UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA - CI
CURSO ENGENHARIA DE COMPUTAÇÃO

EPITÁCIO PESSOA DE BRITO NETO
11506856

RELATÓRIO DE MICROELETRÔNICA – PARTE 2

JOÃO PESSOA
2020

UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA - CI

CURSO ENGENHARIA DE COMPUTAÇÃO

EPITÁCIO PESSOA DE BRITO NETO

11506856

RELATÓRIO DE MICROELETRÔNICA – PARTE 2

Relatório referente à disciplina de Introdução à Microeletrônica do Ensino Superior da Universidade Federal da Paraíba (UFPB) como requisito parcial da avaliação semestral.

Professor: Antonio Carlos Cavalcanti

Hugo Leonardo Davi de Souza Cavalcante

JOÃO PESSOA

2020

RELATÓRIO DE MICROELETRÔNICA – PARTE 2

O objetivo principal deste projeto é a aplicação teórica dos estudos feitos na disciplina de Introdução à Microeletrônica ministrada pelos professores Antonio Carlos e Hugo Leonardo, ofertada pela Universidade Federal da Paraíba (UFPB), para fins didáticos e avaliação do aprendizado durante o percurso da disciplina. Especificamente, este relatório se tratará de aplicações práticas, simulações, desenhos e análises na construção de tecnologias CMOS.

Buscamos, com este relatório, um entendimento sucinto das atividades realizadas em sala, provendo informações do que utilizamos para realizar nossas simulações e dados tanto da listagem de grandezas e componentes utilizados junto com seus respectivos valores.

1 Introdução

Este relatório se remete à, basicamente, aplicações das teorias e análises de técnicas de construção e interpretação de hardwares ministradas nas aulas dos professores Antonio Carlos e Hugo Leonardo. Neste, continuaremos fazendo a construção de modelos esquemáticos via Graal, criação e conversão para linguagem SPICE, simulação e análise de grandezas dos circuitos construídos via SpiceOpus.

Contudo, também demonstraremos o aprendizado de novas tecnologias: construção de circuitos de forma procedural utilizando C e a biblioteca Genlib, funções para posicionar e rotear as standard cells (através do arquivo VHDL resultante do arquivo C) chamadas OCP e Nero, criando, então, as standard cells que fazíamos no relatório 1 manualmente e, também, continuaremos analisando seus comportamentos via SpiceOpus.

2 Transistor de passagem

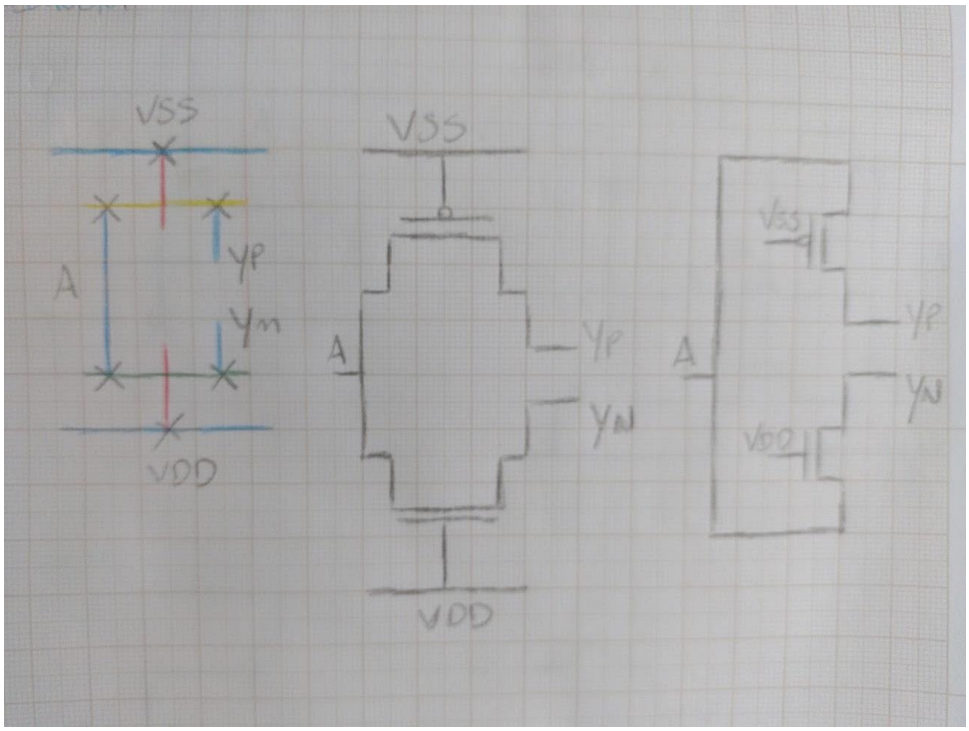

Figura 1: Diagramas de palitos e transistorizados do transistor de passagem.

Ao instanciarmos dois inversores normais e trabalharmos com suas estruturas, chegamos, então ao transistor de passagem:

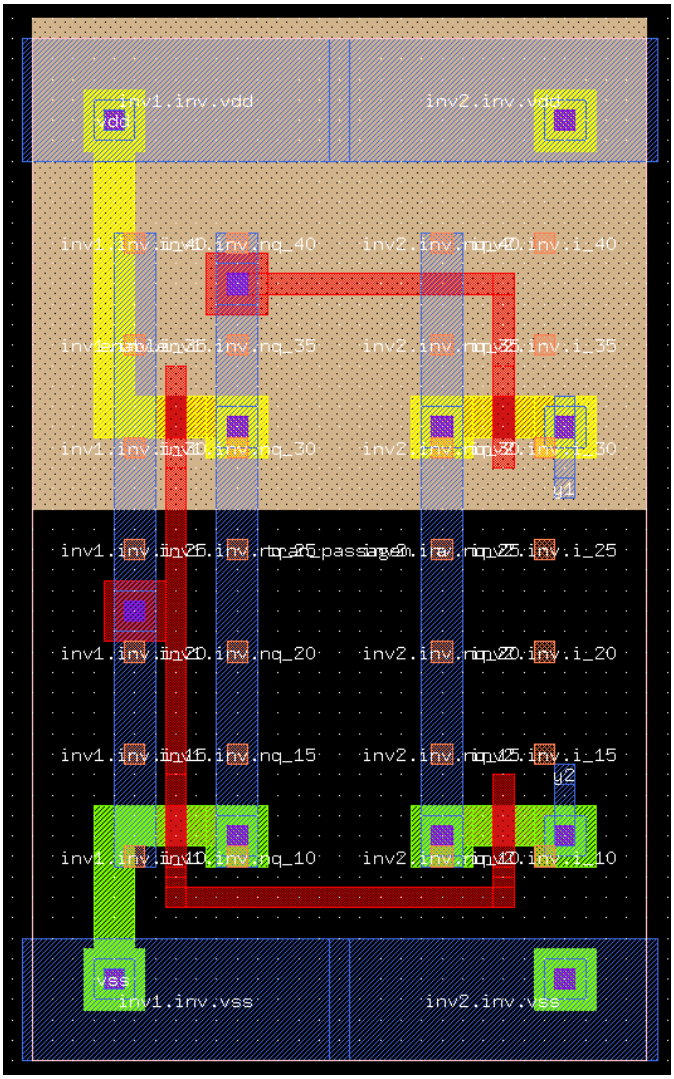

Figura 2: Modelo de construção de um transistor de passagem.

```
1
2 .include tran_passagem.spi
3
4 * INTERF a enable vdd vss y1 y2
5
6 X1 10 11 40 30 20 21 tran_passagem
7 V1 10 0 dc 0.0V
8 V2 11 0 dc 1.8V
9
10 R1 21 0 100k
11 R2 20 0 100k
12
13 Vdd 40 0 1.8V
14 Vss 30 0 -1.8V
15
16 .model tp pmos level = 54
17 .model tn nmos level = 54
18
19 .dc V1 -1.8 1.8 0.001
20 .end
21
```

Figura 3: Arquivo .cir para simulação do transistor de passagem.

Tabela 1: Tabela de cores da simulação do transistor de passagem.

Componentes	***Cor***
V(10) (Fonte V1)	Vermelho
V(11) (Fonte V2)	Verde
V(20) (Saída Y1)	Azul
V(21) (Saída Y2)	Roxo

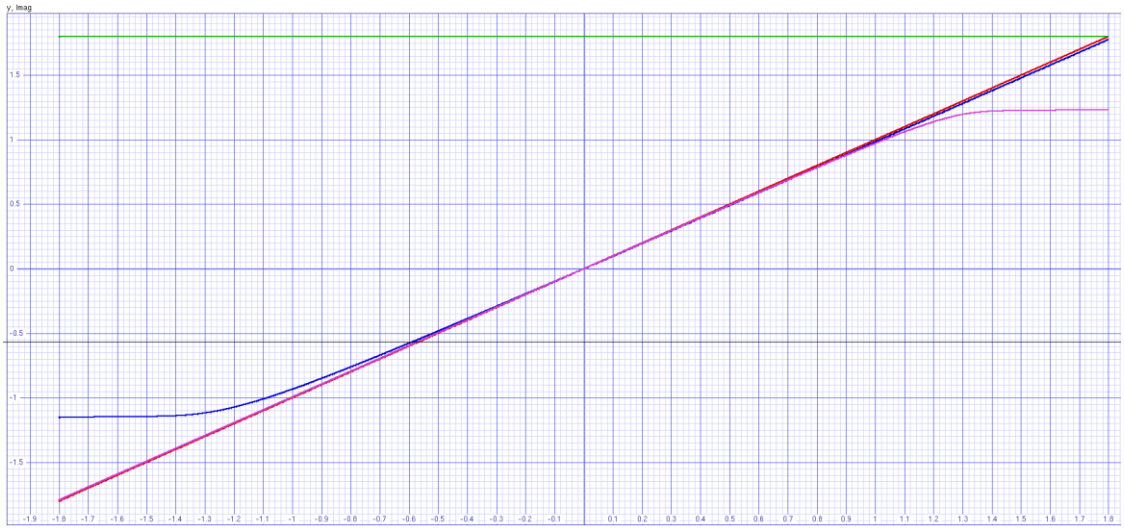

Figura 4: Gráfico do comportamento do transistor de passagem.

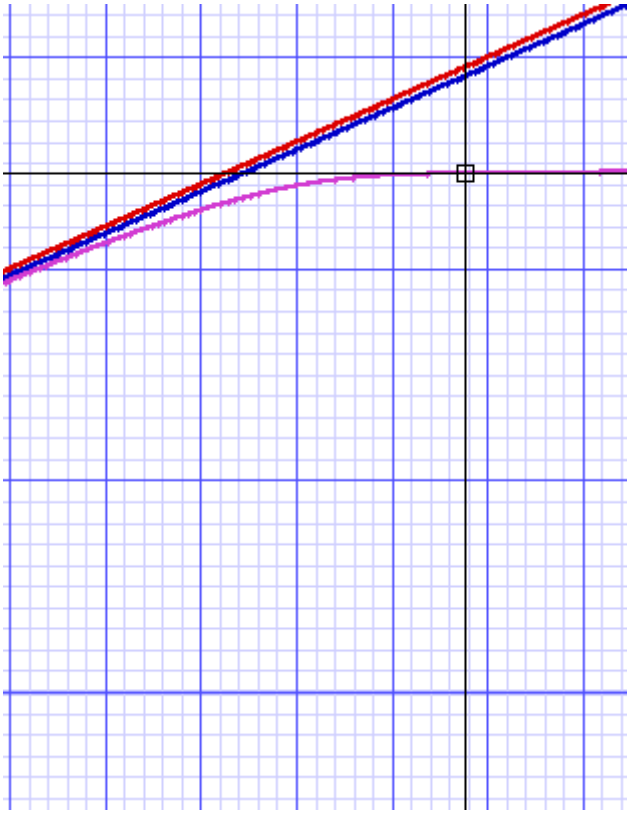

Figura 5: 1 fraco.

Press <space> to identify nearest curve.
x-y grid displaying real vs default.
Cursor: x =1.476343402225597e+00 y =1.226842105263158e+00

Figura 6: Valores do 1 fraco.

3 Transmission Gate

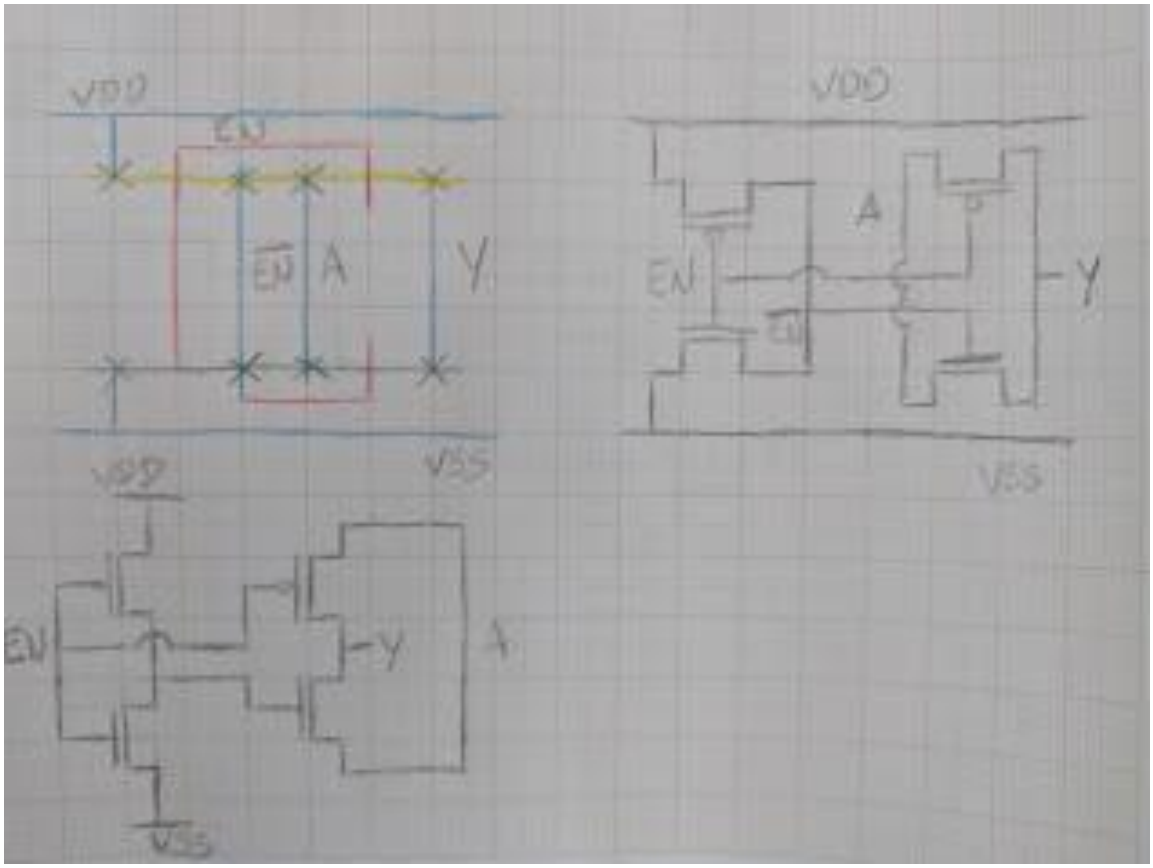

Figura 7: Diagramas de palitos e transistorizados do transmission gate.

