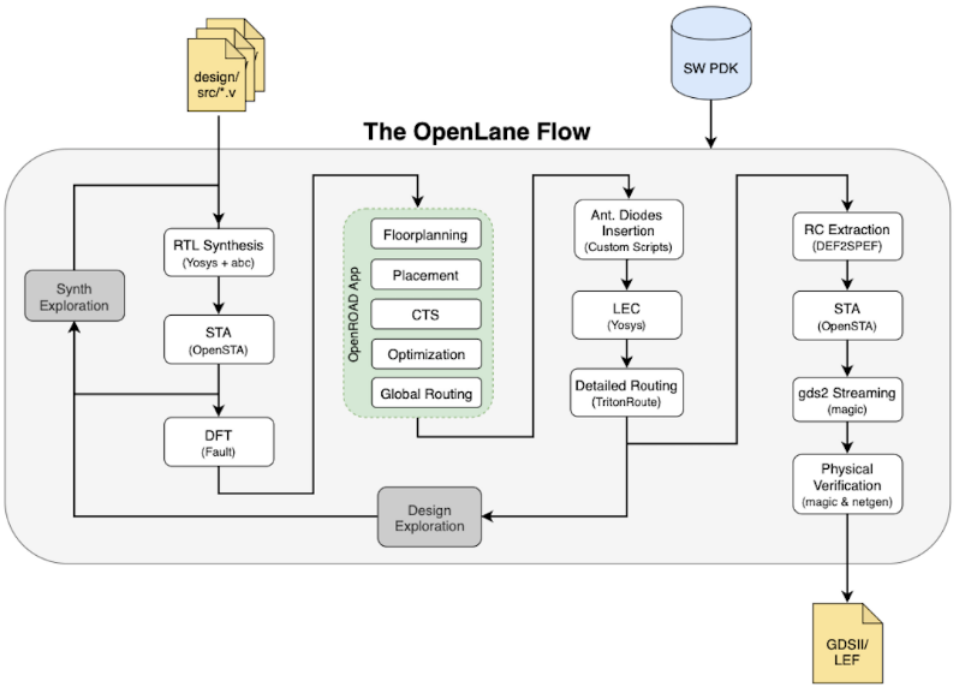
Há alguns anos, algumas empresas sem fins lucrativos iniciaram o desenvolvimento de *PDKs* e ferramentas *open source*.

3.1 Projeto *Openlane*

Para a realização do projeto foram utilizados os *scripts* e ferramentas de um projeto feito na Universidade da California em Santa Cruz, chamado *OpenLane*. Ele promete entregar um fluxo contínuo e automatizado *RTL-GDSII* e para isso utiliza vários *scripts* e ferramentas *open source*.

Abaixo é possível visualizar uma possível arquitetura do fluxo do *Openlane*. Existem outras ferramentas que pode ser executadas para a mesma função.

Figura 5 – Arquitetura do *Openlane*.



Fonte: Documentação do *Openlane*.

O fluxo que será executado no projeto é dado pelas ferramentas:

- *Yosys* - Síntese lógica
- *OpenSTA* - Analise de *Timing*
- *OpenROAD* - Fluxo de *Place & Route*
- *OpenRCX* - Extração de Parasitas
- *Magic* - Verificação de *DRC* (*Design Rule Check*) e geração de *GDSII*
- *Netgen* - Verificação de *LVS* (*Layout vs Schematic*)

O *Openlane* existe em um repositório no *github*. O projeto possui uma documentação detalhada onde explica o seu processo de instalação. O mesmo já instala o repositório do *PDK skyWater*, que será utilizado nas próximas etapas. Também é informado como iniciar um novo projeto de *chip*. Além disso, são explicadas as funcionalidades das variáveis que habilitam os *scripts* que serão rodados no fluxo, dentre outras informações. Importante destacar que o *Openlane* somente é executado pelo terminal de comandos, não possui interface gráfica.

3.1.1 Preparação

Antes de iniciar o desenvolvimento do *chip*, foi necessário inserir dentro do *wrapper* do *XMCPROv0* os *PADs* utilizados. Para cada entrada e saída do *XMCPROCv0* foi associado um *PAD GPIO*, disponível dentro do *PDK Skywater*.

Após isso, com o auxílio da documentação do *Openlane*, foi criado um novo diretório para o projeto dentro da estrutura do *Openlane*. Com o diretório criado, os arquivos *RTL* do *chip* foram movidos para dentro do diretório '*src* estabelecido no Openlane.

A configuração do fluxo é pré estabelecida, basta apenas modificar, dentro do arquivo *config.tcl*, o que for necessário para o projeto que se está projetando.

Dentro do *script config.tcl* foram adicionadas nas variáveis (*EXTRA_LIBS, EXTRA_LEFS, EXTRA_GDS_FILES*) os diretórios dos arquivos de leitura das bibliotecas, *lefs* e *gds* da memória e das *IO cells* (contém os *PADs* e *Fillers*).

Para que a síntese lógica ocorresse, também foi necessário carregar os arquivos *verilog* das bibliotecas. Dessa forma foi adicionada a variável (*VERILOG_FILES_BLACKBOX*) com os arquivos *verilog* das células e da memória. Além disso, seguindo a documentação, também foi possível inserir as variáveis para o *sdc* do *chip*.

Os modos de operação das bibliotecas carregadas foram todos somente para *typical*:

- Células Padrão: *tt_025C_1v80*.
- Memória *SRAM*: *TT_1p8V_25C*.
- Células IO: *tt_025C_1v80*.

3.2 Síntese Lógica

Configurado os diretórios dos arquivos que serão lidos, foram alteradas as variáveis que controlam a síntese lógica feita pela ferramenta *Yosys*.

O *Yosys* é uma ferramenta *EDA open source*. Durante a leitura do *RTL* o *Yosys* não consegue ler algumas funções estabelecidas na linguagem de hardware system verilog. Então, foi necessário converter o código, utilizando a ferramenta que o próprio desenvolvedor do *Yosys* recomenda, o *SystemVerilog to Verilog* (*sv2v*).

Uma das variáveis do fluxo foi alterada para que o *Yosys* interprete o código de forma *FLAT*, já que os blocos estão estruturados de forma *flat*.

Dentro do *Yosys* é possível executar uma ferramenta de análise de *timing* chamada *ABC*, ela é responsável pela checagem de *timing* e mapeamento das células durante essa etapa. É possível alterar a estratégia de resolução, caso se queira uma otimização para *DELAY* ou *ÁREA*. Foi estabelecida a variável para *DELAY*, assim a ferramenta assegura que os caminhos críticos serão corrigidos com maior facilidade.

Após a execução do comando *run_synthesis* no prompt do *Openlane*, foi gerada a *netlist*. Durante esse comando além de fazer a síntese lógica, ao final é feita uma análise de *timing* pela ferramenta *OpenSTA*.

Foi utilizada a ferramenta *OpenSTA* para checar o *timing* dos caminhos críticos. Como a ferramenta não dispõe de comandos para gerar relatórios simplificados de *path groups*, foi criado duas *procs* em linguagem *tcl* para o *script or_sta.tcl*, presente na pasta de *scripts* do *openlane*, que resume os *path groups* (*in2reg*, *reg2reg*, *reg2out*, *in2out*), total de violações e de caminhos.

