

UNIVERSIDADE FEDERAL DE SANTA MARIA  
CENTRO DE TECNOLOGIA  
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA  
CONCEPÇÃO DE CIRCUITOS INTEGRADOS - UFSM00051  
PROF. ORIENTADOR: DR. CESAR AUGUSTO PRIOR

Autores: Anderson Venzke Backes, Artur Fernandez, Murilo Muller

**DESENVOLVIMENTO DE UMA BIBLIOTECA DE PORTAS LÓGICAS  
(STANDARD CELLS) E APLICAÇÃO DESTA NO PROCESSO DE  
SÍNTESE LÓGICA**

Santa Maria, RS  
2023

---

©2023

Todos os direitos autorais reservados a Autores: Anderson Venzke Backes, Artur Fernandez, Murilo Muller .  
A reprodução de partes ou do todo deste trabalho só poderá ser feita mediante a citação da fonte.  
End. Eletr.: andersonvenzkebackes@gmail.com

## LISTA DE FIGURAS

Figura 3.1 – Símbolo e Circuito Esquemático da Porta Inversora .....	6
Figura 3.2 – Variação do tempo de High-Low .....	7
Figura 3.3 – Variação do tempo de Low-High .....	7
Figura 3.4 – High-Low e Low-High em função da proporção dos transistores .....	8
Figura 3.5 – Circuito Esquemático Dimensionado da Porta NOT .....	8
Figura 3.6 – Functional da Porta NOT .....	9
Figura 3.7 – Layout da Porta NOT .....	10
Figura 3.8 – Símbolo e Circuito Esquemático da Porta NAND .....	11
Figura 3.9 – Test Bench da Porta NAND .....	11
Figura 3.10 – Functional da Porta NAND .....	12
Figura 3.11 – Layout da Porta NAND .....	13
Figura 3.12 – Símbolo e Circuito Esquemático da Porta NOR .....	14
Figura 3.13 – Test Bench da Porta NOR .....	14
Figura 3.14 – Functional da Porta NOR .....	15
Figura 3.15 – Layout da Porta NOR .....	16
Figura 3.16 – Símbolo e Circuito Esquemático da Porta AND .....	17
Figura 3.17 – Test Bench da Porta AND .....	17
Figura 3.18 – Functional da Porta AND .....	18
Figura 3.19 – Layout da Porta AND .....	19
Figura 3.20 – Símbolo e Circuito Esquemático da Porta OR .....	20
Figura 3.21 – Test Bench da Porta OR .....	20
Figura 3.22 – Functional da Porta OR .....	21
Figura 3.23 – Layout da Porta OR .....	22
Figura 3.24 – Símbolo e Circuito Esquemático da Porta XOR .....	23
Figura 3.25 – Test Bench da Porta XOR .....	23
Figura 3.26 – Functional da Porta XOR .....	24
Figura 3.27 – Layout da Porta XOR .....	25
Figura 3.28 – Símbolo e Circuito Esquemático da Porta XNOR .....	26
Figura 3.29 – Test Bench da Porta XNOR .....	26
Figura 3.30 – Layout da Porta XNOR .....	27
Figura 3.31 – Símbolo e Circuito Esquemático do Buffer .....	28
Figura 3.32 – Test Bench do Buffer .....	28
Figura 3.33 – Layout do Buffer .....	29
Figura 3.34 – Símbolo e Circuito Esquemático da Tri-State .....	30
Figura 3.35 – Test Bench da Tri-State .....	30
Figura 3.36 – Layout da Tri-State .....	31
Figura 3.37 – Símbolo e Circuito Esquemático do Buffer Tri-State .....	32
Figura 3.38 – Test Bench do Buffer Tri-State .....	32
Figura 3.39 – Layout do Buffer Tri-State .....	33
Figura 3.40 – Símbolo e Circuito Esquemático do Buffer Tri-State .....	34
Figura 3.41 – Test Bench da Transmission Gate .....	34
Figura 3.42 – Layout da Transmission Gate .....	35
Figura 3.43 – Símbolo e Circuito Esquemático do Multiplexador .....	36
Figura 3.44 – Test Bench do Mux .....	36
Figura 3.45 – Functional do Mux .....	37

Figura 3.46 – Layout do Mux .....	38
Figura 3.47 – Símbolo e Circuito Esquemático do Latch D .....	39
Figura 3.48 – Test Bench do Latch D .....	39
Figura 3.49 – Layout do Latch D .....	40
Figura 3.50 – Símbolo e Circuito Esquemático do Flip-Flop D .....	41
Figura 3.51 – Test Bench do Flip-Flop D .....	41
Figura 3.52 – Functional do Flip-Flop D .....	42
Figura 3.53 – Layout do Flip-Flop D .....	42
Figura 4.1 – LEF da tecnologia .....	45
Figura 4.2 – LEF da célula .....	46
Figura 4.3 – Tela inicial do programa Abstract .....	47
Figura 4.4 – View de abstract obtida da porta inversora .....	47
Figura 5.1 – Input-Slew, Output Slew e Cell Delay .....	49
Figura 5.2 – Setup e Hold Time .....	50
Figura 5.3 – Sem mudança .....	50
Figura 5.4 – Removal .....	51
Figura 5.5 – Recuperação .....	51
Figura 5.6 – Representação dos parâmetros obtidos em um plano 2d .....	53
Figura 5.7 – Cell Fall .....	53
Figura 5.8 – Representação dos pin names .....	54
Figura 5.9 – Exemplo de representação .....	54
Figura 5.10 – Representação de Flip-Flops .....	55
Figura 5.11 – Representação de scan cell .....	55
Figura 5.12 – Parte da representação .lib da porta inversora .....	56
Figura 6.1 – Circuito elétrico do Oscilador em anel .....	58
Figura 6.2 – Descrição em VHDL da porta lógica NOT .....	59
Figura 6.3 – Descrição em VHDL da porta lógica NAND .....	59
Figura 6.4 – Descrição em VHDL do oscilador .....	60
Figura 6.5 – Descrição em VHDL do contador .....	61
Figura 7.1 – Script tcl do genus para realizar a síntese lógica do oscilador .....	63
Figura 7.2 – Vista esquemática do oscilador após síntese lógica .....	63
Figura 7.3 – Estimativas de área e potência do circuito oscilador .....	64
Figura 7.4 – Script tcl do genus para realizar a síntese lógica do oscilador .....	64
Figura 7.5 – Vista esquemática do contador/prescaler após síntese lógica .....	65
Figura 7.6 – Estimativas de área e potência do circuito contador/prescaler .....	65
Figura 8.1 – Vista esquemática do oscilador dentro do virtuoso .....	66
Figura 8.2 – Testbench do Oscilador .....	67
Figura 8.3 – Estimativa da área do Oscilador .....	67
Figura 8.4 – Vista esquemática do contador/prescaler dentro do virtuoso .....	68
Figura 8.5 – Testbench do divisor de frequências .....	69
Figura 8.6 – Frequências na saída e consumo @ $f_{in} = 2GHz$ .....	69
Figura 8.7 – Estimativa da área do Prescaler .....	70

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	4
<b>2</b>	<b>OBJETIVOS .....</b>	5
2.1	OBJETIVO PRINCIPAL .....	5
2.2	OBJETIVOS ESPECÍFICOS .....	5
<b>3</b>	<b>BIBLIOTECA DE PORTAS LÓGICAS .....</b>	6
3.1	INVERSORA .....	6
3.2	NAND .....	10
3.3	NOR .....	13
3.4	AND .....	16
3.5	OR .....	19
3.6	XOR .....	22
3.7	XNOR .....	25
3.8	BUFFER .....	27
3.9	TRI-STATE .....	29
3.10	BUFFER TRI-STATE .....	31
3.11	TRANSMISSION GATE .....	33
3.12	MULTIPLEXADOR .....	35
3.13	LATCH D .....	38
3.14	FLIP-FLOP D .....	40
<b>4</b>	<b>LIBRARY EXCHANGE FORMAT .LEF .....</b>	44
4.1	INTRODUÇÃO .....	44
4.1.1	<b>LEF da tecnologia .....</b>	44
4.1.2	<b>LEF da célula .....</b>	45
4.2	CRIAÇÃO DO ARQUIVO .LEF PARA O PROJETO .....	46
<b>5</b>	<b>LIBERTY TIME FILE (.LIB) .....</b>	48
5.1	INTRODUÇÃO .....	48
5.2	CÁLCULO DO ATRASO BASEADO EM CÉLULAS .....	48
5.3	CALCULO DE ATRASO E CHECAGEM DE TIMING .....	48
5.4	INFORMAÇÕES ESSENCIAIS DE UM ARQUIVO .LIB .....	51
5.5	MODELO EM DUAS DIMENSÕES .....	52
5.6	CRIAÇÃO DO ARQUIVO .LIB PARA O PROJETO .....	55
<b>6</b>	<b>CIRCUITOS UTILIZADOS PARA VALIDAÇÃO .....</b>	57
6.1	OSCILADOR .....	58
6.2	CONTADOR/PRESCALER .....	60
<b>7</b>	<b>SÍNTESE LÓGICA .....</b>	62
7.1	OSCILADOR .....	63
7.2	CONTADOR/PRESCALER .....	64
<b>8</b>	<b>IMPORTAÇÃO DOS ARQUIVOS PARA O VIRTUOSO E TESTE DOS MÉTODOS .....</b>	66
8.1	OSCILADOR .....	66
8.2	CONTADOR/PRESCALER .....	68
<b>9</b>	<b>CONCLUSÃO E PERSPECTIVAS FUTURAS .....</b>	71
	<b>REFERÊNCIAS BIBLIOGRÁFICAS .....</b>	72

## **1 INTRODUÇÃO**

Este relatório descreve o desenvolvimento e a implementação bem-sucedidos de uma biblioteca de standard cells para ser utilizada no processo de síntese lógica de circuitos integrados. O objetivo primário foi criar uma coleção eficiente e versátil de elementos lógicos que podem ser facilmente adaptados para diferentes projetos de circuitos digitais.

O presente relatório aborda o processo de síntese lógica aplicado ao design de circuitos integrados. A síntese lógica é uma fase crítica no desenvolvimento de hardware digital, visando traduzir uma descrição funcional em uma implementação física eficiente.

## **2 OBJETIVOS**

### **2.1 OBJETIVO PRINCIPAL**

Desenvolver uma Biblioteca Eficiente: Criar uma coleção diversificada de standard cells otimizadas para atender a uma variedade de requisitos de design.

Facilitar a Síntese Lógica: Projetar as células de forma a simplificar e agilizar o processo de síntese lógica de circuitos integrados.

Garantir Desempenho Satisfatório: Assegurar que a biblioteca ofereça um desempenho satisfatório em termos de área ocupada, consumo de energia e velocidade de operação.

### **2.2 OBJETIVOS ESPECÍFICOS**

Identificar Requisitos de Design: Analisar as necessidades de diferentes projetos para identificar os requisitos específicos a serem atendidos pela biblioteca.

Projetar Células Lógicas Específicas: Desenvolver standard cells para funções lógicas específicas, otimizando-as para desempenho, área e potência.

Garantir Consistência na Interface: Manter uma interface consistente entre as células para facilitar a integração em projetos diversos, simplificando o processo de síntese.

Conduzir Simulações Abrangentes: Realizar simulações detalhadas para validar o comportamento das standard cells em diversas condições operacionais.

Integrar a Biblioteca em Fluxos de Design: Garantir que a biblioteca seja facilmente integrada em fluxos de design existentes, colaborando com ferramentas de síntese lógica.

Avaliar Desempenho em Projetos Reais: Testar a biblioteca em projetos reais para avaliar o desempenho em situações práticas e identificar oportunidades de melhoria.

Documentar Adequadamente a Biblioteca: Preparar documentação abrangente para os usuários, incluindo descrições técnicas, guias de uso e informações sobre as características de desempenho.

### 3 BIBLIOTECA DE PORTAS LÓGICAS

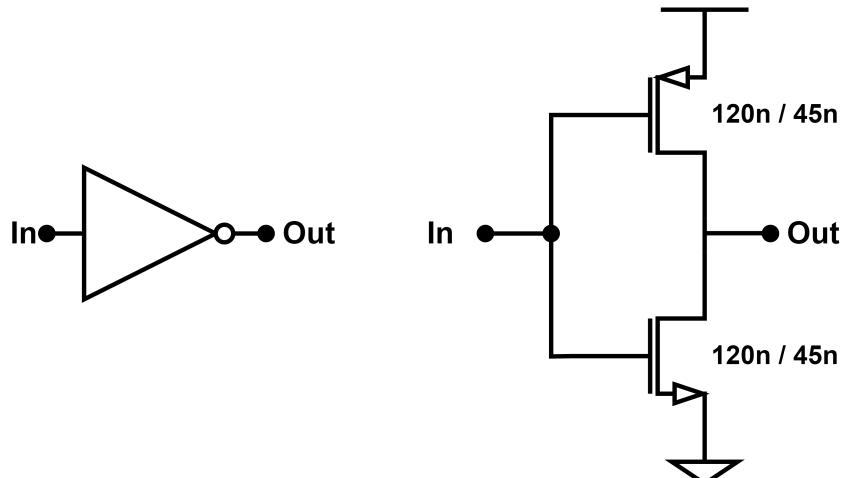
A biblioteca de standard cells é composta por um conjunto diversificado de elementos lógicos, como portas AND, OR, NOT, flip-flops, latches, entre outros. Cada célula foi projetada para atender a requisitos específicos de área, potência e velocidade, oferecendo flexibilidade no design de circuitos integrados.

O processo de desenvolvimento da biblioteca seguiu uma metodologia rigorosa que incluiu a análise de requisitos, o mapeamento lógico para elementos específicos, a otimização de desempenho e a validação por meio de simulações extensivas.

### 3.1 INVERSORA

Iniciando pela lógica digital mais básica, a porta inversora desenvolvida pode ser visualizada na figura 3.1

Figura 3.1 – Símbolo e Circuito Esquemático da Porta Inversora



Fonte: Produzido pelos autores.

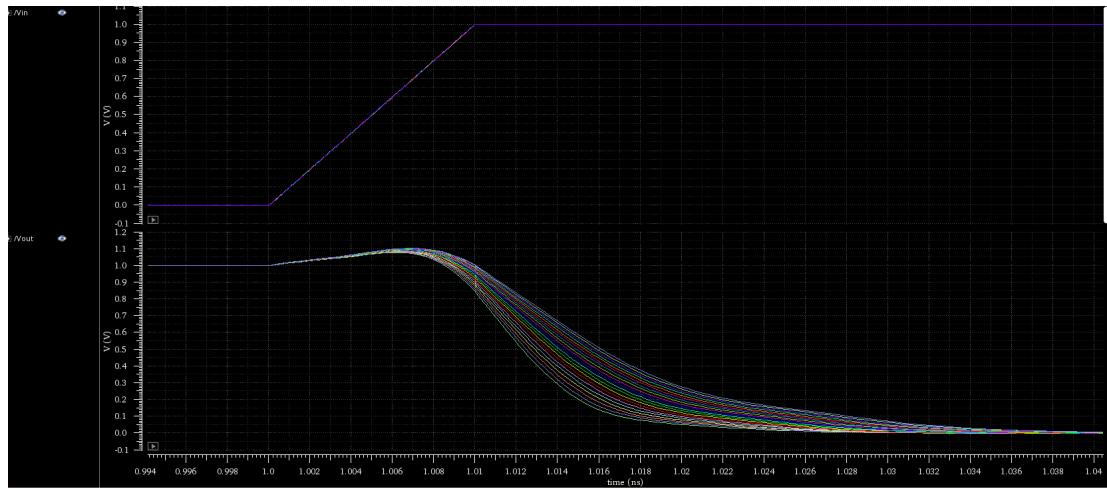
Tabela 3.1 – Tabela Verdade da Porta NOT

IN	Out
0	1
1	0

Utilizando os transitores mínimos da tecnologia, percebe-se que existe uma diferença na capacidade de condução de corrente entre os transistores Pmos e Nmos, impactando negativamente nos tempos de High-Low e Low-High da porta, essa diferença na condução dos transistores se deve ao fato de que no Nmos ocorre o deslocamento de elétrons, já no Pmos ocorre o deslocamento de lacunas, que são mais lentas. Para compensar essa diferença, é necessário aumentar o W do transistor Pmos, dessa forma realizou-se simulações paramétricas alterando a proporção entre o W do Pmos e do W do Nmos.

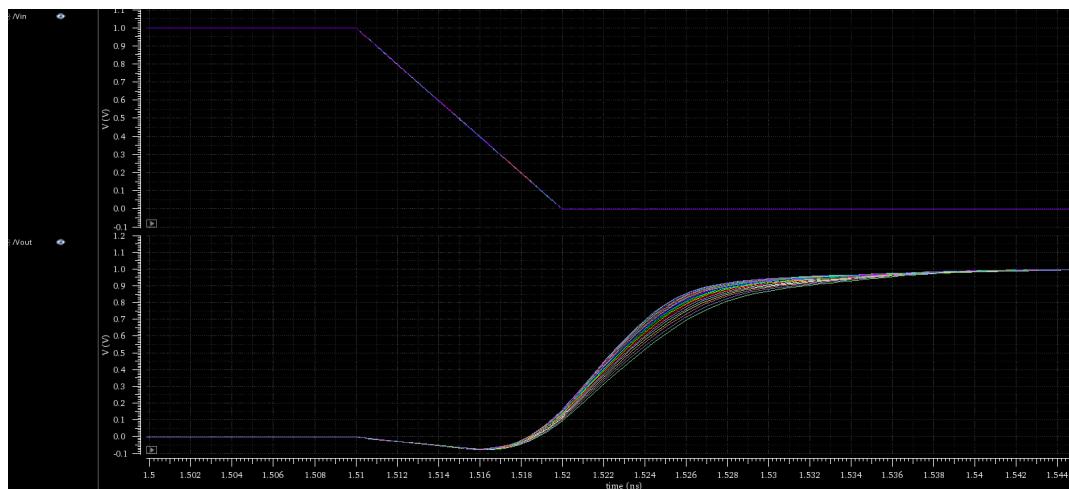
A cada alteração na proporção, foi analisado o tempo de High-Low e de Low-High [1], quando ambas os parâmetros são iguais, significa que o tempo para descarregar a capacitância do nó de saída é o mesmo que para carregar, logo a capacidade de corrente de ambos os transistores são iguais.

Figura 3.2 – Variação do tempo de High-Low



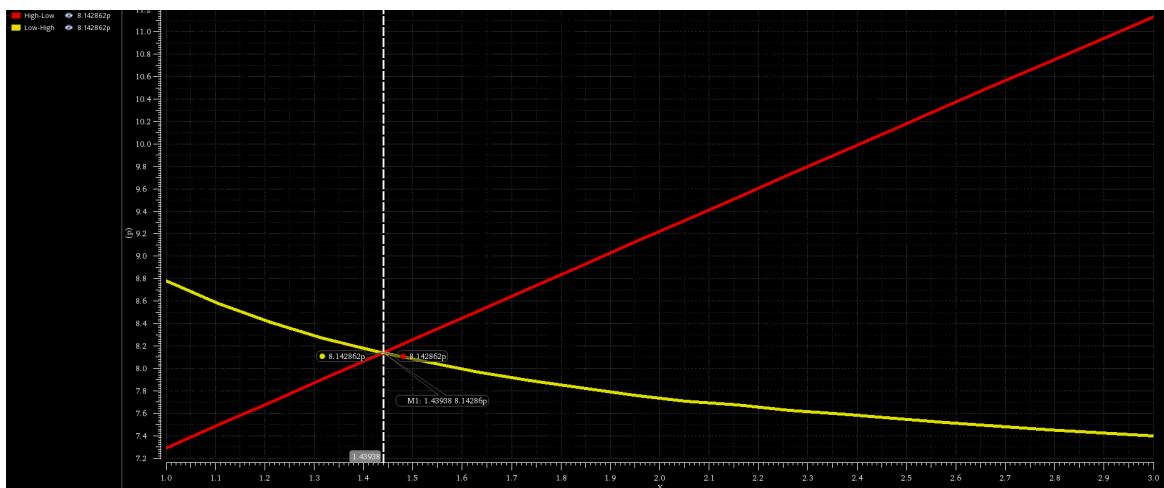
Fonte: Produzido pelos autores.

Figura 3.3 – Variação do tempo de Low-High



Fonte: Produzido pelos autores.

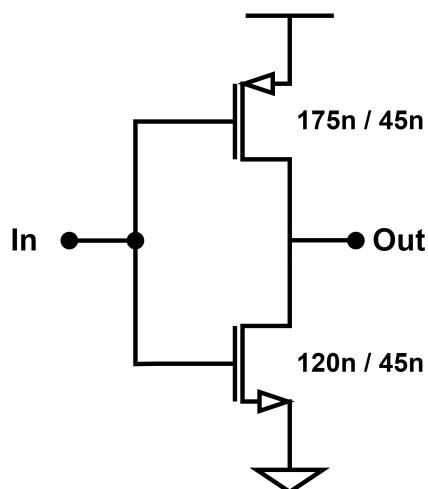
Figura 3.4 – High-Low e Low-High em função da proporção dos transistores



Fonte: Produzido pelos autores.

A proporção ideal foi definida como 1.45833, dessa maneira a porta NOT final pode ser visualizada na figura 3.5

Figura 3.5 – Circuito Esquemático Dimensionado da Porta NOT



Fonte: Produzido pelos autores.

Essa proporção foi mantida para todas as portas lógicas dessa biblioteca, além disso, para implementar algumas lógicas é necessário dispor os transistores em série, para manter a capacidade de corrente o W é multiplicado pela quantia de transistores colocados em série.

Em sequência, foi realizada a descrição funcional da porta lógica.