Feito a partir do transistor de passagem anterior, onde aproveitamos aquela standard cell como base para alterá-lo, chegamos, então, ao transmission gate:

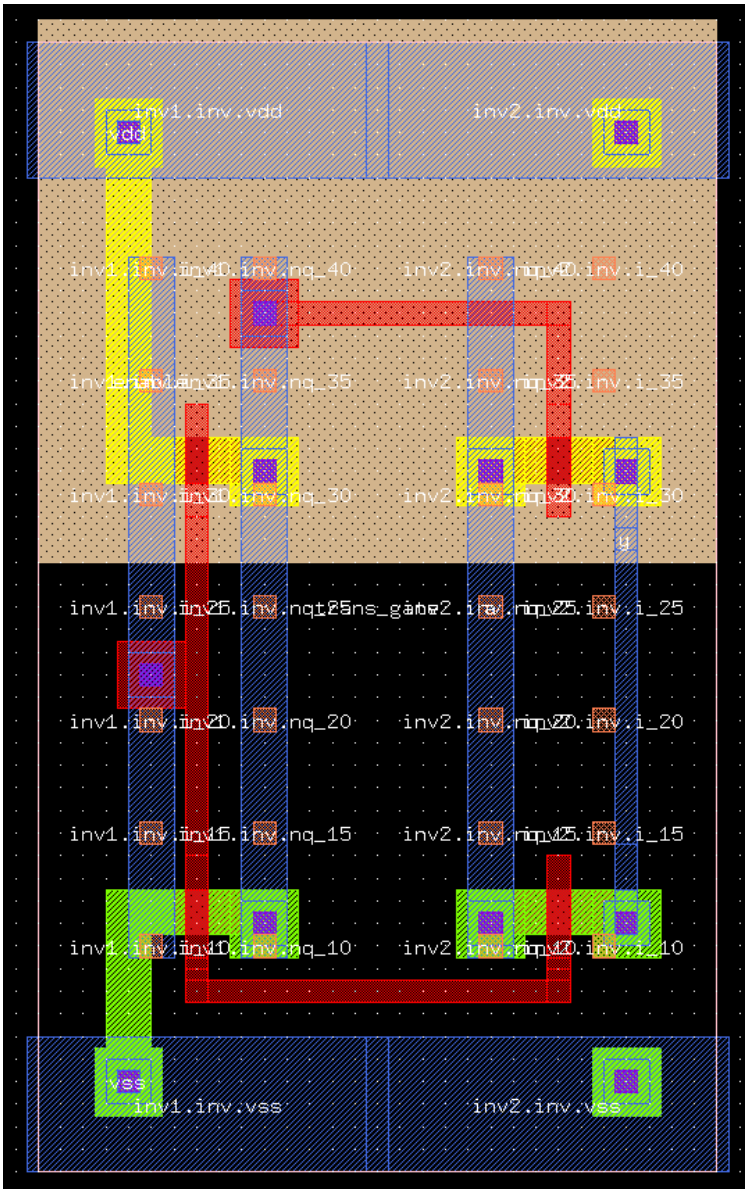

Figura 8: Modelo de construção de um transmission gate.

```
1
2 .include trans_gate.spi
3
4 * INTERF a enable vdd vss y
5
6 X1 10 11 40 30 20 trans_gate
7 V1 10 0 dc 0.0V
8 V2 11 0 dc 1.8V
9
10 R1 20 0 1k
11
12 Vdd 40 0 1.8V
13 Vss 30 0 -1.8V
14
15 .model tp pmos level = 54
16 .model tn nmos level = 54
17
18 .dc V1 -1.8 1.8 0.001
19 .end
20
```

Figura 9: Arquivo .cir para simulação do transmission gate.

Tabela 2: Tabela de cores da simulação do transmission gate.

Componentes	***Cor***
V(10) (Fonte V1)	Vermelho
V(11) (Fonte V2)	Verde
V(20) (Saída Y)	Azul

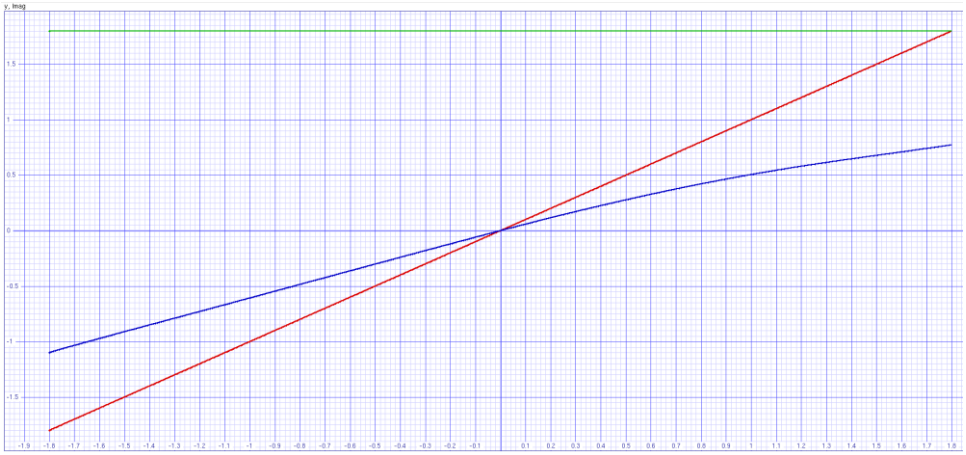

Figura 10: Gráfico do comportamento do transmission gate.

3.1 Análise dinâmica

```
1
2 .include trans_gate.spi
3
4 * INTERF a enable vdd vss y
5
6 X1 10 11 40 30 20 trans_gate
7 V1 10 0 sine(0 1.8V 1G)
8 V2 11 0 pulse(-1.8V 1.8V 2n 1p 1p 10n 20n)
9
10 R1 20 0 1Meg
11
12 Vdd 40 0 1.8V
13 Vss 30 0 -1.8V
14
15 .model tp pmos level = 54
16 .model tn nmos level = 54
17
18 .tran 0.001ns 40ns
19 .end
```

Figura 11: Arquivo .cir para simulação dinâmica do transmission gate.

Tabela 3: Tabela de cores da simulação do transmission gate.

Componentes	***Cor***

V(10) (Fonte V1)	Vermelho
V(11) (Fonte V2)	Verde
V(20) (Saída Y)	Azul

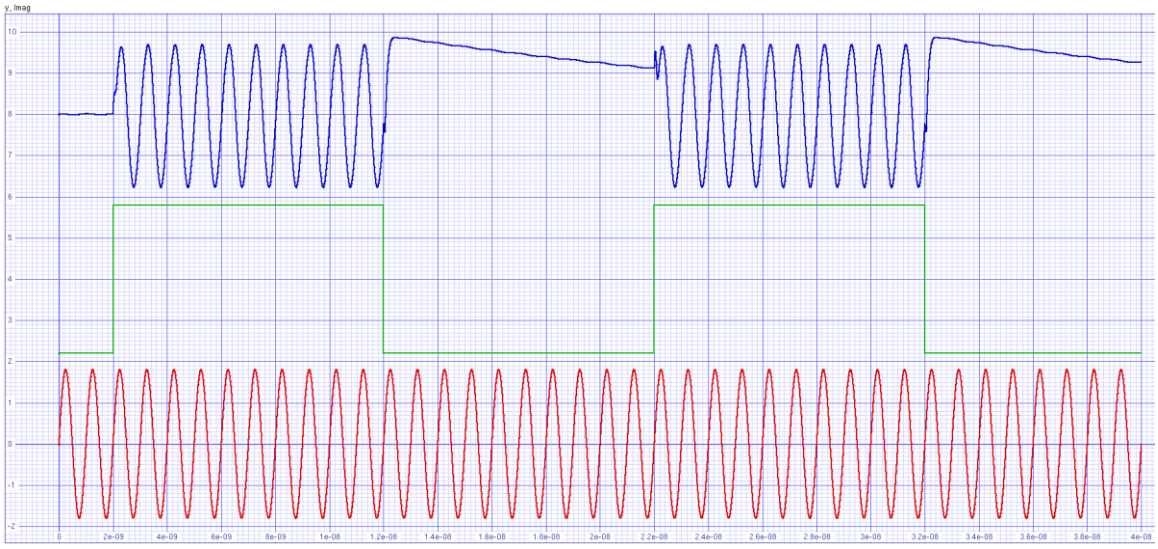

Figura 12: Gráfico do comportamento do transmission gate na análise dinâmica.

4 Tristate Inverter

Tabela 4: Tabela verdade do Tristate Inverter.

EN/EN barra	A	Y
0/1	0	Z
0/1	1	Z
1/0	0	0
1/0	1	1

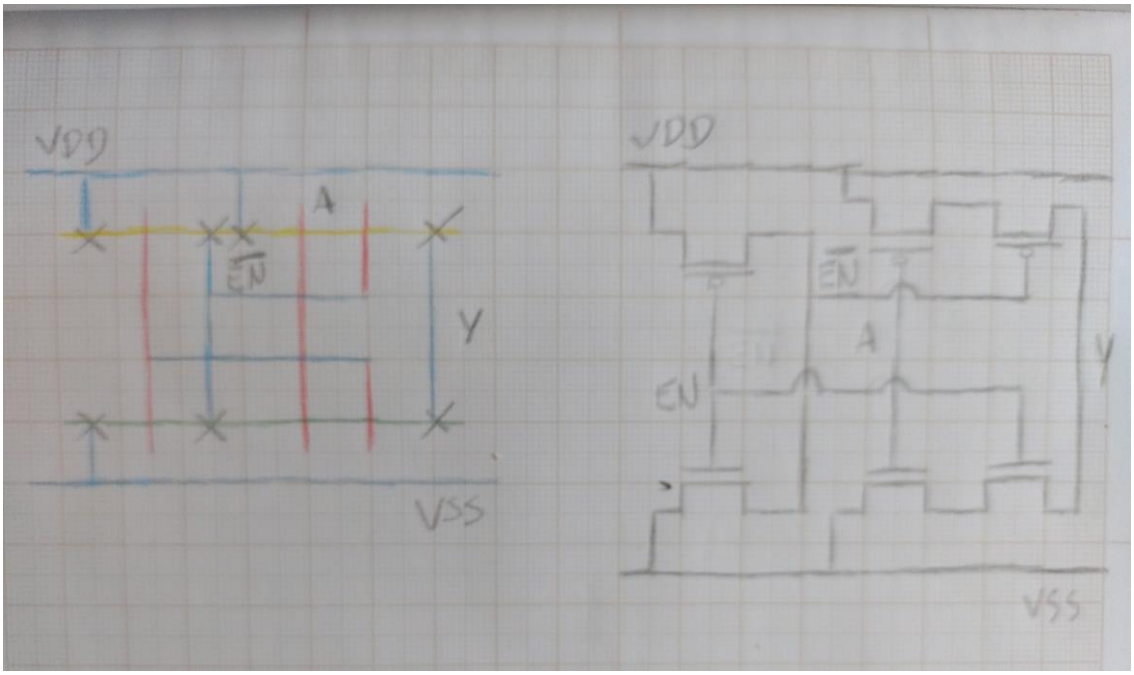

Figura 13: Diagrama de palitos e transistorizados do Tristate Inverter.

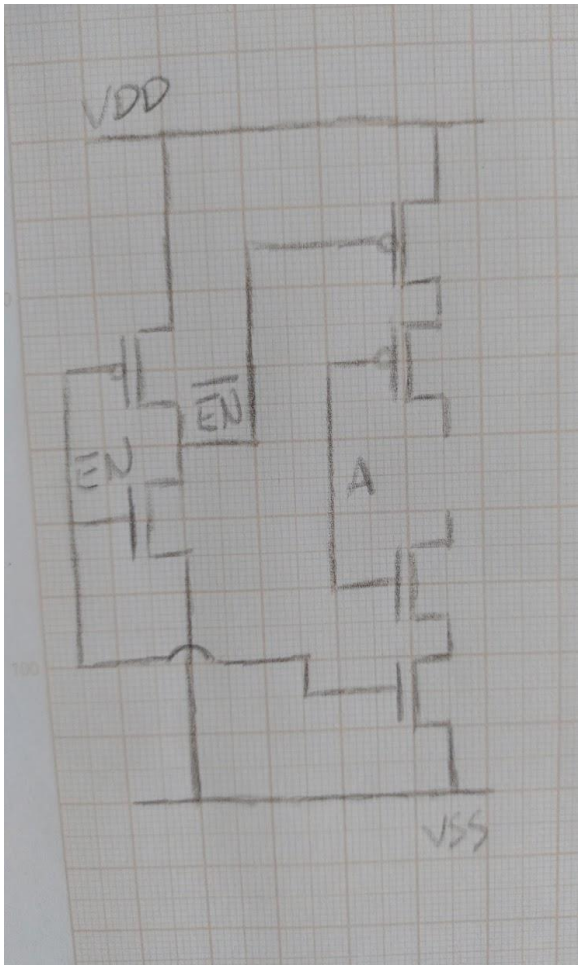

Figura 14: Continuação dos diagramas transistorizados do Tristate Inverter.

O tristate inverter também foi feito a partir da base inicial feita no transistor de passagem, ou seja, a partir de dois inversores, alteramo-los de forma a chegar no seguinte modelo:

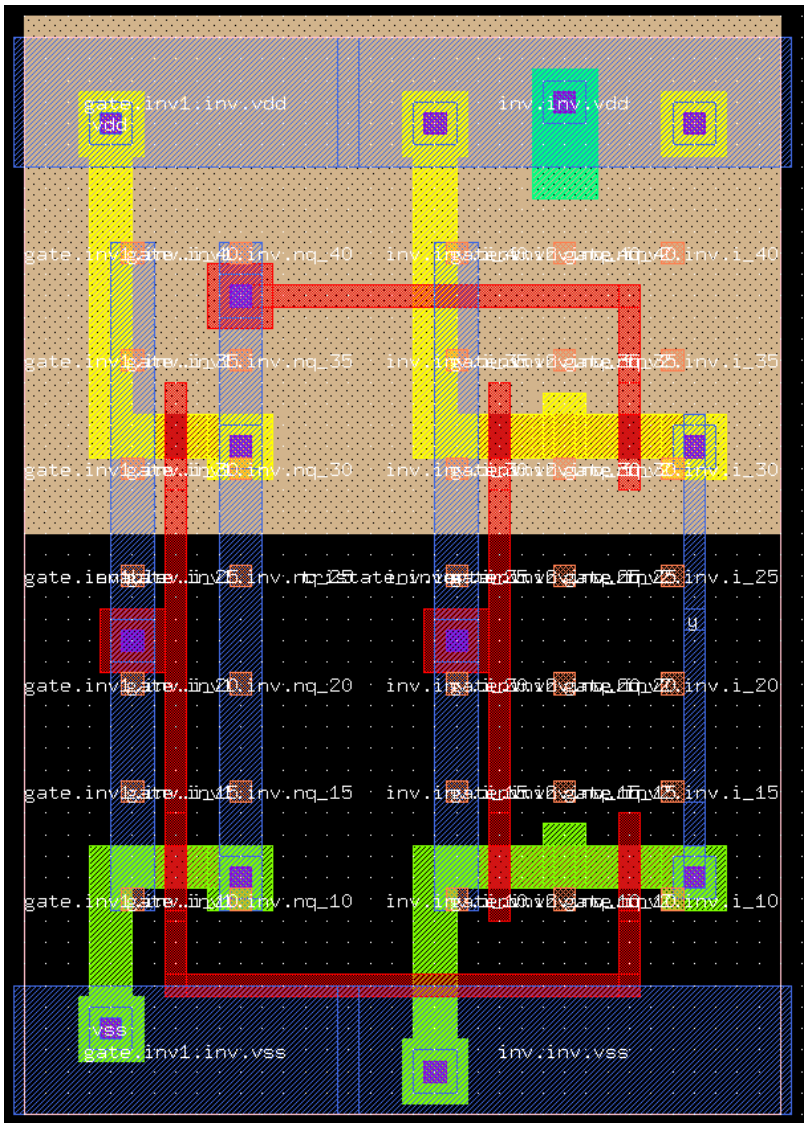

Figura 15: Modelo de construção de um tristate inverter.

```
1
2 .include tristate_inverter.spi
3 .include inversor_1.spi
4
5 * INTERF a enable vdd vss y
6 * INTERF a vdd vss y
7
8 X1 10 11 40 30 20 tristate_inverter
9 X2 20 40 30 21 inversor_1
10
11 V1 10 0 sine(0 1.8V 100Meg)
12 V2 11 0 pulse(-1.8V 1.8V 20n 1p 1p 100n 200n)
13
14 *R1 20 0 10Meg
15
16 Vdd 40 0 1.8V
17 Vss 30 0 -1.8V
18
19 .model tp pmos level = 54
20 .model tn nmos level = 54
21
22 .tran 0.01ns 400ns
23 .end
```

Figura 16: Arquivo .cir para simulação do tristate inverter.