Figura 6 – Relatório de *timing* após a síntese do *XMCPROCv0* por meio da ferramenta *OpenSTA*

```
                                Setup Mode:
 (reg2reg)              (reg2out)              (in2reg)               (in2out)
TNS(ns):0.0000          TNS(ns):0.0000         TNS(ns):0.0000         TNS(ns):0.0000
WNS(ns):3.9565          WNS(ns):18.5939        WNS(ns):17.3754        WNS(ns):0.0000
Total Paths: 3667       Total Paths: 1         Total Paths: 103       Total Paths: 0
Total Violations : 0    Total Violations : 0   Total Violations : 0   Total Violations : 0

                                Hold Mode:
 (reg2reg)              (reg2out)              (in2reg)               (in2out)
TNS(ns):0.0000          TNS(ns):0.0000         TNS(ns):0.0000         TNS(ns):0.0000
WNS(ns):0.1283          WNS(ns):21.3640        WNS(ns):0.8754         WNS(ns):0.0000
Total Paths: 3667       Total Paths: 1         Total Paths: 103       Total Paths: 0
Total Violations : 0    Total Violations : 0   Total Violations : 0   Total Violations : 0
```

Fonte: Autor

Também foi checado por meio do *OpenSTA* o relatório de violações de *DRV* de *max capacitance*, *max fanout* e *max slew*. Não existe relatório resumido para *DRV*, foi montado um *script* para tornar a visualização mais prática.

Figura 7 – Relatório de *DRV* após a síntese do *XMCPROCv0* por meio da ferramenta *OpenSTA*

```
Max Slew Violations: 830
Max Capacitance Violations: 8
Max Fanout Violations: 0
```

Fonte: Autor

Foram reportadas diversas violações de *skew* e algumas de *max capacitance*, elas serão corrigidas durante as próximas etapas da implementação.

3.3 Implementação

Para o desenvolvimento da implementação o *openlane* utiliza a ferramenta *OpenROAD* na maior parte do fluxo, utilizando também o *OpenSTA* para análises de *timing*.

3.3.1 Floorplan

Utilizando a *netlist* gerada e os arquivos das bibliotecas, foram definidas as especificações de tamanho do *chip* como: tamanho do *die* 1550x1650 e do *core* 1050x1150, existem variáveis específicas para isso. Também foi inserida a memória, o seu posicionamento é feito de forma padrão, atentando-se para o local dos seus pinos na *macro*.

Os *scripts* do *Openlane* ainda não estão muito adaptados à inserção de *PADs*, não existem variáveis que facilitem o processo. Além disso, os scripts automatizados não estão

chamando as bibliotecas dos *PADs* para análise de *timing*. Uma outra maneira de inserí-los é utilizando alguns comandos próprios do *OpenROAD*.

Existe um comando dentro do *OpenROAD* chamado *ICeWall*. Esse comando é capaz de fazer o *floorplan* do *chip*, são passados alguns arquivos com informações de tamanho do *DIE* e do *CORE*, o posicionamento em que os *PADs* serão colocados e deve-se associar as instâncias dos *PADs* aos sinais dos pinos que farão a conexão com a lógica do circuito.

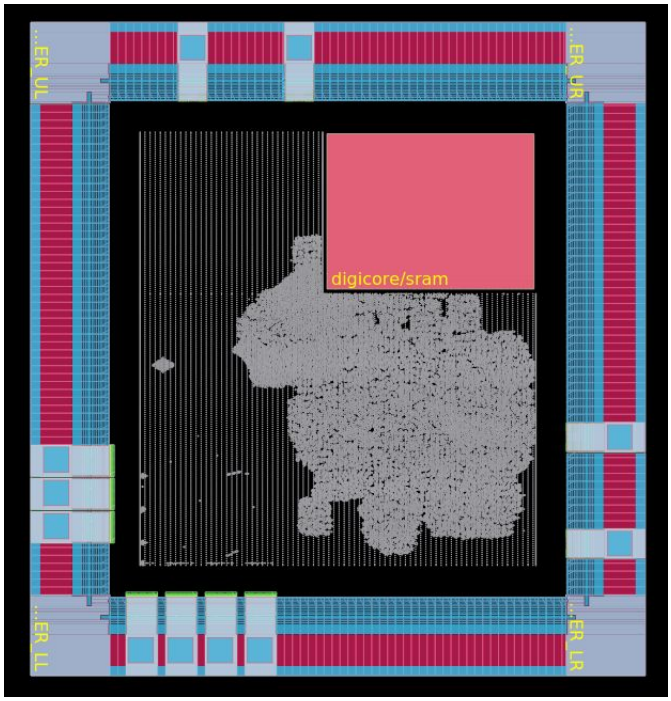
Dessa forma, foram feitas alterações dentro do *script or_floorplan.tcl* da pasta do *OpenROAD* (dentro do diretório do *Openlane*) com os comandos e arquivos necessários para a criação dos *PADs*. Também foi alterado o carregamento das bibliotecas para que fossem carregadas corretamente dentro do *OpenROAD*.

Após fazer todas as alterações, foi rodado o comando *run_floorplan* no terminal do *Openlane*. Ao final, é gerado alguns arquivos no diretório de resultados, dentre eles o *.def*.

O *Openlane* não possibilita a visualização do *chip* de forma clara, existe uma maneira de gerar o print do *DEF* por meio da ferramenta *KLayout*. No entanto, afim de se obter uma visualização mais elaborada, foi instalado o *OpenROAD* e carregado o *.def* na sua interface.

Na Figura 11 é possível observar o *floorplan* do *XMCPROCv0*.

Figura 8 – Print do *floorplan* do XMCPROCv0 a partir da interface da ferramenta OpenROAD



Fonte: Autor

Como pode ser visto, foram colocadas células dentro do *core*. No entanto, as células ainda não estão legalizadas.

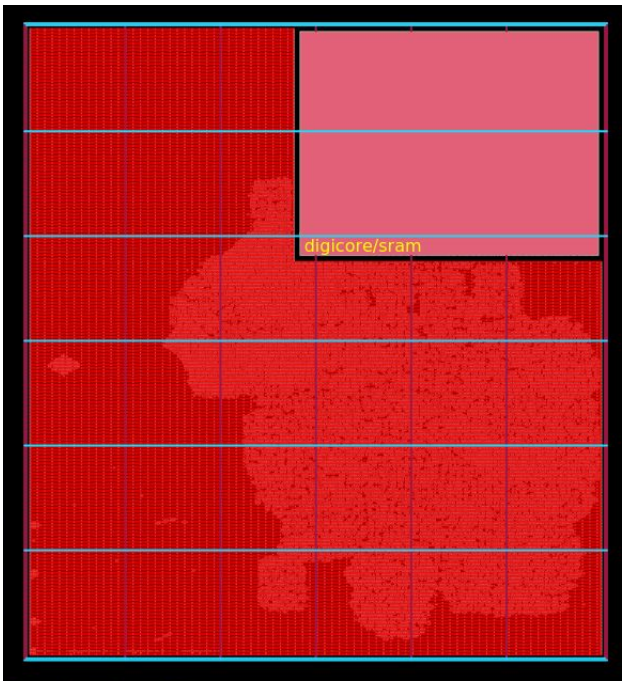
3.3.2 *Powerplan*

Para gerar o *powerplan* (malha de alimentação), o *Openlane* possui um *script* chamado *common_pdn.tcl*. A partir do *config.tcl* é possível alterar algumas variáveis para auxiliar na construção da malha, como o *ring* ao redor do *core*, os comprimentos dos fios, entre outras. Foi necessário alterar dentro do *script common_pdn.tcl* para inserir os pinos de *power* da memória e dos *PADs* nas nets globais.

A malha de alimentação possui um anel ao redor do *core* em uma tensão de 1.8V que alimenta as células. Os *PADs* são alimentdos por uma outra alimentação, também de 1.8V.

