UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
INTRODUÇÃO A MICROELETRÔNICA

UFPB

RELATÓRIO X

THIAGO ALVES DE ARAUJO

2016019787

João Pessoa
2018

1 – SOMADOR DE 1 BIT (COM INVERSOR E NAND FEITOS A MÃO)

O somador faz parte da família dos circuitos aritméticos. Estes são muito utilizados nas Unidades Lógicas Aritméticas (ULA) dos computadores. Um Somador Básico Completo é um Circuito Lógico que faz a adição entre dois números de 1 bit com o bit de transporte, ET. O Somador Completo consiste em uma Porta OU e dois Semi-Somadores. O circuito gera duas Saídas: S e ST.

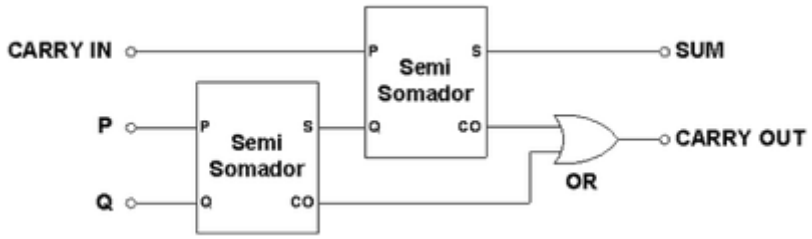


Figura 136 – Somador Completo

1.1 – EXPRESSÕES

A expressão da saída S e da saída Cout podem ser vistas abaixo na figura abaixo.

$$S = A \oplus B \oplus Cin$$

$$Cout = A.B + A.Cin + B.Cin$$

Figura 137 – Expressões para as saídas

Como podemos perceber, a expressão utiliza portas XOR, AND e OR. Como algumas destas portas não foram desenvolvidas anteriormente, podemos utilizar as leis De Morgan para que a mesma expressão seja representada apenas com portas NAND e INVERSOR. Abaixo está o passo a passo da simplificação da expressão.

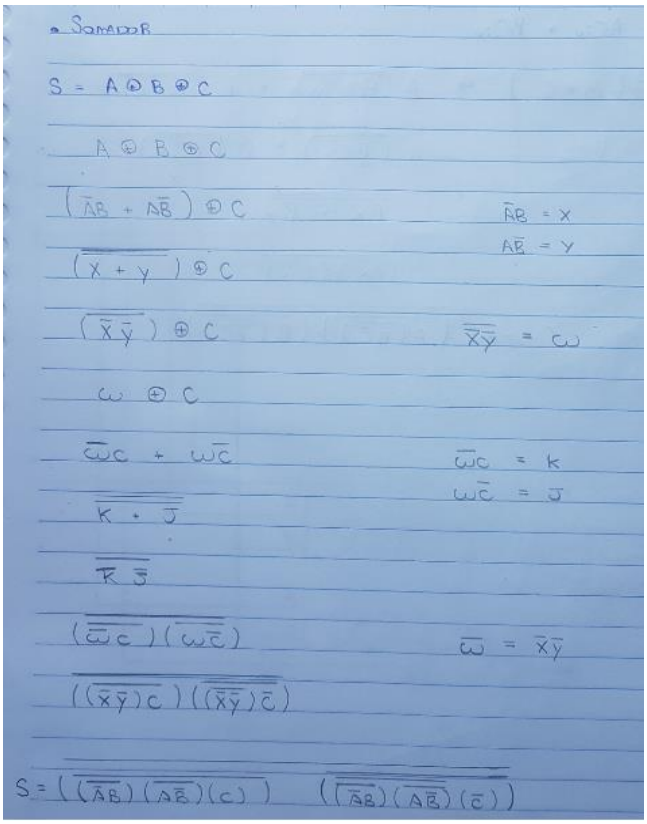


Figura 138 – Expansão da expressão da saída S

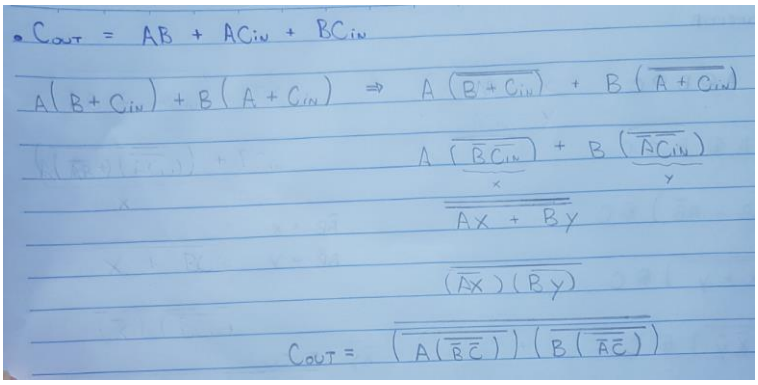


Figura 139 – Expansão da expressão da saída Cout

1.2 – CRIANDO CIRCUITO EM ALTO NIVEL

Para criarmos o somador, vamos utilizar um processo diferente dos utilizados anteriormente. Ao invés de conectarmos todas as células manualmente em baixo nível, vamos criar um código em linguagem de alto nível (C) que vai criar todas as conexões de forma automática. Para isso, devemos descrever todas as entradas, saídas e conexões

internas. Abaixo está o desenho do circuito com todos os nomes das entradas, saídas e fios de conexão interna.