Tabela 5: Tabela de cores da simulação do tristate inverter.

Componentes	***Cor***
V(10) (Fonte V1)	Vermelho
V(11) (Fonte V2)	Verde
V(20) (Saída tristate inverter)	Azul
V(21) (Saída inversor)	Roxo

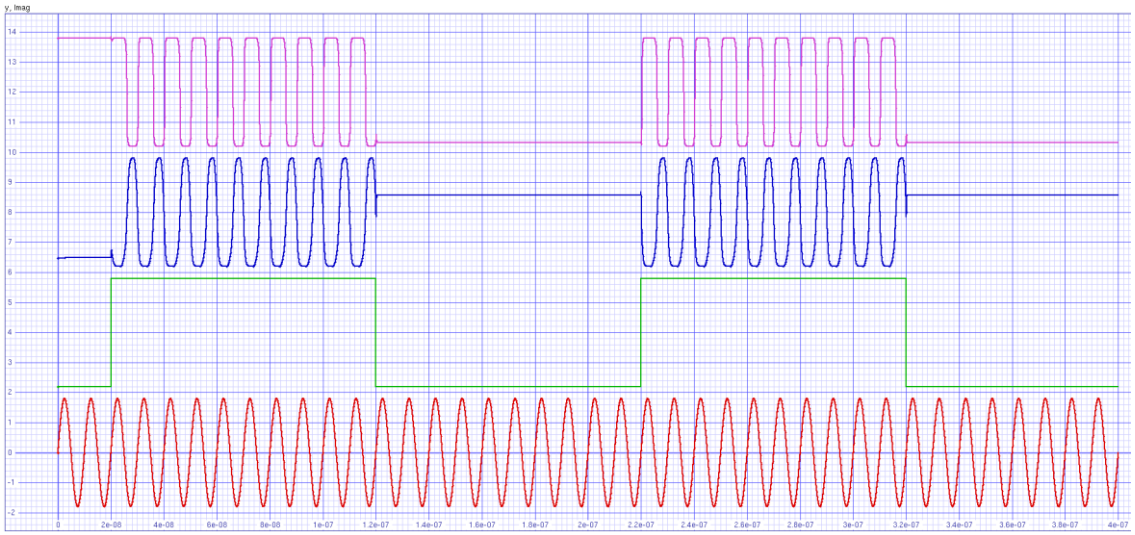

Figura 17: Gráfico do comportamento do tristate inverter.

4.1 Tristate

Este modelo de tristate foi construído com objetivo de construção de um Mux 2:1 que poderá ser visto posteriormente. Portanto, temos:

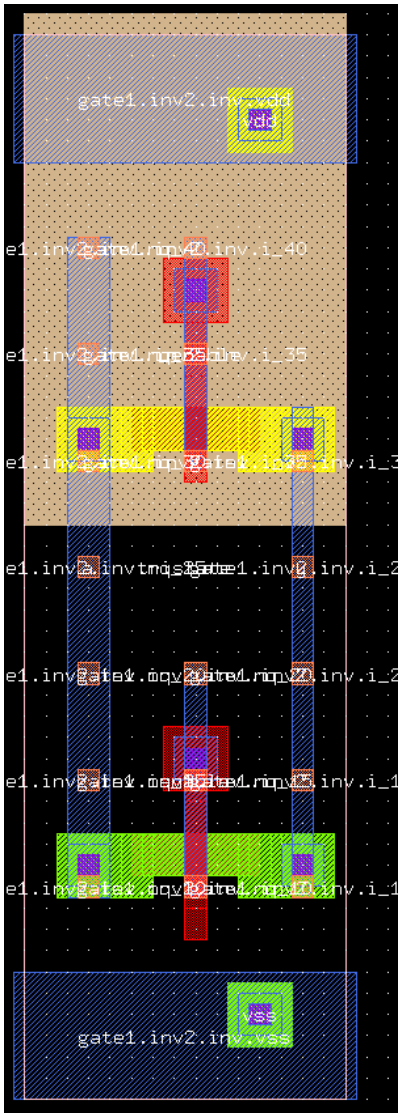

Figura 18: Modelo de construção de um tristate.

4.2 Tristate Inverter sem Inversor de entrada

Assim como o tristate anterior, este também foi construído com objetivo de construção de outra peça. Neste caso, este tristate fez parte da construção do Inverting Mux, que pode ser visto posteriormente. Assim:

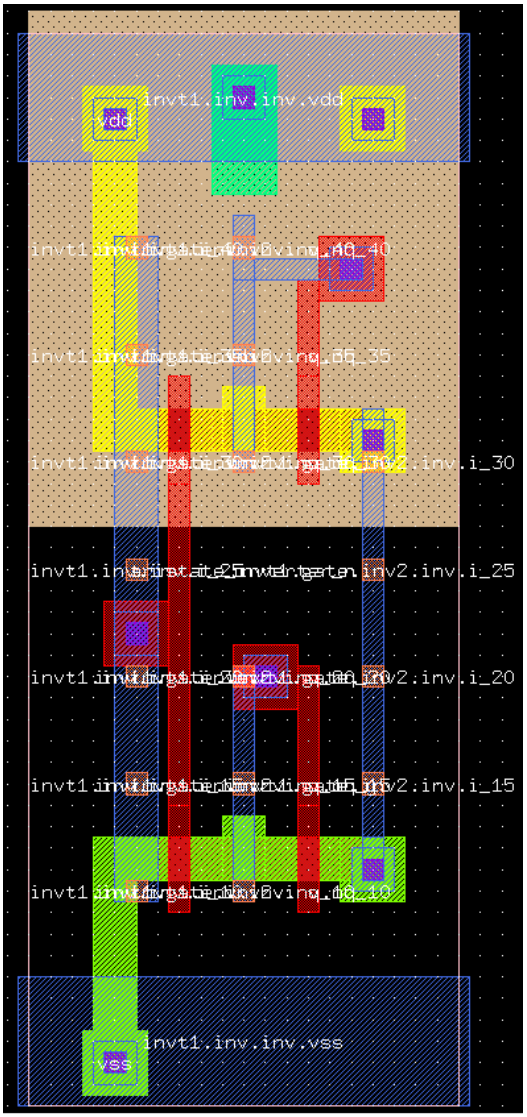

Figura 19: Modelo de construção de um tristate Inverter sem Inversor de entrada.

5 Mux 2:1

Tabela 6: Tabela verdade do Mux 2:1.

S/S~	D1	D0	Y
0/1	X	0	0
0/1	X	1	1
1/0	0	X	0
1/0	1	X	1

A partir do tristate mostrado no tópico 4.1, chegamos ao seguinte modelo de Mux 2:1:

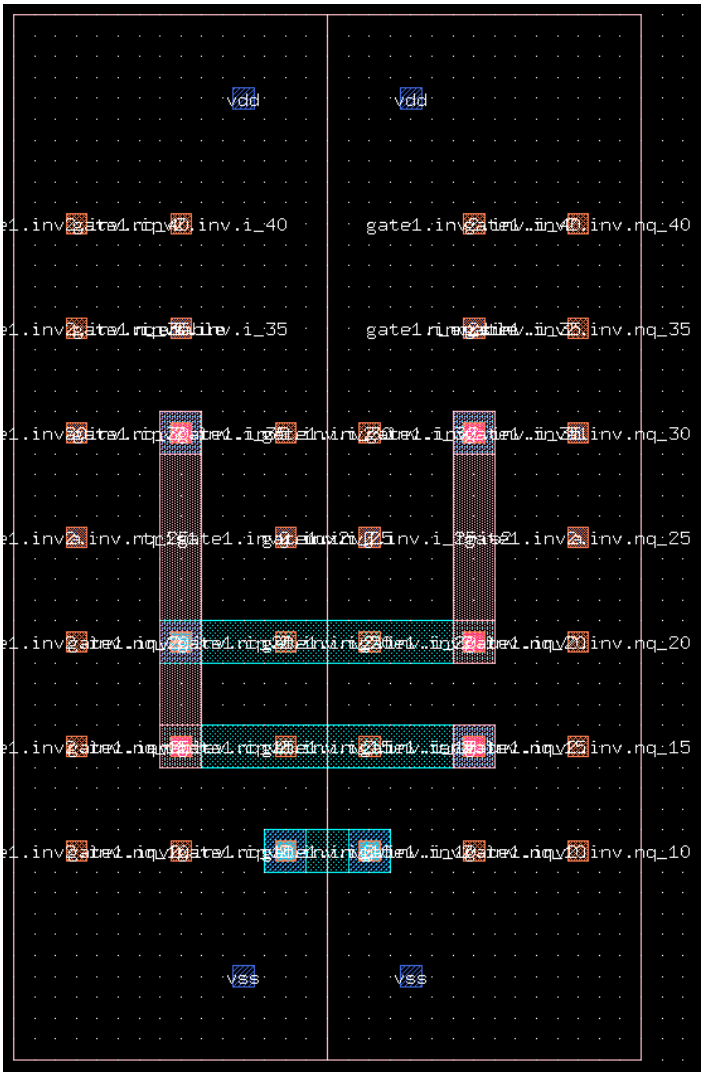

Figura 20: Modelo de construção de um Mux 2:1.

```
1
2 .include mux2.spi
3 .include inversor_1.spi
4
5 * INTERF a0 a1 n_s s vdd vss y
6 * INTERF a vdd vss y
7
8 X1 10 11 12 13 40 30 20 mux2
9 X2 13 40 30 12 inversor_1
10
11 V0 10 0 sine(0 1.8V 70Meg)
12 V1 11 0 sine(0 1.8V 40Meg)
13 V2 13 0 pulse(-1.8V 1.8V 20n 1p 1p 100n 200n)
14
15
16 *R1 20 0 10Meg
17
18 Vdd 40 0 1.8V
19 Vss 30 0 -1.8V
20
21 .model tp pmos level = 54
22 .model tn nmos level = 54
23
24 .tran 0.01ns 400ns
25 .end
```

Figura 21: Arquivo .cir para simulação do Mux 2:1.

Tabela 7: Tabela de cores da simulação do Mux 2:1.

Componentes	***Cor***
V(10) (Fonte V0)	Vermelho
V(11) (Fonte V1)	Verde
V(13) (Fonte V2)	Azul
V(12) (Saída inversor)	Roxo
V(20) (Saída Mux 2:1)	Preto

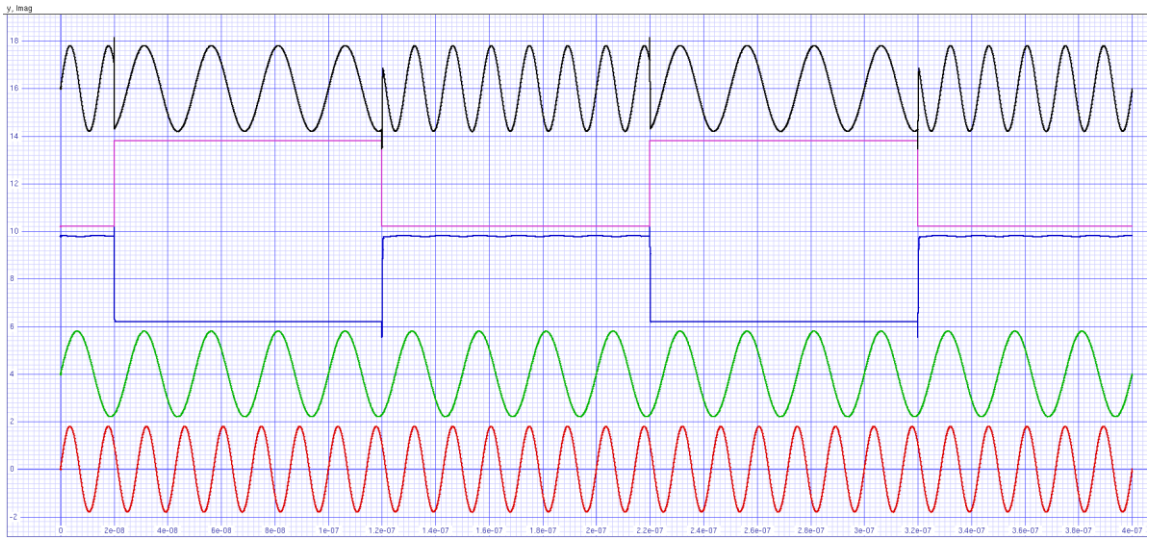

Figura 22: Gráfico do comportamento do Mux 2:1.

6 Mux 4:1 non-Inverter

Tabela 8: Tabela verdade do Mux 4:1.

S1	S0	Y
0	0	A0
0	1	A1
1	0	A2
1	1	A3

Este modelo de Mux 4:1 non-Inverter foi feito a partir de três Inverting Mux que pode ser visto no tópico 7, em conjunto de dois inversores normais. Portanto, chegamos à seguinte configuração:

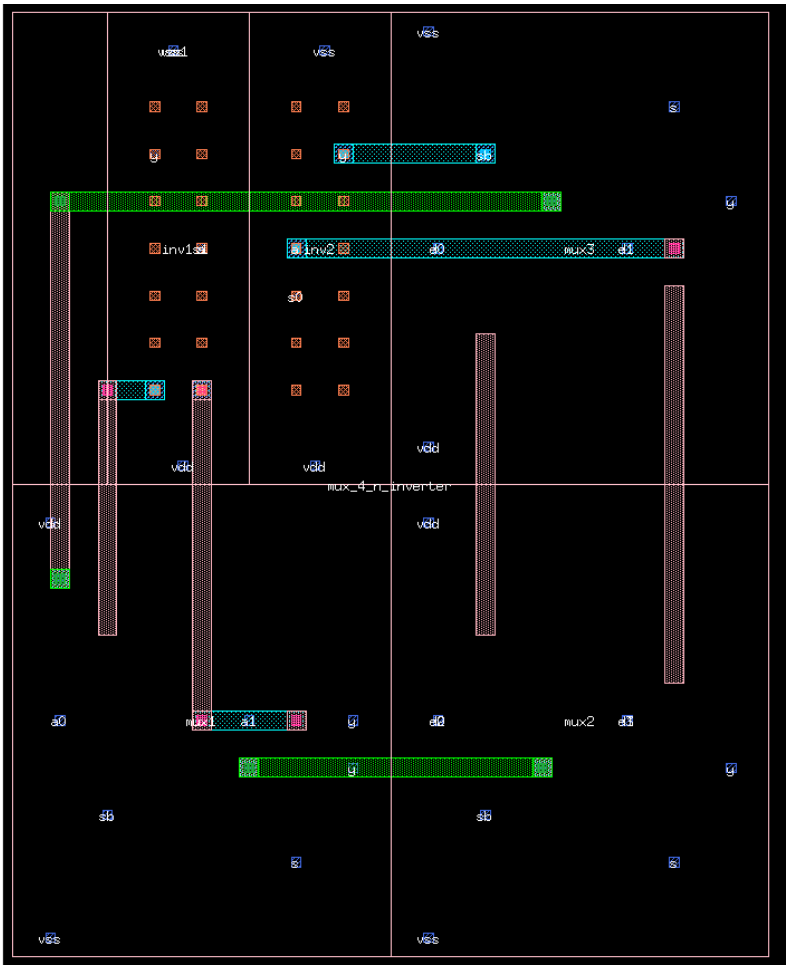

Figura 23: Modelo de construção de um Mux 4:1 non-Inverter.

```
1
2 .include mux_4_n_inverter.spi
3
4 * INTERF d0 d1 d2 d3 s0 s1 vdd vss vss1 y
5
6 X1 10 11 12 13 20 21 40 30 30 50 mux_4_n_inverter
7
8 V0 10 0 sine(0 1.8V 40Meg)
9 V1 11 0 sine(0 1.8V 70Meg)
10 V2 12 0 sine(0 1.8V 100Meg)
11 V3 13 0 sine(0 1.8V 120Meg)
12 V4 20 0 pulse(-1.8V 1.8V 100n 1p 1p 100n 200n)
13 V5 21 0 pulse(-1.8V 1.8V 200n 1p 1p 200n 400n)
14
15
16 *R1 20 0 10Meg
17
18 Vdd 40 0 1.8V
19 Vss 30 0 -1.8V
20
21 .model tp pmos level = 54
22 .model tn nmos level = 54
23
24 .tran 0.01ns 800ns
25 .end
```

Figura 24: Arquivo .cir para simulação do Mux 4:1 non-Inverter.