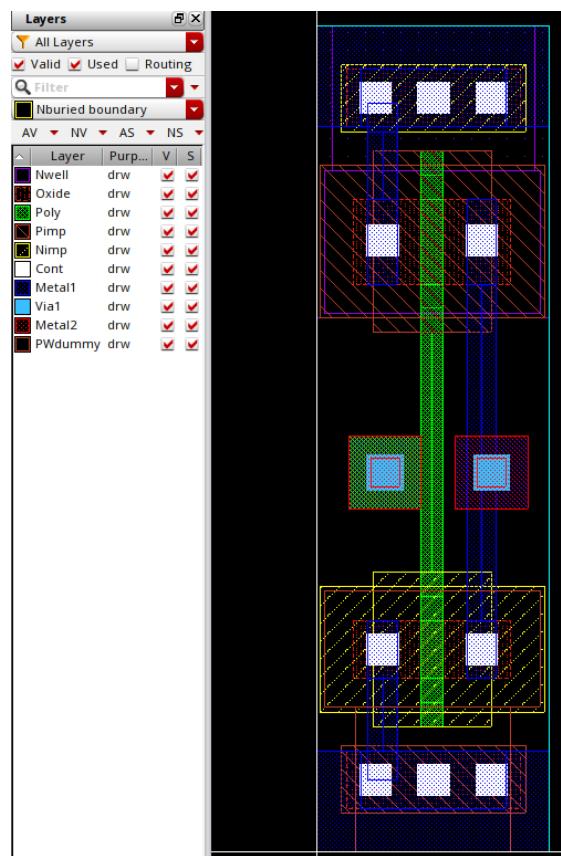
Figura 3.6 – Functional da Porta NOT

```
// type:  
`timescale 1ns/1ps  
`celldefine  
module NOT_1x (OUT, IN);  
    output OUT;  
    input IN;  
  
    // Function  
    not (OUT, IN);  
  
    // Timing  
    specify  
        (IN => OUT) = 0;  
    endspecify  
endmodule  
`endcelldefine
```

Fonte: Produzido pelos autores.

Por fim, foi desenvolvido o Layout da porta NOT.

Figura 3.7 – Layout da Porta NOT



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

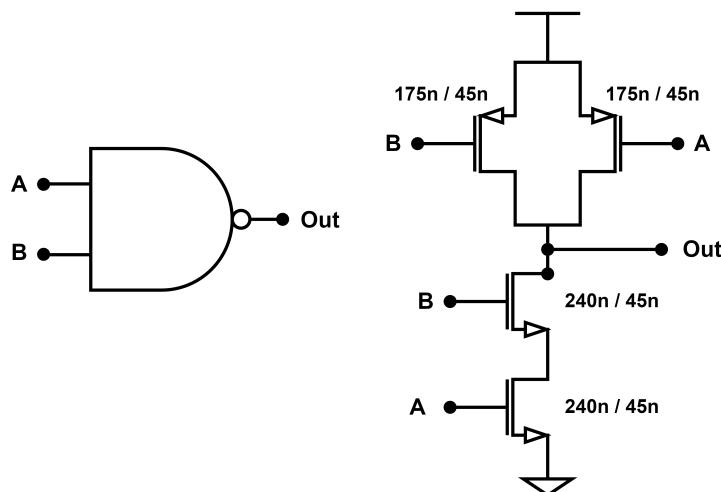
Tabela 3.2 – Área da Porta NOT

Área Standard Cells ( $\mu m^2$ )	Área Full Custom ( $\mu m^2$ )
0.8208	0.8208

### 3.2 NAND

A porta NAND desenvolvida pode ser visualizada na figura 3.8

Figura 3.8 – Símbolo e Circuito Esquemático da Porta NAND



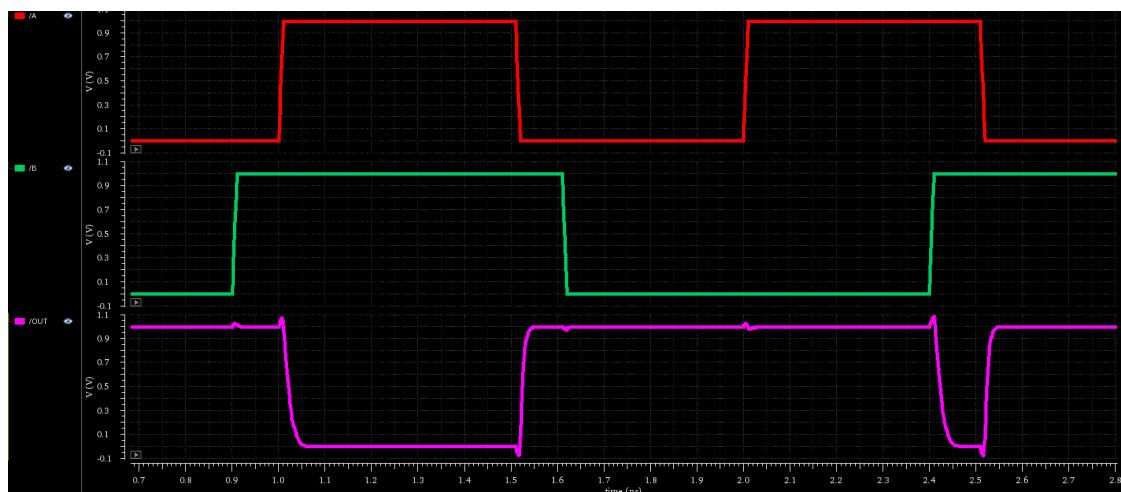
Fonte: Produzido pelos autores.

Tabela 3.3 – Tabela Verdade da Porta NAND

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.9 – Test Bench da Porta NAND



Fonte: Produzido pelos autores.

Em sequência, foi realizada a descrição funcional da porta lógica.

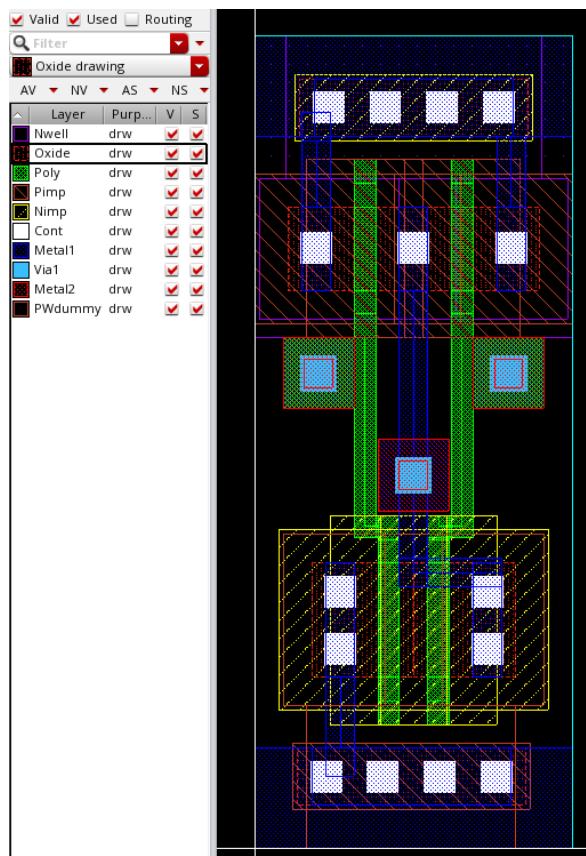
Figura 3.10 – Functional da Porta NAND

```
// type:  
`timescale 1ns/1ps  
`celldefine  
module NAND_1x (OUT, A, B);  
    output OUT;  
    input A, B;  
  
    // Function  
    wire A__bar, int_fwire_0;  
  
    not (A__bar, A);  
    and (int_fwire_0, A__bar, B);  
    not (OUT, int_fwire_0);  
  
    // Timing  
    specify  
        (A => OUT) = 0;  
        (B => OUT) = 0;  
    endspecify  
endmodule  
`endcelldefine
```

Fonte: Produzido pelos autores.

Por fim, foi desenvolvido o Layout da porta NAND.

Figura 3.11 – Layout da Porta NAND



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

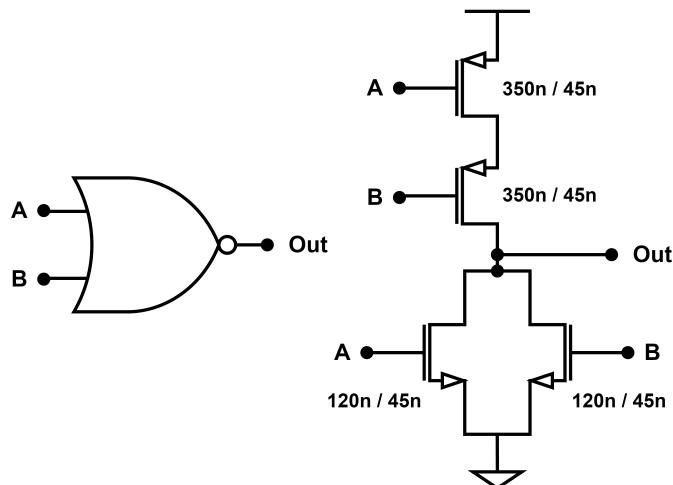
Tabela 3.4 – Área da Porta NAND

Área Standard Cells ( $\mu\text{m}^2$ )	Área Full Custom ( $\mu\text{m}^2$ )
1.1457	1.1457

### 3.3 NOR

A porta NOR desenvolvida pode ser visualizada na figura 3.12

Figura 3.12 – Símbolo e Circuito Esquemático da Porta NOR



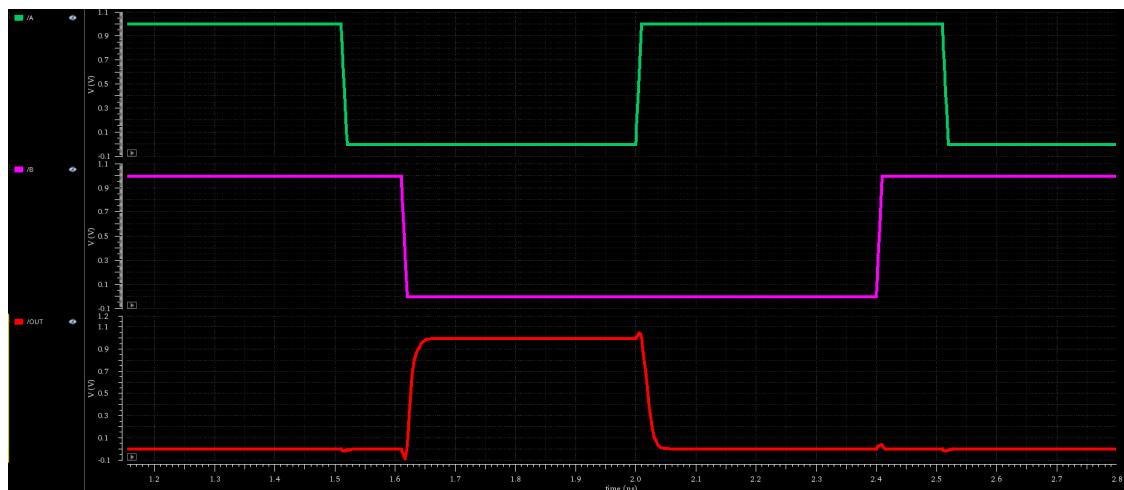
Fonte: Produzido pelos autores.

Tabela 3.5 – Tabela Verdade da Porta NOR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.13 – Test Bench da Porta NOR



Em sequência, foi realizada a descrição funcional da porta lógica.

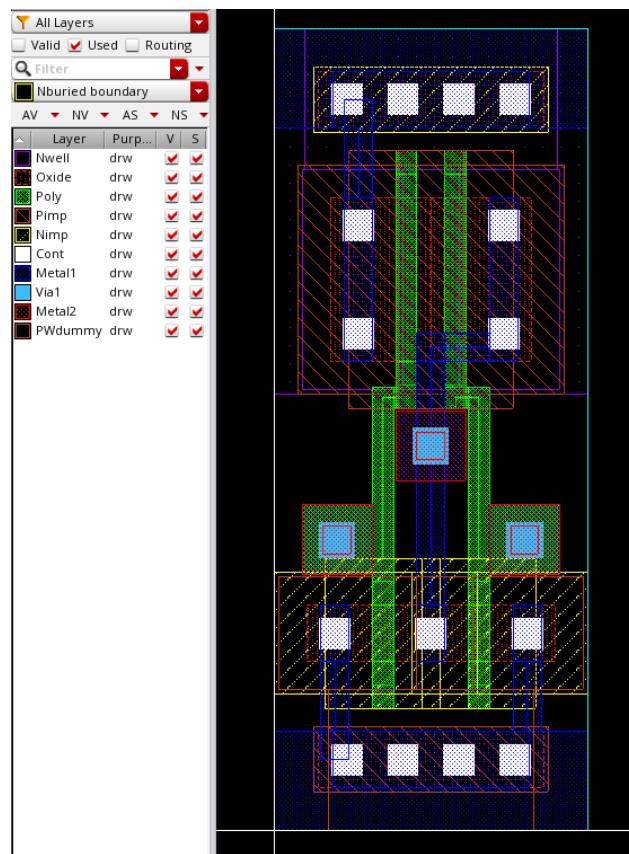
Figura 3.14 – Functional da Porta NOR

```
// type:  
`timescale 1ns/1ps  
`celldefine  
module NOR_1x (OUT, A, B);  
    output OUT;  
    input A, B;  
  
    // Function  
    wire A__bar, int_fwire_0;  
  
    not (A__bar, A);  
    or (int_fwire_0, A__bar, B);  
    not (OUT, int_fwire_0);  
  
    // Timing  
    specify  
        (A => OUT) = 0;  
        (B => OUT) = 0;  
    endspecify  
endmodule  
`endcelldefine
```

Fonte: Produzido pelos autores.

Por fim, foi desenvolvido o Layout da porta NOR.

Figura 3.15 – Layout da Porta NOR



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

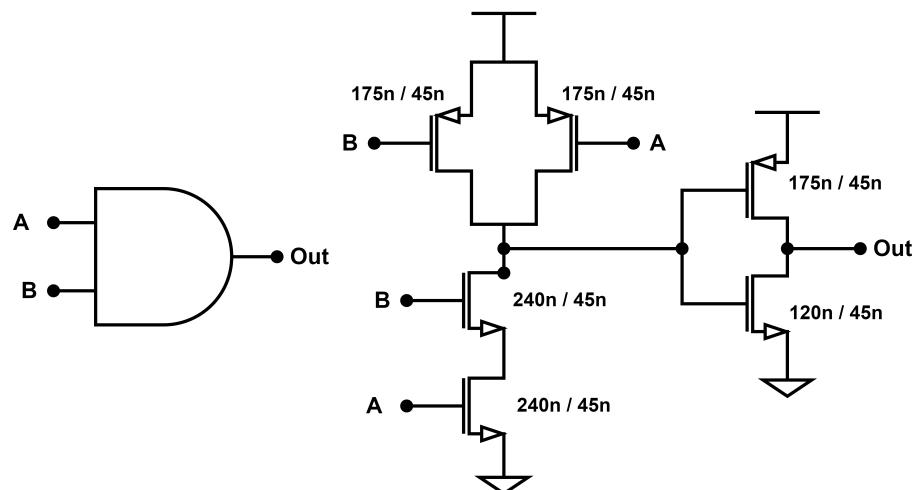
Tabela 3.6 – Área da Porta NOR

Área Standard Cells ( $\mu m^2$ )	Área Full Custom ( $\mu m^2$ )
1.1457	1.1457

### 3.4 AND

A porta AND desenvolvida pode ser visualizada na figura 3.16

Figura 3.16 – Símbolo e Circuito Esquemático da Porta AND



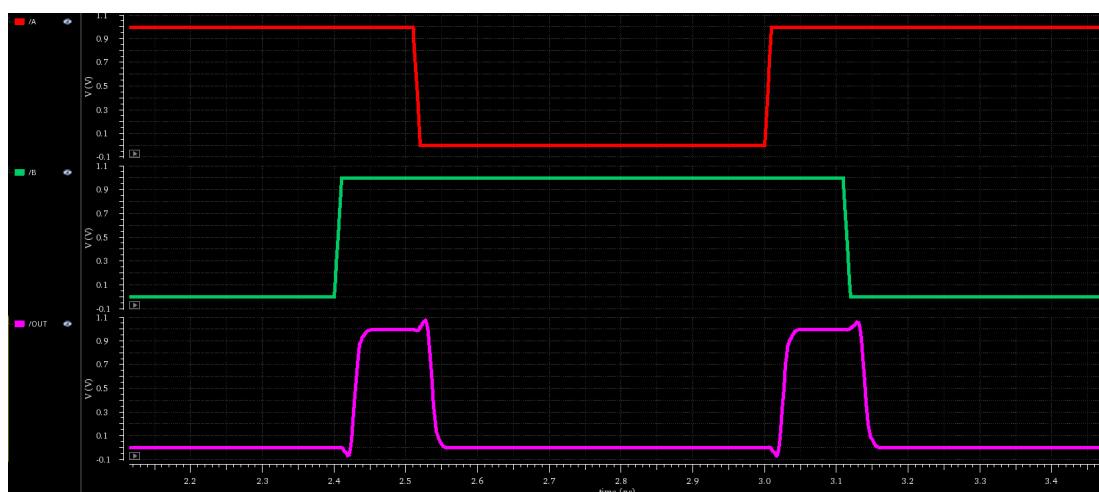
Fonte: Produzido pelos autores.

Tabela 3.7 – Tabela Verdade da Porta AND

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.17 – Test Bench da Porta AND



Em sequência, foi realizada a descrição funcional da porta lógica.

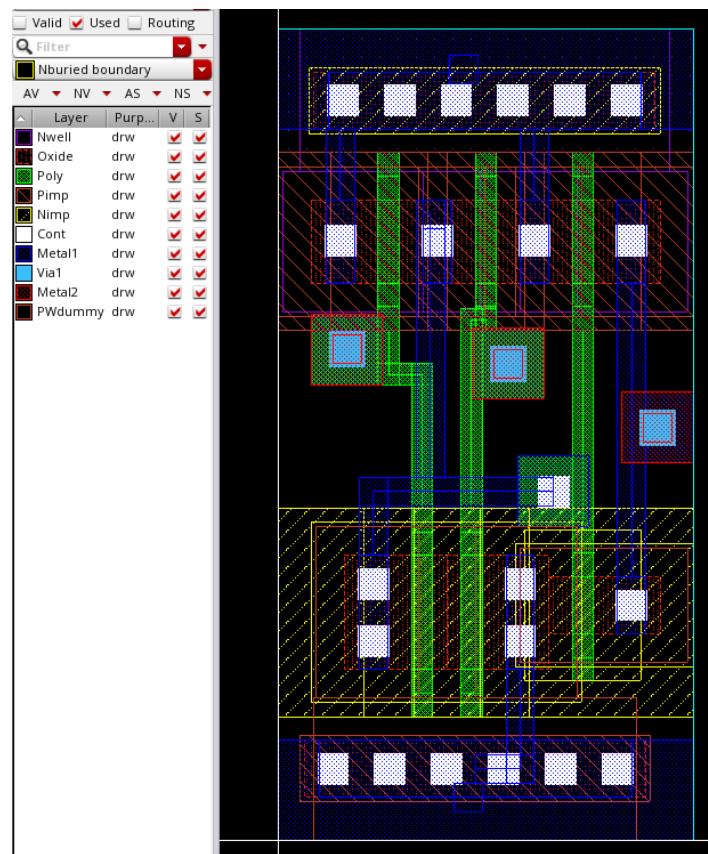
Figura 3.18 – Functional da Porta AND

```
// type:  
`timescale 1ns/1ps  
`celldefine  
module AND_1x (OUT, A, B);  
    output OUT;  
    input A, B;  
  
    // Function  
    and (OUT, A, B);  
  
    // Timing  
    specify  
        (A => OUT) = 0;  
        (B => OUT) = 0;  
    endspecify  
endmodule  
`endcelldefine
```

Fonte: Produzido pelos autores.

Por fim, foi desenvolvido o Layout da porta AND.

Figura 3.19 – Layout da Porta AND



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

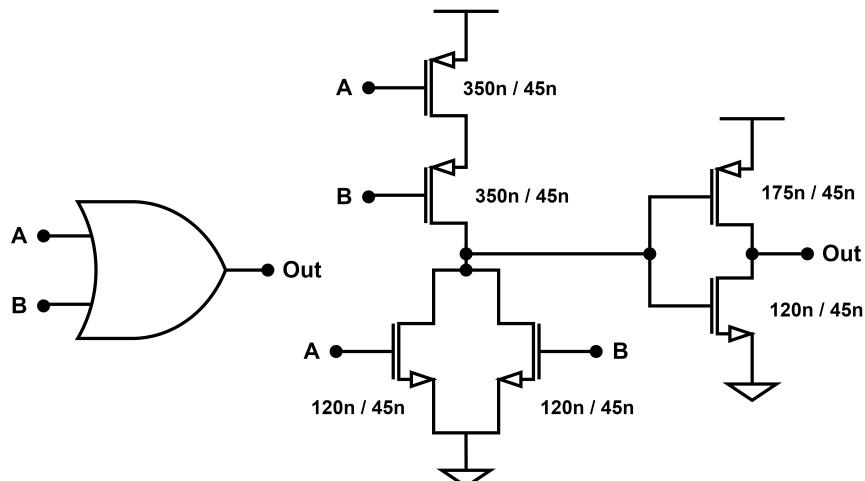
Tabela 3.8 – Área da Porta AND

Área Standard Cells ( $\mu m^2$ )	Área Full Custom ( $\mu m^2$ )
1.9665	1.4962

### 3.5 OR

A porta OR desenvolvida pode ser visualizada na figura 3.20

Figura 3.20 – Símbolo e Circuito Esquemático da Porta OR



Fonte: Produzido pelos autores.

Tabela 3.9 – Tabela Verdade da Porta OR

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.21 – Test Bench da Porta OR



Em sequência, foi realizada a descrição funcional da porta lógica.