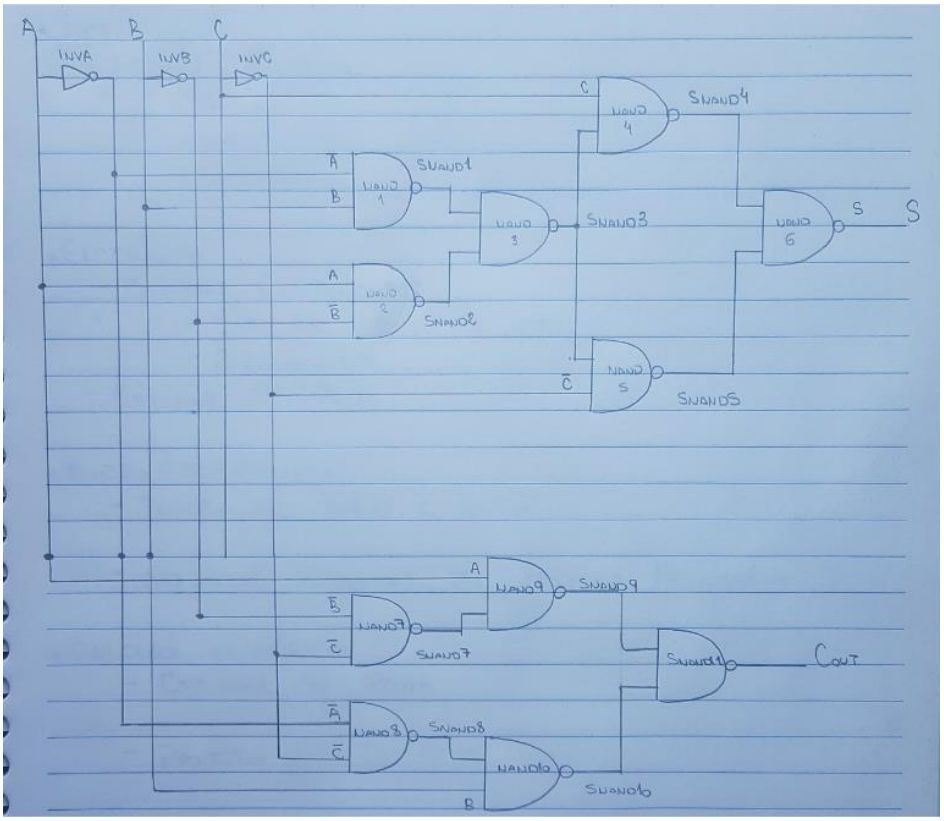


Figura 140 – Esquema das portas logicas do somador

Com os desenhos em mãos, podemos facilmente criar o código em C que vai gerar o circuito. Abaixo está o código gerado.

```c
#include <genlib.h>

int main()
{

GENLIB_DEF_LOFIG("somador_genlib");

GENLIB_LOCON("a", IN, "a");
GENLIB_LOCON("b", IN, "b");
GENLIB_LOCON("c", IN, "c");
GENLIB_LOCON("vdd", IN, "vdd");
GENLIB_LOCON("vss", IN, "vss");
GENLIB_LOCON("s", OUT, "s");
GENLIB_LOCON("cout", OUT, "cout");

//Inversores "entrada, saída, vdd, vss"

GENLIB_LOINS("inversor", "invA", "a", "na", "vdd", "vss", NULL);
GENLIB_LOINS("inversor", "invB", "b", "nb", "vdd", "vss", NULL);
GENLIB_LOINS("inversor", "invC", "c", "nc", "vdd", "vss", NULL);

//Nanda "entrada a, entrada b, saída, vdd, vss"
//saida S

GENLIB_LOINS("nand2", "nand1", "na", "b", "snand1", "vdd", "vss", NULL);
GENLIB_LOINS("nand2", "nand2", "a", "nb", "snand2", "vdd", "vss", NULL);
GENLIB_LOINS("nand2", "nand3",  "snand1", "snand2", "snand3", "vdd", "vss", NULL);
GENLIB_LOINS("nand2", "nand4",  "c", "snand3", "snand4","vdd", "vss", NULL);
GENLIB_LOINS("nand2", "nand5", "nc", "snand3", "snand5","vdd", "vss", NULL);
GENLIB_LOINS("nand2", "nand6", "snand4", "snand5", "s", "vdd", "vss", NULL);


//saida cout

GENLIB_LOINS("nand2", "nand7", "nb", "na", "snand7","vss", "vdd",NULL);
GENLIB_LOINS("nand2", "nand8", "na", "nc", "snand8", "vss", "vdd",NULL);
GENLIB_LOINS("nand2", "nand9", "a", "snand7", "snand9", "vss", "vdd",NULL);
GENLIB_LOINS("nand2", "nand10", "snand8", "b","snand10", "vss", "vdd",NULL);
GENLIB_LOINS("nand2", "nand11", "snand9", "snand10", "cout", "vss", "vdd",NULL);

GENLIB_SAVE_LOFIG();

return 0;

}
```

Figura 141 – Somador_genlib.c

A função GENIB_DEF_LOFIG define o nome que o arquivo vai ser extraído pelo genlib posteriormente. A função GENLIB_LOINS recebe os seguintes parâmetros: O primeiro parâmetro é o nome referente ao componente que está sendo instanciado. O segundo parâmetro é o nome que o programador escolhe para representar o componente no circuito (no desenho feitos a mão, cada nand recebe um nome). Os demais parâmetros são as conexões da porta instanciada (é possível ver a ordem das conexões extraindo o arquivo "vst" da célula).