Tabela 9: Tabela de cores da simulação do Mux 4:1 non-Inverter.

Componentes	***Cor***
V(10) (Fonte V0)	Vermelho
V(11) (Fonte V1)	Verde
V(12) (Fonte V2)	Azul
V(13) (Fonte V3)	Roxo
V(20) (Fonte V4)	Preto
V(21) (Fonte V5)	Amarelo
V(50) (Saída Mux 4:1 non-Inverter)	Rosa

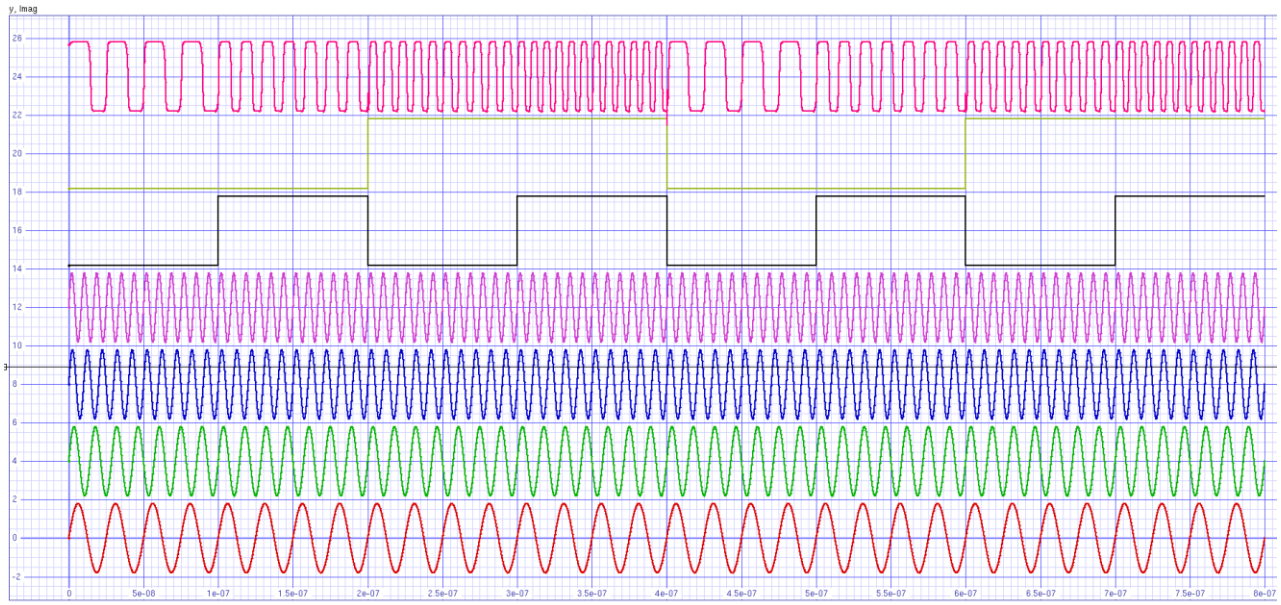

Figura 25: Gráfico do comportamento do Mux 4:1 non-Inverter.

7 Inverting Mux

O modelo a seguir foi feito a partir dos tristates inverters sem inversor de entrada que podem ser vistos anteriormente no tópico 4.2. Com as devidas ligações, chegamos ao seguinte layout:

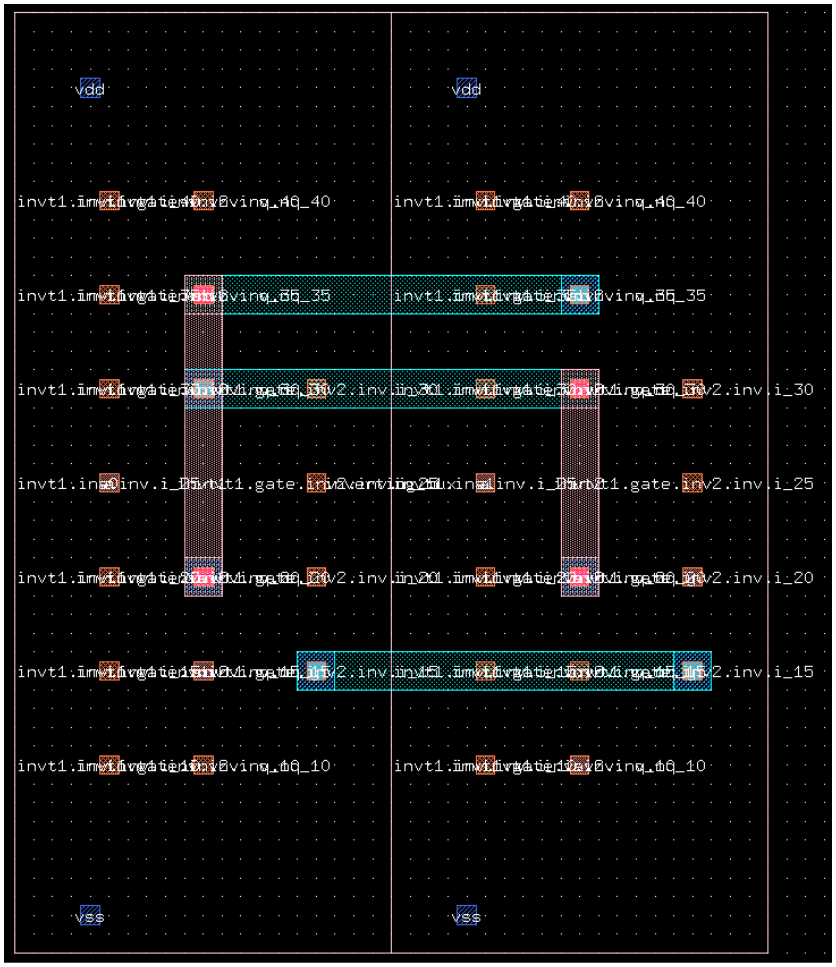

Figura 26: Modelo de construção de um Inverting Mux.

```
1
2 .include inverting_mux.spi
3 .include inversor_1.spi
4
5 * INTERF a0 a1 s sb vdd vss y
6 * INTERF a vdd vss y
7
8 X1 10 11 12 13 40 30 20 inverting_mux
9 X2 13 40 30 12 inversor_1
10
11 V0 10 0 sine(0 1.8V 70Meg)
12 V1 11 0 sine(0 1.8V 40Meg)
13 V2 13 0 pulse(-1.8V 1.8V 20n 1p 1p 100n 200n)
14
15
16 *R1 20 0 10Meg
17
18 Vdd 40 0 1.8V
19 Vss 30 0 -1.8V
20
21 .model tp pmos level = 54
22 .model tn nmos level = 54
23
24 .tran 0.01ns 400ns
25 .end
```

Figura 27: Arquivo .cir para simulação do Inverting Mux.

Tabela 10: Tabela de cores da simulação do Inverting Mux.

Componentes	***Cor***
V(10) (Fonte V0)	Vermelho
V(11) (Fonte V1)	Verde
V(13) (Fonte V2)	Azul
V(12) (Saída inversor)	Roxo
V(21) (Saída Inverting Mux)	Preto

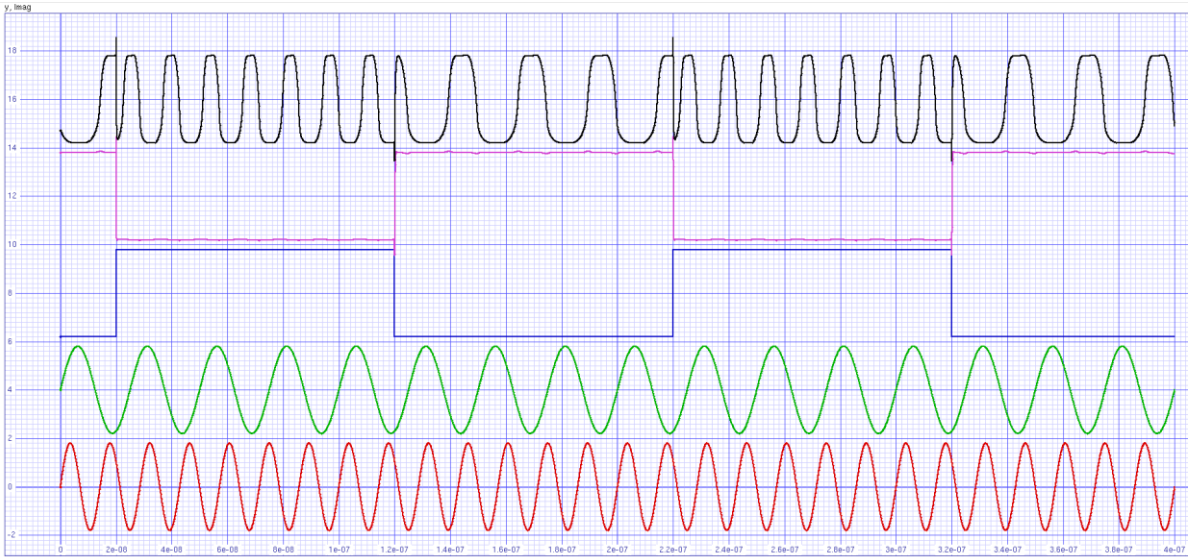

Figura 28: Gráfico do comportamento do Inverting Mux.

7.1 Tempo de propagação Low-High e High-Low

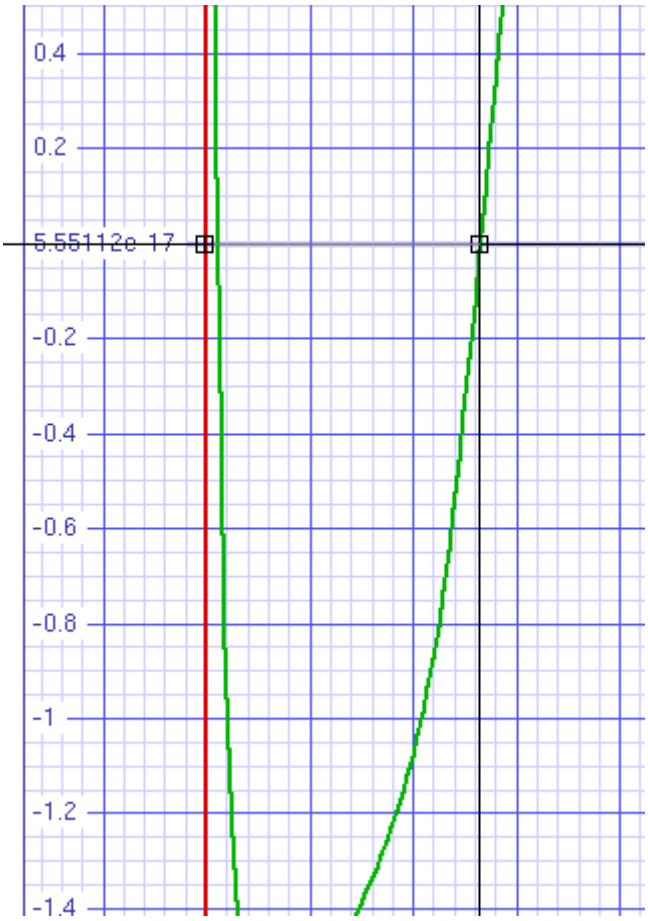

Figura 29: Tempo de propagação Low-High do Inverting Mux.

Press <space> to identify nearest curve.
x-y grid displaying real vs default.
Marker: x =1.997919032439295e-08 y =-1.307086966238801e-03
Cursor: x =2.263839889872107e-08 y =-1.307086966238801e-03
Delta : dx=2.659208574328123e-09 dy=0.000000000000000e+00

Figura 30: Resultado da análise Low-High do Inverting Mux.

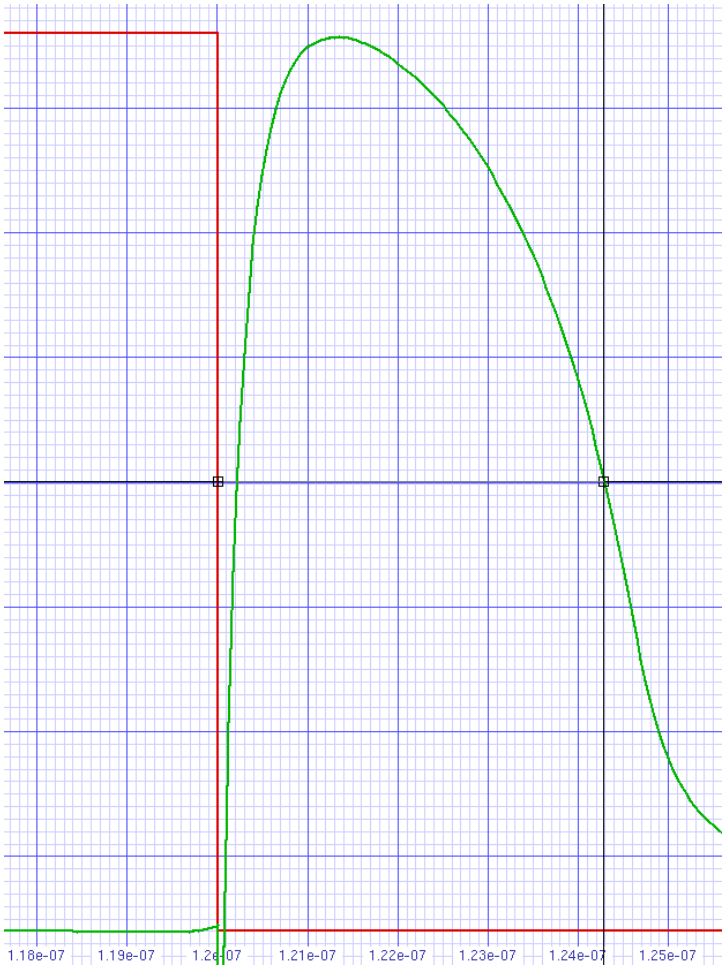

Figura 31: Tempo de propagação High-Low do Inverting Mux.

Press <space> to identify nearest curve.
x-y grid displaying real vs default.
Marker: x =1.200068364117085e-07 y =3.321643018124639e-03
Cursor: x =1.242809035488258e-07 y =3.321643018124639e-03
Delta : dx=4.274067137117224e-09 dy=0.000000000000000e+00

Figura 32: Resultado da análise High-Low do Inverting Mux.

7.2 Tempo de subida e descida

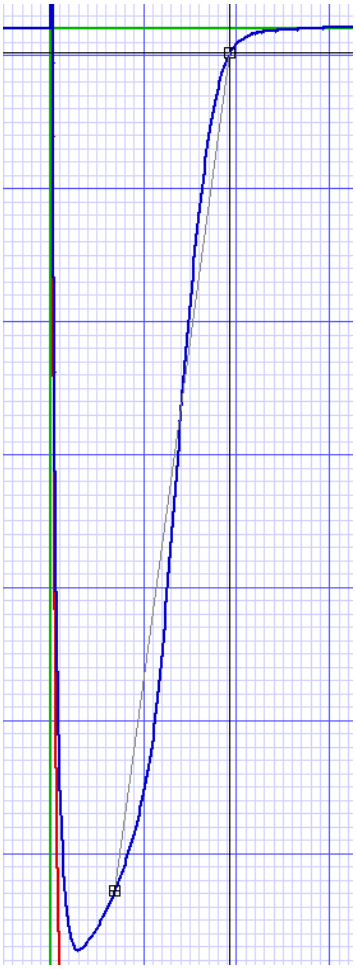

Figura 33: Tempo de subida do Inverting Mux.

```
Press <space> to identify nearest curve.
x-y grid displaying real vs default.
Marker: x =2.138158920165840e-08   y =3.610376563418409e-01
Cursor: x =2.385049409270861e-08   y =3.515294908358627e+00
Delta : dx=2.468904891050218e-09   dy=3.154257252016786e+00
```

Figura 34: Valores do tempo de subida do Inverting Mux.