O *powerplan* também é gerado ainda durante o comando *run_floorplan* do *Openlane*. A figura abaixo mostra a estrutura do chip com a malha de alimentação.

Figura 9 – Estrutura do *floorplan* do *XMCPROCv0* a partir da interface da ferramenta *OpenROAD*



Fonte: Autor

É possível observar os aneis ao redor do *core*. Por motivos desconhecidos, a ferramenta não conseguiu criar o anel ao redor da memória mesmo estão configurada para isso, no entanto, ainda assim as *nets* de *power* foram conectadas a ela. Na sua estrutura,

existe um *blockage* até o metal 4, ou seja, é impedido de se criar trilhas. O metal 5 é possível, pois internamente a memória não possui metais dessa camada.

3.3.3 *Placement*

O comando do *Openlane* para inserir e legalizar as células do *chip* é o *run_placement*, existem variáveis para habilitar que seja executado com otimizações dirigidas para rotabilidade ou para *timing*. Foi estabelecido uma otimização para *timing*. Também foi alterada para que o *placement* fosse feito com uma densidade máxima de células de 80%. Além disso, por padrão o *Openlane* muda o tamanho de algumas células para resolver problemas de *skew*.

Após rodar todo o *placement*, não foram geradas violações de *timing*, nem de *DRV*. O relatório de *timing* analisado pela ferramenta *OpenSTA* e estrutura do *chip* com as células colocadas:

Figura 10 – Relatório de *timing* após o *placement* do *XMCPROCv0* por meio da ferramenta *OpenSTA*

```
                                Setup Mode:
 (reg2reg)              (reg2out)              (in2reg)               (in2out)
TNS(ns):0.0000         TNS(ns):0.0000         TNS(ns):0.0000         TNS(ns):0.0000
WNS(ns):3.9227         WNS(ns):18.4175        WNS(ns):17.6694        WNS(ns):0.0000
Total Paths: 3667      Total Paths: 1         Total Paths: 103       Total Paths: 0
Total Violations : 0   Total Violations : 0   Total Violations : 0   Total Violations : 0
                                Hold Mode:
 (reg2reg)              (reg2out)              (in2reg)               (in2out)
TNS(ns):0.0000         TNS(ns):0.0000         TNS(ns):0.0000         TNS(ns):0.0000
WNS(ns):0.1502         WNS(ns):21.4871        WNS(ns):0.8713         WNS(ns):0.0000
Total Paths: 3667      Total Paths: 1         Total Paths: 103       Total Paths: 0
Total Violations : 0   Total Violations : 0   Total Violations : 0   Total Violations : 0
```

Fonte: Autor

3.3.4 Árvore de *Clock*

A configuração da síntese da árvore de *clock* é feita por meio do comando *run_cts*. As configurações, como os *clocks* balanceados, algumas otimizações, divisões em *clusters* e a análise de *timing* por meio do *OpenSTA*, também são rodadas por padrão.

Os relatórios de *timing* foram iguais aos do *placement*.

As trilhas do roteamento da *CTS* somente são feitas após a etapa de roteamento.

3.3.5 Roteamento

As trilhas das *nets* de dados e *clock* são criadas durante o roteamento com o comando *run_route*. Novamente, o *Openlane* utiliza o *OpenROAD* por baixo. No *OpenROAD*, o

roteamento é feito em duas etapas *global route* (são feitas estimações de parasitas) e *detail route* em que violações de DRC são corrigidas utilizando o algoritmo chamado *tritonRoute*.

Durante o roteamento, algumas células com nomeclatura "*sky130_fd_sc_hd___fa_ **"estavam gerando erros de acesso aos pinos. Para resolver esse problema foi inserido ela no conjunto de células "*set_dont_use*", de forma que não fossem inserida na *netlist* durante a síntese lógica.

Abaixo está um *print* do erro:

Figura 11 – Erro ocorrido durante roteamento.

```
[ERROR DRT-0073] no ap for _1058_/A
terminate called after throwing an instance of 'std::runtime_error'
  what():  DRT-0073
```

Fonte: Autor

O relatório de violações de *DRC* utilizando o algorítmo padrão do *tritonRoute*, gerado após o roteamento:

Figura 12 – Violações corrigidas durante a 6ª iteração.

```
[INFO DRT-0195] Start 6th optimization iteration.
    Completing 10% with 4 violations.
    elapsed time = 00:00:00, memory = 1497.32 (MB).
    Completing 20% with 4 violations.
    elapsed time = 00:00:00, memory = 1497.32 (MB).
    Completing 30% with 4 violations.
    elapsed time = 00:00:00, memory = 1497.32 (MB).
    Completing 40% with 4 violations.
    elapsed time = 00:00:00, memory = 1497.32 (MB).
    Completing 50% with 4 violations.
    elapsed time = 00:00:00, memory = 1497.32 (MB).
    Completing 60% with 4 violations.
    elapsed time = 00:00:00, memory = 1497.32 (MB).
    Completing 70% with 4 violations.
    elapsed time = 00:00:00, memory = 1497.32 (MB).
    Completing 80% with 3 violations.
    elapsed time = 00:00:00, memory = 1497.32 (MB).
    Completing 90% with 3 violations.
    elapsed time = 00:00:00, memory = 1497.32 (MB).
    Completing 100% with 0 violations.
    elapsed time = 00:00:00, memory = 1497.32 (MB).
[INFO DRT-0199]   Number of violations = 0.
```

Fonte: Autor

Após corrigidas as violações ocorre a extração de parasitas do circuito. Por padrão o *Openlane* utiliza a ferramenta *OpenRCX* integrada ao *OpenROAD*. Após isso foi analisado o *timing* e *DRV* a partir do *OpenSTA*, não resultaram violações:

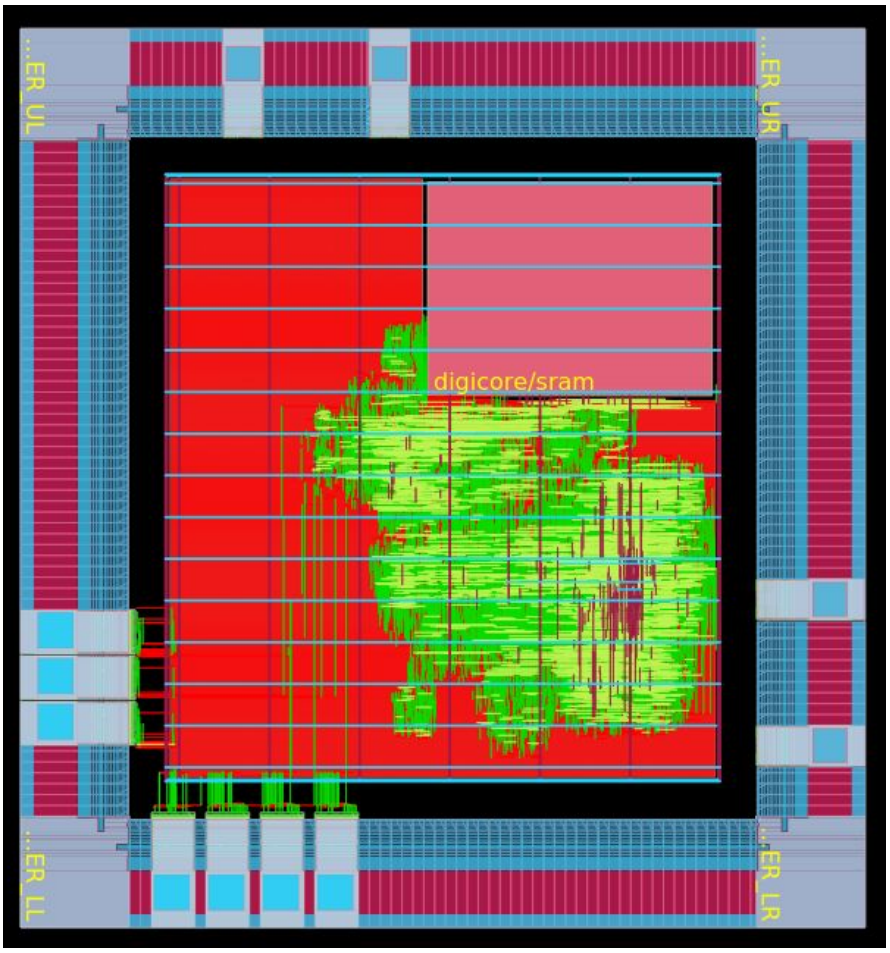
```
                                Setup Mode:
 (reg2reg)              (reg2out)              (in2reg)               (in2out)
TNS(ns):0.0000         TNS(ns):0.0000         TNS(ns):0.0000         TNS(ns):0.0000
WNS(ns):3.6542         WNS(ns):17.1512        WNS(ns):17.2121        WNS(ns):0.0000
Total Paths: 3667      Total Paths: 1         Total Paths: 103       Total Paths: 0
Total Violations : 0   Total Violations : 0   Total Violations : 0   Total Violations : 0

                                Hold Mode:
 (reg2reg)              (reg2out)              (in2reg)               (in2out)
TNS(ns):0.0000         TNS(ns):0.0000         TNS(ns):0.0000         TNS(ns):0.0000
WNS(ns):0.1283         WNS(ns):21.3640        WNS(ns):0.8754         WNS(ns):0.0000
Total Paths: 3667      Total Paths: 1         Total Paths: 103       Total Paths: 0
Total Violations : 0   Total Violations : 0   Total Violations : 0   Total Violations : 0
```