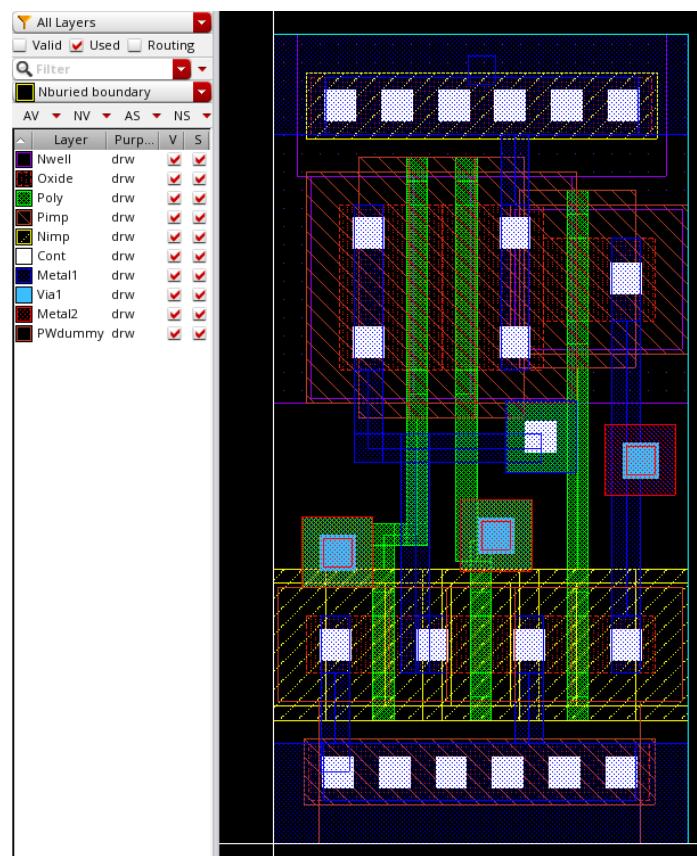
Figura 3.22 – Functional da Porta OR

```
// type:  
`timescale 1ns/1ps  
`celldefine  
module OR_1x (OUT, A, B);  
    output OUT;  
    input A, B;  
  
    // Function  
    or (OUT, A, B);  
  
    // Timing  
    specify  
        (A => OUT) = 0;  
        (B => OUT) = 0;  
    endspecify  
endmodule  
`endcelldefine
```

Fonte: Produzido pelos autores.

Por fim, foi desenvolvido o Layout da porta OR.

Figura 3.23 – Layout da Porta OR



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

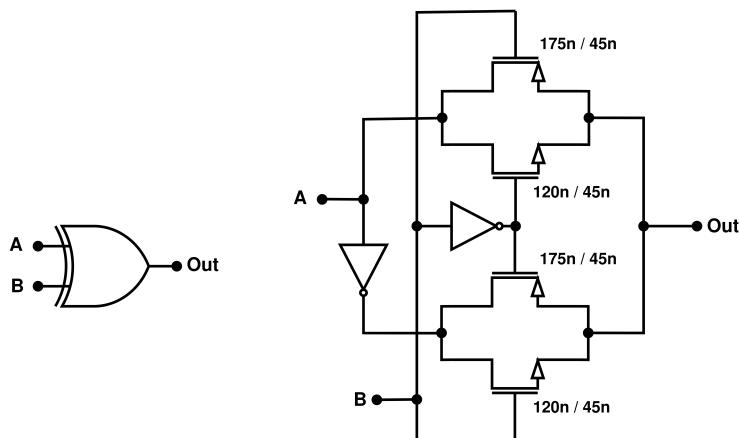
Tabela 3.10 – Área da Porta OR

Área Standard Cells ( $\mu\text{m}^2$ )	Área Full Custom ( $\mu\text{m}^2$ )
1.9665	1.4962

### 3.6 XOR

A porta XOR desenvolvida pode ser visualizada na figura 3.24

Figura 3.24 – Símbolo e Circuito Esquemático da Porta XOR



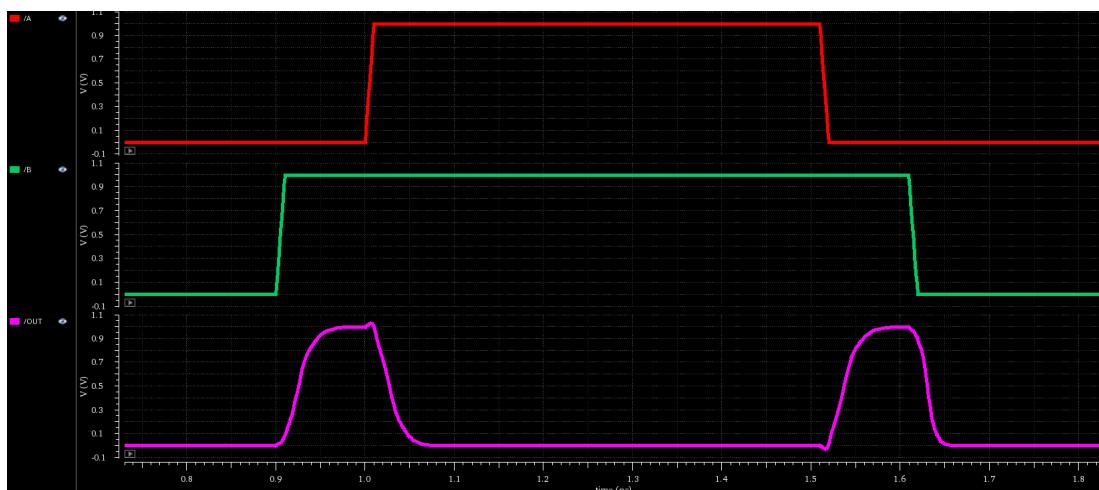
Fonte: Produzido pelos autores.

Tabela 3.11 – Tabela Verdade da Porta XOR

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.25 – Test Bench da Porta XOR



Em sequência, foi realizada a descrição funcional da porta lógica.

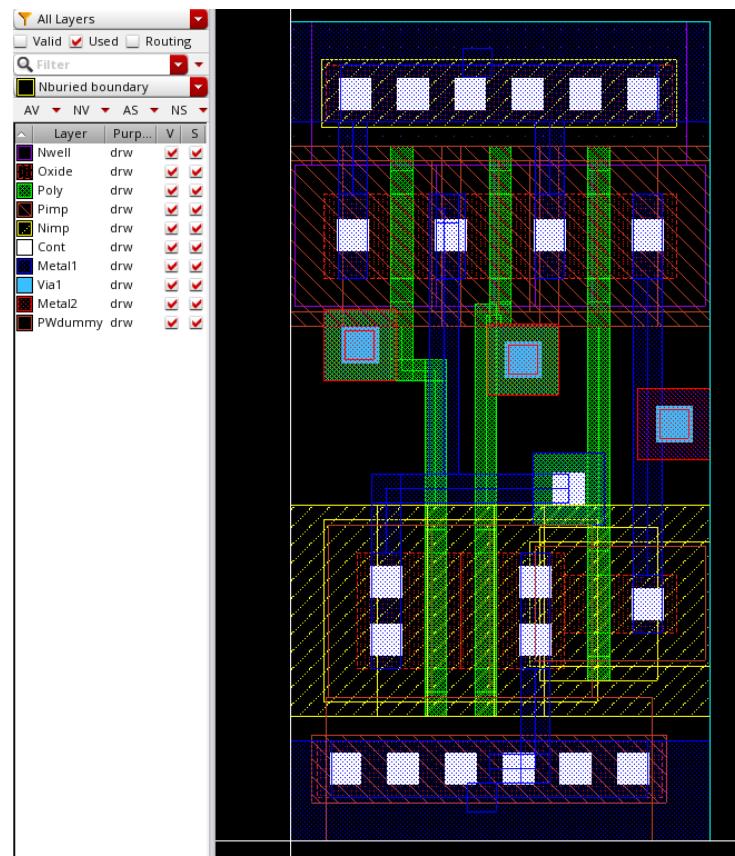
Figura 3.26 – Functional da Porta XOR

```
// type:  
`timescale 1ns/1ps  
`celldefine  
module XOR (OUT, A, B);  
    output OUT;  
    input A, B;  
  
    // Function  
    xor (OUT, A, B);  
  
    // Timing  
    specify  
        (posedge A => (OUT:A)) = 0;  
        (negedge A => (OUT:A)) = 0;  
        (posedge B => (OUT:B)) = 0;  
        (negedge B => (OUT:B)) = 0;  
    endspecify  
endmodule  
`endcelldefine
```

Fonte: Produzido pelos autores.

Por fim, foi desenvolvido o Layout da porta XOR.

Figura 3.27 – Layout da Porta XOR



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

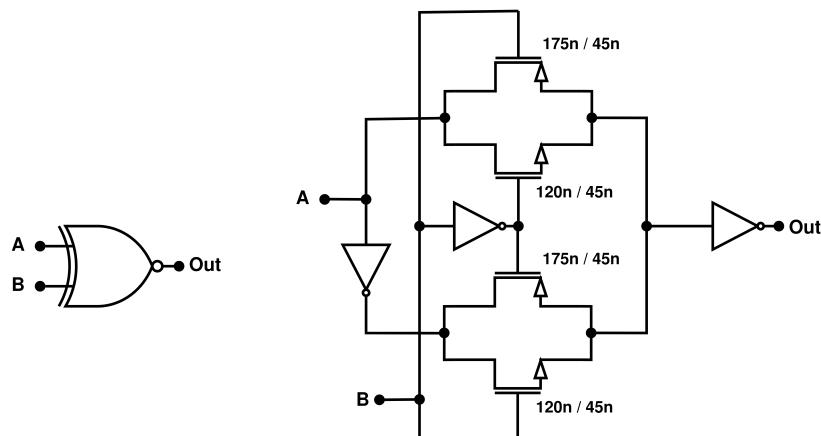
Tabela 3.12 – Área da Porta XOR

Área Standard Cells ( $\mu m^2$ )	Área Full Custom ( $\mu m^2$ )
4.8222	1.8468

### 3.7 XNOR

A porta XNOR desenvolvida pode ser visualizada na figura 3.28

Figura 3.28 – Símbolo e Circuito Esquemático da Porta XNOR



Fonte: Produzido pelos autores.

Tabela 3.13 – Tabela Verdade da Porta XNOR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

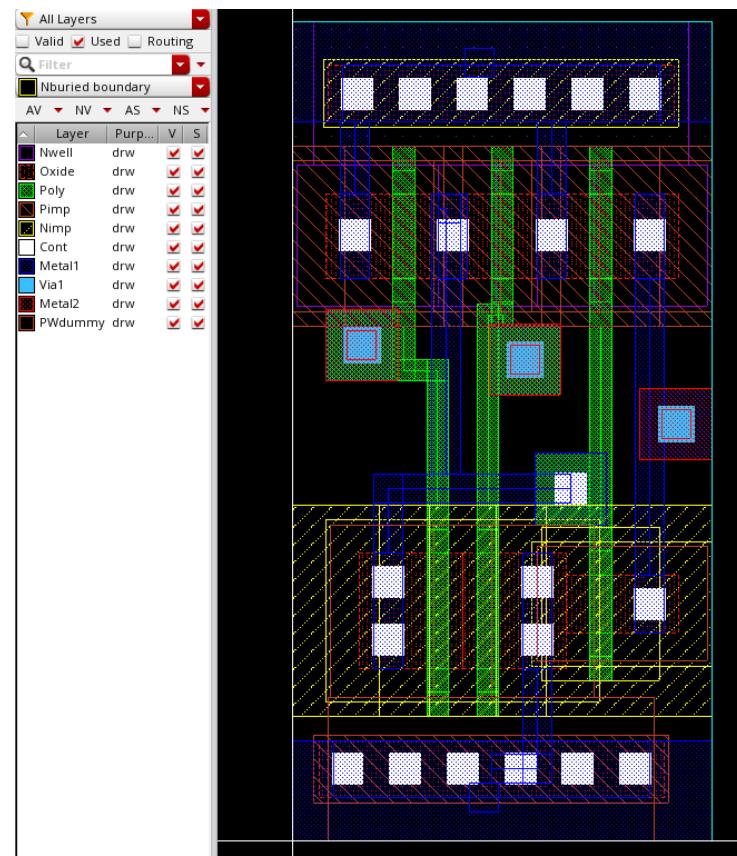
Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.29 – Test Bench da Porta XNOR



Por fim, foi desenvolvido o Layout da porta XNOR.

Figura 3.30 – Layout da Porta XNOR



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

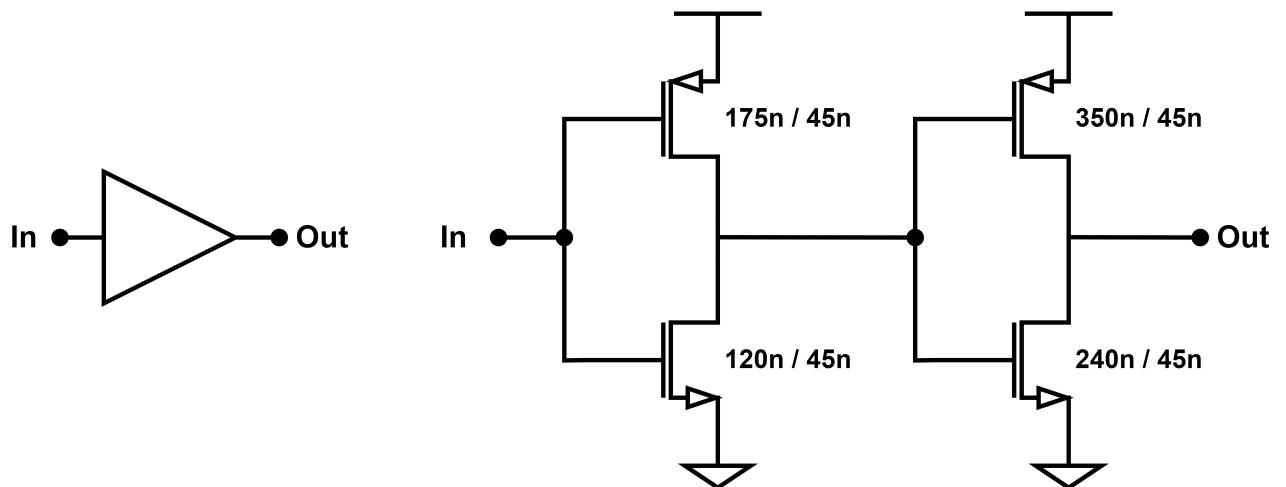
Tabela 3.14 – Área da Porta XNOR

Área Standard Cells ( $\mu\text{m}^2$ )	Área Full Custom ( $\mu\text{m}^2$ )
5.6430	2.5393

### 3.8 BUFFER

O Buffer desenvolvida pode ser visualizado na figura 3.31

Figura 3.31 – Símbolo e Circuito Esquemático do Buffer



Fonte: Produzido pelos autores.

Tabela 3.15 – Tabela Verdade do Buffer

In	Out
0	0
1	1

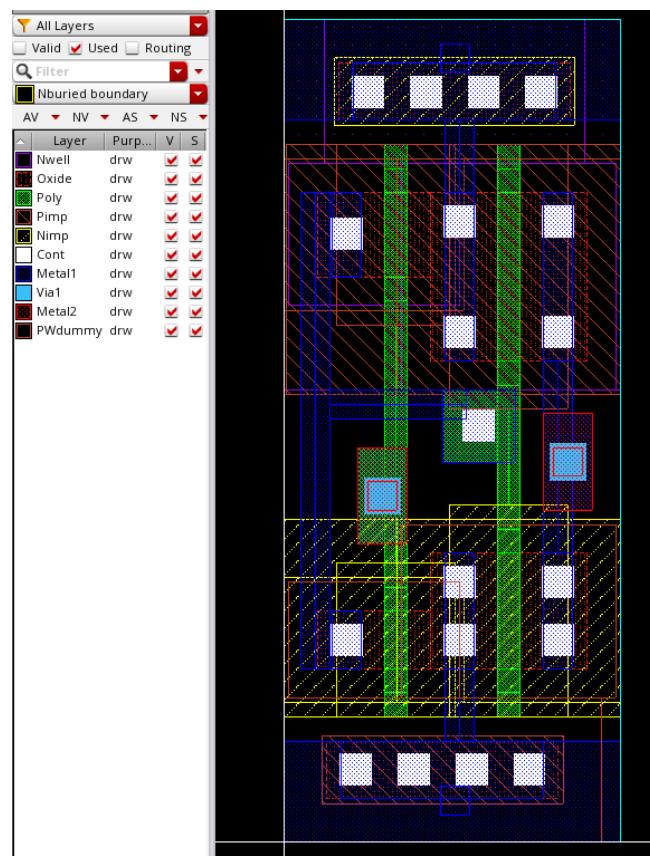
Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.32 – Test Bench do Buffer



Por fim, foi desenvolvido o Layout do Buffer.

Figura 3.33 – Layout do Buffer



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

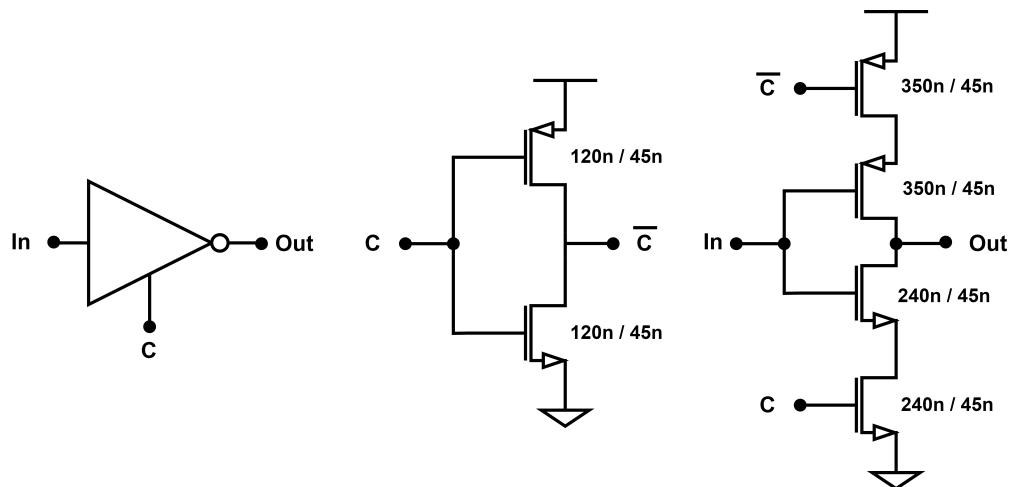
Tabela 3.16 – Área do Buffer

Área Standard Cells ( $\mu m^2$ )	Área Full Custom ( $\mu m^2$ )
1.6416	1.1970

### 3.9 TRI-STATE

A porta Tri-State desenvolvida pode ser visualizada na figura 3.34

Figura 3.34 – Símbolo e Circuito Esquemático da Tri-State



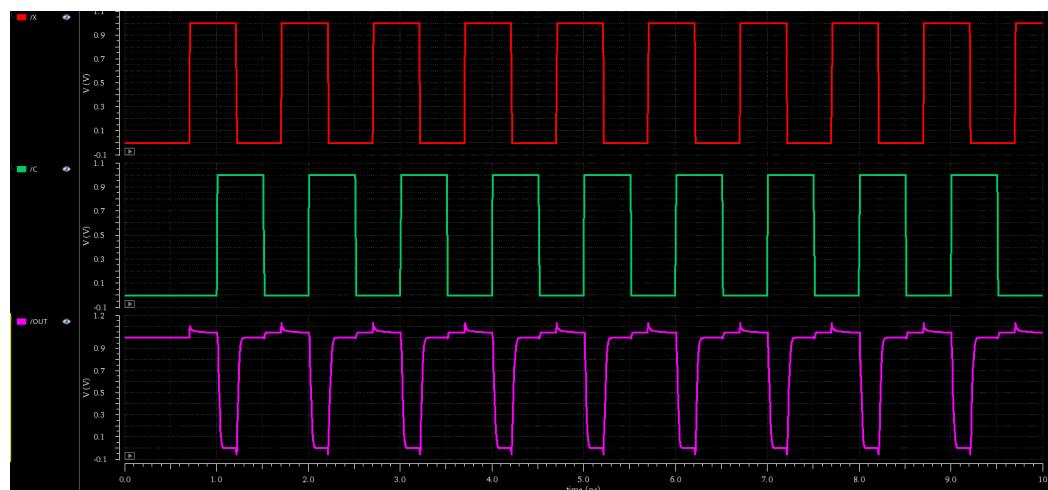
Fonte: Produzido pelos autores.

Tabela 3.17 – Tabela Verdade da Tri-State

In	C	Out
0	0	Z
0	1	1
1	0	Z
1	1	0

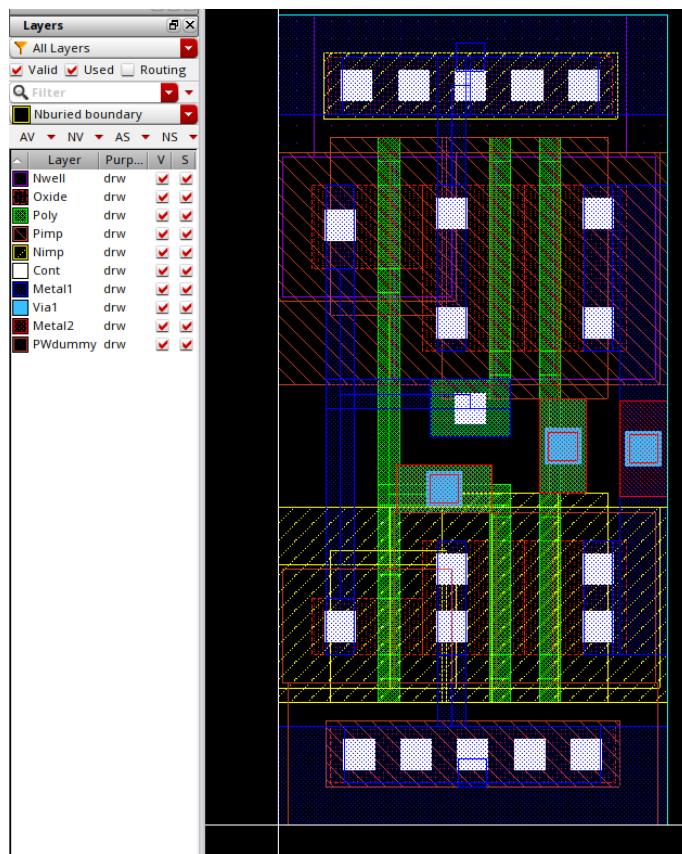
Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.35 – Test Bench da Tri-State



Por fim, foi desenvolvido o Layout da Tri-State.