1.3 – GENADO OS ARQUIVOS DE SIMULAÇÃO

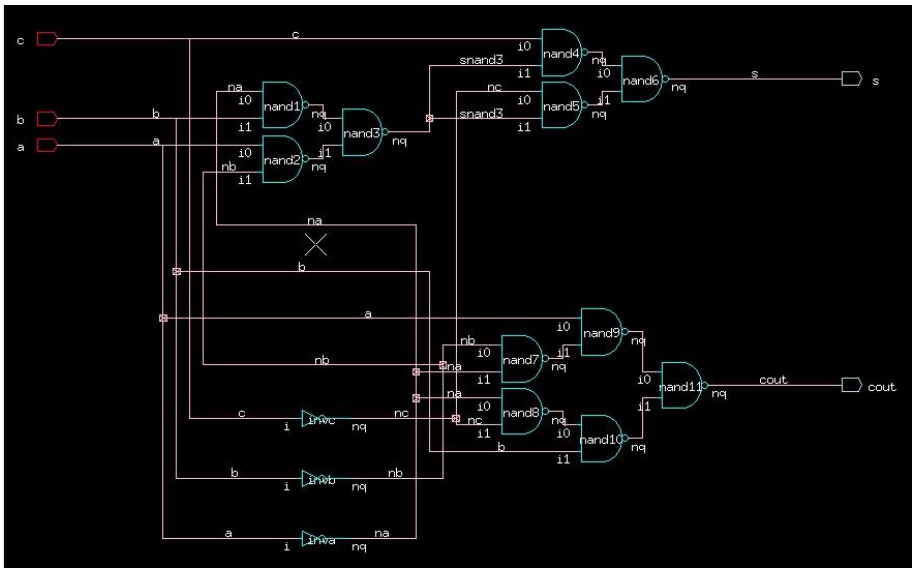
Para gerarmos o arquivo "vst", vamos utilizar uma ferramenta chamada genlib. O **genlib** é um conjunto de funções dedicadas a fins de produção procedural. Do ponto de vista do

usuário, o genlib é uma linguagem de descrição de circuitos que permite o controle de fluxo do C padrão, o uso de variáveis e funções especializadas que manipulam objetos VLSI. Com o arquivo C pronto, podemos extrair os arquivos “vst” usando os seguintes comandos:

export MBK_OUT_LO=vst

genlib somador_genlib

Após isso, podemos verificar o circuito no xsch para conferir as conexões.



Feito isso, inicia-se o processo de representação física do circuito. O primeiro passo deste processo é o cálculo da posição das standard cells na estrutura física. Este cálculo é executado pela ferramenta **ocp**, (Over Cell Placement), que recebe como entrada o arquivo de descrição lógica (netlist). Para executar o processo executamos o seguinte comando:

ocp somador_genlib somador_posicionado

Após extraído o arquivo, podemos tentar visualiza-lo no graal, porém, devido à falta de algumas informações internas nos arquivos das células criadas anteriormente, o programa não vai conseguir exibi-la.

2 – SOMADOR DE 1 BIT (COM INVERSOR E NAND DA BIBLIOTECA DO SISTEMA)

Como as portas NAND e portas INVERSORAS não funcionaram, podemos repetir o processo anterior, porem desta vez utilizando as portas logicas que o próprio sistema disponibiliza. Para isso, basta ir no arquivo C e mudar o nome das portas instanciadas.

```c
#include <genlib.h>

int main()
{

GENLIB_DEF_LOFIG("somador_genlib_1bit");

GENLIB_LOCON("a", IN, "a");
GENLIB_LOCON("b", IN, "b");
GENLIB_LOCON("s", OUT, "s");
GENLIB_LOCON("c", IN, "c");
GENLIB_LOCON("cout", OUT, "cout");
GENLIB_LOCON("vdd", IN, "vdd");
GENLIB_LOCON("vss", IN, "vss");

//Inversores "entrada, saída, vdd, vss"

GENLIB_LOINS("inv_x1", "invA", "a", "na", "vdd", "vss", NULL);
GENLIB_LOINS("inv_x1", "invB", "b", "nb", "vdd", "vss", NULL);
GENLIB_LOINS("inv_x1", "invC", "c", "nc", "vdd", "vss", NULL);

//Nanda "entrada a, entrada b, saída, vdd, vss"
//saida S

GENLIB_LOINS("na2_x1", "nand1", "na", "b", "snand1", "vdd", "vss", NULL);
GENLIB_LOINS("na2_x1", "nand2", "a", "nb", "snand2", "vdd", "vss", NULL);
GENLIB_LOINS("na2_x1", "nand3",  "snand1", "snand2", "snand3", "vdd", "vss", NULL);
GENLIB_LOINS("na2_x1", "nand4",  "c", "snand3", "snand4","vdd", "vss", NULL);
GENLIB_LOINS("na2_x1", "nand5", "nc", "snand3", "snand5","vdd", "vss", NULL);
GENLIB_LOINS("na2_x1", "nand6", "snand4", "snand5", "s", "vdd", "vss", NULL);

//saida cout

GENLIB_LOINS("na2_x1", "nand7", "nb", "na", "snand7","vss", "vdd",NULL);
GENLIB_LOINS("na2_x1", "nand8", "na", "nc", "snand8", "vss", "vdd",NULL);
GENLIB_LOINS("na2_x1", "nand9", "a", "snand7", "snand9", "vss", "vdd",NULL);
GENLIB_LOINS("na2_x1", "nand10", "snand8", "b","snand10", "vss", "vdd",NULL);
GENLIB_LOINS("na2_x1", "nand11", "snand9", "snand10", "cout", "vss", "vdd",NULL);

GENLIB_SAVE_LOFIG();

return 0;

}
```

Figura 142 – Somador_genlib.c

2.1 – GERANDO OS ARQUIVOS DE SIMULAÇÃO

Após extrair o arquivo “ap” com os comandos descritos anteriormente, podemos finalmente visualizar o circuito no graal. Note que todas as células dos inversores e nand foram dispostas de forma a ocupar o menor espaço possível. Também é possível notar que algumas células estão “espelhadas” ou invertidas para que as alimentações fiquem mais próximas.