Figura 13 – Relatório de *timing*, após o roteamento, feito pela ferramenta *OpenSTA*.

Por fim, o resultado do roteamento:

Figura 14 – Estrutura do *chip* após o roteamento *OpenSTA*.



Fonte: Autor

4 Desenvolvimento em Ferramentas Comerciais

Também é possível realizar o desenvolvimento de *chips* utilizando o *PDK SkyWater*, em ferramentas comerciais. Afim de comparação entre os fluxos e ferramentas, foi decidido realizar o fluxo nas ferramentas da *Cadence*.

Cediado em um repositório github a parte do *PDK*, existem alguns scripts feitos por projetistas independentes, que realizam a junção dos arquivos necessários para carregar nas ferramentas comerciais.

Com isso foi possível obter além dos arquivos das bibliotecas, também os arquivos *capTbl* em modo *typical* (utilizado para extração de parasitas) e o arquivo de mapeamento da tecnologia, que será utilizado para geração do *GDS*.

As ferramentas comerciais utilizadas para a elaboração do *chip* foram:

- *Cadence*: *Genus* - Síntese Lógica
- *Cadence*: *Innovus* - *Place* & *Route*
- *Cadence*: *Tempus* - Análise de *Timing*

Até o momento ainda não existem os arquivos necessários para se carregar nas ferramentas da *Mentor* (utilizada para checagem de *DRC* e *LVS*) e nem no *QRC* da *Cadence*. A extração de parasitas foi feita pelo próprio *Innovus* com um esforço baixo.

4.1 Preparação

Para a realização do fluxo de síntese lógica e física, utilizou-se os scripts feitos pelo Laboratório de Excelência em Microeletrônica do Nordeste da Universidade Federal de Campina Grande. O script estava pronto para o desenvolvimento do *XMCPROCv0* em uma tecnologia de 180nm da *xfab*. Algumas alterações foram necessárias para se adequar ao *PDK* da SkyWater.

4.2 Síntese Lógica

Foram inseridos os diretórios das bibliotecas do *PDK SkyWater* dentro dos scripts do Genus, assim como o RTL feito em systemverilog do projeto. Também foi estabelecido

o mesmo arquivo de constraints utilizado durante o desenvolvimento nas ferramentas comerciais.

O próprio *Genus* faz análises de *timing*, fazendo otimizações na *netlist*. Após rodar os scripts, a *netlist* foi gerada sem nenhuma violação de *timing*. Abaixo é possível visualizar o relatório do pior slack e o de *drv*.

Figura 15 – Relatório da análise pelo Genus do pior slack de *timing*.

```
  Slack   |                Endpoint                    | Cost Group
----------------------------------------------------------------------
 3.316ns  |   digicore/xcrc_prl/xcrc16/crc16_reg_7/D   |   C2C
```

Fonte: Autor

Figura 16 – Relatório de *DRV* após a síntese lógica.

```
        Design Rule Check
        -----------------

Max_transition design rule: no violations.

Max_capacitance design rule: no violations.

Max_fanout design rule: no violations.
```

Fonte: Autor

4.3 Implementação

A implementação do *Place & Route* foi feita com o *Innovus* da *Cadence*. Foram carregadas as bibliotecas necessárias.

4.3.1 *Floorplan*

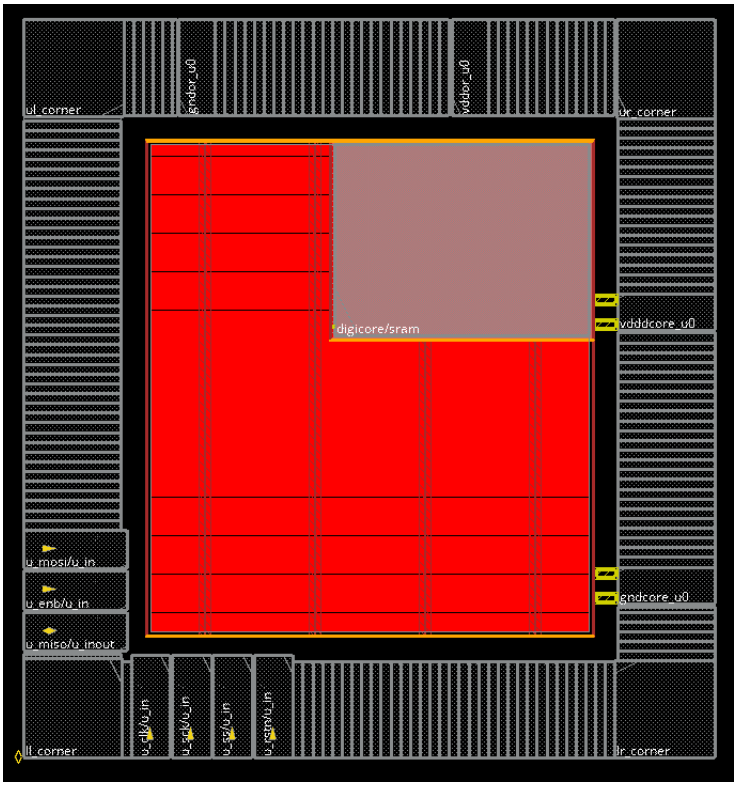
As especificações de tamanho do *DIE* e do *CORE* foram as mesmas utilizadas no desenvolvimento do *chip* com ferramentas *open-source*. Foram inseridas a memória *sram*, bem como as *tapwells* e *endcaps*, necessárias para o processo de manufatura. Os *PADs* também foram posicionados, inserindo as *fillers* entre eles.