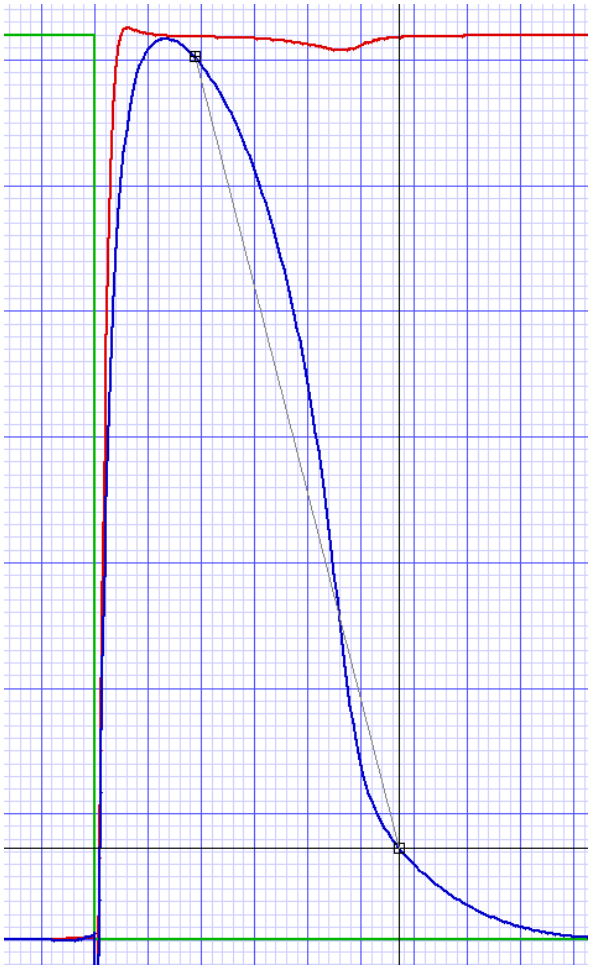

Figura 35: Tempo de descida do Inverting Mux.

```
Press <space> to identify nearest curve.
x-y grid displaying real vs default.
Marker: x =1.218833179012736e-07    y =3.514752191012741e+00
Cursor: x =1.257164224918831e-07    y =3.633521861992974e-01
Delta : dx=3.833104590609510e-09    dy=-3.151400004813444e+00
```

Figura 36: Valores do tempo de descida do Inverting Mux.

8 Non-Overlapping Clock (NOC)

Para a construção deste, utilizamos os modelos prontos de inversores e NORs disponíveis na biblioteca e, após a construção, chegamos ao seguinte modelo:

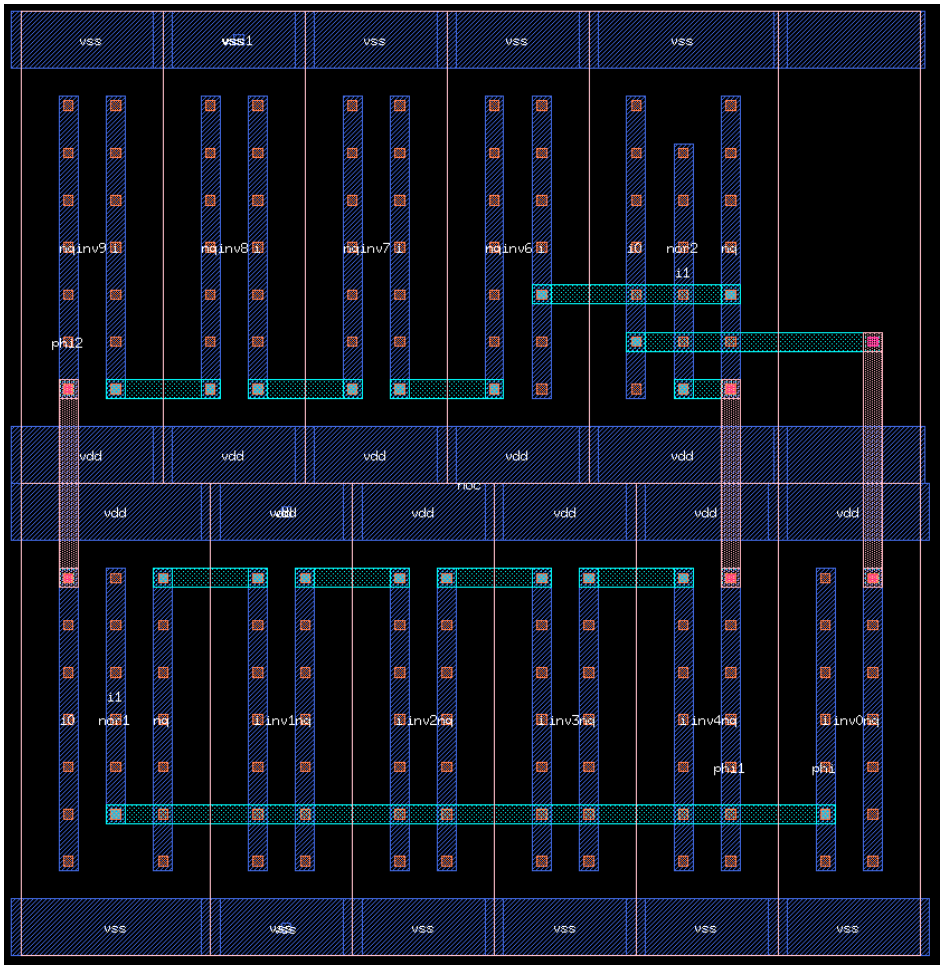

Figura 37: Modelo de construção de um Non-Overlapping Clock (NOC).

```
1
2 .include noc.spi
3
4 * INTERF phi phi1 phi2 vdd vss vss1
5
6 X1 20 10 11 40 30 30 noc
7
8 V1 20 0 pulse(0V 1.8V 5n 1p 1p 5n 10n)
9
10 *R1 20 0 10Meg
11
12 Vdd 40 0 1.8V
13 Vss 30 0 0V
14
15 .model tp pmos level = 54
16 .model tn nmos level = 54
17
18 .tran 0.01ns 20ns
19 .end
```

Figura 38: Arquivo .cir de um Non-Overlapping Clock (NOC).

Tabela 11: Tabela de cores da simulação do Non-Overlapping Clock (NOC).

Componentes	***Cor***
V(20) (Fonte V1)	Vermelho
V(10) (Phi1)	Verde
V(11) (Phi2)	Azul

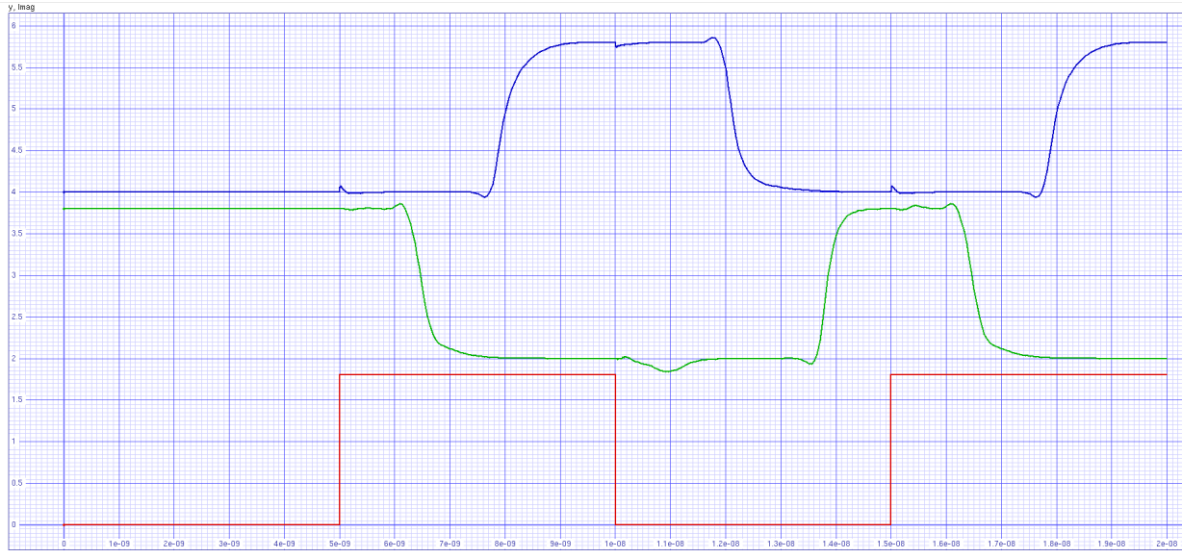

Figura 39: Gráfico do comportamento do Non-Overlapping Clock (NOC).

9 Non-Overlapping Clock (NOC) utilizando o Genlib

Aprendemos, então, a fazer circuitos de forma procedural por meio da biblioteca Genlib disponível na linguagem C. A partir deste conhecimento, pudemos chegar no circuito final, utilizando comandos como o genlib para extraírmos um arquivo VHDL a partir do arquivo C, de posicionamento utilizando o OCP e de roteamento utilizando o Nero, desta forma, chegaríamos num modelo standard cell que estamos habituados a criar durante a disciplina e, assim, podemos analisar o comportamento dos mesmos via SpiceOpus. Este procedimento que será demonstrado com o NOC a seguir, também foi utilizado para os demais circuitos que sucedem este.

```c
#include <genlib.h>

int main(void){
        GENLIB_DEF_LOFIG("noc");
        //Pinos externos
        //Sintaxe: “nome”, IN para entrada e OUT para saída, “nome”.
        GENLIB_LOCON("ck", IN, "ck");
        GENLIB_LOCON("i5", OUT, "i5");//phy_0
        GENLIB_LOCON("i10", OUT, "i10");//phy_1
        GENLIB_LOCON("vdd", IN, "vdd");
        GENLIB_LOCON("vss", IN, "vss");
        GENLIB_LOINS("inv_x1", "inv0", "ck", "ck_b", "vdd", "vss", NULL);

        int i = 0;

        //GENLIB_NAME concatena string com inteiro e gera strig - se aplica sobretudo a NOMES com numeração sequencial
        for(i = 1; i <5;i++){
                GENLIB_LOINS("inv_x1", GENLIB_NAME("inv%d", i), GENLIB_NAME("i%d", i), GENLIB_NAME("i%d", i+1), "vdd", "vss", NULL);
        }
        for(i = 6; i<10;i++){
                GENLIB_LOINS("inv_x1", GENLIB_NAME("inv%d", i), GENLIB_NAME("i%d", i), GENLIB_NAME("i%d", i+1), "vdd", "vss", NULL);
        }

        GENLIB_LOINS("no2_x1", "nor1", "ck", "i10", "i1", "vdd", "vss", NULL);
        GENLIB_LOINS("no2_x1", "nor2", "ck_b", "i5", "i6", "vdd", "vss", NULL);
        GENLIB_SAVE_LOFIG();
return 0;
```

Figura 40: Arquivo C com a descrição do NOC.

EXPORT MBK_OUT_LO=vst

Genlib noc_genlib

EXPORT MBK_IN_LO=vst

Xsch –l ‘noc’&

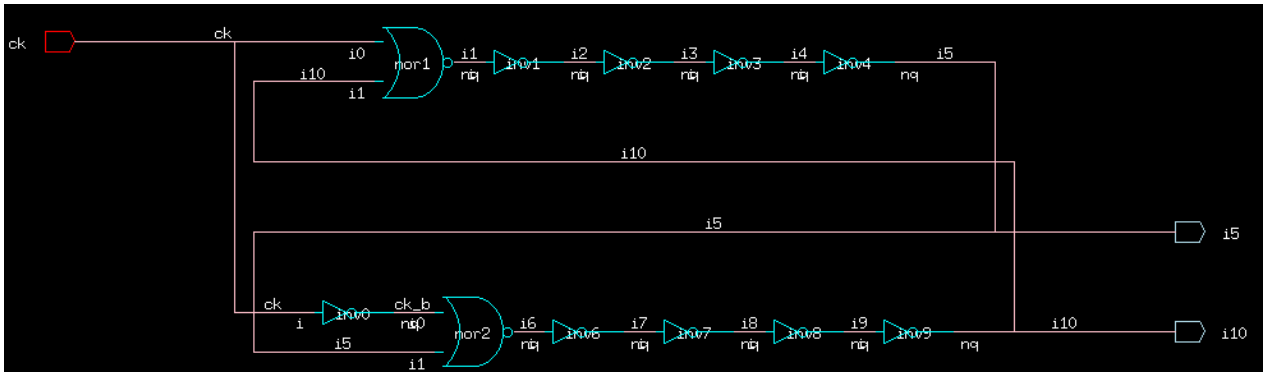

Figura 41: Diagrama lógico do NOC.

Alliance-ocp noc noc_posicionado

Graal –l ‘noc_posicionado’&

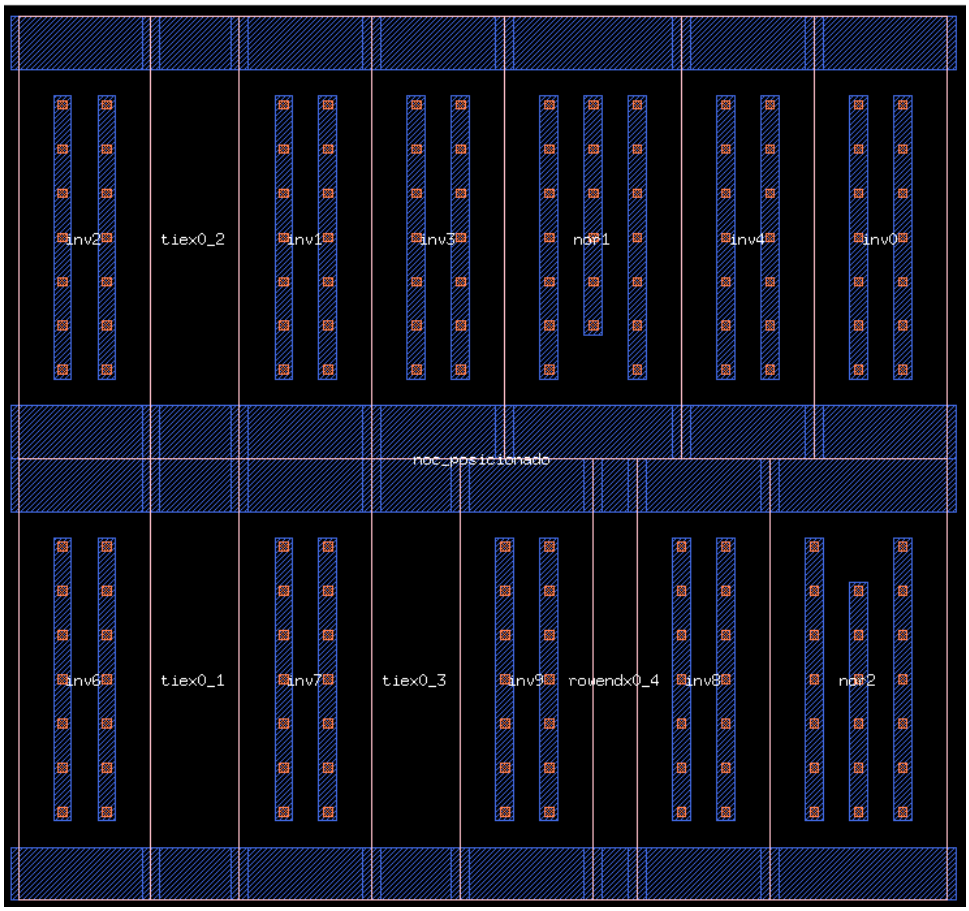

Figura 42: Resultado do posicionamento dos componentes.