Figura 3.36 – Layout da Tri-State



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

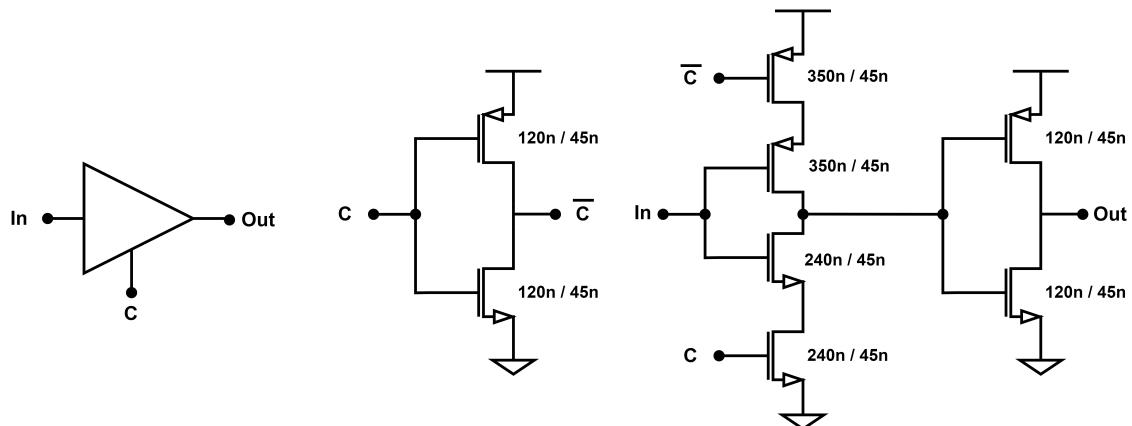
Tabela 3.18 – Área da Tri-State

Área Standard Cells ( $\mu m^2$ )	Área Full Custom ( $\mu m^2$ )
1.4022	1.4022

### 3.10 BUFFER TRI-STATE

O Buffer Tri-State desenvolvido pode ser visualizado na figura 3.37

Figura 3.37 – Símbolo e Circuito Esquemático do Buffer Tri-State



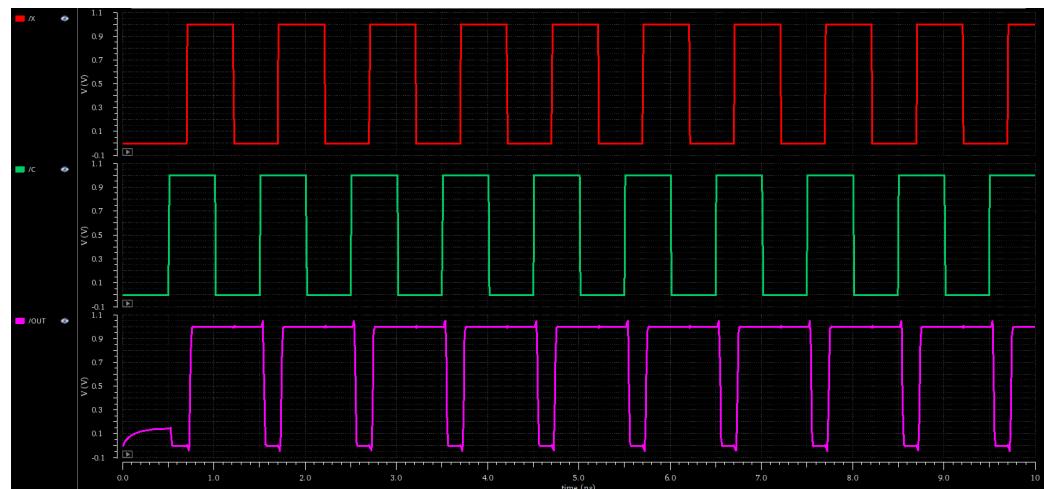
Fonte: Produzido pelos autores.

Tabela 3.19 – Tabela Verdade do Buffer Tri-State

In	C	Out
0	0	Z
0	1	0
1	0	Z
1	1	1

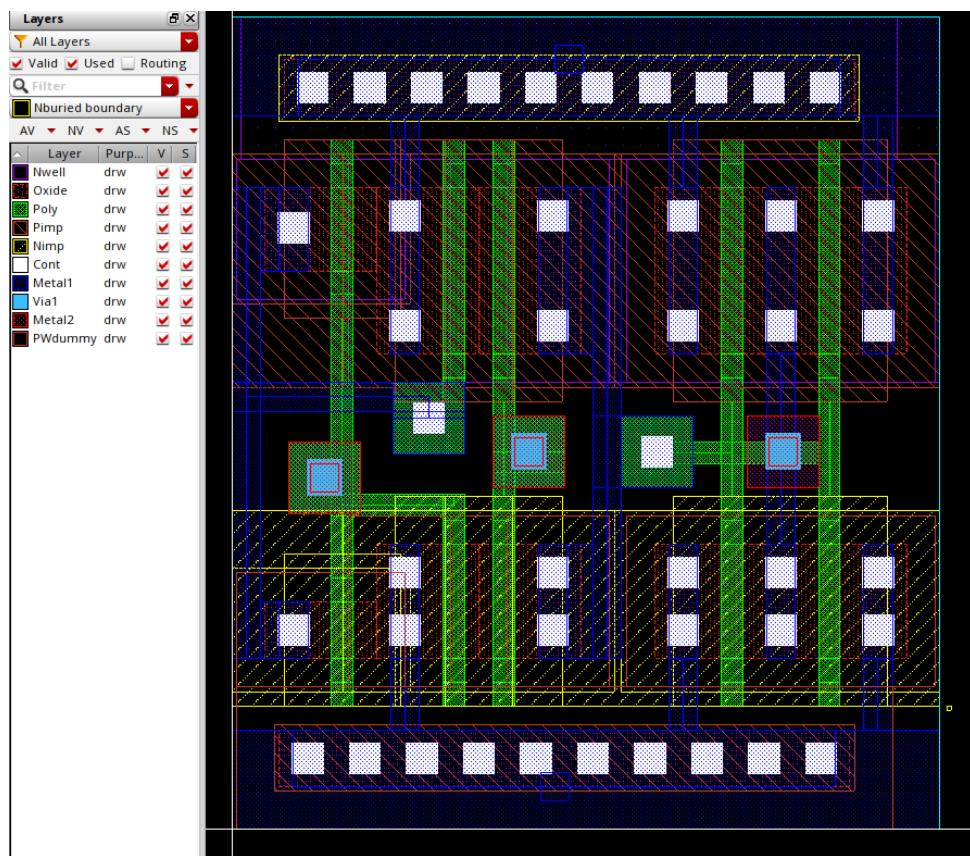
Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.38 – Test Bench do Buffer Tri-State



Por fim, foi desenvolvido o Layout do Buffer Tri-State.

Figura 3.39 – Layout do Buffer Tri-State



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

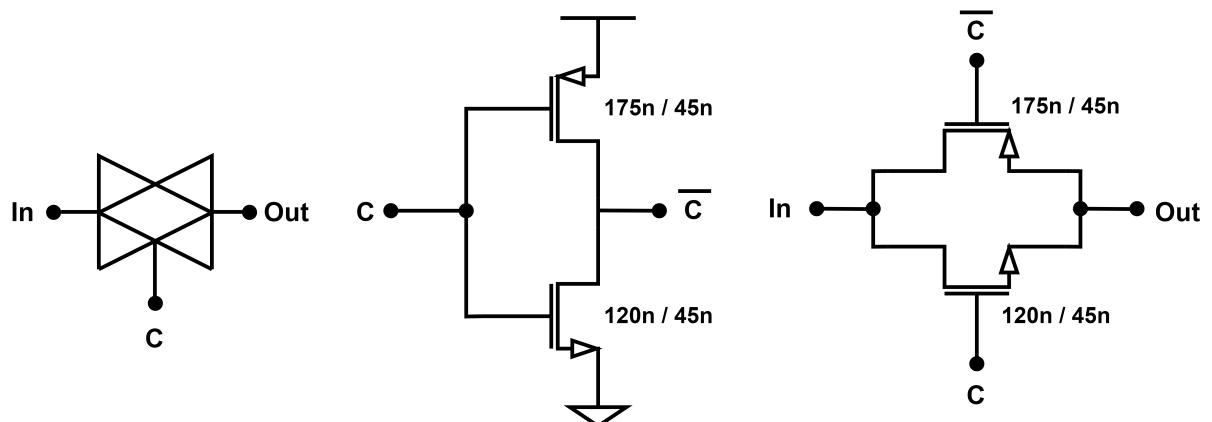
Tabela 3.20 – Área do Buffer Tri-State

Área Standard Cells ( $\mu m^2$ )	Área Full Custom ( $\mu m^2$ )
2.2230	2.5479

### 3.11 TRANSMISSION GATE

A Transmission Gate desenvolvida pode ser visualizada na figura 3.40

Figura 3.40 – Símbolo e Circuito Esquemático do Buffer Tri-State



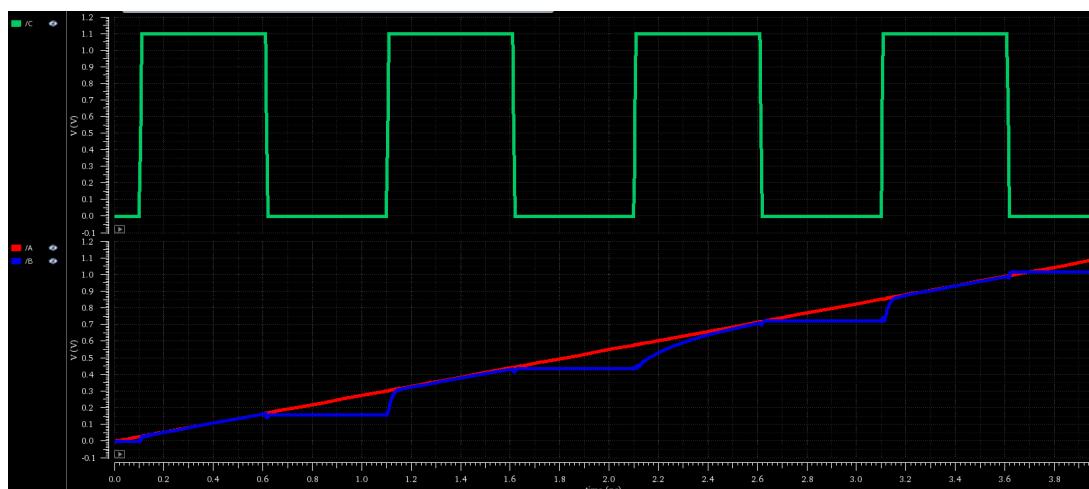
Fonte: Produzido pelos autores.

Tabela 3.21 – Tabela Verdade da Trasmission Gate

In	C	Out
0	0	Z
0	1	0
1	0	Z
1	1	1

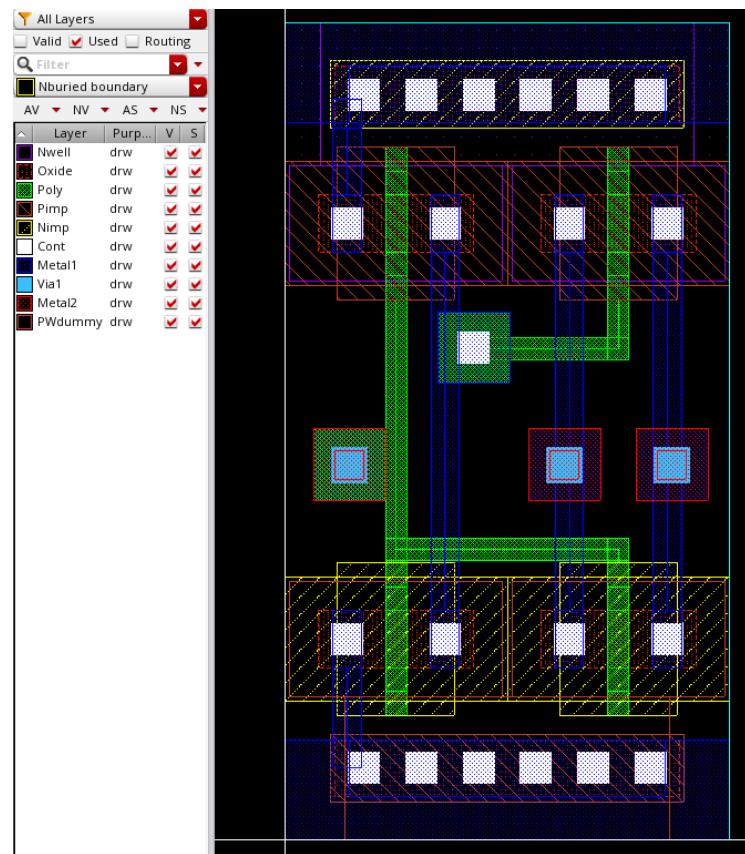
Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.41 – Test Bench da Transmission Gate



Por fim, foi desenvolvido o Layout da Transmission Gate.

Figura 3.42 – Layout da Transmission Gate



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

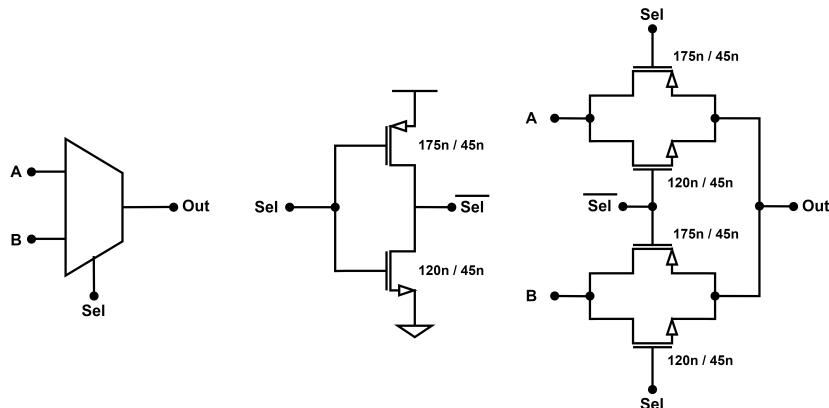
Tabela 3.22 – Área da Transmission Gate

Área Standard Cells ( $\mu m^2$ )	Área Full Custom ( $\mu m^2$ )
1.5903	1.5903

### 3.12 MULTIPLEXADOR

O Multiplexador desenvolvido pode ser visualizado na figura 3.43

Figura 3.43 – Símbolo e Circuito Esquemático do Multiplexador



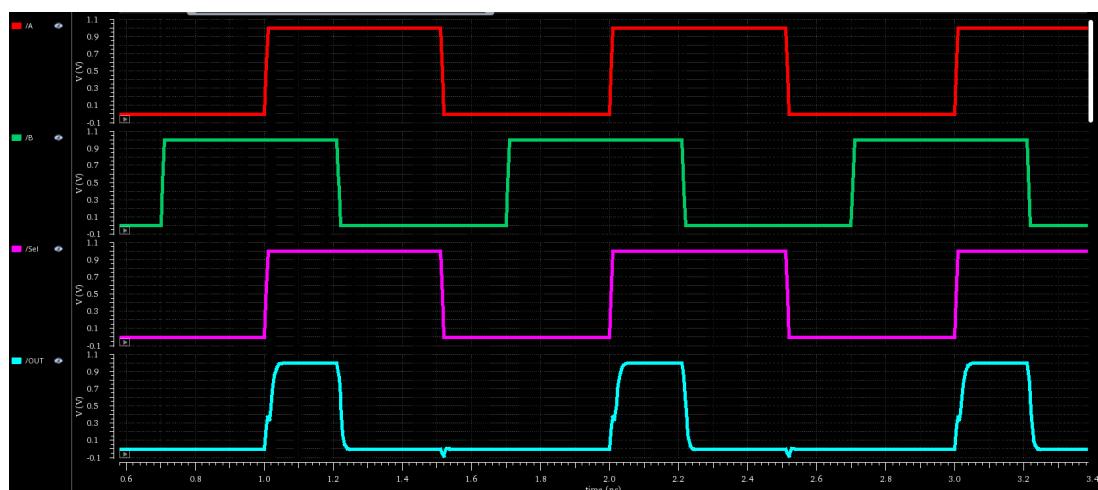
Fonte: Produzido pelos autores.

Tabela 3.23 – Tabela Verdade do Multiplexador

A	B	Sel	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.44 – Test Bench do Mux



Em sequência, foi realizada a descrição funcional da porta lógica.

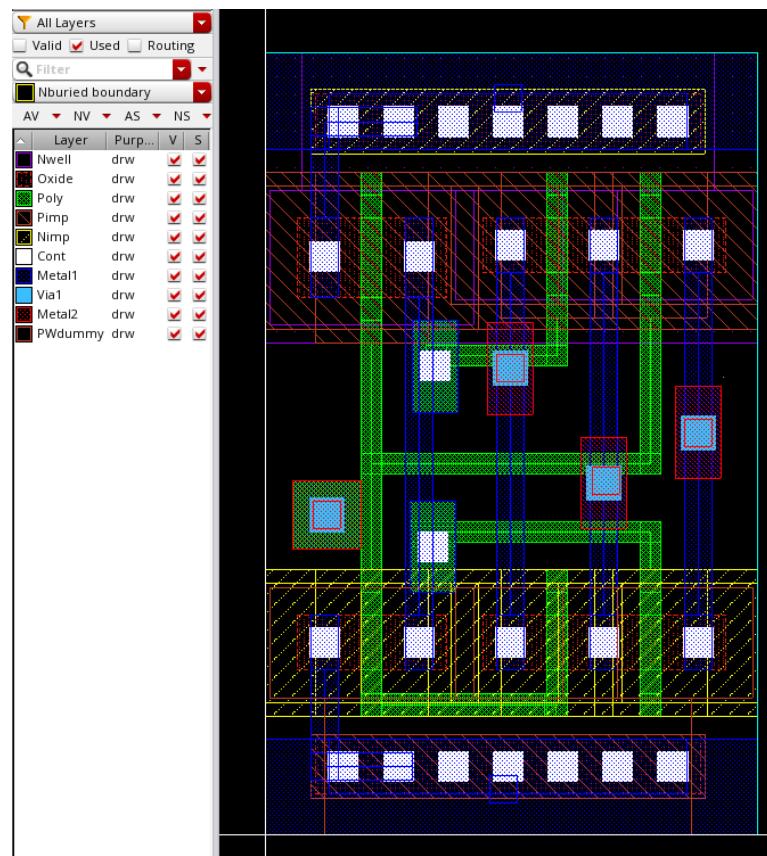
Figura 3.45 – Functional do Mux

```
// type:  
`timescale 1ns/1ps  
`celldefine  
module MUX_2x1 (OUT, A, B, Sel);  
    output OUT;  
    input A, B, Sel;  
  
    // Function  
    wire int_fwire_0, int_fwire_1, S0__bar;  
  
    not (S0__bar, Sel);  
    and (int_fwire_0, S0__bar, A);  
    and (int_fwire_1, Sel, B);  
    or (OUT, int_fwire_1, int_fwire_0);  
  
    // Timing  
    specify  
        (A => OUT) = 0;  
        (B => OUT) = 0;  
        (posedge Sel => (OUT:Sel)) = 0;  
        (negedge Sel => (OUT:Sel)) = 0;  
    endspecify  
endmodule  
`endcelldefine
```

Fonte: Produzido pelos autores.

Por fim, foi desenvolvido o Layout do Multiplexador.

Figura 3.46 – Layout do Mux



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

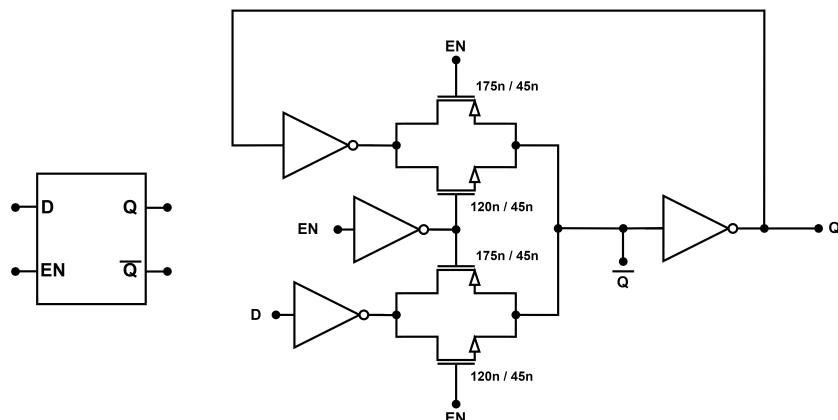
Tabela 3.24 – Área do Multiplexador

Área Standard Cells ( $\mu m^2$ )	Área Full Custom ( $\mu m^2$ )
4.0014	1.8382

### 3.13 LATCH D

O Latch D desenvolvido pode ser visualizado na figura 3.47

Figura 3.47 – Símbolo e Circuito Esquemático do Latch D



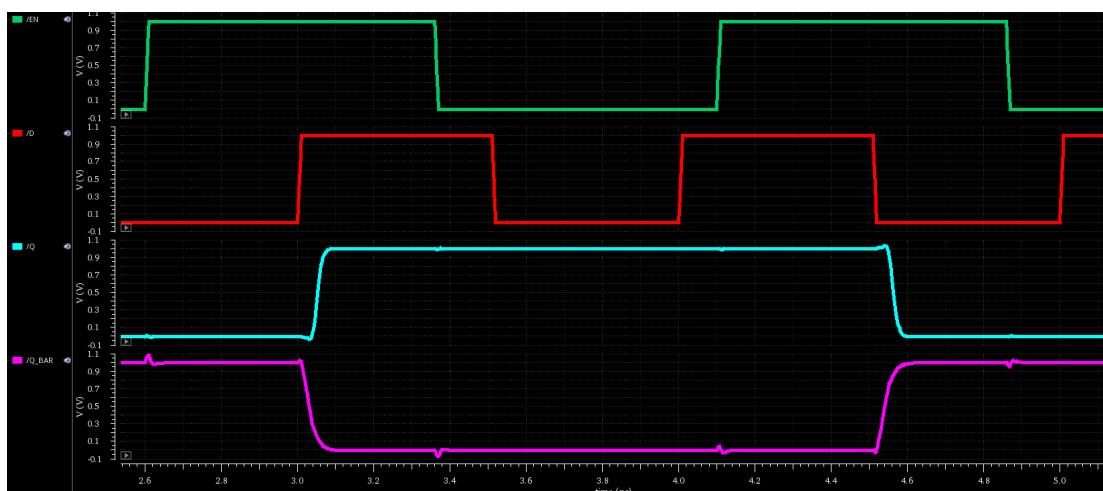
Fonte: Produzido pelos autores.