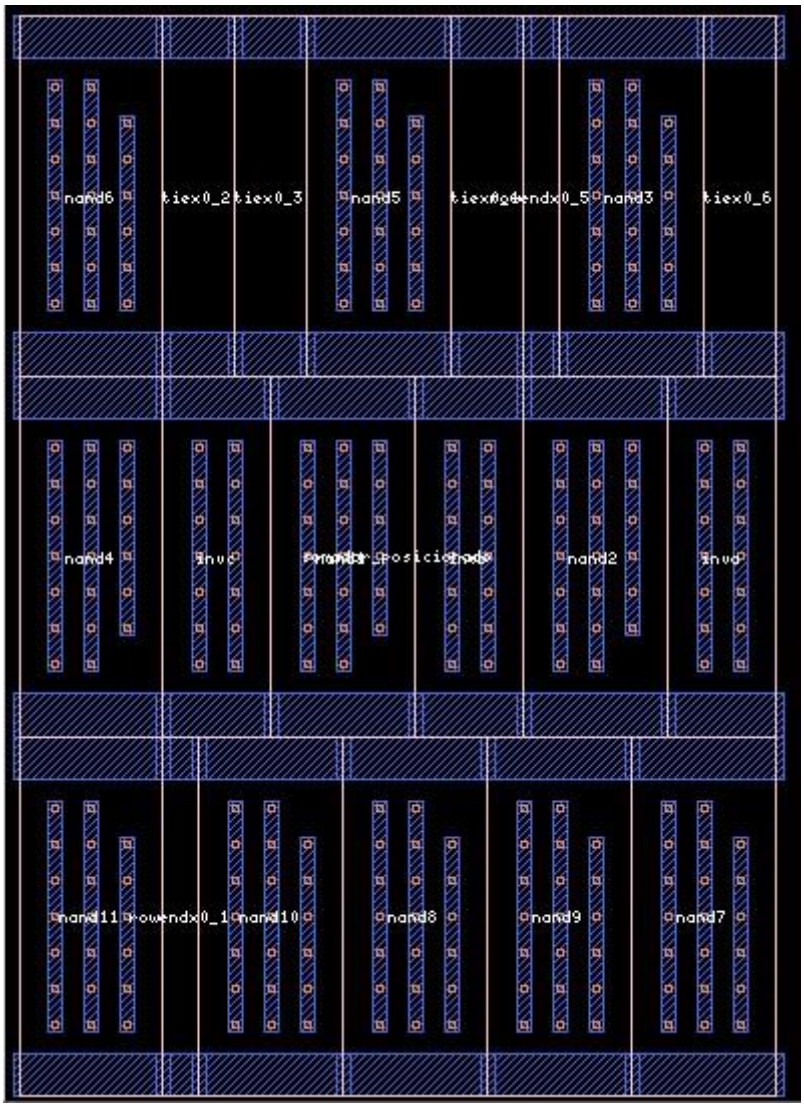


Figura 143 – Somador_genlib_posicionado.ap

Com a célula posicionada pronta, podemos iniciar o processo de roteamento para criar as trilhas e conexões do circuito. Para isso, podemos utilizar uma ferramenta do sistema chamada nero. Para rotear a célula, basta executar o seguinte comando:

nero -p somador_posicionado somador_genlib somador_roteado

O circuito resultante pode ser visto abaixo:

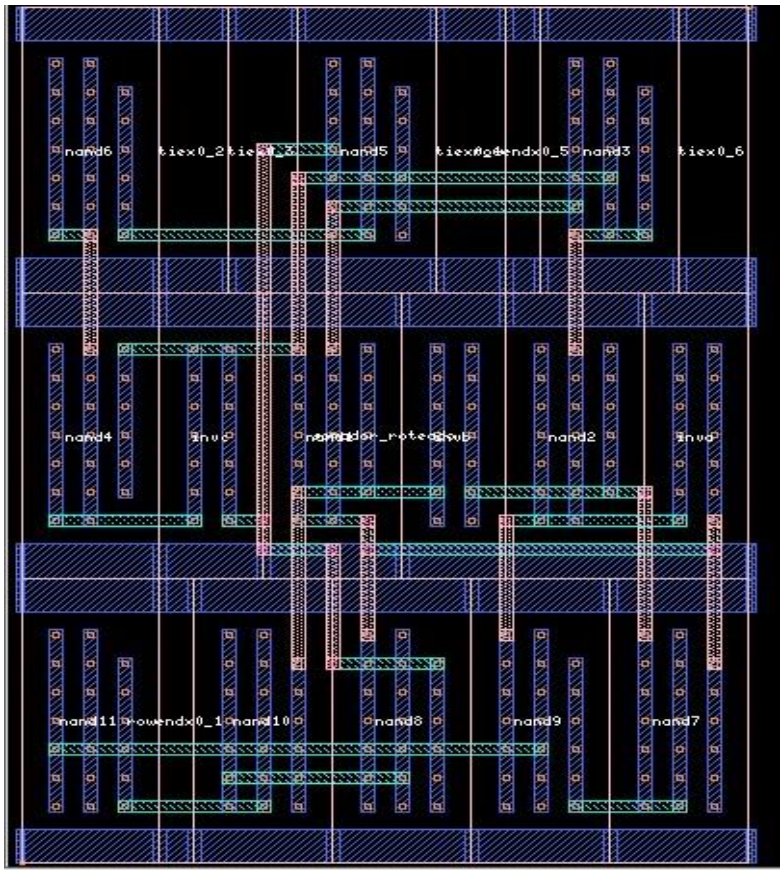


Figura 144 – Somador_genlib_roteado.ap

Após isso, vamos criar novamente as células a partir do código em C, porém desta vez a célula vai ser distribuídas horizontalmente. Para isso, utilizamos o seguinte comando:

aliance-ocp rows 1 somador_genlib somador_posicionado_rows1

O resultado pode ser visto abaixo:

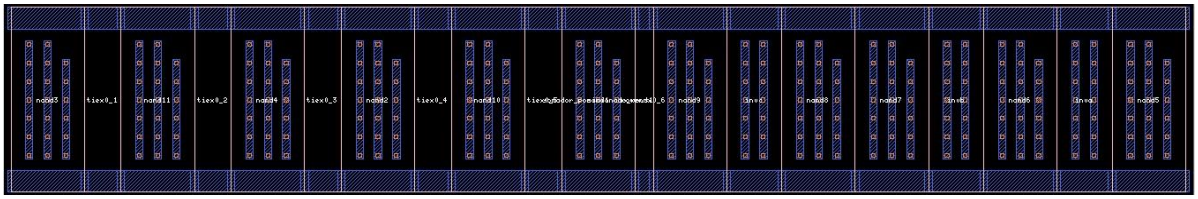


Figura 145 – Somador_posicionado_rows1.ap

Repetindo o processo anterior de roteamento, executamos o seguinte comando:

nero -p somador_posicionado_rows1 somador_genlib somador_roteado_rows1

e obtemos o seguinte resultado:

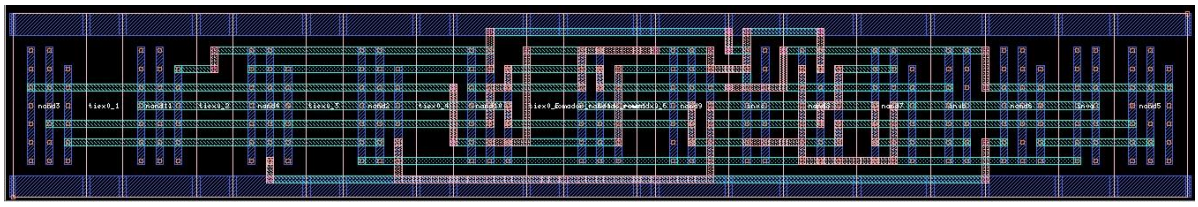


Figura 146 – Somador_roteado_rows1.ap

Com o somador roteado pronto, podemos visualiza-lo no xsch para verificar suas conexões. Para isso, basta exportar o arquivo “vst” e executa-lo no xsch

export MBK_OUT_LO=vst

cougar somador_roteado_rows1

Xsch&

2.2 – SIMULAÇÕES

Utilizando a ferramenta spice opus, podemos simular o somador para verificar os resultados de saída. Abaixo está o arquivo de simulação utilizado.

```
.include somador1bit.spi
.model tp pmos level=54
.model tn nmos level=54

*    a  b  c cout s  vdd vss
x1 (10 11 12  21  20  40  30) somador1bit

v1 10 30 pulse(0 1.8 4ns 1ps 1ps 4ns 8ns)
v2 11 30 pulse(0 1.8 8ns 1ps 1ps 8ns 16ns)
v3 12 30 pulse(0 1.8 16ns 1ps 1ps 16ns 32ns)
v4 40 30 1.8
v5 30 0 0V

.control
tran 0.00001 32ns
plot v(10)+8 v(11)+6 v(12)+4 v(20)+2 v(21)
.endc
.end
```

Figura 147 – Simulação somador de 1bit

Figura 148 – Somador de 1 bit

transição	tipo de atraso	atraso (ns)	causa
a	*low-high*	1,243	*A*
b	*glitch (h→h)*	—	*A,B*
c	*high-low*	2,100	*A*
d	*low-high*	2,194	*A,B,C*
e	*high-low*	2,134	*A*
f	—	—	*A,B*
g	*low-high*	1,334	*A*

3 – SOMADOR DE 4 BITS

O circuito somador de 4 bits é um circuito lógico aritmético que executa a soma binária entre dois números de 4 bits. Este circuito nada mais é do que um arranjo em cascata de quatro somadores de 1 bit, que executam a soma entre cada um dos dígitos dos números de entrada, e transporta o dígito de *carry out* para o bloco somador seguinte.