Nero –p noc_posicionado noc noc_roteado

Graal –l 'noc_roteado'&

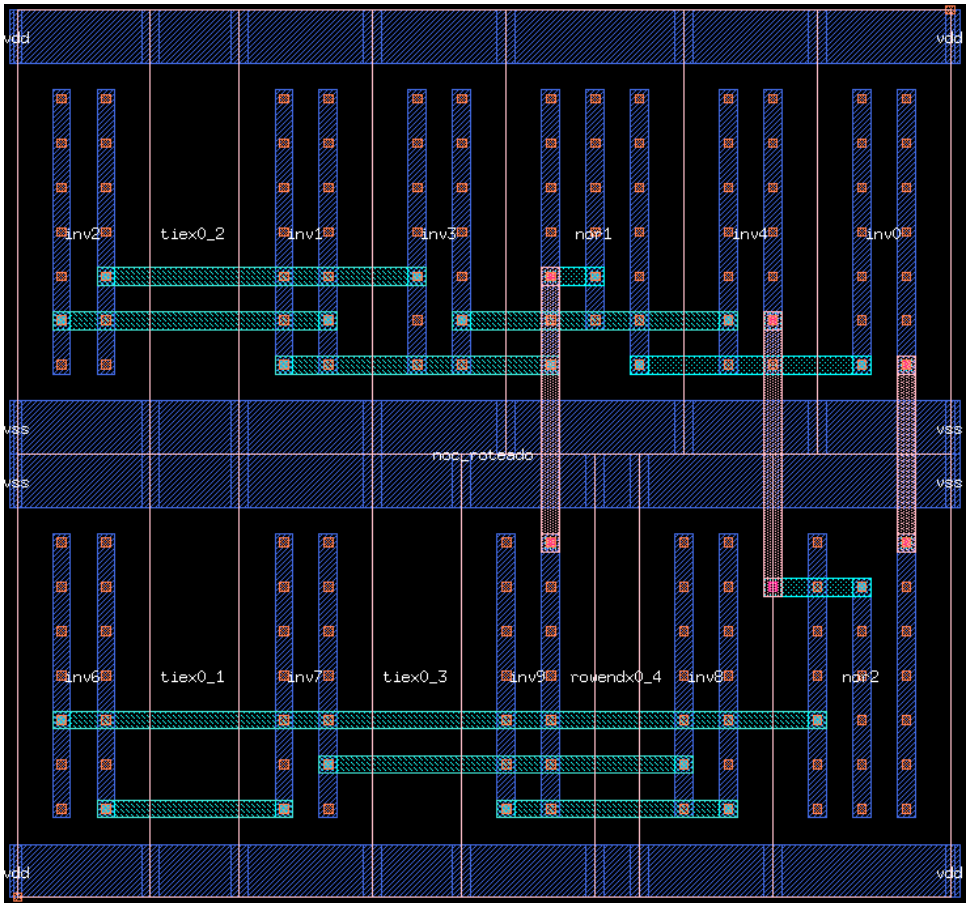

Figura 43: Resultado do roteamento dos componentes.

Após posicionado e roteado, exportaremos a standard cell resultante para arquivo SPICE, parecido como fazíamos com os modelos anteriores, com alteração na sintaxe do comando Cougar de –ar para –ac, para não extrair as resistências, de forma a diminuir os equipotenciais na extração do arquivo spi.

EXPORT MBK_OUT_LO=spi

Cougar –t –ac noc_roteado

```
1
2 .include noc.spi
3 .include noc_roteado.spi
4
5 * INTERF phi phi1 phi2 vdd vss vss1
6 * INTERF ck i10 i5 vdd vss
7
8 X1 20 10 11 40 30 30 noc
9 X2 20 12 13 40 30 noc_roteado
10
11 V1 20 0 pulse(0V 1.8V 7n 1p 1p 5n 10n)
12
13 *R1 20 0 10Meg
14
15 Vdd 40 0 1.8V
16 Vss 30 0 0V
17
18 .model tp pmos level = 54
19 .model tn nmos level = 54
20
21 .tran 0.01ns 40ns
22 .end
```

Figura 44: Arquivo .cir do NOC feito “à mão” no tópico 8 e o NOC resultante das operações anteriores.

Tabela 12: Tabela de cores da simulação do Non-Overlapping Clock (NOC) junto com o NOC feito “à mão”.

Componentes	***Cor***
V(20) (Clock)	Vermelho
V(10) (Phi1)	Verde
V(11) (Phi2)	Azul
V(12) (i10)	Roxo
V(13) (i5)	Preto

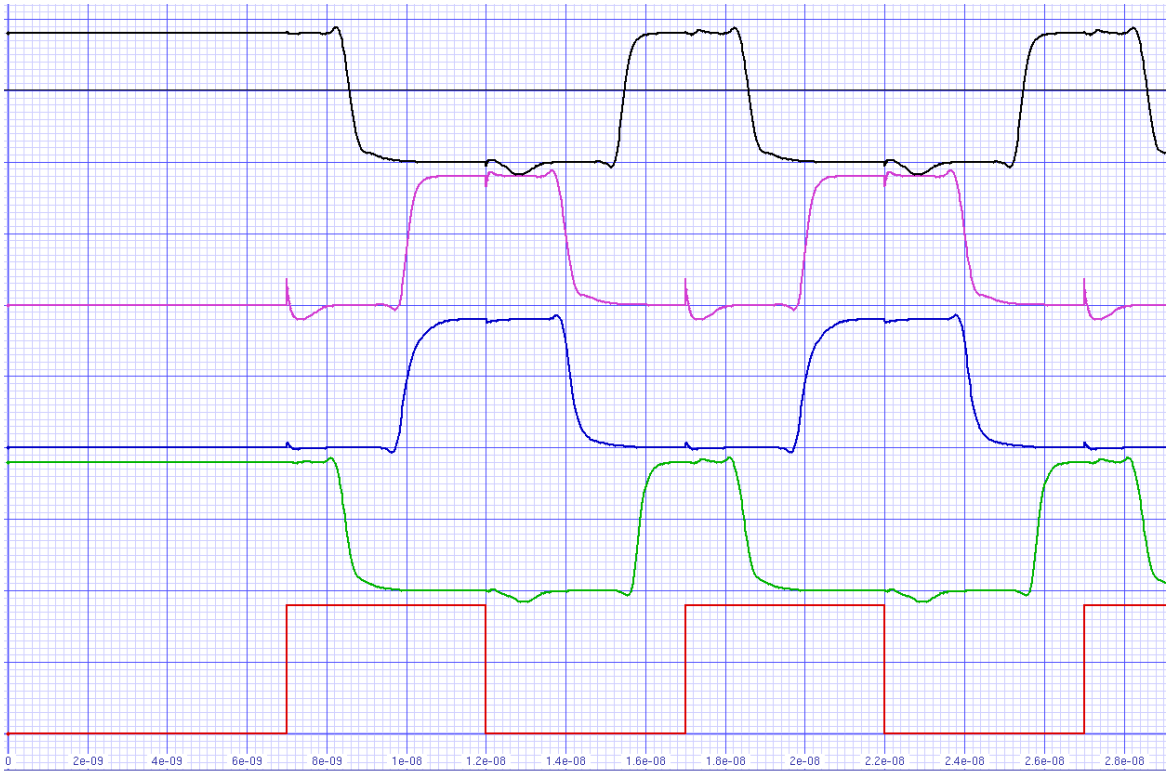

Figura 45: Gráfico do comportamento dos NOCs.

10 Latch D

```
#include <genlib.h>
int main(void){
GENLIB_DEF_LOFIG("latch_d");
//Pinos externos
//Sintaxe: “nome”, IN para entrada e OUT para saída, “nome”.
GENLIB_LOCON("a", IN, "a");
GENLIB_LOCON("ck", IN, "ck");
GENLIB_LOCON("q", OUT, "q");
GENLIB_LOCON("vdd", IN, "vdd");
GENLIB_LOCON("vss", IN, "vss");
//Instâncias
//Sintaxe: (“arquivo da porta lógica”, “nome da instância”, “entrada”, “saída”,
//“vdd, vss”, NULL (fim da lista);
GENLIB_LOINS("inv_x1", "inv_1", "b", "q", "vdd", "vss", NULL);
GENLIB_LOINS("inv_x1", "inv_2", "q", "q_b", "vdd", "vss", NULL);
GENLIB_LOINS("mx2_x2", "mux", "ck", "q_b", "a", "b", "vdd", "vss", NULL);
GENLIB_SAVE_LOFIG();
return 0;
}
```

Figura 46: Arquivo C com a descrição do Latch D.

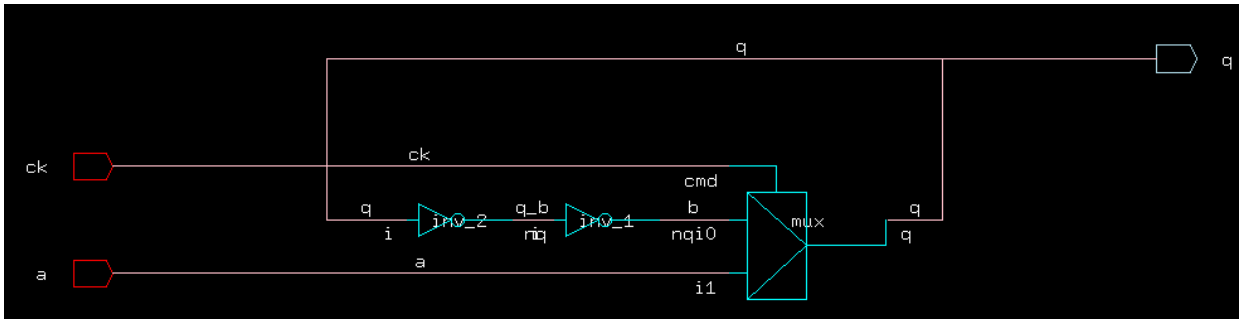

Figura 47: Diagrama lógico do Latch D.

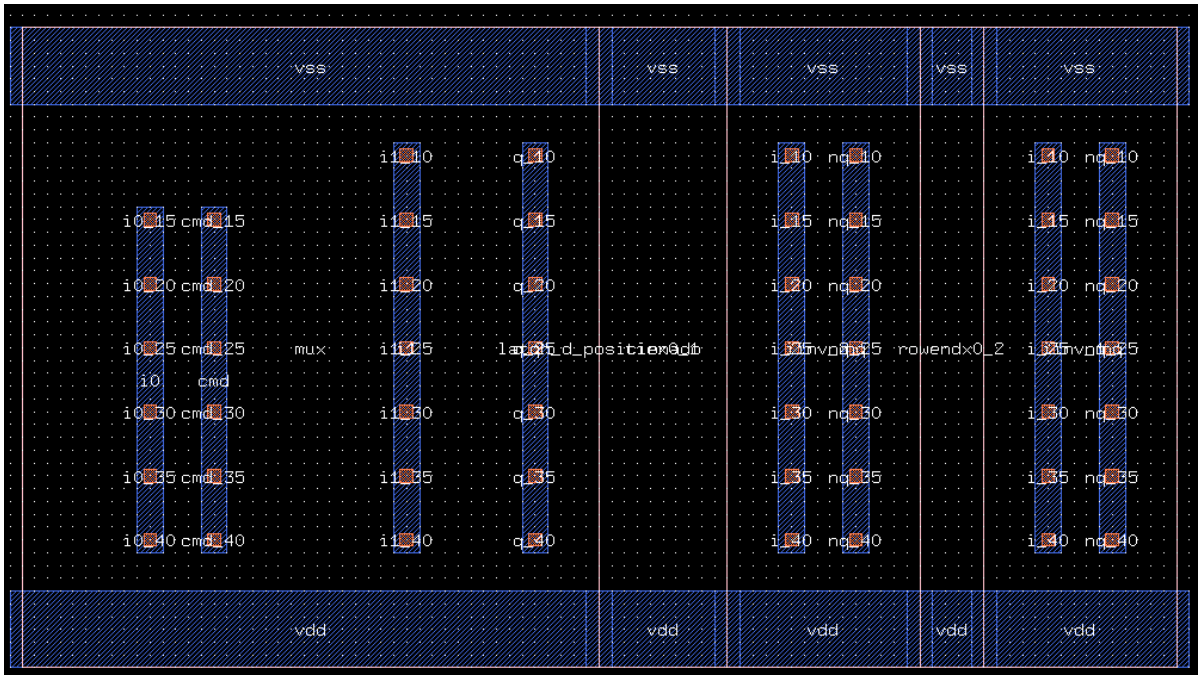

Figura 48: Resultado do posicionamento do Latch D.

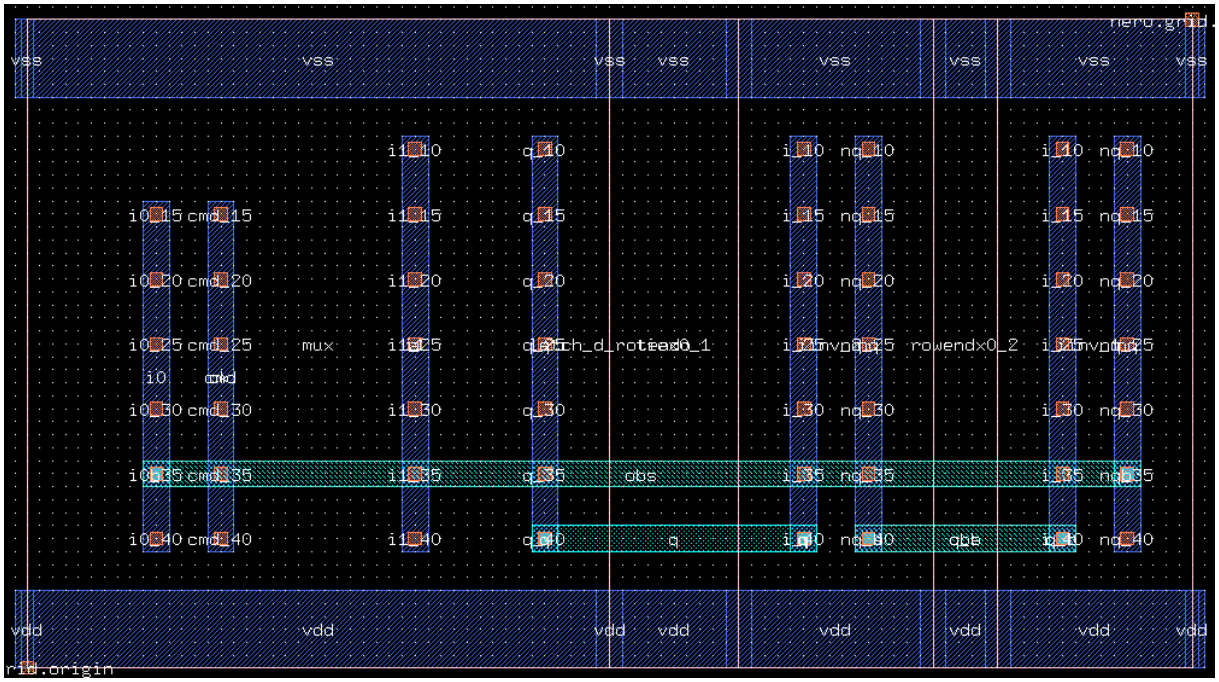

Figura 49: Resultado do roteamento do Latch D.

```
1
2 .include latch_d_roteado.spi
3
4 * INTERF a ck q vdd vss
5
6 X1 20 12 13 40 30 latch_D_roteado
7
8 V1 12 30 pulse(0V 1.8V 7n 1p 1p 5n 10n)
9 V2 20 30 pulse(0V 1.8V 0 1p 1p 12n 24n)
10
11 *R1 20 0 10Meg
12
13 Vdd 40 0 1.8V
14 Vss 30 0 0V
15
16 .model tp pmos level = 54
17 .model tn nmos level = 54
18
19 .tran 0.01ns 96ns
20 .end
```

Figura 50: Arquivo .cir para simulação do Latch D.

Tabela 13: Tabela de cores da simulação do Latch D.