4.3.1.1 *Powerplan*

A estrutura da malha de alimentação foi feita seguindo o mesmo princípio estabelecido durante o desenvolvimento em ferramentas *open source*. Um anel ao redor do *core* com tensão de alimentação de 1.8V. Foi criado, também, o anel ao redor da memória, foram conectados os pinos da memória ao anel criado. Os *PADs* que alimentam as células

foram conectados ao anel do *core*. Abaixo é possível visualizar o *floorplan* junto com a malha de alimentação.

Figura 17 – Estrutura do *floorplan* junto com a malha de alimenração feito pelo *Innovus*



Fonte: Autor

4.3.2 *Placement*

Durante o *placement* são feitas otimizações para correção de violações de *DRV* e de *timing*. O *Innovus* checa o *timing* de *setup* e *hold* dos caminhos. As violações de *hold* são corrigidas, já as violações de *setup* são deixadas para serem corrigidas durante as próximas etapas.

Ao final foi gerado o relatório de *timing* e *DRV*:

Figura 18 – Relatório de *timing* e *DRV* gerado pelo *Innovus* após a colocação das células

```
Setup views included:
 funcional_libset_typ_rc-typ

+--------------------+---------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg |reg2cgate| in2reg  | reg2out | in2out  | default |
+--------------------+---------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  3.839  |  3.839  | 14.680  | 17.821  | 17.791  |   N/A   |  0.000  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |    0    |   N/A   |    0    |
|          All Paths:|  3968   |  3859   |   98    |   44    |    1    |   N/A   |    0    |
+--------------------+---------+---------+---------+---------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |             Real              |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |      0 (0)       |   0.000    |     10 (10)      |
|   max_tran     |      0 (0)       |   0.000    |      0 (0)       |
|   max_fanout   |      0 (0)       |     0      |      0 (0)       |
|   max_length   |      0 (0)       |     0      |      0 (0)       |
+----------------+------------------+------------+------------------+
```

Fonte: Autor

Foram reportado 10 violações de *DRV* do tipo *max capacitance*, no entanto não são reais. Provavelmente serão corrigidas no relatório de *timing* das próximas etapas.

4.3.3 Árvore de *clock*

Foi configurada a síntese da árvore de *clock* com os *buffers* balanceados, para a *net* de *clock* foi utilizada o mesmo comprimento e largura das demais *nets* de roteamento.

Após rodado todos os comandos os relatórios de *timing* e *DRV* gerados foram:

Figura 19 – Relatório de *timing* e *DRV* gerado pelo *Innovus* após a síntese da árvore de clock

```
+--------------------+---------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg |reg2cgate| in2reg  | reg2out | in2out  | default |
+--------------------+---------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  3.950  |  3.950  | 14.113  | 17.316  | 17.836  |   N/A   |  0.000  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |    0    |   N/A   |    0    |
|          All Paths:|  3968   |  3859   |   98    |   44    |    1    |   N/A   |    0    |
+--------------------+---------+---------+---------+---------+---------+---------+---------+

+--------------------+---------+---------+---------+---------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg |reg2cgate| in2reg  | reg2out | in2out  | default |
+--------------------+---------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  0.085  |  0.085  |  0.712  |  0.703  | 22.156  |   N/A   |  0.000  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |    0    |   N/A   |    0    |
|          All Paths:|  3968   |  3859   |   98    |   44    |    1    |   N/A   |    0    |
+--------------------+---------+---------+---------+---------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |             Real              |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |      0 (0)       |   0.000    |      0 (0)       |
|   max_tran     |      0 (0)       |   0.000    |      0 (0)       |
|   max_fanout   |      0 (0)       |     0      |      0 (0)       |
|   max_length   |      0 (0)       |     0      |      0 (0)       |
+----------------+------------------+------------+------------------+
```

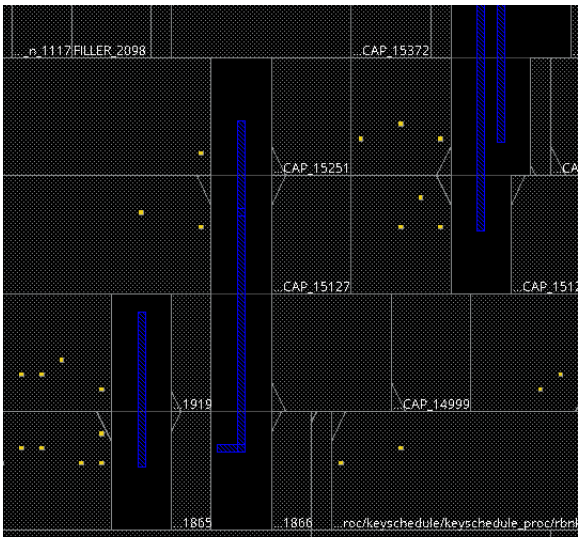
Fonte: Autor

4.3.4 Roteamento

Para o roteamento, foram inseridas as células de diodo para correção de violações de antenna.

Os pinos das células, na tecnologia da *SkyWater*, estão presentes na camada *li1*. Após o roteamento é necessário inserir as *fillers* para gerar continuidade no circuito, e as *decaps*, para evitar problemas de queda de tensão. No entanto, a ferramenta quando estava roteando, criava *nets* na camada *li1*, impedindo que células de *fillers* e decaps (que possuem metal em *li1*), fossem inseridas naquela região, como pode ser visto na figura abaixo:

Figura 20 – Problemas na inserção de *fillers* e *decaps* após o roteamento.



Fonte: Autor

A solução foi criar um *blockage* em toda a estrutura do *chip* antes do roteamento e deletá-lo após isso. Também foi inserido o comando *setNanoRouteMode -routeWithViaOnlyForStandardCellPin true* antes de rotear, para que a ferramenta conecte os pinos das células com vias, evitando assim roteamentos nas camadas mais baixas.

Após rodar todos os comandos do roteamento, o relatório de *timing* gerado:

Figura 21 – Relatório de *timing* e *DRV* gerado pelo *Innovus* após o roteamento

```
Setup views included:
 funcional_libset_typ_rc-typ

+--------------------+---------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg |reg2cgate| in2reg  | reg2out | in2out  | default |
+--------------------+---------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  3.636  |  3.636  | 13.626  | 17.375  | 17.819  |   N/A   |  0.000  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |    0    |   N/A   |    0    |
|          All Paths:|  3968   |  3859   |   98    |   44    |    1    |   N/A   |    0    |
+--------------------+---------+---------+---------+---------+---------+---------+---------+
|funcional_libset_typ_rc-typ
|                    |  3.636  |  3.636  | 13.626  | 17.375  | 17.819  |   N/A   |  0.000  |
|                    |  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |  0.000  |
|                    |    0    |    0    |    0    |    0    |    0    |   N/A   |    0    |
|                    |  3968   |  3859   |   98    |   44    |    1    |   N/A   |    0    |
+--------------------+---------+---------+---------+---------+---------+---------+---------+

+----------------+--------------------------------+------------------+
|                |             Real               |       Total      |
|    DRVs        +------------------+-------------+------------------|
|                |  Nr nets(terms)  | Worst Vio   |  Nr nets(terms)  |
+----------------+------------------+-------------+------------------+
|   max_cap      |      0 (0)       |   0.000     |      0 (0)       |
|   max_tran     |      0 (0)       |   0.000     |      0 (0)       |
|   max_fanout   |      0 (0)       |     0       |      0 (0)       |
|   max_length   |      0 (0)       |     0       |      0 (0)       |
+----------------+------------------+-------------+------------------+
```

Fonte: Autor

Figura 22 – Relatório de *timing* e *DRV* gerado pelo *Innovus* após o roteamento