Tabela 3.25 – Tabela Verdade do Latch D

D	EN	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	Q	$\bar{Q}$
1	1	1	0

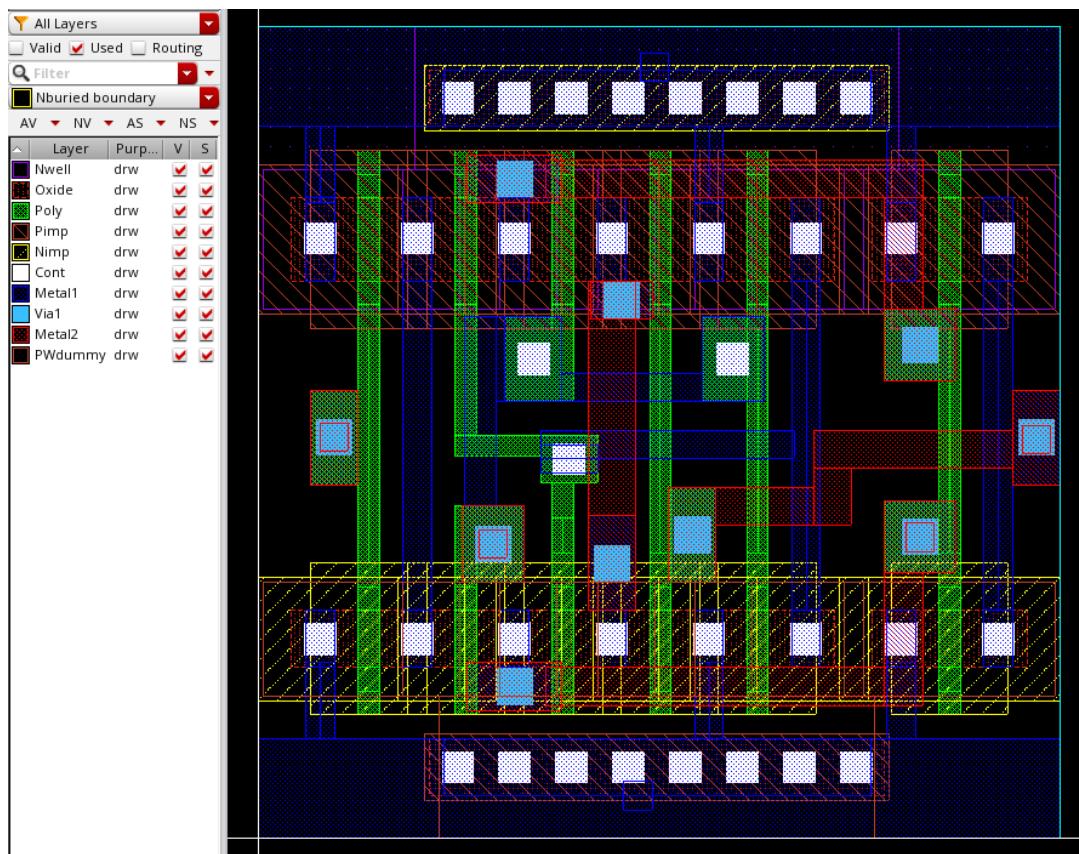
Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.48 – Test Bench do Latch D



Por fim, foi desenvolvido o Layout do Latch D.

Figura 3.49 – Layout do Latch D



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

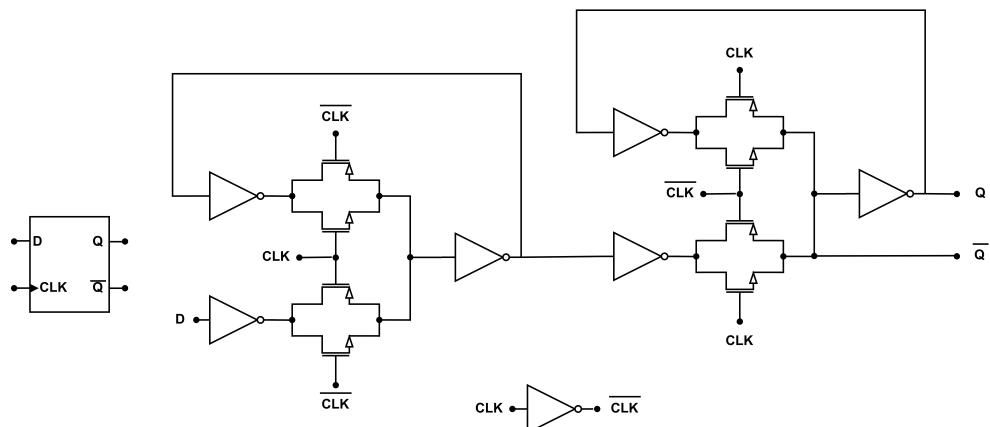
Tabela 3.26 – Área do Latch D

Área Standard Cells ( $\mu\text{m}^2$ )	Área Full Custom ( $\mu\text{m}^2$ )
6.4638	2.8899

### 3.14 FLIP-FLOP D

O Flip-Flop D desenvolvido pode ser visualizado na figura 3.50

Figura 3.50 – Símbolo e Circuito Esquemático do Flip-Flop D



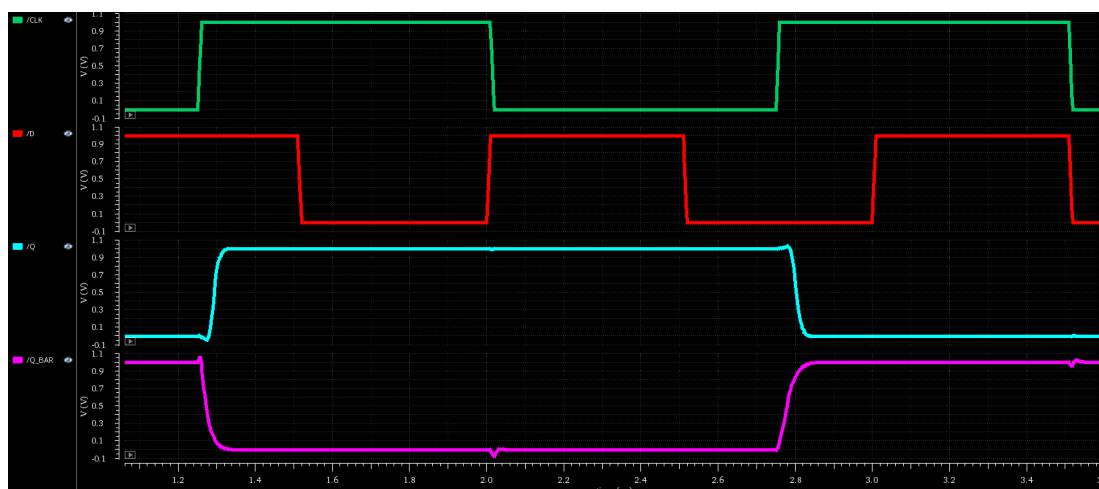
Fonte: Produzido pelos autores.

Tabela 3.27 – Tabela Verdade do Flip-Flop D

D	$\uparrow$ CLK	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	Q	$\bar{Q}$
1	1	1	0

Foi realizado o Test Bench da porta com uma inversora mínima como carga para comprovar seu funcionamento no melhor caso de carga possível.

Figura 3.51 – Test Bench do Flip-Flop D



Em sequência, foi realizada a descrição funcional da porta lógica.

Figura 3.52 – Functional do Flip-Flop D

```
// type:
`timescale 1ns/1ps
`celldefine
module Multiplexer_FF (Q, _Q, D, CLK);
    output Q, _Q;
    input D, CLK;
    reg notifier;
    wire delayed_D, delayed_CK;

    // Function
    wire int_fwire_N30, int_fwire_QBINT, xcr_0;

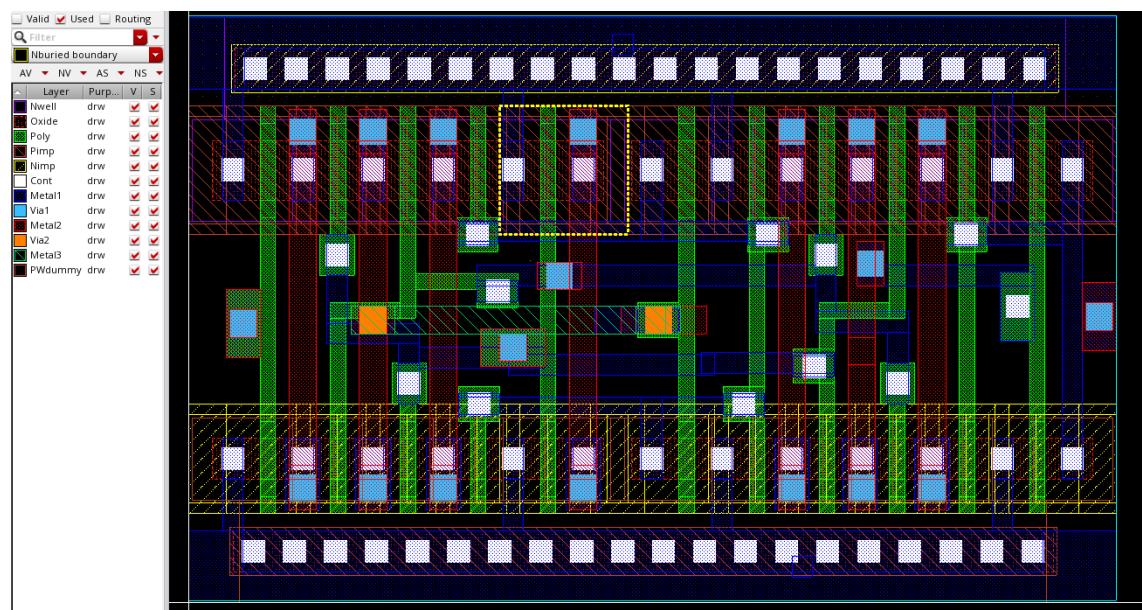
    altos_dff_err (xcr_0, delayed_CK, delayed_D);
    altos_dff (int_fwire_N30, notifier, delayed_CK, delayed_D, xcr_0);
    buf (Q, int_fwire_N30);
    not (int_fwire_QBINT, int_fwire_N30);
    buf (_Q, int_fwire_QBINT);

    // Timing
    specify
        (posedge CLK => (Q+:D)) = 0;
        (posedge CLK => (_Q:D)) = 0;
        $setuphold (posedge CLK, posedge D, 0, 0, notifier,,, delayed_CK, delayed_D);
        $setuphold (posedge CLK, negedge D, 0, 0, notifier,,, delayed_CK, delayed_D);
        $width (posedge CLK, 0, 0, notifier);
        $width (negedge CLK, 0, 0, notifier);
    endspecify
endmodule
`endcelldefine
```

Fonte: Produzido pelos autores.

Por fim, foi desenvolvido o Layout do Flip-Flop D.

Figura 3.53 – Layout do Flip-Flop D



Fonte: Produzido pelos autores.

O layout foi feito visando a menor área possível, na tabela abaixo é possível comparar o layout através de Standard Cells rotiadas com o layout Full Custom.

Tabela 3.28 – Área do Flip-Flop D

Área Standard Cells ( $\mu m^2$ )	Área Full Custom ( $\mu m^2$ )
12.1068	4.6426

## 4 LIBRARY EXCHANGE FORMAT .LEF

### 4.1 INTRODUÇÃO

O *Library Exchange Format*, é um arquivo de dados em formato ASCII, e é utilizado para a especificação detalhada de uma biblioteca de células padrão [3], fornecendo informações críticas relativas às regras de design para o roteamento e ao resumo das células. O arquivo no formato LEF (*Library Exchange Format*) compreende diversas seções distintas e fundamentais:

- **Tecnologia:** Esta seção engloba parâmetros essenciais, tais como camadas, regras de design, definições de vias e capacidade de metal.
- **Site:** Nesta seção, são especificadas extensões de site relevantes.
- **Macros:** Esta é uma parte crucial do arquivo LEF, abrangendo descrições detalhadas das células, dimensões das células, disposição de pinos e bloqueios, bem como informações sobre capacidades.

A descrição da tecnologia é articulada por meio de declarações de Camada e Via. Para cada camada, uma série de atributos é associada, incluindo:

- **Tipo:** Podendo ser categorizado como roteamento, corte (contato), masterslice (poli, ativo) ou sobreposição.
- **Regras de largura/afastamento/espaçamento.**
- **Direção.**
- **Resistência e capacidade por unidade quadrada.**
- **Fator de antena.**

Essa estrutura fornece uma abordagem técnica abrangente, delineando os elementos essenciais para a compreensão e implementação eficiente da biblioteca de células padrão.

#### 4.1.1 LEF da tecnologia

A parte de Tecnologia (Technology) do arquivo LEF contém informações sobre todas as interconexões de metal, dados sobre vias e regras de design relacionadas, enquanto a parte de célula (Cell) do arquivo LEF contém informações relacionadas à geometria de cada célula. Um exemplo ilustrativo está apresentado abaixo para demonstrar as informações contidas na parte de Tecnologia do LEF.

Figura 4.1 – LEF da tecnologia

```
VERSION 5.7 ;
BUSBITCHARS "[]" ;
DIVIDERCHAR "/" ;

PROPERTYDEFINITIONS
    LAYER LEF58_TYPE STRING ;
    LAYER LEF58_ENCLOSURE STRING ;
    LAYER LEF58_SPACING STRING ;
    LAYER LEF58_WIDTH STRING ;
END PROPERTYDEFINITIONS

UNITS
    CAPACITANCE PICOFARADS 1 ;
    DATABASE MICRONS 2000 ;
END UNITS
MANUFACTURINGGRID 0.005 ;
LAYER OVERLAP
    TYPE OVERLAP ;
END OVERLAP

LAYER PWell
    TYPE MASTERSLICE ;
    PROPERTY LEF58_TYPE "TYPE PWell ;" ;
    PROPERTY LEF58_SPACING "SPACING 0.3 ;" ;
    PROPERTY LEF58_WIDTH "WIDTH 0.3 ;" ;
END PWell

LAYER Nwell
    TYPE MASTERSLICE ;
    PROPERTY LEF58_TYPE "TYPE NWELL ;" ;
    PROPERTY LEF58_SPACING "SPACING 0.6 ;" ;
    PROPERTY LEF58_WIDTH "WIDTH 0.3 ;" ;
END Nwell
```

Fonte: Produzido pelos autores

#### 4.1.2 LEF da célula

A parte de célula (Cell) do arquivo LEF contém informações relacionadas a cada célula presente na biblioteca de células padrão, organizadas em seções distintas. A título de exemplo, apresenta-se abaixo uma figura para facilitar a compreensão do formato.

Figura 4.2 – LEF da célula

```
MACRO AND_1x
  CLASS CORE ;
  ORIGIN 0 0 ;
  FOREIGN AND_1x 0 0 ;
  SIZE 0.875 BY 1.71 ;
  SYMMETRY X Y ;
  SITE CoreSite ;
  PIN B
    DIRECTION INPUT ;
    USE SIGNAL ;
    PORT
      LAYER Metal1 ;
      RECT 0.41 0.93 0.56 1.08 ;
      LAYER Metal2 ;
      RECT 0.41 0.93 0.56 1.08 ;
      LAYER Vial ;
      RECT 0.45 0.97 0.52 1.04 ;
  END
END B
PIN A
  DIRECTION INPUT ;
  USE SIGNAL ;
  PORT
    LAYER Metal1 ;
    RECT 0.07 0.96 0.22 1.11 ;
    LAYER Metal2 ;
    RECT 0.07 0.96 0.22 1.11 ;
    LAYER Vial ;
    RECT 0.11 1 0.18 1.07 ;
  END
END A
```

Fonte: Produzido pelos autores

Em suma, o arquivo .lef é um arquivo utilizado por ferramentas de roteamento e síntese no fluxo de design, obtendo características de localização dos pinos e regras das standard cells, dessa forma realizando um roteamento correto. Em outras palavras, obtem-se uma vista abstrata do layout da célula.

## 4.2 CRIAÇÃO DO ARQUIVO .LEF PARA O PROJETO

Para obtenção do arquivo .lef referente às células produzidas para o projeto, utilizou-se o software *Abstract*, que realiza de forma automática a geração do arquivo, utilizando os arquivos de layout produzidos.

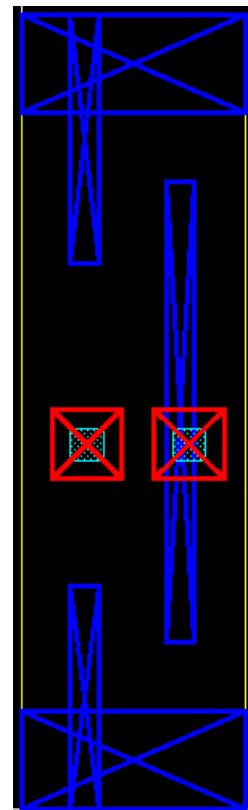
Figura 4.3 – Tela inicial do programa Abstract

Abstract - Logic_Gate_Library							
File	Bins	Cells	Flow				
Bin	Cells	Cells		Layout	Logical	Pins	Extract
Core	19			✓	✓	✓	✓
IO	0			✓	---	✓	---
Corner	0			✓	---	✓	---
Block	0			✓	---	✓	---
Ignore	10			✓	---	✓	---
				✓	✓	✓	✓
				✓	✓	✓	✓
				✓	---	✓	---
				✓	---	✓	---
				✓	---	✓	---
				✓	---	✓	---
				✓	---	✓	---

Fonte: Produzido pelos autores

Através dessa extração, podemos obter a vista abstrata das células, contendo suas informações de localização e pinagem. Abaixo podemos observar a vista de *abstract* obtida, referente à porta inversora.

Figura 4.4 – View de abstract obtida da porta inversora



Fonte: Produzido pelos autores

## 5 LIBERTY TIME FILE (.LIB)

### 5.1 INTRODUÇÃO

O arquivo .lib é uma representação ASCII dos parâmetros de tempo e potência associados à qualquer célula padrão em um processo de fabricação da indústria dos semicondutores [2]. Estes parâmetros são obtidos através de simulações onde as células são impostas diferentes condições de operação, e os resultados destas simulações representados no formato .lib.

O arquivo .lib contém modelos e dados de timing que possibilitam o cálculo de:

- Atrasos nos caminhos de Input e Output;
- Checagem nos valores de timing;
- Atraso em interconexões;

Características como os atrasos de caminho em circuito dependem do comportamento elétrico das interconexões de uma célula. Essa informação pode ser obtida pelo layout do design em questão, mas deve ser estimada mesmo sem a disponibilidade de layout.

O arquivo .lib não pode prever variações no processo de fabricação, tensão e temperatura, porém, para garantia de funcionamento, fatores de redução na capacidade dos circuitos podem ser incluídos de maneira à compensar por essas variações.

### 5.2 CÁLCULO DO ATRASO BASEADO EM CÉLULAS

O cálculo do atraso baseado em células (Cell-based delay calculation) é modelado caracterizando o atraso da célula e o tempo de transição de saída (slew de saída) como uma função do tempo de transição de entrada (slew de entrada) e da carga capacitiva na saída da célula. As verificações de tempo também são funções do slew de entrada e da carga capacitiva de saída.

Cada célula possui um número específico de caminhos de entrada para saída. Os atrasos do caminho podem ser descritos para cada transição do sinal de entrada que afeta um sinal de saída.

O atraso do caminho também pode depender de sinais em outras entradas (dependências de estado). Em muitas células sequenciais, o atraso do caminho de um pino de entrada para um pino de saída pode depender do atraso do caminho de outro pino de saída para este pino de saída.