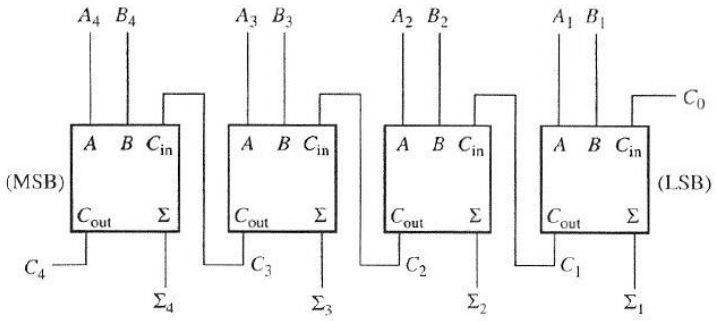


Figura 149 – Diagrama de blocos somador de 4 bit

Para utilizarmos as funções do genlib para criar o somador de 4 bits, podemos criar um código em C que instancie quatro somadores de 1 bit criados anteriormente, um em cima do outro. Para isso, utilizamos o seguinte código:

```c
#include <genlib.h>

int main()
{

        GENLIB_DEF_LOFIG("somador4bit_genlib");
        GENLIB_LOCON("a[3:0]", IN, "a[3:0]");
        GENLIB_LOCON("b[3:0]", IN, "b[3:0]");
        GENLIB_LOCON("s[3:0]", OUT, "s[3:0]");
        GENLIB_LOCON("c0", IN, "c0");
        GENLIB_LOCON("c4", OUT, "c4");
        GENLIB_LOCON("vdd", IN, "vdd");
        GENLIB_LOCON("vss", IN, "vss");

        int i;
        for(i = 0; i < 4; i++){
                GENLIB_LOINS("somador_genlib_1bit", GENLIB_NAME("somador_%d", i), GENLIB_ELM("a", i),
                GENLIB_ELM("b", i), GENLIB_ELM("s", i), GENLIB_NAME("c%d", i) , GENLIB_NAME("c%d", i+1), "vdd", "vss", NULL);
        }

        GENLIB_SAVE_LOFIG();

        return 0;

}
```

Figura 150 – Descrição em alto nível

Após isso extraímos o arquivo “vst” para visualizarmos as conexões do circuito. Fazendo isso, obtemos o seguinte esquemático:

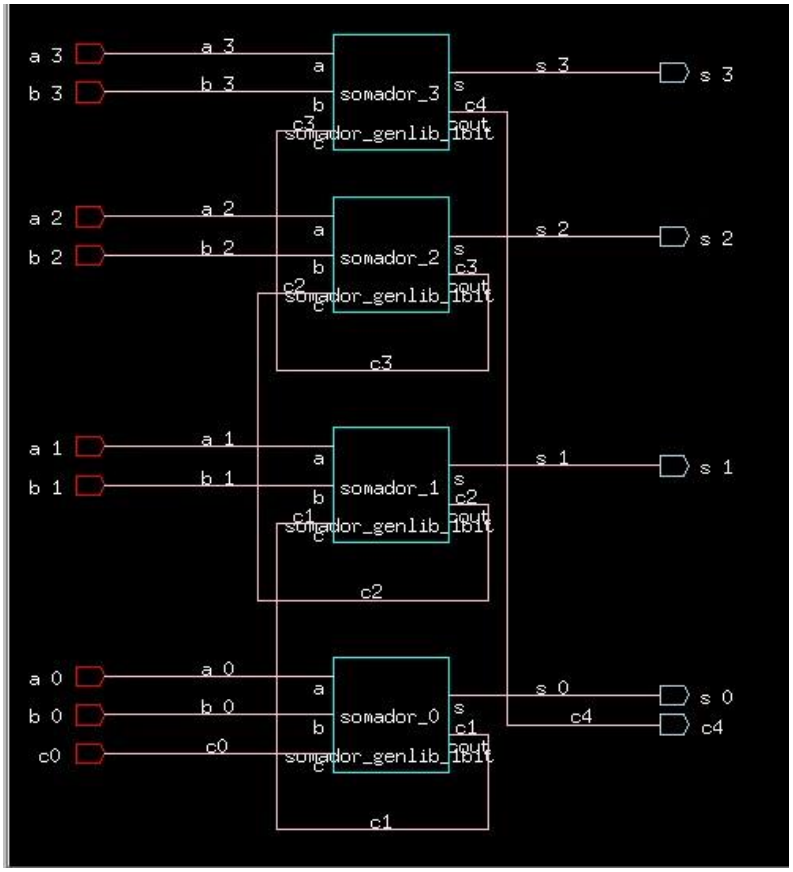


Figura 151 – Somador 4 bit xsch

Extraindo também o arquivo “ap” do circuito, podemos ter a visão física do circuito. O resultado pode ser visto abaixo:

Figura 152 – Somador 4bit.ap

Como podemos perceber, o programa criado anteriormente “empilhou” quarto instancias do somador de 1 bit. Para visualizar as células internamente, podemos dar um flat.