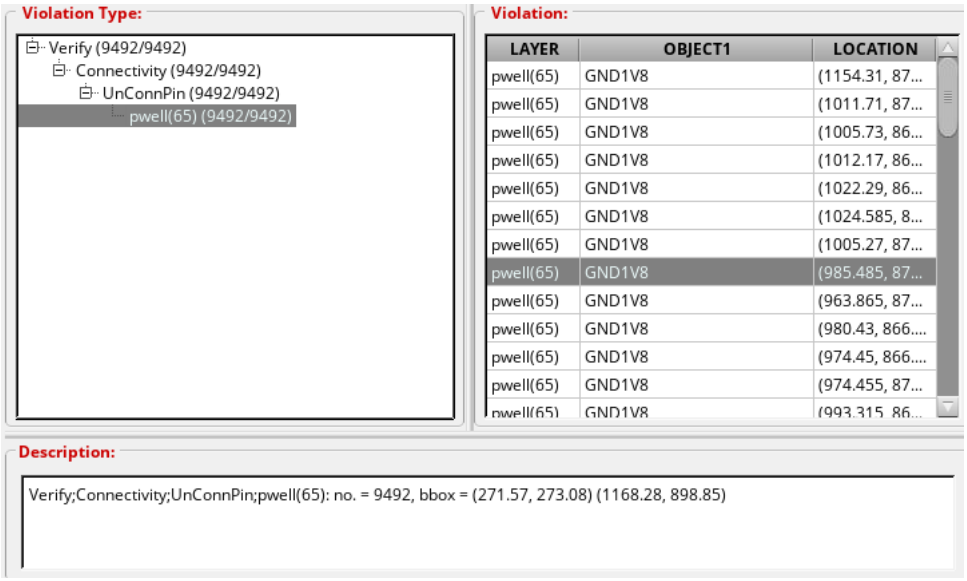
```
Hold  views included:
 funcional_libset_typ_rc-typ

+--------------------+---------+---------+---------+---------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg |reg2cgate| in2reg  | reg2out | in2out  | default |
+--------------------+---------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  0.084  |  0.084  |  0.705  |  0.698  | 22.177  |   N/A   |  0.000  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |    0    |   N/A   |    0    |
|          All Paths:|  3968   |  3859   |   98    |   44    |    1    |   N/A   |    0    |
+--------------------+---------+---------+---------+---------+---------+---------+---------+
|funcional_libset_typ_rc-typ
|                    |  0.084  |  0.084  |  0.705  |  0.698  | 22.177  |   N/A   |  0.000  |
|                    |  0.000  |  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |  0.000  |
|                    |    0    |    0    |    0    |    0    |    0    |   N/A   |    0    |
|                    |  3968   |  3859   |   98    |   44    |    1    |   N/A   |    0    |
+--------------------+---------+---------+---------+---------+---------+---------+---------+
```

Fonte: Autor

Após rodar o roteamento ainda são inseridas *decaps* e *fillers* para resolver problemas de connectividade. O *Innovus* disponhe de checagem de *DRC* por meio do comando *verify_drc*, além disso checou-se a connectividade (*verify_connectivity*), no caso de haver circuitos abertos e o *verifyPowerVia*, para checar se a malha criou as vias que alimentam o circuito corretamente. Apenas erros de conectividade foram reportados.

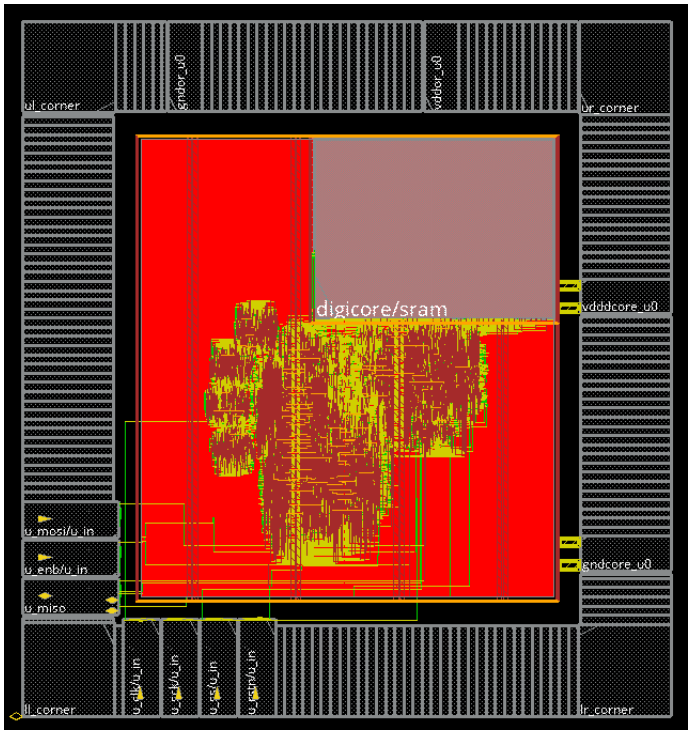
Figura 23 – Relatório de conectividade após o roteamento.



Fonte: Autor

No relatório não é informado o nome da violação, sabe-se apenas que é em uma camada interna chamada *pwell*. Decidiu-se esperar para a verificação física de *DRC*, pois será checado de forma mais detalhada.

Figura 24 – Estrutura do chip após o roteamento.



Fonte: Autor

5 Verificação Física e Análise Comparativa entre as Ferramentas

5.0.1 Checagem de Equivalência Lógica

Após a síntese lógica foi comparada a lógica da netlist gerada pelo Genus com a lógica do RTL por meio da ferramenta Conformal LEC da Cadence. O resultado deu equivalente como sugere o relatório abaixo.

Figura 25 – Relatório de consumo do projeto feito com ferramentas *open source*

```
================================================================================
Mapped points: SYSTEM class
--------------------------------------------------------------------------------
Mapped points     PI     PO     Z         Total
--------------------------------------------------------------------------------
Golden            24     12     24        60
--------------------------------------------------------------------------------
Revised           24     12     24        60
================================================================================
// Command: remodel -seq_merge
// Command: add compare points -all
// 12 compared points added to compare list
// Command: compare
================================================================================
Compared points      PO        Total
--------------------------------------------------------------------------------
Equivalent           12        12
================================================================================
```

Fonte: Autor

No projeto com ferramentas *open source*, o *Yosys* não consegue fazer a comparação entre o *RTL* e a *netlist*, somente entre duas *netlist*. Afim de checar se a *netlist* gerada pelo *Yosys* estava correta logicamente e se após o roteamento não ocasionou nenhum erro lógico, foram feitas as duas análises utilizando o *LEC* no *Genus*. O resultado foi equivalente em ambas, como mostra a imagem abaixo:

Figura 26 – Relatório da equivalência lógica entre a netlist e o RTL do projeto com ferramentas open source.

```
================================================================================
Mapped points: SYSTEM class
--------------------------------------------------------------------------------
Mapped points     PI     PO     Z         Total
--------------------------------------------------------------------------------
Golden            24     12     24        60
--------------------------------------------------------------------------------
Revised           24     12     24        60
================================================================================
// Command: remodel -seq_merge
// Command: add compare points -all
// 12 compared points added to compare list
// Command: compare
================================================================================
Compared points      PO        Total
--------------------------------------------------------------------------------
Equivalent           12        12
================================================================================
```

Fonte: Autor

5.1 Análise de Regras de Projeto

Como não existe arquivos para conseguir checar *DRC* em ferramentas comerciais, foi utilizada a ferramenta *Magic*, do projeto *Openlane*, para os dois projetos.