### 5.3 CALCULO DE ATRASO E CHECAGEM DE TIMING

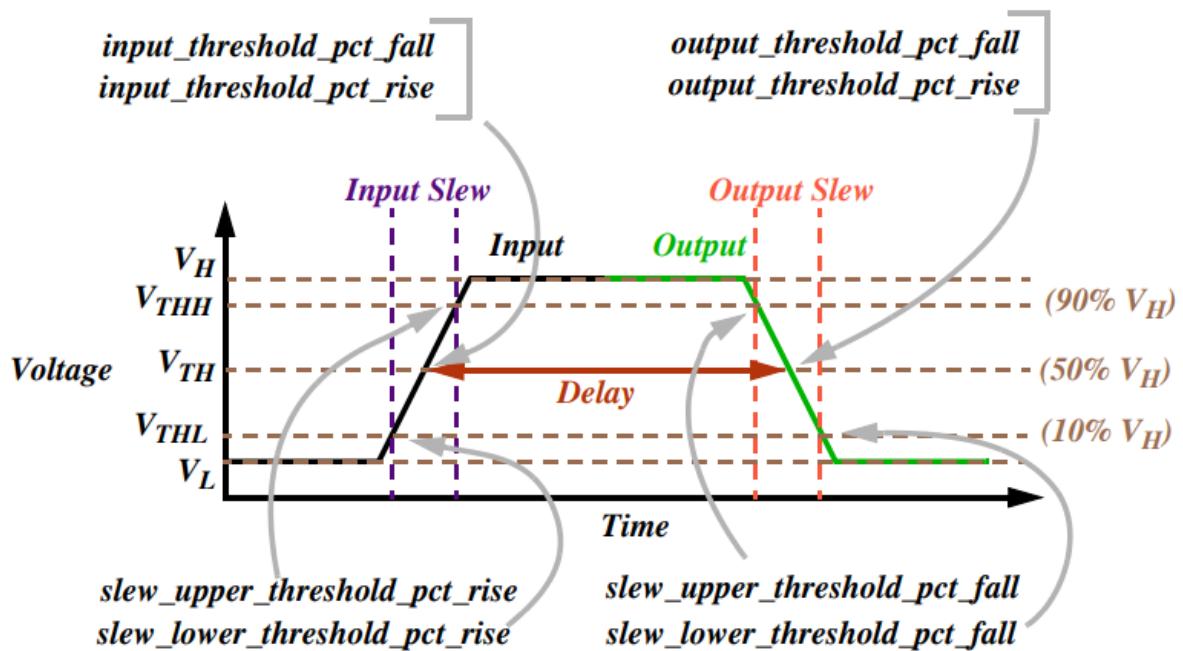
No contexto de projeto VLSI avançado, a Delay Calculation (cálculo de atraso) e Timing Checks (verificações de tempo) são processos fundamentais para assegurar o desempenho eficiente de circuitos integrados complexos. O cálculo de atraso envolve a estimativa

precisa dos atrasos temporais ao longo dos caminhos de sinal, considerando fatores como capacidade, resistência e características do dispositivo. Isso é vital para garantir que os sinais atinjam seus destinos dentro dos limites temporais especificados.

Por outro lado, as Timing Checks referem-se à verificação sistemática de intervalos temporais críticos, como setup e hold times, para garantir a sincronização adequada entre os sinais. Isso inclui a análise de diversas condições operacionais e considerações de layout para prever e corrigir potenciais violações temporais. Ambos os processos são essenciais para otimizar o desempenho e a confiabilidade de dispositivos VLSI avançados, especialmente diante da crescente complexidade e velocidade operacional desses circuitos.

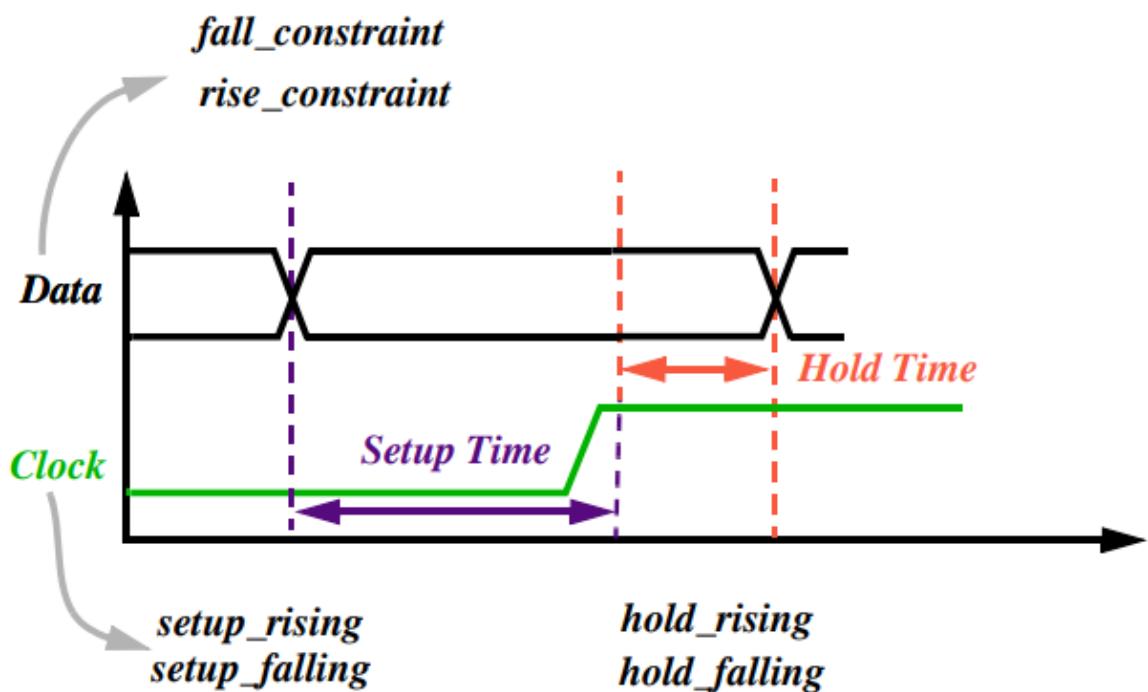
Nas figuras abaixo, podemos observar uma representação de como obter os dados de timing e atraso.

Figura 5.1 – Input-Slew, Output Slew e Cell Delay



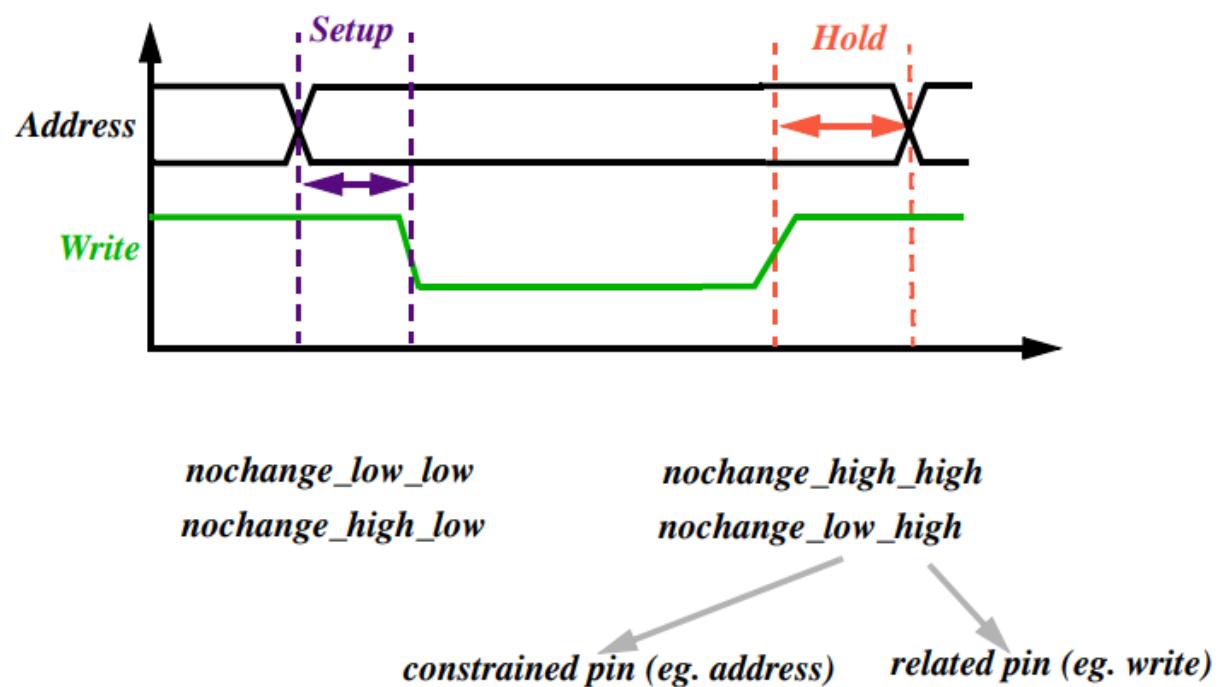
Fonte: [2]

Figura 5.2 – Setup e Hold Time



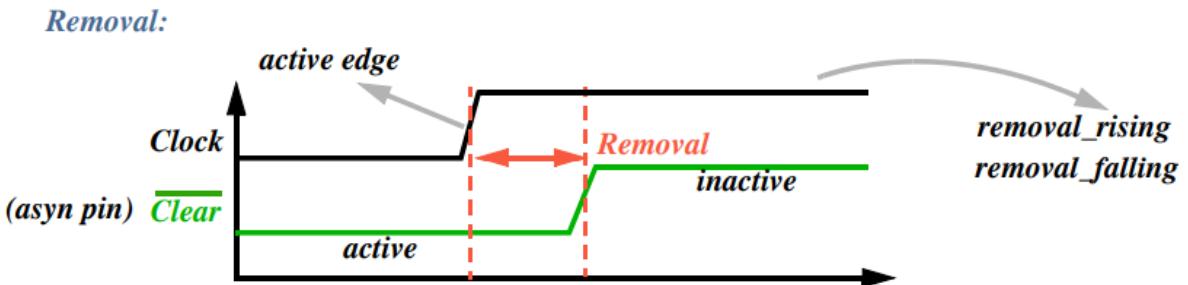
Fonte: [2]

Figura 5.3 – Sem mudança



Fonte: [2]

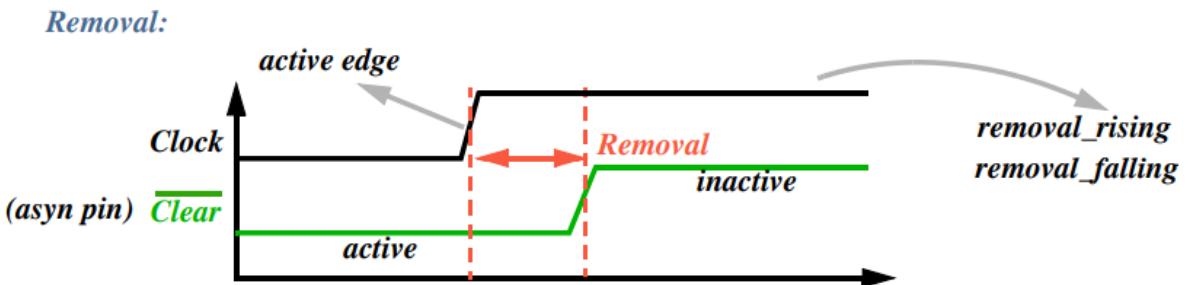
Figura 5.4 – Removal



Fonte: [2]

Teste de texto

Figura 5.5 – Recuperação



Fonte: [2]

## 5.4 INFORMAÇÕES ESSENCIAIS DE UM ARQUIVO .LIB

Em um arquivo de biblioteca .lib são necessárias as seguintes informações:

- Informações de cabeçalho;
- Condições de operação;
- Fatores de derating;
- Limites e unidades;

Normalmente, são requeridos três valores diferentes, representando diferentes casos: Típico, melhor e pior. Porém, para obtenção desses valores são necessário parâmetros e modelos dos transistores e do desvio do processo de fabricação. Informações como essas são disponibilizadas apenas por fabricantes.

Para simulações serão utilizados modelos SPICE, desta forma, é possível variar parâmetros como temperatura de operação e modificar valores com taxas de mais ou menos 5 ou 10 por cento.

Condições de operação, fatores de derating, limites e unidades:

- *library and delay model*: Forneça um nome de biblioteca e o modelo de atraso a ser utilizado. Estaremos utilizando o modelo de busca em tabela (atraso não linear).
- *nom process property*: Especifica os pontos de referência para a escalonamento de processo usados para a caracterização das células. Nosso arquivo conterá valores para apenas um ponto de processo, e assim, será utilizado 1.0. No entanto, podemos criar três arquivos diferentes para os casos típico, pior e melhor.
- *nom temperature and nom voltage*: Especifica os pontos de referência de temperatura e voltagem.
- *operating conditions*: Define os valores de processo, temperatura e voltagem no nível da biblioteca, juntamente com as condições de operação padrão.
- *slew and delay threshold points*: Valores de limite baixo e alto para o cálculo de subida (pontos 10% - 90%) e o limite para cálculos de atraso (pontos 50%)
- *default values for fanout, capacitance, slew*: Especifica os limites de subida máxima de entrada em um pino de entrada, capacitância de pino de entrada/saída e a capacidade máxima de saída em qualquer pino de saída
- *units*: Especifica as unidades utilizadas para tempo, capacidade, potência, tensão, corrente, etc.
- *derating factors and wire-load models*: Como discutido anteriormente, precisamos de informações detalhadas sobre o processo, bem como parasitas extraídos de designs anteriores.
- *Lookup table templates*: Defina modelos de informações comuns a serem usados em tabelas de pesquisa. Esses modelos são estabelecidos para arcos de temporização, verificações de potência e temporização que serão incluídos nas definições de células.

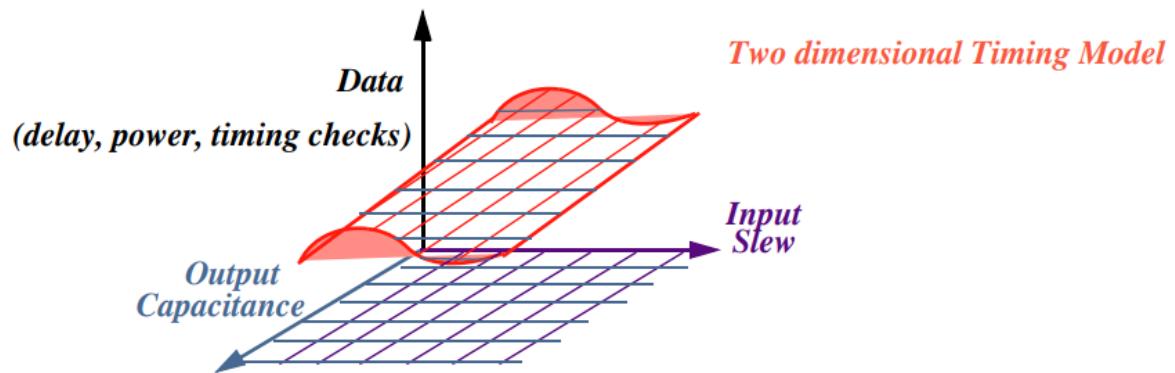
Definições da célula:

- *Cell(cell\_name)*: O nome dado à célula;
- *Area*: Especifica a área usada pela célula, utilizada durante a síntese lógica e a análise de temporização, sem unidades;
- *<lookup tables>(lookup\_table\_template\_name)*: Especifica os modelos de temporização, potência e checagem a serem utilizados no caminho específico do circuito.

## 5.5 MODELO EM DUAS DIMENSÕES

Os dois eixos variáveis e independentes são o input slew e a capacidade de carga da saída.

Figura 5.6 – Representação dos parâmetros obtidos em um plano 2d



Fonte: [2]

A seguir, vemos um exemplo das tabelas que realizam a modelagem 2d da célula:

Figura 5.7 – Cell Fall

```
cell_fall(fall_template_name n x m)
    index_1 (value1, value2, ... , value n)
    index_2 (value1, value2, ... , value m)
    values (
        data_max11:data_typ11:data_min11, ..., data_max1m:data_typ1m:data_min1m \
        ..... \
        data_maxn1:data_typn1:data_minn1, ..., data_maxnm:data_typnm:data_minnm);
```

Fonte: [2]

Figura 5.8 – Representação dos pin names

***pin(pin\_name)***

*direction* : *input, output, inout, internal*

*clock\_pin*

*function(expression)*

Used for output or bidirectional pins. The expression defines the value of the output pin as a function of input pins

***max\_capacitance***

The maximum output capacitive load that an output pin can drive

***capacitance***

The capacitive load of an input, inout, output or internal pin. Usually defined as 0 for output pins

***internal\_power()***

Output pins in combinational cells, define the rise\_power and fall\_power to a related input pin. Input and clock pins also define this in sequential cells

***timing()***

Output pins in combinational cells, define the rise\_delay, fall\_delay, rise\_transition and fall\_transition to a related input pin

Fonte: [2]

Figura 5.9 – Exemplo de representação

***timing() contd.***

Timing checks are also defined for sequential cells

***pulse width definitions, recovery, removal***

Required for clocks, asynchronous set and reset pins

Fonte: [2]

Para a definição de flip flops e latches, é necessário usar a definição de grupos ff e latch.

Figura 5.10 – Representação de Flip-Flops

```
ff(<state of noninverting output>, <state of inverting output>) {  
    clocked_on: <clock pin name>  
    next_state: <input combination that produces the next state>  
    clear: <active value of the clear input>  
    preset: <active value of the preset input>  
    clear_preset_var1: <value of noninverting output when both active>  
    clear_preset_var2: <value of inverting output when both active> }  
latch()  
latch is similar but requires enable and data in, instead of clock and next state
```

Fonte: [2]

Para teste e debugging de circuitos, são utilizados scan cells, as scan cells são definidas dentro dos grupos ff e latch, utilizando dois grupos ff, um na célula e outro dentro do grupo de test\_cell.

Figura 5.11 – Representação de scan cell

```
test_cell(){  
    Inside the test_cell group all pins are defined and test related pins are given a  
    signal_type or test_output_only attribute  
    signal_type can be:  
        test_scan_in: scan input pin  
        test_scan_in_inverted: inverted scan input  
        test_scan_out: scan output pin  
        test_scan_out_inverted: inverted scan output  
        test_scan_enable: high on this pin puts it in test mode (scan and shift)  
        test_scan_enable_inverted: same as above but inverted  
        test_scan_clock: test scan clock for clocked-scan  
        other clocks defined for LSSD scan}
```

Fonte: [2]

## 5.6 CRIAÇÃO DO ARQUIVO .LIB PARA O PROJETO

Para realização do arquivo .lib para a biblioteca de standard cells da disciplina, utilizamos do arquivo .lib disponível na biblioteca do processo tecnológico de 45nm utilizado no GPKD Cadence, especificamente na biblioteca GSCLIB (General Standard Cell Library).

Utilizando o arquivo .lib disponível na biblioteca, sobrescrevemos o arquivo com informações referentes às portas confeccionadas durante a disciplina, de forma que, ao realizar a síntese lógica, arquivo .lib estaria representando a operação das mesmas. Abaixo podemos ver uma parte da representação de uma das portas no arquivo .lib:

Figura 5.12 – Parte da representação .lib da porta inversora

```
cell (NOT_1x) {
    area : 0.684;
    pg_pin (vdd!) {
        pg_type : primary_power;
        voltage_name : "vdd!";
    }
    pg_pin (gnd!) {
        pg_type : primary_ground;
        voltage_name : "gnd!";
    }
    leakage_power () {
        value : 37.1549;
        when : "IN";
        related_pg_pin : vdd!;
    }
    leakage_power () {
        value : 55.2757;
        when : "!IN";
        related_pg_pin : vdd!;
    }
    leakage_power () {
        value : 46.2153;
        related_pg_pin : vdd!;
    }
    pin (OUT) {
        direction : "output";
        function : "(!IN)";
        related_ground_pin : gnd!;
        related_power_pin : vdd!;
        max_capacitance : 0.25;
        timing () {
            related_pin : "IN";
            timing_sense : negative_unate;
            timing_type : combinational;
            cell_rise (delay_template_2x2) {
                index_1 ("0.008, 0.28");
                index_2 ("0.01, 0.25");
                values ( \
                    "0.0430046, 0.929301", \
                    "0.125925, 1.05666" \
                );
            }
        }
    }
}
```

Fonte: Produzido pelos autores

Para teste, utilizamos apenas a porta inversora, substituindo o nome da célula e seus pinos para aqueles utilizados na produção de portas lógicas realizadas durante a disciplina.

Para que pudessemos sintetizar circuitos mais complexos e de maneira mais otimizada, foram adicionados elementos de memória (Flip Flops tipo D) e um multiplexador, também foi adicionado uma porta NAND para construção de lógica.

## 6 CIRCUITOS UTILIZADOS PARA VALIDAÇÃO

O processo de desenvolvimento e validação de circuitos digitais é essencial para garantir que um design atenda aos requisitos funcionais antes da implementação física. Neste contexto, a linguagem de descrição de hardware VHDL, a ferramenta GHDL e o visualizador GTKWave foram empregados para criar e testar circuitos, assegurando que a síntese lógica ocorra de maneira eficaz.

O processo começou com a descrição do(s) circuito(s) em VHDL. Nesse estágio, utilizou-se a linguagem VHDL para modelar a lógica digital, estabelecer hierarquias de design, e definir a interconexão entre componentes. Essa descrição foi a base para a simulação e a futura síntese lógica.

Após a descrição do circuito em VHDL, os códigos foram compilados utilizando a ferramenta GHDL. A compilação envolveu a transformação do código VHDL em uma representação que poderia ser executada em uma simulação digital. Esse processo é fundamental para a preparação do design para a simulação.

A fase de elaboração foi conduzida para criar um modelo executável do design. Esse estágio envolveu a integração das diferentes unidades de design compiladas em uma única entidade, proporcionando um modelo completo do circuito para fins de simulação.

A simulação dos circuitos foi realizada utilizando o GHDL. Esse processo permitiu a análise do comportamento do design em diferentes condições de entrada. A execução da simulação gerou dados que poderiam ser posteriormente visualizados.

O GTKWave foi empregado como ferramenta de visualização para analisar os resultados da simulação. Essa interface gráfica permitiu a observação das formas de onda, transições lógicas e outros parâmetros relevantes para uma compreensão detalhada do comportamento do circuito.

A utilização do GTKWave facilitou a análise detalhada dos resultados da simulação. O visualizador proporcionou uma representação gráfica intuitiva das formas de onda, permitindo a identificação de possíveis problemas ou comportamentos indesejados.

Ao seguir esse processo iterativo de descrição em VHDL, compilação, elaboração, simulação e visualização, foi possível testar e validar os circuitos antes da síntese lógica. Essa abordagem proativa foi crucial para garantir que o design atendia aos requisitos estabelecidos antes de avançar para etapas subsequentes do processo de desenvolvimento de hardware digital.

Foram adotadas duas metodologias de descrição para testar a síntese posteriormente, uma descrição estrutural e outra descrição comportamental.

Na descrição estrutural, o foco está na organização física e na interconexão dos componentes que compõem o circuito. Nela, descrevemos como os diversos elementos de hardware estão conectados para formar o sistema completo. Os componentes estruturais podem incluir portas lógicas, flip-flops, multiplexadores, entre outros. A descrição estrutural é frequentemente utilizada para representar designs complexos em um nível mais baixo de abstração.