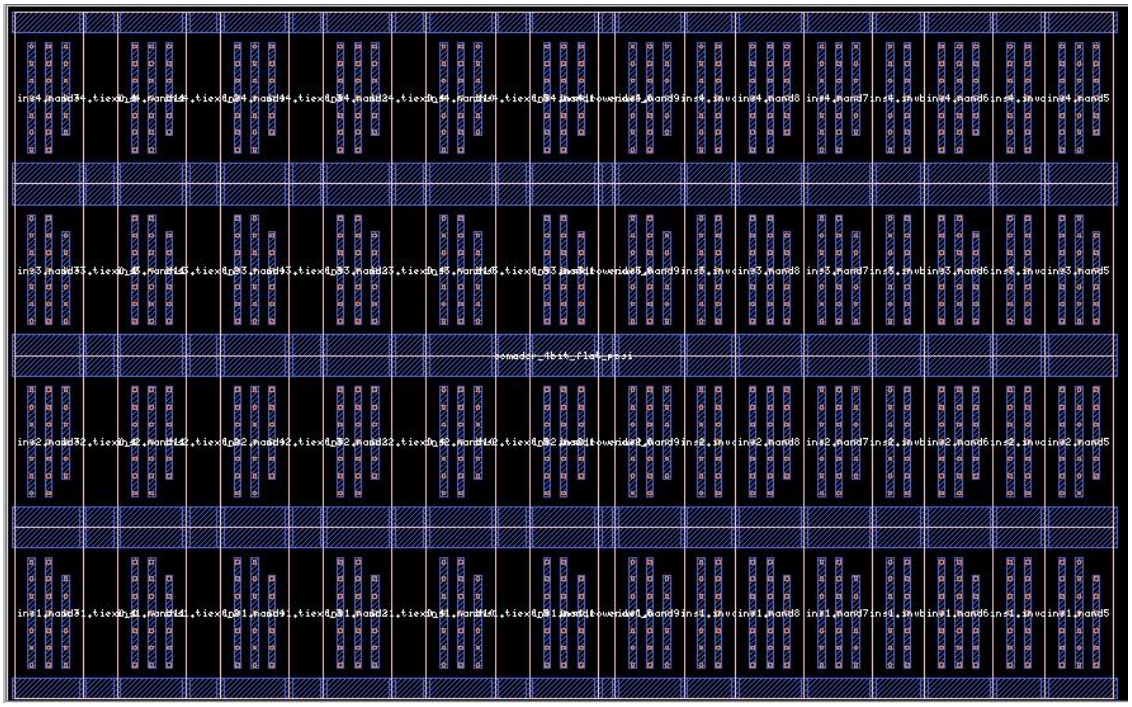


Figura 153 – Somador 4bit posicionado (flat)

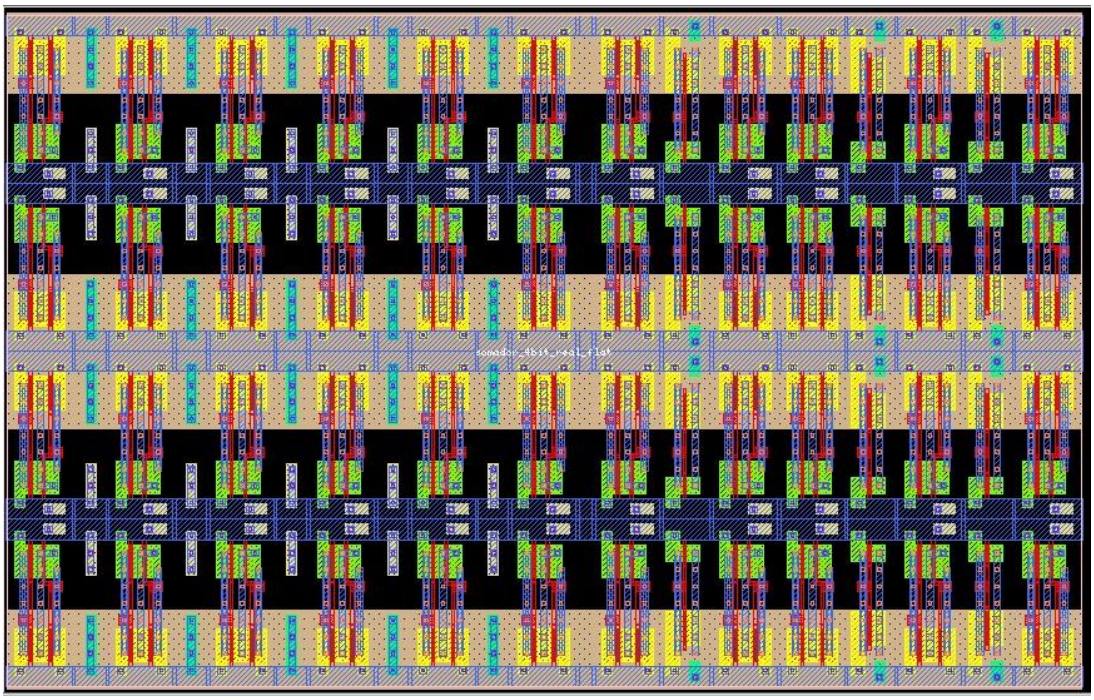


Figura 154 – Somador 4bit (flat)

Após isso, fazemos o processo de roteamento para criar as trilhas de conexão e extraímos os arquivos necessários para a simulação.

```
export MBK_IN_LO=vst
export MBK_IN_PH=ap
export MBK_OUT_PH=ap
nero -p soma4bit_genlib_ocp soma4bit_genlib soma4bit_genlib_ocp_nero
graal -l soma4bit_genlib_ocp_nero
```