Tanto para o projeto com ferramentas open source como para o com ferramentas comerciais. O resultado final pode ser visto abaixo:

Figura 27 – Relatório de *DRC* do projeto com ferramentas *open source* utilizando o *Magic*.

```
Magic DRC Summary:
Violation Message: "Metal4 > 3um spacing to unrelated m4 < 0.4um (met4.5b)" found 66 times.
Violation Message: "Metal3 > 3um spacing to unrelated m3 < 0.4um (met3.3d)" found 47 times.
Violation Message: "All nwells must contain metal-connected N+ taps (nwell.4)" found 764 times.
Violation Message: "Metal1 > 3um spacing to unrelated m1 < 0.28um (met1.3b)" found 6 times.
Total Magic DRC Violations 883.
```

Fonte: Autor

Para o projeto com ferramentas comerciais, também ocasionaram as mesmas violações, no entanto com quantidades diferentes.

Figura 28 – Relatório de *DRC* do projeto com ferramentas comerciais utilizando o *Magic*.

```
Magic DRC Summary:
Violation Message: "Metal4 > 3um spacing to unrelated m4 < 0.4um (met4.5b)" found 66 times.
Violation Message: "Metal3 > 3um spacing to unrelated m3 < 0.4um (met3.3d)" found 26 times.
Violation Message: "All nwells must contain metal-connected N+ taps (nwell.4)" found 887 times.
Violation Message: "Metal1 > 3um spacing to unrelated m1 < 0.28um (met1.3b)" found 2 times.
Total Magic DRC Violations 981.
```

Fonte: Autor

5.2 *Layout vs Schematic*

Os dois relatórios de *LVS* tiveram violações.

Figura 29 – Relatório de *LVS* do projeto com ferramentas open source utilizando o *Netgen*.

```
Result: Netlists do not match.
Logging to file "/openLANE_flow/designs/digitop/runs/17-10_13-24/results/lvs/digitop.lvs.lef.log" disabled
LVS Done.
LVS reports:
    net count difference = 24
    device count difference = 0
    unmatched nets = 1968
    unmatched devices = 163
    unmatched pins = 0
    property failures = 0

Total errors = 2155
[ERROR]: There are LVS errors in the design according to Netgen LVS.
```

Fonte: Autor

Para o projeto com ferramentas comerciais, também ocasionaram violações.

Figura 30 – Relatório de *LVS* do projeto com ferramentas comerciais utilizando o *Netgen*.

```
Result: Netlists do not match.
Logging to file "/openLANE_flow/designs/digitop/runs/17-10_14-50/results/lvs/digitop.lvs.lef.log" disabled
LVS Done.
LVS reports:
    net count difference = 13
    device count difference = 0
    unmatched nets = 1452
    unmatched devices = 112
    unmatched pins = 0
    property failures = 0

Total errors = 1577
```

Fonte: Autor

5.3 *Static Timing Analysis*

Como pôde ser visto, análise de *timing* no projeto com ferramentas *opensource* foi feita durante todo o fluxo do *Openlane*, o roteamento não apresentou violações de *timing*.

Para o projeto utilizando ferramentas comerciais, utilizou-se o *Tempus* para a análise de *timing*. Para as duas soluções o relatório não gerou nenhuma violações de *timing*, nem violações de *DRV*.

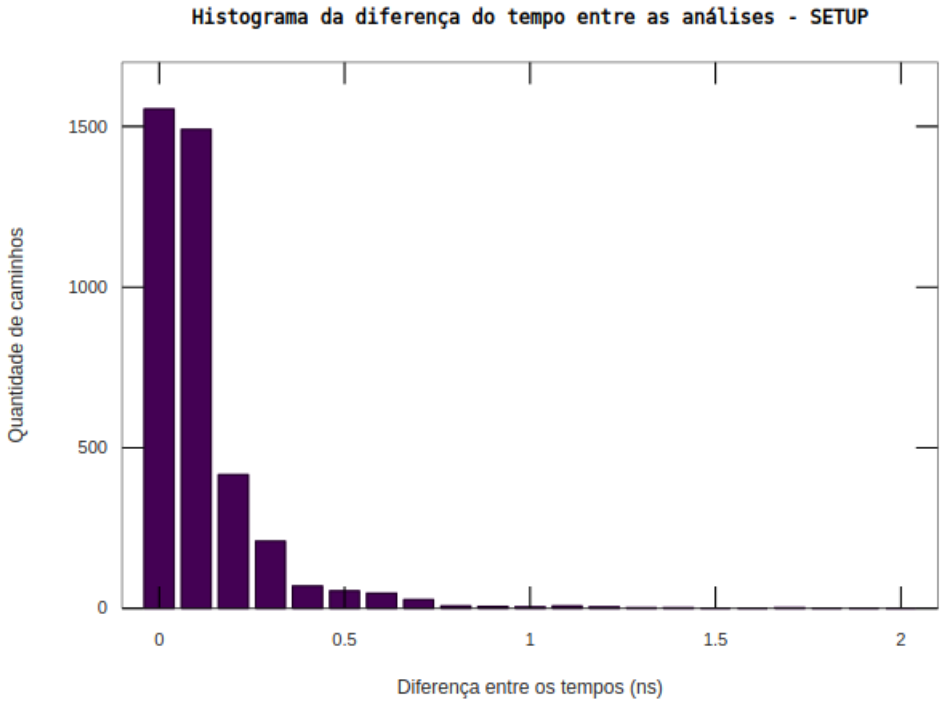
5.4 Análise Comparativa entre *OpenSTA* e *Tempus*

Foi comparado os resultados obtidos durante a análise de *timing* na ferramenta *Tempus* e na ferramenta *OpenSTA* com a mesma *netlist*. O objetivo é ver o quão distante estão os valores para os mesmos caminhos.

Inicialmente foram coletados todos os slacks dos caminhos de setup e todos os caminhos de *hold* encontrados pelo *Tempus*. A partir de um scripts em tcl, gerou-se um arquivo para ser carregado no OpenSTA e assim fosse possível coletar os slacks calculados pelo OpenSTA para os mesmos caminhos. Ao final, montou-se dois histogramas em valores absolutos, um para os caminhos de *hold* e outro para os de *setup*. A elaboração do histograma foi feito utilizando o *Octave*.

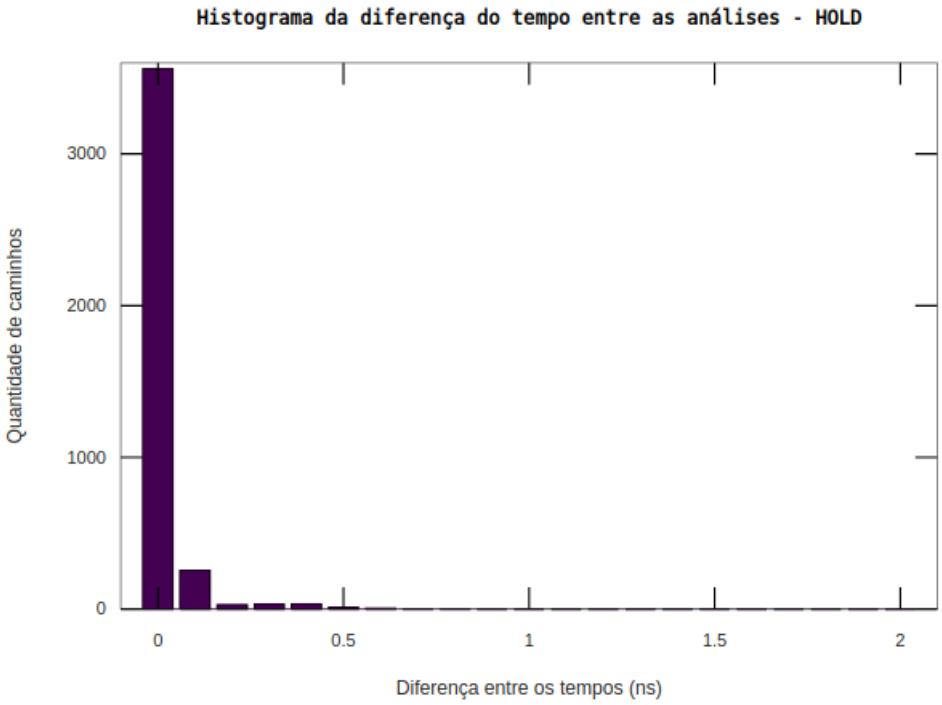
O resultado pode ser observado abaixo:

Figura 31 – Histograma da diferença entre os tempos dos *slacks* de *setup* gerados pelas duas ferramentas.