Na descrição comportamental, o foco é na funcionalidade do circuito, sem enfatizar a estrutura interna. Nessa abordagem, você descreve o comportamento do sistema em termos de operações e relações lógicas, sem entrar nos detalhes específicos dos componentes internos. Essa abordagem é mais abstrata e facilita a compreensão do propósito do circuito.

## 6.1 OSCILADOR

Para testar a descrição estrutural, criamos um circuito eletrônico utilizado para gerar sinais oscilantes, conhecido como oscilador em anel. Este tipo de oscilador é conhecido por sua simplicidade e eficácia em diversas aplicações, desde circuitos de clock em microprocessadores até geradores de sinais em sistemas de comunicação.

O oscilador em anel é construído com uma cadeia (ou anel) de inversores, onde a saída de cada inversor é conectada à entrada do próximo.

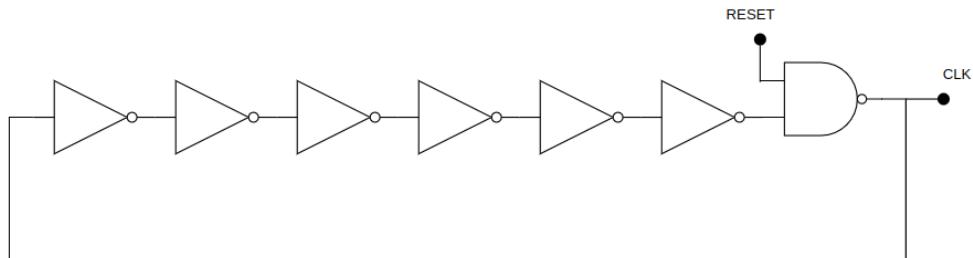
Inicialmente, um sinal de entrada (por exemplo, um pulso) é aplicado ao primeiro inversor do anel. O sinal percorre o anel, passando por cada inversor, invertendo sua polaridade a cada estágio. A retroalimentação positiva garante que o sinal continue circulando no anel, gerando uma onda oscilante contínua.

O atraso de propagação em cada inversor é crucial para o funcionamento do oscilador em anel. Ele deve ser suficiente para garantir que o sinal percorra todo o anel antes de iniciar um novo ciclo.

A frequência de oscilação é determinada pelo tempo que o sinal leva para dar uma volta completa no anel. Pode ser ajustada variando-se o número de estágios (inversores) no anel.

Na figura 6.1 pode ser visualizado o esquema elétrico do oscilador descrito.

Figura 6.1 – Circuito elétrico do Oscilador em anel



Fonte: Os Autores.

Inicialmente, foi descrita uma simples inversora, que pode ser visualizada na figura 6.2.

Figura 6.2 – Descrição em VHDL da porta lógica NOT

```
library ieee;
use ieee.std_logic_1164.all;

-- Descrição da inversora
entity inv is
    Port ( input  : in std_logic;
           output : out std_logic
        );
end entity inv;

architecture Behavioral_inv of inv is
begin
    output <= not input;
end architecture Behavioral_inv;
```

Fonte: Os Autores.

Prosssegui-se com a descrição de uma porta lógica nand para servir de controle do oscilador e criar o sinal de início da osilação. Esta, que pode ser visualizada na figura 6.3.

Figura 6.3 – Descrição em VHDL da porta lógica NAND

```
library ieee;
use ieee.std_logic_1164.all;

-- Descrição da nand
entity nand_gate is
    Port ( a : in std_logic;
           b : in std_logic;
           y : out std_logic
        );
end entity nand_gate;

architecture Behavioral_nand of nand_gate is
begin
    y <= not (a and b);
end architecture Behavioral_nand;
```

Fonte: Os Autores.

Por último, as duas portas descritas anteriormente foram instanciadas na entidade de topo do oscilador, onde foram realizadas as conexões apropriadas. O resultado da descrição do oscilador pode ser visualizado na figura 6.4.

Figura 6.4 – Descrição em VHDL do oscilador

```
-- Descrição: Oscilador em anel para trabalho da disciplina CCI-2023
-- Autores: Anderson, Artur e Murilo
-- Versão: 1.0

library ieee;
use ieee.std_logic_1164.all;

-- Descrição do oscilador
entity oscilador is
    port (
        rst : in std_logic := '0';
        clk : out std_logic
    );
end entity oscilador;

architecture flow of oscilador is
    signal inv_outs : std_logic_vector(4 downto 0) := (others => '0');
    signal nand_out : std_logic := '0';
    signal nand_inv : std_logic := '0';

    component inv is
        port ( input : in std_logic;
               output : out std_logic
        );
    end component inv;

    component nand_gate is
        port ( a : in std_logic;
               b : in std_logic;
               y : out std_logic
        );
    end component nand_gate;

begin
    nand_gate_inst: nand_gate port map (rst, nand_inv, nand_out);

    inversor_0: inv port map (nand_out, inv_outs(0));
    inversor_1: inv port map (inv_outs(0), inv_outs(1));
    inversor_2: inv port map (inv_outs(1), inv_outs(2));
    inversor_3: inv port map (inv_outs(2), inv_outs(3));
    inversor_4: inv port map (inv_outs(3), inv_outs(4));
    inversor_5: inv port map (inv_outs(4), nand_inv);

    clk <= nand_out;

end architecture flow;
```

Fonte: Os Autores.

Não foi possível realizar a simulação deste oscilador diretamente pela compilação do código em VHDL, pois o simulador considera-o como ideal sem contar com tempos de propagação das portas lógicas e o estado inicial delas é indefinido. Contudo, a descrição foi coerente com o circuito elétrico projetado e a análise do ghdl demonstrou que não houveram erros de sintaxe.

## 6.2 CONTADOR/PRESCALER

Na descrição comportamental foi realizada uma descrição em VHDL de um contador de 12 bits que opera como um prescaler. O prescaler é uma técnica frequentemente empregada em eletrônica para dividir a frequência de um sinal, sendo valioso em aplicações

como medições de frequência, controle de temporizadores e modulação de sinais.

Este prescaler gera saídas sequenciais, cada uma delas operando em uma frequência menor que a anterior, de acordo com a potência de 2.

O prescaler utiliza um contador de 12 bits. Esse contador é incrementado a cada pulso do sinal de clock de entrada. Cada uma das 12 saídas é associada a um bit específico do contador. A primeira saída está associada ao bit mais significativo (MSB), e a última saída ao bit menos significativo (LSB).

A sequência de ativação segue uma ordem decrescente, onde a primeira saída é ativada a cada dois pulsos, a segunda saída a cada quatro pulsos, e assim por diante.

Ex: Se  $clk = 1MHz$ , então,  $Cont\_bin[0] = 500kHz$ ,  $Cont\_bin[1] = 250kHz$ ,  $Cont\_bin[2] = 125kHz$  e assim por diante.

Na figura 6.5 pode ser visualizada a descrição comportamental em VHDL do contador como prescaler.

Figura 6.5 – Descrição em VHDL do contador

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Contador is
    port (
        Cont_bin : out std_logic_vector(11 downto 0);
        clk : in std_logic
    );
end entity Contador;

architecture flow of Contador is
begin
    process (clk)
        variable temp : unsigned(11 downto 0):=(others =>'0');
    begin
        if (rising_edge(clk)) then
            temp := temp + 1;
        end if;

        Cont_bin <= std_logic_vector(temp);
    end process;
end architecture;
```

Fonte: Os Autores.

## 7 SÍNTESE LÓGICA

Para a Síntese Lógica, foi utilizado o software Genus. O Genus, uma ferramenta de síntese lógica desenvolvida pela *Cadence*, desempenha um papel crucial no fluxo de design digital. Essa ferramenta avançada é projetada para realizar a síntese lógica, convertendo descrições de hardware em linguagens de descrição de hardware (HDL), como VHDL ou Verilog, em uma representação de nível lógico otimizada.

Ao lidar com uma variedade de descrições de design, incluindo RTL (Register-Transfer Level) e gate-level netlists, o Genus emprega técnicas avançadas de otimização global e local. O objetivo é melhorar a área, a velocidade e o consumo de energia do design, garantindo eficiência e desempenho ideais.

Essa ferramenta é parte integrante do fluxo de design da Cadence e se integra a outros produtos, como o Encounter Digital Implementation System, o Innovus Implementation System e o Tempus Timing Signoff Solution. Essa integração contínua possibilita um fluxo de trabalho eficiente e coeso.

Para realizar a síntese de maneira adequada é necessário fornecer a descrição do design em uma linguagem de descrição de hardware (HDL), como VHDL ou Verilog, no nosso caso foi adotada a linguagem VHDL.

Deve-se especificar as restrições temporais do design caso necessário, como restrições de setup e hold. Essas restrições são cruciais para garantir que o design atenda aos requisitos de temporização desejados.

Deve-se, ainda, informar os parâmetros específicos da tecnologia que está sendo utilizada. Isso inclui informações sobre o processo de fabricação, como tamanho da célula, velocidades de transição, capacidades, consumo das portas lógicas entre outros.

Se necessário, deve-se definir as metas de otimização para o Genus. Essas metas podem incluir a ênfase em área, desempenho (velocidade), consumo de energia ou uma combinação desses fatores.

Caso o seu design for hierárquico, como por exemplo o oscilador descrito neste projeto, devemos fornecer informações sobre a hierarquia das diferentes partes do design.

Dentro do contexto do Genus da Cadence, os arquivos .lib e .lef desempenham papéis específicos durante o processo de síntese lógica e design dos circuitos.

Os arquivos .lib no contexto do Genus contêm informações cruciais sobre a biblioteca de células padrão que será utilizada durante a síntese lógica. Essas bibliotecas contêm características elétricas e temporais das células padrão, essenciais para as fases de otimização de área, potência e desempenho. O Genus utiliza as informações dos arquivos .lib para realizar a otimização global e local, escolhendo as células mais adequadas para cada caso.

Os arquivos .lef no contexto do Genus fornecem dados de layout físico das células padrão presentes na biblioteca. O Genus utiliza as informações dos arquivos .lef para garantir que a disposição física das células respeite as restrições de layout, evitando violações de regras de design e garantindo uma implementação bem-sucedida.

## 7.1 OSCILADOR

Na figura 7.1 abaixo pode ser visualizado o script que foi criado para realizar a síntese lógica do circuito do oscilador descrito em VHDL. O script em questão foi rodado na ferramenta GENUS.

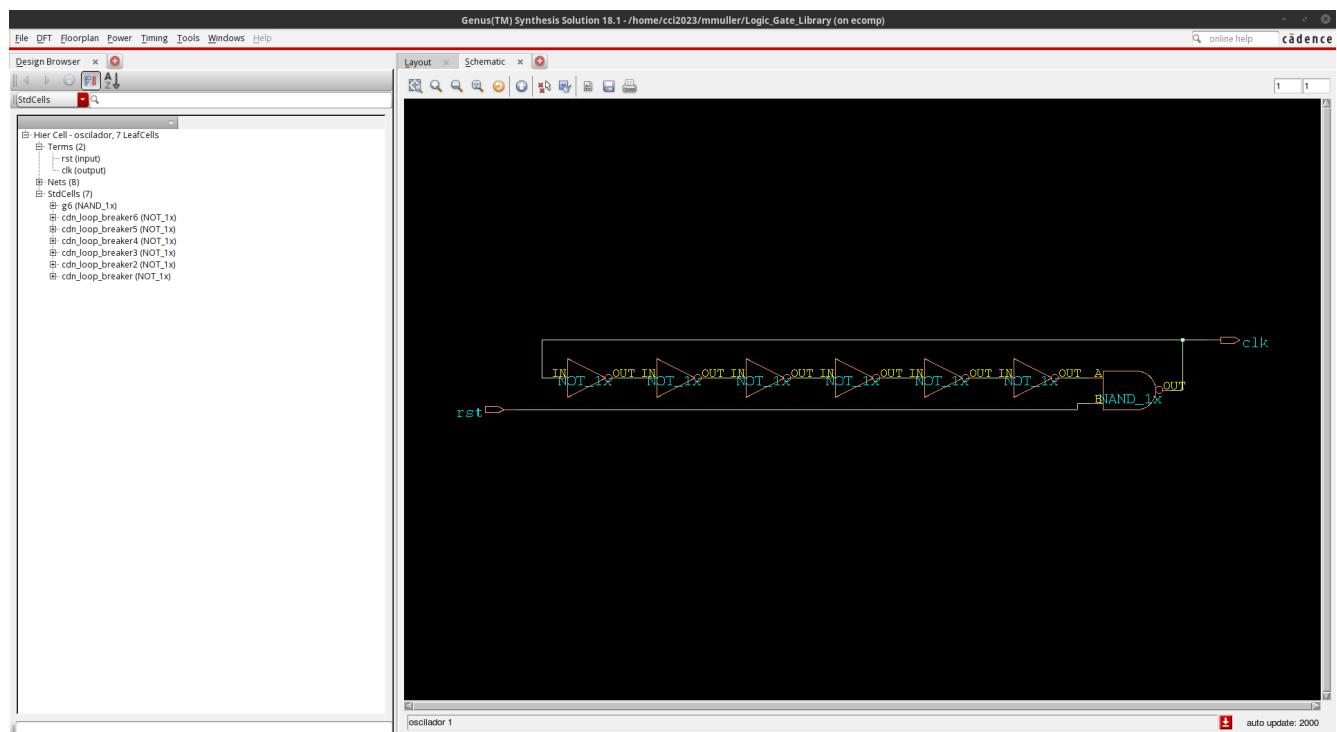
Figura 7.1 – Script tcl do genus para realizar a síntese lógica do oscilador

```
set_db library /home/cci2023/mmuller/Logic_Gate_Library/fast_vdd1v0_basicCells_cci.lib
set_db lef_library /home/cci2023/mmuller/Logic_Gate_Library/abstract.lef
read_hdl -vhdl /home/cci2023/mmuller/Logic_Gate_Library/inv.vhd nand_gate.vhd oscilador.vhd
elaborate
syn_generic
syn_map
write_hdl > /home/cci2023/mmuller/Logic_Gate_Library/oscilador.v
```

Fonte: Os Autores.

O resultado da síntese lógica do oscilador pode ser visualizado na figura 7.2.

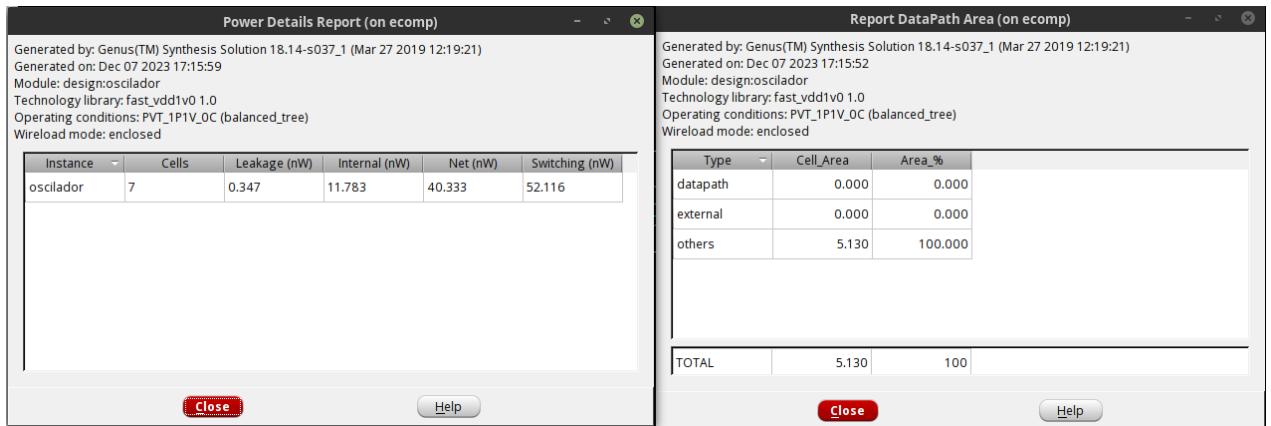
Figura 7.2 – Vista esquemática do oscilador após síntese lógica



Fonte: Os Autores.

Na figura 7.3 podem ser visualizadas as informações de área e potência estimadas neste circuito.

Figura 7.3 – Estimativas de área e potência do circuito oscilador



Fonte: Os Autores.

## 7.2 CONTADOR/PRESCALER

Na figura 7.4 abaixo. pode ser visualizado o script que foi criado para realizar a síntese lógica do circuito do Contador/Prescaler descrito em VHDL.

Figura 7.4 – Script tcl do genus para realizar a síntese lógica do oscilador

```
set_db library /home/cci2023/mmuller/Logic_Gate_Library/fast_vdd1v0_basicCells_cci.lib
set_db lef_library /home/cci2023/mmuller/Logic_Gate_Library/abstract.lef