Componentes	***Cor***
V(20) (a)	Vermelho
V(12) (Clock)	Verde
V(13) (Q)	Azul

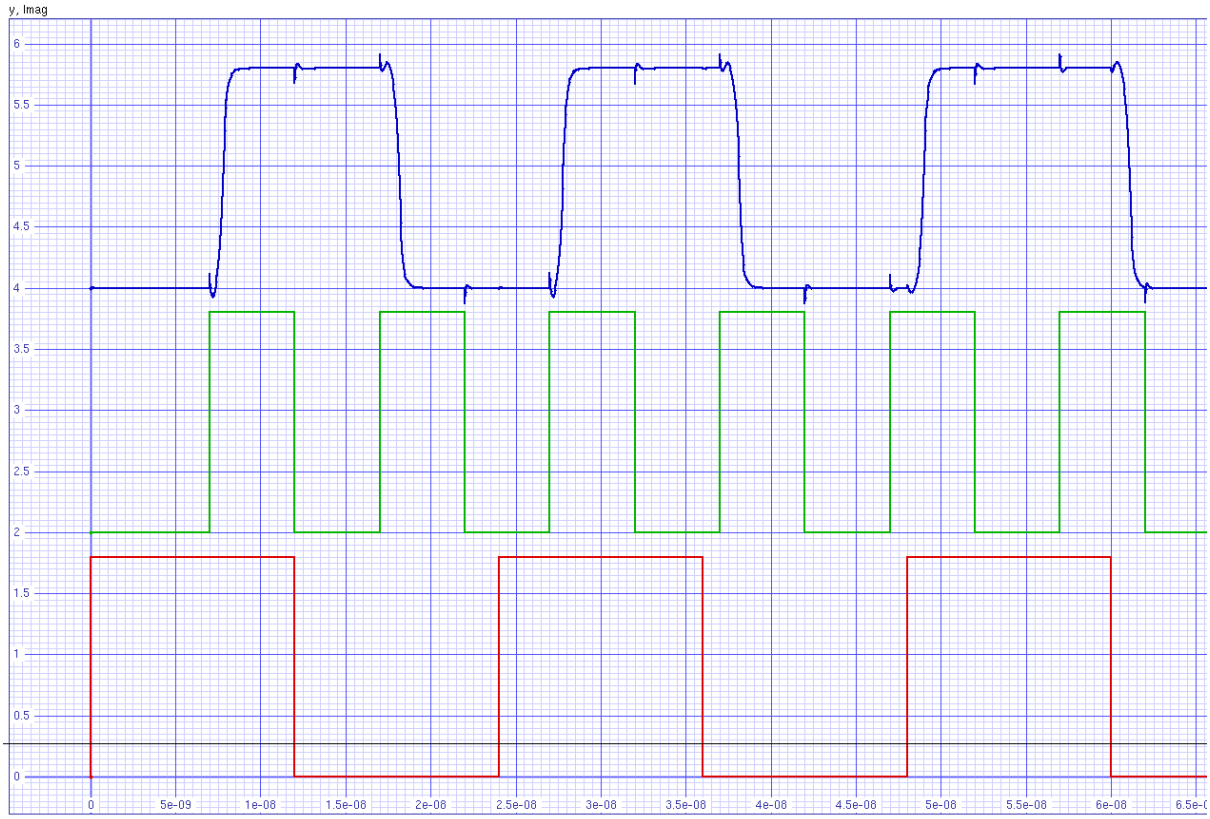

Figura 51: Gráfico do comportamento do Latch D.

11 Flip-Flop Mestre-Escravo

```c
#include <genlib.h>
int main(void){
GENLIB_DEF_LOFIG("ff_m_e");
//Pinos externos
//Sintaxe: “nome”, IN para entrada e OUT para saída, “nome”.
GENLIB_LOCON("a", IN, "a");
GENLIB_LOCON("ck_m", IN, "ck_m");
GENLIB_LOCON("ck_e", IN, "ck_e");
GENLIB_LOCON("q", OUT, "q");
GENLIB_LOCON("vdd", IN, "vdd");
GENLIB_LOCON("vss", IN, "vss");
//Instâncias
//Sintaxe: (“arquivo da porta lógica”, “nome da instância”, “entrada”, “saída”,
//“vdd, vss ”, NULL (fim da instância)); Instanciação explícita: pino a pino a => b
GENLIB_LOINSE("latch_d", "la_m", "a =>a", "ck =>ck_m", "q =>q_m", "vdd =>vdd",
"vss =>vss", NULL);
GENLIB_LOINSE("latch_d", "la_e", "a =>q_m", "ck =>ck_e", "q =>q", "vdd =>vdd",
"vss =>vss", NULL);
GENLIB_SAVE_LOFIG();
return 0;
}
```

Figura 52: Arquivo C do Flip-Flop Mestre-Escravo.

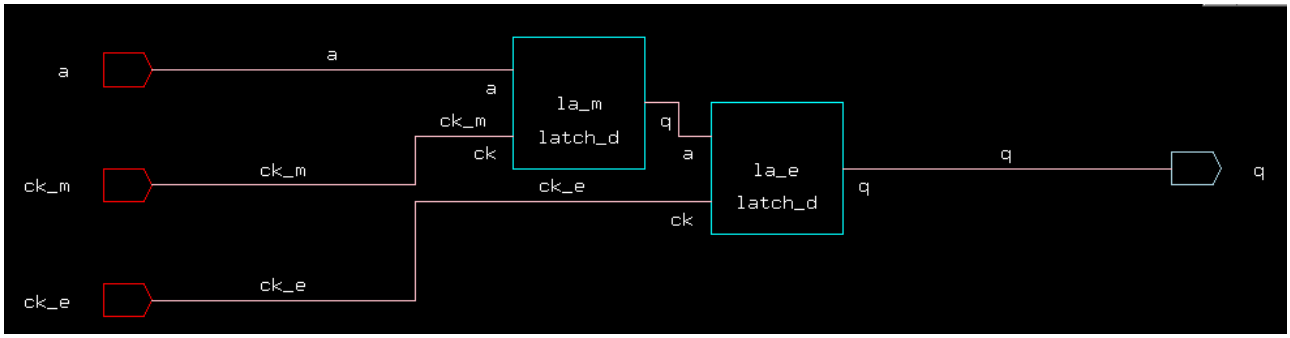

Figura 53: Diagrama lógico do Flip-Flop Mestre-Escravo.

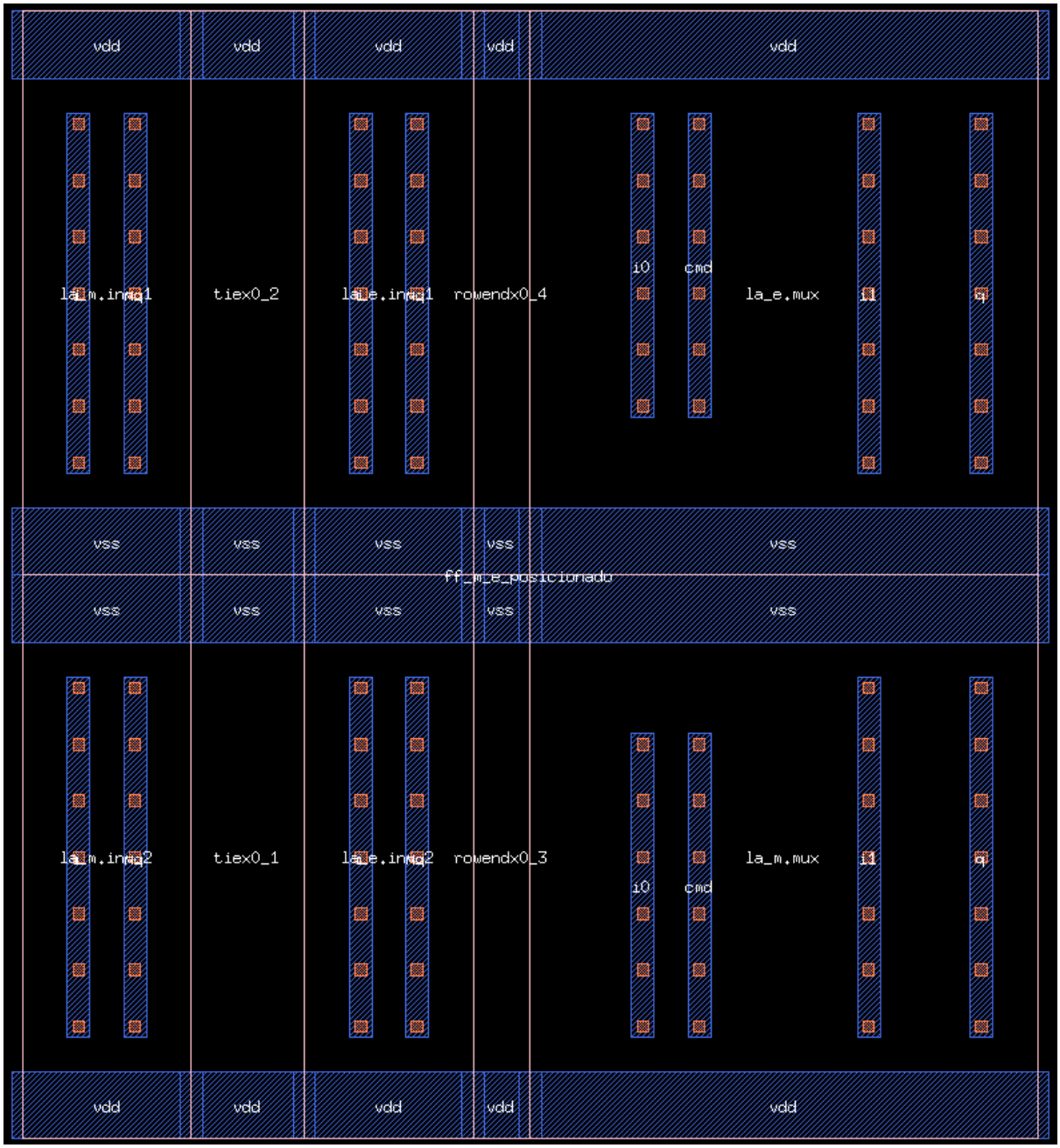

Figura 54: Resultado do posicionamento do Flip-Flop Mestre-Escravo.

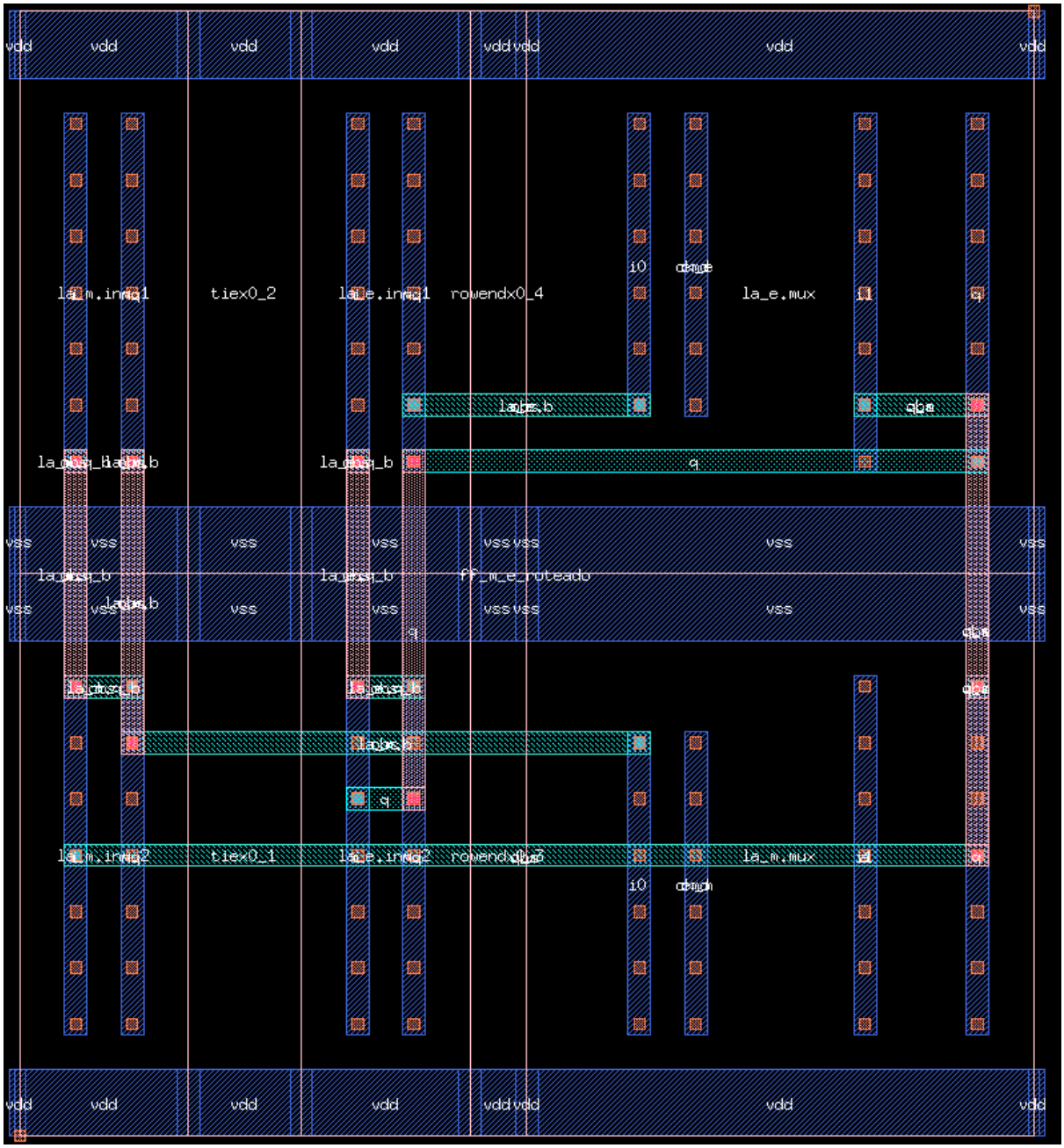

Figura 55: Resultado do roteamento do Flip-Flop Mestre-Escravo.

12 Flip-Flop Mestre-Escravo com Non-Overlapping Clock (NOC)

```c
#include <genlib.h>
int main(void){
GENLIB_DEF_LOFIG("ff_m_e_noc");
//Pinos externos
//Sintaxe: “nome”, IN para entrada e OUT para saída, “nome”.
GENLIB_LOCON("a", IN, "a");
GENLIB_LOCON("ck", IN, "ck");
GENLIB_LOCON("q", OUT, "q");
GENLIB_LOCON("phi_1", OUT, "phi_1");
GENLIB_LOCON("phi_2", OUT, "phi_2");
GENLIB_LOCON("vdd", IN, "vdd");
GENLIB_LOCON("vss", IN, "vss");
//Instâncias conectadas (netlist) - aqui descrita em "C"

GENLIB_LOINSE("noc", "nock", "ck =>ck", "i5 =>phi_1", "i10 =>phi_2", "vdd =>vdd", "vss =>vss", NULL);
GENLIB_LOINSE("ff_m_e", "ff", "a =>a", "ck_m =>phi_2", "ck_e =>phi_1", "q =>q", "vdd =>vdd", "vss =>vss", NULL);
GENLIB_SAVE_LOFIG();
return 0;
}
```

Figura 56: Arquivo C do Flip-Flop Mestre-Escravo com NOC.

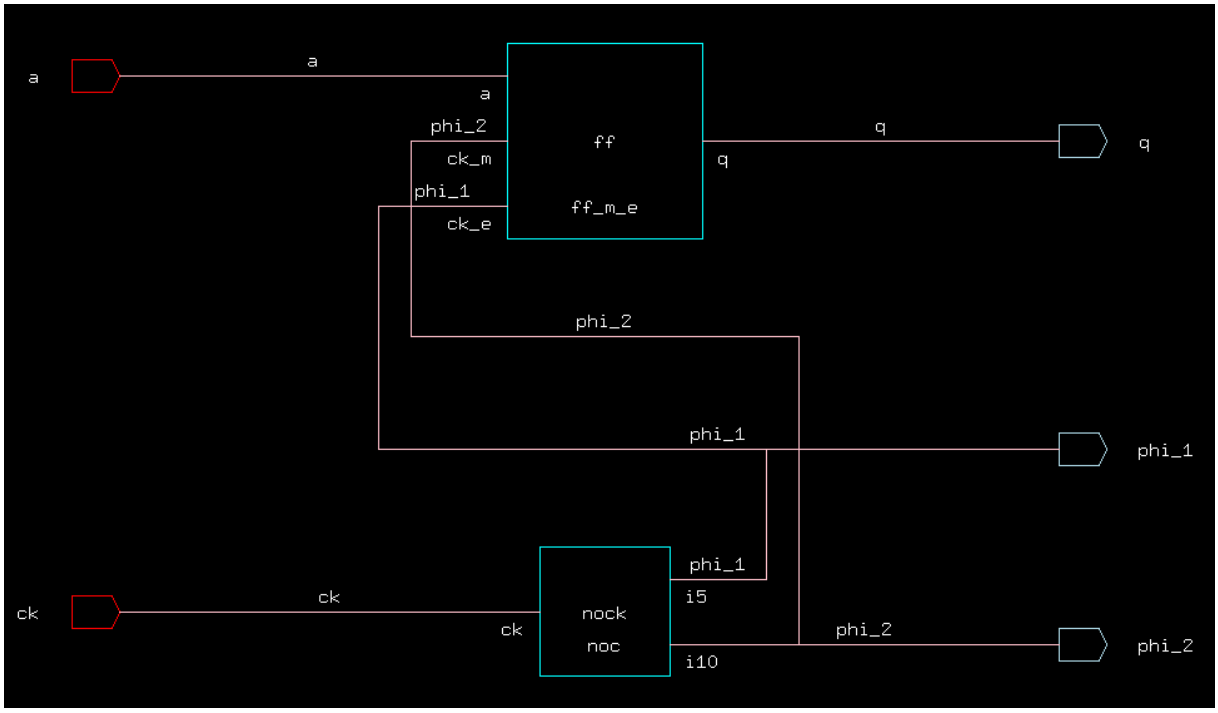

Figura 57: Diagrama lógico do Flip-Flop Mestre-Escravo com NOC.