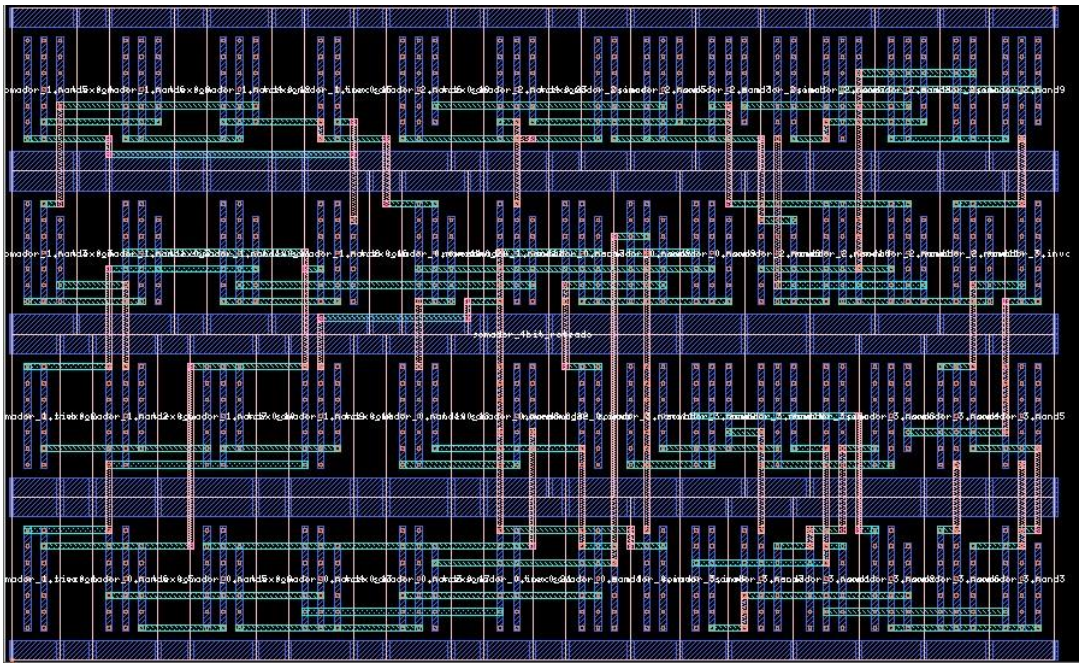


Figura 153 – Somador 4bit roteado

3.1 – SIMULAÇÃO

Para a simulação do somador de 4 bits, criamos um circuito com pulsos de entrada que variam para gerar todas as combinações possíveis para 4 bit. Abaixo está a descrição do código:

```
.include somador_4bit_roteado.spi

* INTERF a[0] a[1] a[2] a[3] b[0] b[1] b[2] b[3] c0 c4 s[0]
s[1] s[2] s[3]
* INTERF vdd vss

X1 10 11 12 13 14 15 16 17 18 24 20 21 22 23 40 30 somador_
4bit_roteado

V1 10 30 pulse(0 1.8 4ns 1ps 1ps 4ns 8ns)
V2 11 30 pulse(0 1.8 8ns 1ps 1ps 8ns 16ns)
V3 12 30 pulse(0 1.8 16ns 1ps 1ps 16ns 32ns)
V4 13 30 pulse(0 1.8 32ns 1ps 1ps 32ns 64ns)
V5 14 30 pulse(0 1.8 64ns 1ps 1ps 64ns 128ns)
V6 15 30 pulse(0 1.8 128ns 1ps 1ps 128ns 256ns)
V7 16 30 pulse(0 1.8 256ns 1ps 1ps 256ns 512ns)
V8 17 30 pulse(0 1.8 512ns 1ps 1ps 512ns 1024ns)
V9 18 30 pulse(0 1.8 1024ns 1ps 1ps 1024ns 2048ns)

V10 40 30 1.8V
V11 30 0 0V

.model tp pmos level=54
.model tn nmos level=54
.tran 0.01 2050ns

.end
```

Figura 154 – Simulação 4bit.cir

Após esperar um bom tempo para terminar a simulação, podemos exibir o resultado da saída.

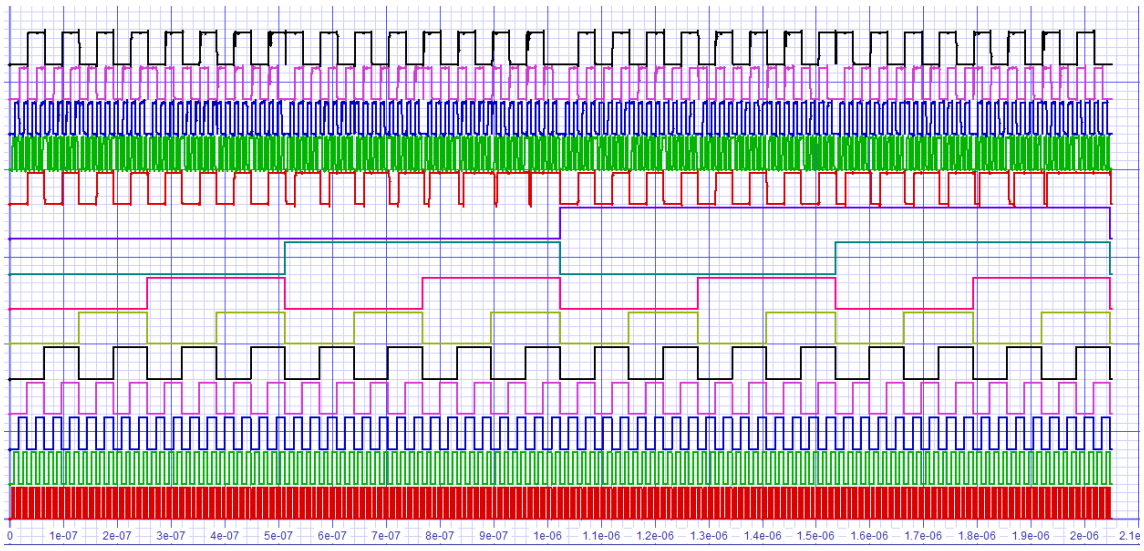


Figura 155 – Somador de 4 bit