Fonte: Autor

Figura 32 – Histograma da diferença entre os tempos dos *slacks* de *hold* gerados pelas duas ferramentas.



Fonte: Autor

Pode ser observado que a maior quantidade de caminhos estão nas frequências mais baixas. Em números, 93% dos caminhos de setup estão abaixo de 0.3ns de diferença. Quanto aos de hold, 98% dos caminhos estão abaixo de 0.3ns. Pode-se concluir que existe uma semelhança entre os tempos.

6 Conclusão

A realização do trabalho permitiu o entendimento de um fluxo de projeto de circuito integrado, mais precisamente de um *SoC*, que acarreta a inserção de *PAD's* na estrutura do seu floorplan. Além disso, foi utilizado um *PDK* e ferramentas de código aberto, sendo assim, é possível trabalhar com uma tecnologia de alto nível sem que grandes custos estejam envolvidos.

Também foi possível o desenvolvimento do *chip* utilizando algumas ferramentas comerciais, isso possibilitou uma comparação entre ferramentas. Apesar do fluxo com ferramentas *open source* possuir menos utilitários que facilitam a interação, elas se apresentaram bastante úteis e confiáveis.

Algumas violações foram diagnosticados durante as etapas de verificação de *LVS* e *DRC* utilizando o fluxo do *Openlane* e não foi possível solucionar. Como o *PDK SkyWater* ainda não está preparado para ser carregado nas ferramentas do Calibre, não foi possível checar o *DRC* e *LVS* nas ferramentas comerciais.

O fluxo *Openlane* não faz a inserção de metal fill. Nem o próprio *PDK* informa como fazer a inserção, o conjunto de regras ainda está em andamento. Apesar do fluxo não ter sido completado, ainda assim, conseguiu-se abstrair muitas informações sobre como as ferramentas funcionam.

Bibliografia

AJAYI, D., BLAAUM, David. ***OpenROAD: Toward a Self-Driving, open source Digital Layout Implementation Tool Chain***. Disponível em: <https://vlsicad.ucsd .edu/ Publications/Conferences/370/c370.pdf>. Acesso em: 11 de Outubro de 2021.

BHUVA, D. ***VLSI Physical Design Methodology for ASIC Development with a Flavor of IP Hardening***. Disponível em: <https://www.design-reuse.com/articles/48852/vlsi-physical-design-methodology-for-asic-development-with-a-flavor-of-ip-hardening.html>. Acesso em: 20 de Setembro de 2021.

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

MEDEIROS, J. L. P. ***Design Físico de um IP. Trabalho de Conclusão de Curso em Engenharia Elétrica*** – Universidade Federal de Campina Grande, Campina Grande, 2018.

Laboratório XMEN. ***XMCPROCv0 Block Guide***.

Documentação Openlane. Disponível em: https://github.com/The-OpenROAD-Project/OpenLane. Acesso em: 15 de Setembro de 2021.

Documentação OpenROAD. Disponível em: <https://github.com/The-OpenROAD-Project/OpenROAD>. Acesso em: 10 de Outubro de 2021.

ANEXO A – Script TCL

Código A.1 – Código para gerar relatório de *DRV*

```
set fp [open "nomereportdrv" r]
set file_data [read $fp]
set outfile2 [open "drv_report.rpt" w+]
set slew 0
set cap 0
set fan 0
set c_slew 0
set c_cap 0
set c_fan 0
set data [split $file_data "\n"]
foreach line $data {

  if {[string first "max slew" $line] != -1} {
    set slew 1
    set cap 0
    set fan 0
  }
  if {[string first "max capacitance" $line] != -1} {
      set slew 0
      set cap 1
      set fan 0
    }
  if {[string first "max fanout" $line] != -1} {
      set slew 0
      set cap 0
      set fan 1
    }
  if {$slew == 1 && [string first "VIOLATED" $line] != -1} {
    set c_slew [expr $c_slew + 1]
  }
  if {$cap == 1 && [string first "VIOLATED" $line] != -1} {
    set c_cap [expr $c_cap + 1]
  }
  if {$fan == 1 && [string first "VIOLATED" $line] != -1} {
    set c_fan [expr $c_fan + 1]
  }
}
puts $outfile2 "Max Slew Violations: $c_slew"
puts $outfile2 "Max Capacitance Violations: $c_cap"
puts $outfile2 "Max Fanout Violations: $c_fan"

```

```
42 close $outfile2
43 close $fp
```

Código A.2 – Procs para gerar relatório de *setup e hold*

```
1 proc timing_report_setup {path_group rpt_out path_count step} {
2
3   set sta_geral_out_filename "timing_setup.rpt"
4
5   set paths [find_timing_paths -path_delay max -group_count $path_count
     -path_group $path_group]
6   set sta_total_out [open $sta_geral_out_filename "a+"]
7
8   set count_s 0
9   set tns_s 0
10  set wns_s 10000
11  foreach path $paths {
12    set startpoint_name [get_property [get_property $path startpoint]
     full_name]
13    set endpoint_name [get_property [get_property $path endpoint]
     full_name]
14    set slack_s [get_property $path slack]
15    if {$slack_s < 0} {
16       set count_s [expr $count_s + 1]
17       set tns_s [expr $tns_s + $slack_s]
18    }
19    if {$slack_s < $wns_s} {
20       set wns_s $slack_s
21    }
22  }
23
24    puts $sta_total_out [format "\n Setup Mode: ($path_group) \nTNS(ns)
     :%.4f \nWNS(ns):%.4f \nTotal Paths: [llength $paths] \nTotal
     Violations : $count_s" $tns_s $wns_s]
25  close $sta_total_out
26 }
27
28 proc timing_report_hold {path_group rpt_out path_count step} {
29  set sta_geral_out_filename "timing_hold.rpt"
30
31
32  set paths [find_timing_paths -path_delay min -group_count $path_count
     -path_group $path_group]
33  set sta_total_out [open $sta_geral_out_filename "a+"]
34
35  set count_s 0
36  set tns_s 0
37  set wns_s 10000
```

```
38   foreach path $paths {
39     set startpoint_name [get_property [get_property $path startpoint]
     full_name]
40     set endpoint_name [get_property [get_property $path endpoint]
     full_name]
41     set slack_s [get_property $path slack]
42     if {$slack_s < 0} {
43        set count_s [expr $count_s + 1]
44        set tns_s [expr $tns_s + $slack_s]
45     }
46     if {$slack_s < $wns_s} {
47        set wns_s $slack_s
48     }
49   }
50
51     puts $sta_total_out [format "\n Setup Mode: ($path_group) \nTNS(ns)
     :%.4f \nWNS(ns):%.4f \nTotal Paths: [llength $paths] \nTotal
     Violations : $count_s" $tns_s $wns_s]t
52   close $sta_total_out
53 }
```