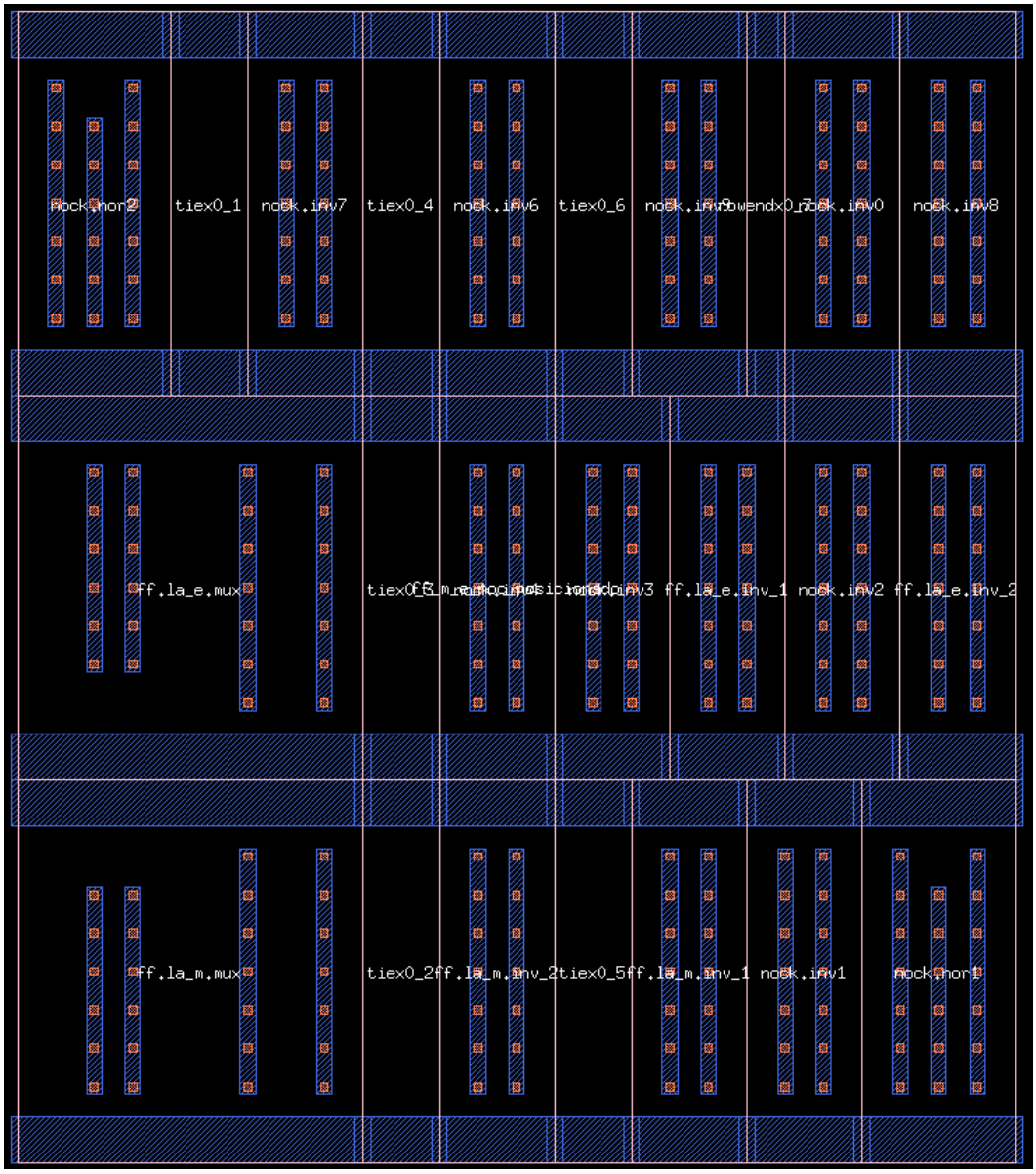

Figura 58: Resultado do posicionamento do Flip-Flop Mestre-Escravo com NOC.

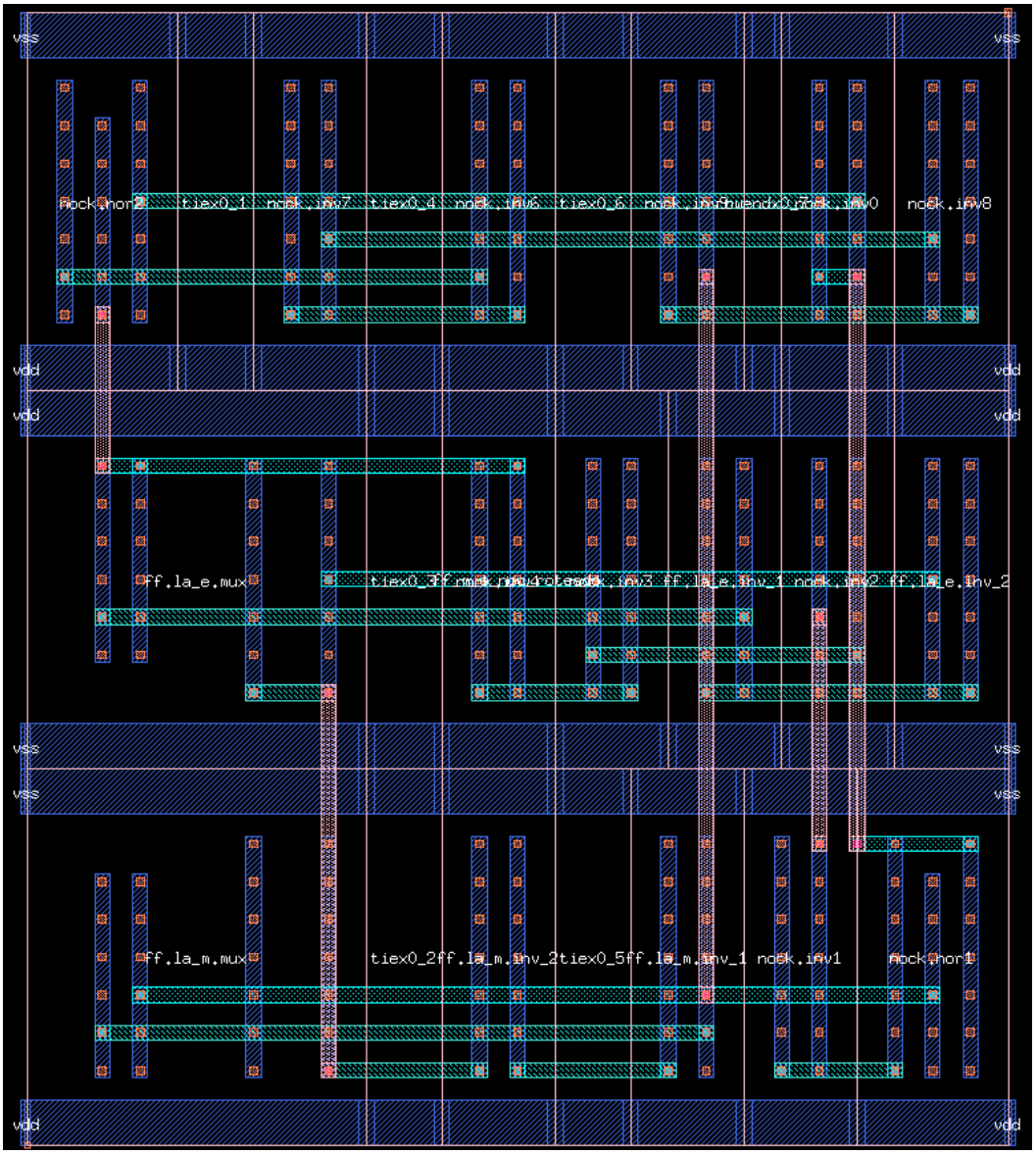

Figura 59: Resultado do roteamento do Flip-Flop Mestre-Escravo com NOC.

```
1
2 .include ff_m_e_noc_roteado.spi
3 .include latch_d_roteado.spi
4
5 * INTERF a ck phi_1 phi_2 q vdd vss
6 * INTERF a ck q vdd vss
7
8 X1 20 12 13 14 15 40 30 ff_m_e_noc_roteado
9 X2 20 12 16 40 30 latch_d_roteado
10
11 V1 12 30 pulse(0V 1.8V 50n 1p 1p 50n 100n)
12 V2 20 30 pulse(0V 1.8V 20n 1p 1p 120n 240n)
13
14 *R1 20 0 10Meg
15
16 Vdd 40 0 1.8V
17 Vss 30 0 0V
18
19 .model tp pmos level = 54
20 .model tn nmos level = 54
21
22 .tran 0.1ns 960ns
23 .end
```

Figura 60: Arquivo .cir para simulação do Flip-Flop Mestre-Escravo com NOC junto ao Latch D feito anteriormente.

Tabela 14: Tabela de cores da simulação do Flip-Flop Mestre-Escravo com NOC e Latch D.

Componentes	***Cor***
V(20) (a)	Vermelho
V(12) (clock)	Verde
V(13) (Phi1)	Azul
V(14) (Phi2)	Roxo
V(15) (Q Flip-Flop)	Preto
V(16) (Q latch D)	Amarelo

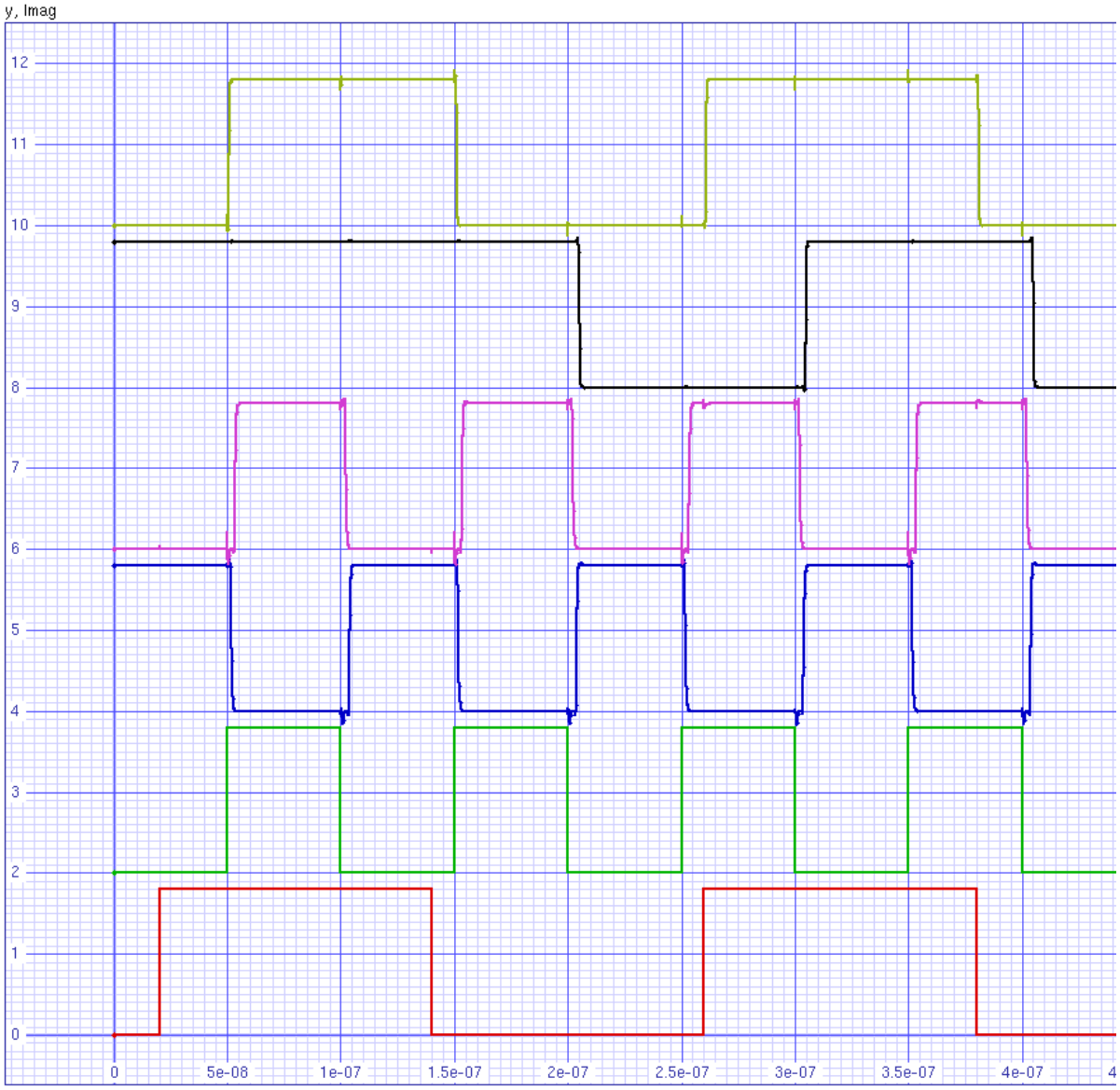

Figura 61: Gráfico do comportamento do Flip-Flop Mestre-Escravo com NOC e do Latch D.

13 Conclusão

Durante a construção dos circuitos a partir do transistor de passagem até o Non-Overlapping Clock (NOC), utilizamos o mesmo método de criação que fizemos na parte 1 do relatório, ou seja, os circuitos foram apresentados e feitos durante os encontros síncronos. Em suma, construímos a standard cell via Graal dos circuitos apresentados e o simulamos para análise de seus comportamentos via SpiceOpus.

Dito isto, prosseguimos para a abordagem procedural, utilizando a linguagem C e a biblioteca Genlib. Desta forma, a construção destes circuitos se tornou menos trabalhosa, dado que as tecnologias contribuem (e muito) com sua sintaxe simples (tanto pela biblioteca de Genlib quanto para ajustes de posição e roteamento usando o OCP e Nero pela linha de

comando) e a preservação das regras de desenho que são exigidas na construção via Graal, que também demandava um certo tempo.

Conseguimos perceber um avanço considerável na eficiência (em relação ao tempo de construção) quando comparamos o método padrão de criação de standard cells manualmente em relação à forma procedural de construção. Vale notar que, ao utilizarmos funções como OCP e Nero, é normal que os circuitos sejam posicionados e roteados de formas diferentes à cada execução.

Desta forma, se pegarmos o NOC feito manualmente e compararmos com o NOC feito de forma procedural, notamos que, neste caso, o NOC manual possui menos ligações de alumínio com relação ao NOC que foi roteado via Nero, então, deve-se notar que existe um pequeno prejuízo que, dependendo da aplicação ou do tamanho do projeto, pode ser considerado. Entretanto, ao fazer este procedimento por meio de uma tecnologia, evitamos o erro humano ao fazermos o roteamento manualmente que, com certeza, trará um prejuízo menor e uma eficiência bem maior.