read_hdl -vhdl /home/cci2023/mmuller/Logic_Gate_Library/Contador.vhd

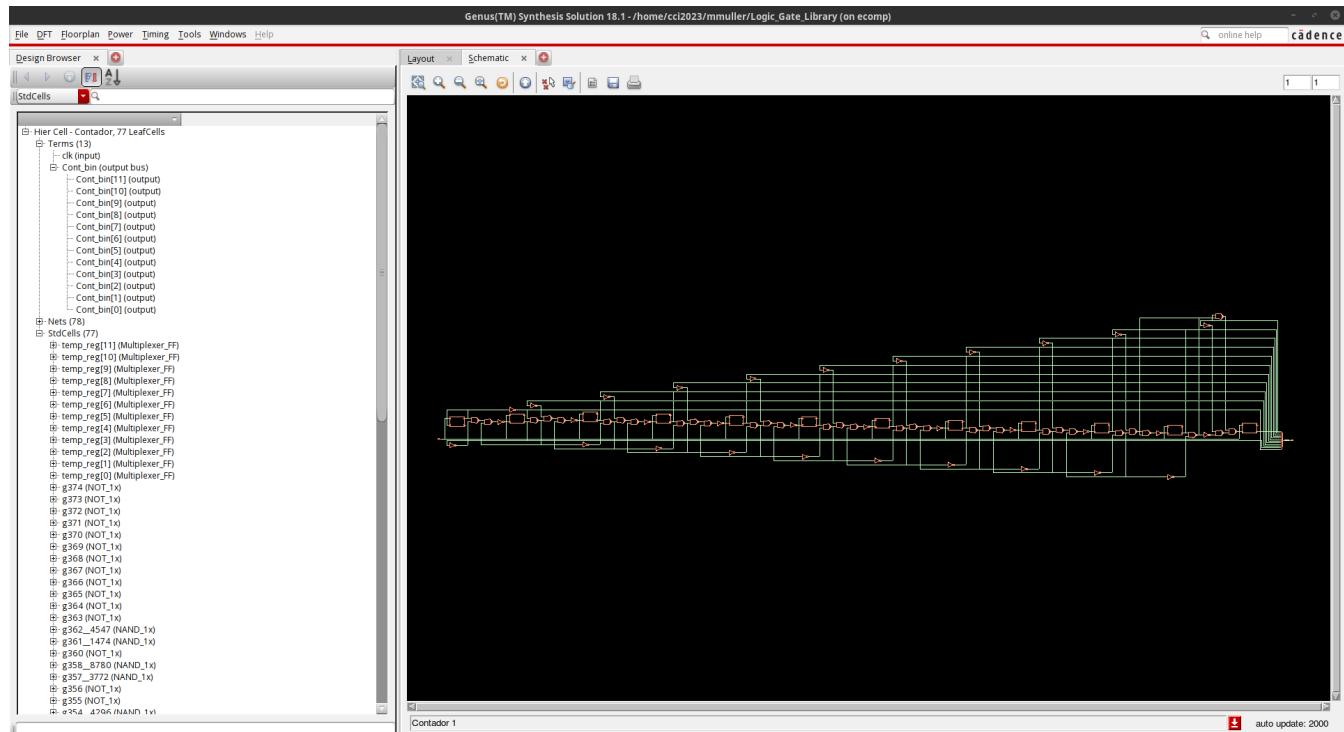
elaborate
syn_generic
syn_map

write_hdl > /home/cci2023/mmuller/Logic_Gate_Library/Contador.v
```

Fonte: Os Autores.

O resultado da síntese lógica do contador/prescaler pode ser visualizado na figura 7.5.

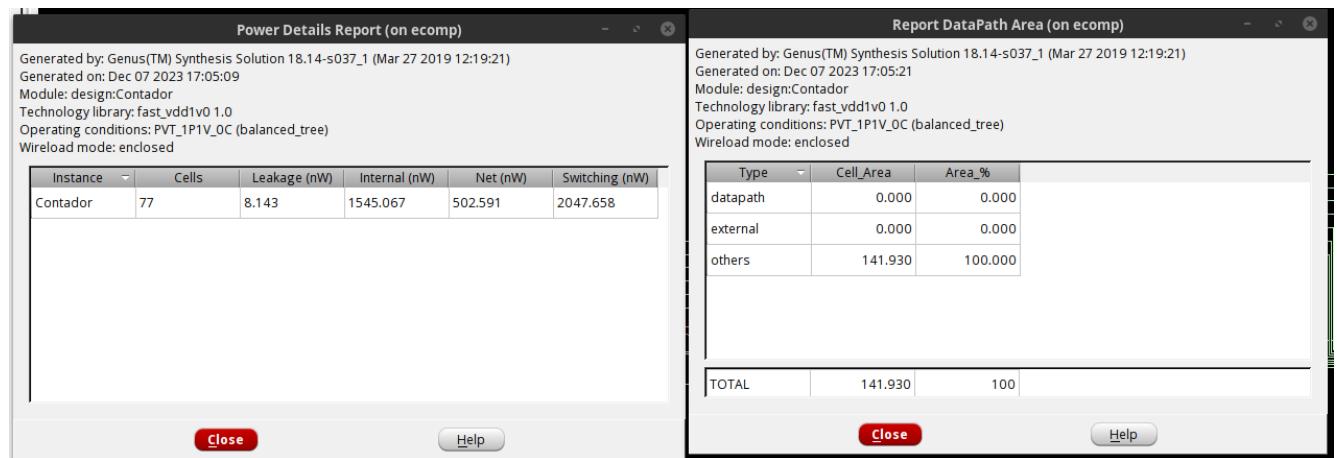
Figura 7.5 – Vista esquemática do contador/prescaler após síntese lógica



Fonte: Os Autores.

Na figura 7.6 podem ser visualizadas as informações de área e potência estimada.

Figura 7.6 – Estimativas de área e potência do circuito contador/prescaler



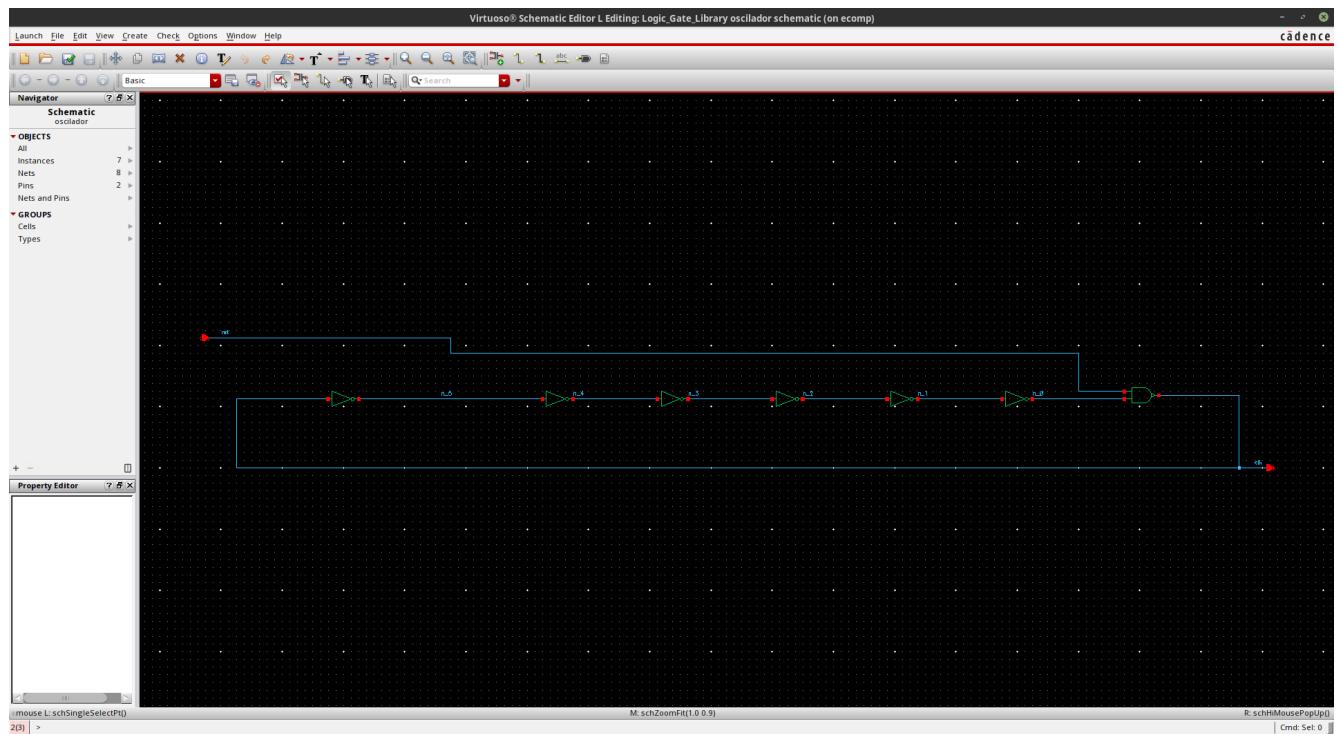
Fonte: Os Autores.

## 8 IMPORTAÇÃO DOS ARQUIVOS PARA O VIRTUOSO E TESTE DOS MESMOS

### 8.1 OSCILADOR

Na figura 8.1 pode ser visualizada a vista esquemática do oscilador dentro do virtuoso, onde foram utilizadas as portas lógicas projetadas.

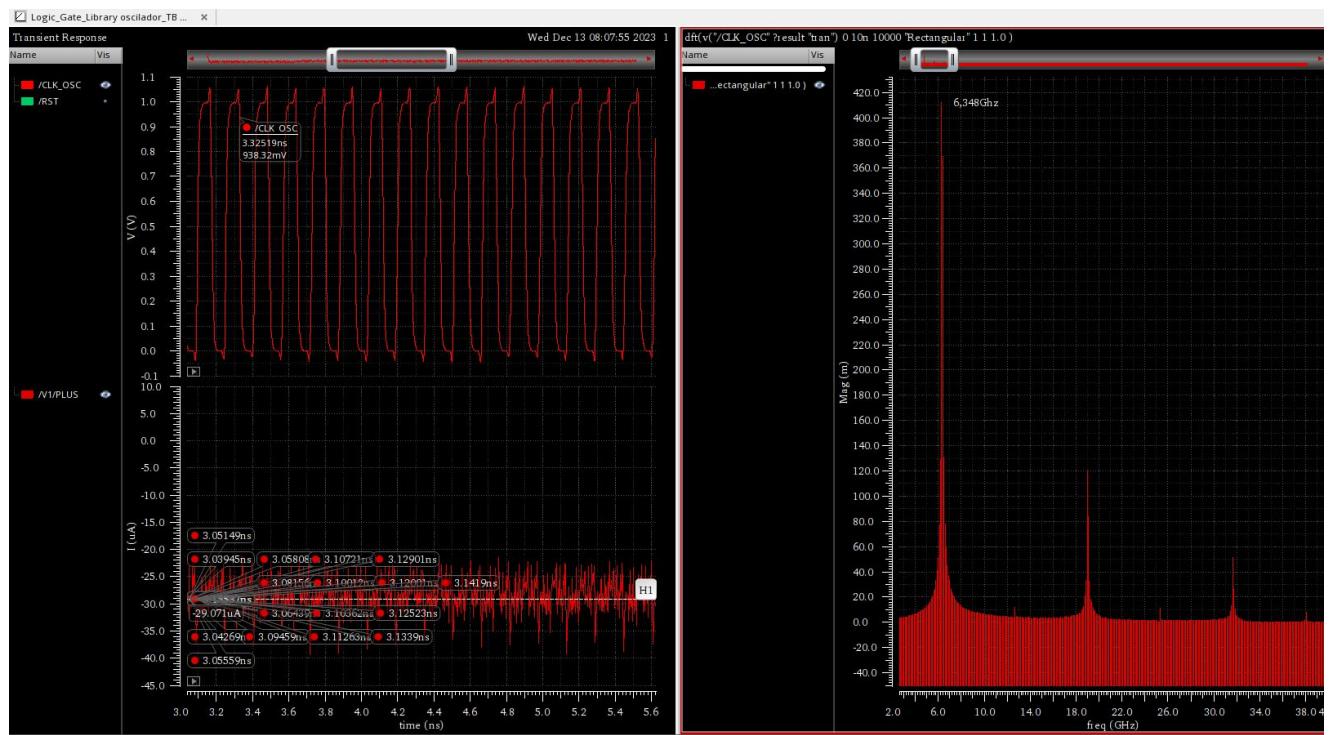
Figura 8.1 – Vista esquemática do oscilador dentro do virtuoso



Fonte: Os Autores.

Na figura 8.2 podem ser visualizadas as formas de onda obtidas no testbench do oscilador.

Figura 8.2 – Testbench do Oscilador

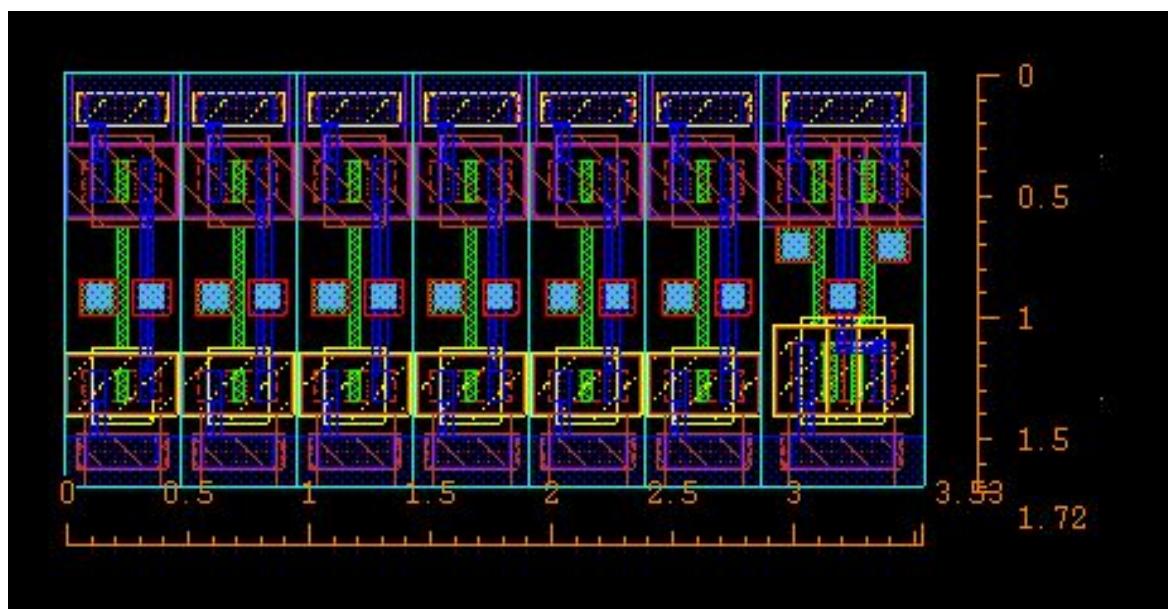


Fonte: Os Autores.

Os resultados indicam a frequencia de oscilação de  $6,348\text{GHz}$  e o consumo, nesta frequência, de  $29,11\mu\text{W}$ .

Abaixo, na figura 8.3 pode ser visualizada a estimativa da área do oscilador, que ficou em aproximadamente  $6,04\mu\text{m}^2$

Figura 8.3 – Estimativa da área do Oscilador

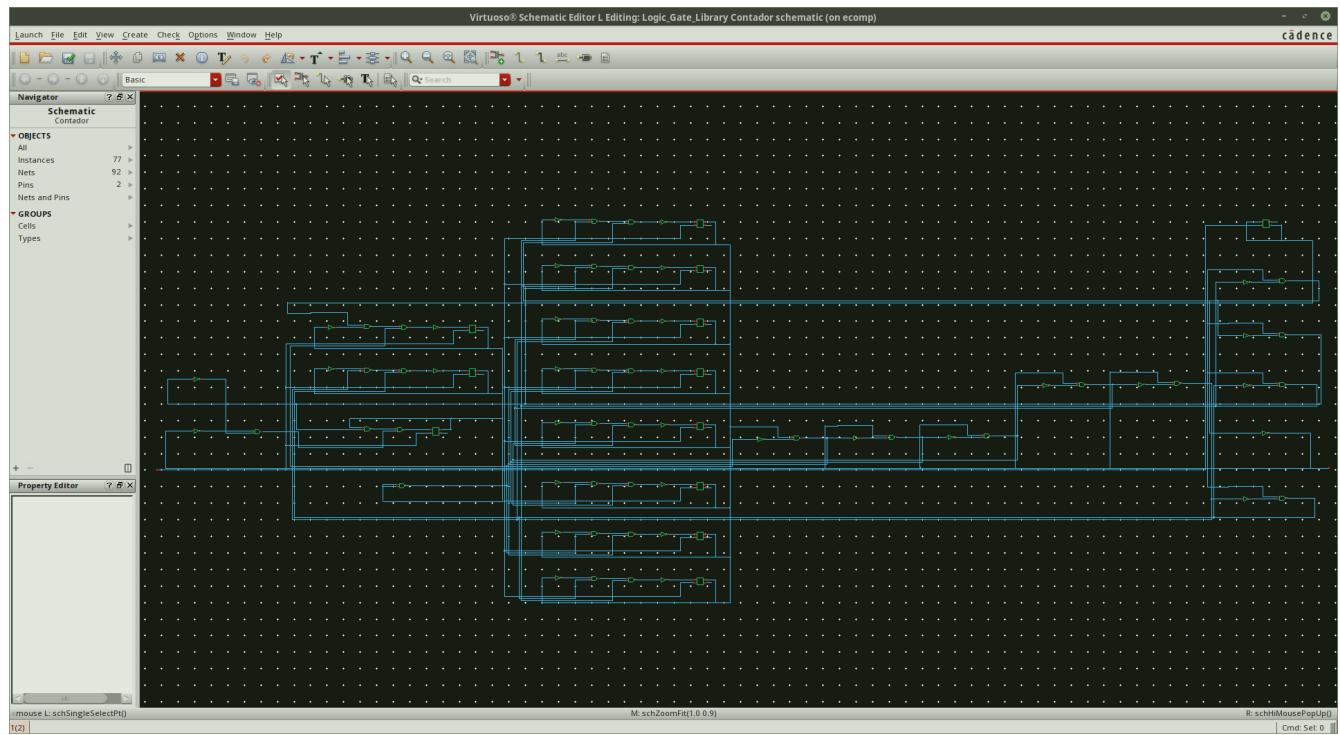


Fonte: Os Autores.

## 8.2 CONTADOR/PRESCALER

Na figura 8.4 pode ser visualizada a vista esquemática do contador/prescaler dentro do virtuoso, onde também pode-se visualizar que foram utilizadas as portas lógicas projetadas.

Figura 8.4 – Vista esquemática do contador/prescaler dentro do virtuoso



Fonte: Os Autores.

O resultado do testbench do contador/prescaler oriundo da síntese lógica pode ser visto na figura 8.5.

Figura 8.5 – Testbench do divisor de frequências



Fonte: Os Autores.

Os resultados das frequências na saída e consumo geral extraídos do testbench feito estão indicados na figura 8.6.

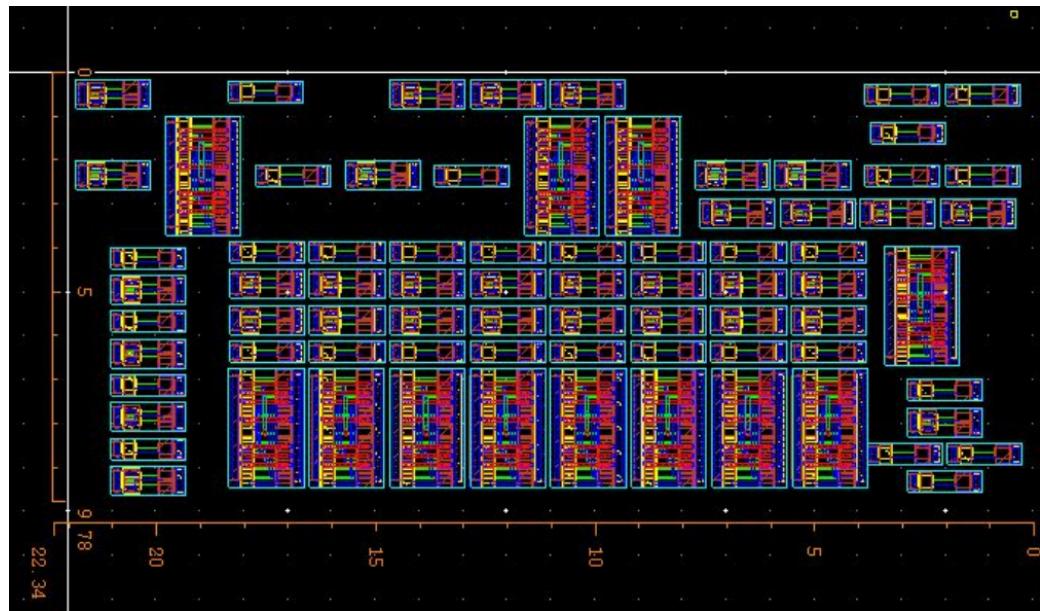
Figura 8.6 – Frequências na saída e consumo @  $f_{in} = 2GHz$

frequency(leafValue( VT("/Cont_b..."))	
Expression	frequency(v("/CLK_IN_DIV" ?result "tran"))
Value	2.000E9
Expression	frequency(leafValue( VT("/Cont_bin<11:0>") "bit" "/Cont_bin<0>" ))
Value	1.000E9
Expression	frequency(leafValue( VT("/Cont_bin<11:0>") "bit" "/Cont_bin<1>" ))
Value	500.0E6
Expression	frequency(leafValue( VT("/Cont_bin<11:0>") "bit" "/Cont_bin<2>" ))
Value	250.0E6
Expression	frequency(leafValue( VT("/Cont_bin<11:0>") "bit" "/Cont_bin<3>" ))
Value	125.0E6
Expression	frequency(leafValue( VT("/Cont_bin<11:0>") "bit" "/Cont_bin<4>" ))
Value	62.50E6
Expression	frequency(leafValue( VT("/Cont_bin<11:0>") "bit" "/Cont_bin<5>" ))
Value	31.25E6
Expression	frequency(leafValue( VT("/Cont_bin<11:0>") "bit" "/Cont_bin<6>" ))
Value	15.63E6
Expression	frequency(leafValue( VT("/Cont_bin<11:0>") "bit" "/Cont_bin<7>" ))
Value	7.813E6
Expression	frequency(leafValue( VT("/Cont_bin<11:0>") "bit" "/Cont_bin<9>" ))
Value	4.020E9
Expression	frequency(leafValue( VT("/Cont_bin<11:0>") "bit" "/Cont_bin<10>" ))
Value	4.000E9
Expression	frequency(leafValue( VT("/Cont_bin<11:0>") "bit" "/Cont_bin<11>" ))
Value	4.000E9
Expression	average(i("/V1/PLUS" ?result "tran"))
Value	-31.26E-6

Fonte: Os Autores.

Abaixo, na figura 8.7 pode ser visualizada a estimativa da área do divisor de frequências, que ficou em aproximadamente  $200\mu m^2$

Figura 8.7 – Estimativa da área do Prescaler



Fonte: Os Autores.

## 9 CONCLUSÃO E PERSPECTIVAS FUTURAS

Em se tratando da criação da biblioteca de Standard Cells, concluímos que a mesma foi realizada com sucesso e o processo de criação destas levou como princípio o processo otimizado de criação *Full Custom* no qual a área obtida foi reduzida em comparação com processos onde portas lógicas menos complexas são reaproveitadas.

O processo de adaptação das Standard Cells projetadas para uso na síntese lógica exigiu que fossem criadas as *Cellviews Schematic, Symbol, Layout, Functional, Extracted* e *Abstract* e neste processo fez-se o uso da ferramenta *ABSTRACT*, na qual tudo ocorreu conforme esperado.

A criação das bibliotecas *.LIB* e *.LEF* foram essenciais no processo de síntese lógica e exigiram cuidado para que houvesse *match* entre as views criadas no processo anterior em termos dos pinos que foram utilizados em ambos os arquivos e também na estrutura lógica adotada para descrever os funcionamento das standard cells.

Os circuitos utilizados para testar as bibliotecas criadas no processo da síntese lógica foram validados nas ferramentas *GHDL* e *GTKWAVE*, utilizadas para análise, simulação e visualização das formas de onda obtidas antes de aplicar as descrições no software *GENUS*, responsável por realizar a síntese lógica dos mesmos.

O software *GENUS*, após realização da síntese, informou estimativas de área a consumo condizentes com as standard cells da tecnologia *GF-45nm*. Os arquivos gerados pela ferramenta foram importados para dentro do *VIRTUOSO* para que o processo de testes a análise dos resultados pudessem ser feitos.

As simulações pós síntese foram bem sucedidas e os circuitos cumpriram com os propósitos para os quais foram projetados.

Ao criar a vista *Layout* para estimar as áreas dos circuitos foi comprovado mais uma vez o uso bem sucedido das células projetadas.

Pôde-se concluir, após importação dos arquivos para o virtuoso, que as portas lógicas utilizadas no processo foram as mesmas que foram projetadas na primeira etapa deste projeto, o que indica sucesso no desenvolvimento de todos os passos no processo inteiro, onde, a partir de circuitos descritos em linguagem de descrição de hardware conseguimos obter suas equivalências a nível elétrico utilizando a tecnologia CMOS em um fluxo de design VLSI por meio das ferramentas da plataforma *CADENCE*.

Como perspectivas futuras, para continuação do projeto, pretende-se utilizar os dados extraídos individualmente, por simulação, de cada célula projetada nesta biblioteca e aplicação destes nas *Lookup tables* da biblioteca *.LIB* para que as estimativas de potência, tempos de propagação, capacitâncias nos nós e demais informações pertinentes sejam utilizadas para o processo de síntese lógica e que este permita comprovação posterior nas estimativas de área e consumo se comparados ao processo realizado à mão dentro do ambiente de projetos *VIRTUOSO* da *CADENCE*.

Vem sendo desenvolvido um tutorial completo com o passo-a-passo de todo o processo realizado até o momento, para que o mesmo seja utilizado, posteriormente, para fins acadêmicos.

Em anexo, segue uma pasta com todos os arquivos desenvolvidos neste projeto até o momento. Os arquivos incluem as bibliotecas *.LIB* e *.LEF*, a descrição dos circuitos utilizados para teste, os arquivos script rodados na ferramenta *GENUS* os arquivos *.V* resultantes da síntese lógica, os slides da apresentação e uma versão incompleta do tutorial.

## **REFERÊNCIAS BIBLIOGRÁFICAS**

- [1] Rabaey, J. M. (2003). Digital Integrated Circuits: A Design Perspective (2<sup>a</sup> ed.). Prentice Hall.
- [2] University of Maryland, Baltimore County. (S/D). Liberty Timing File. Disponível em: [https://redirect.cs.umbc.edu/cpatel2/links/641/slides/lect05\\_LIB.pdf](https://redirect.cs.umbc.edu/cpatel2/links/641/slides/lect05_LIB.pdf).
- [3] University of Maryland, Baltimore County. (S/D). Library Exchange Format (LEF). Disponível em: [https://redirect.cs.umbc.edu/cpatel2/links/641/slides/lect04\\_LEF.pdf](https://redirect.cs.umbc.edu/cpatel2/links/641/slides/lect04_LEF.pdf).