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

## EPITÁCIO PESSOA DE BRITO NETO 11506856

# RELATÓRIO DE MICROELETRÔNICA – PARTE 3

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 3

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 3

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.

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, criaremos um chip a partir da linguagem procedural criada em C, converteremos, quando necessário, a descrição e extração dos arquivos necessários para construção, de forma que possamos utilizar as diferentes descrições para uso das ferramentas que nos foram apresentadas durante o andamento da disciplina.

Também faremos várias séries de teste para que essas conversões de descrições, de fato, traduza a descrição para a outra sem que haja perda, incoerência lógica ou estrutural. Desta forma, com auxílio de uma série de arquivos disponibilizados na especificação do trabalho, conseguiremos chegar a um modelo de chip com um núcleo e pads inseridos nele.

Este passo a passo será feito da seguinte maneira: serão explicados os procedimentos de construção, como o uso das ferramentas (o que elas fazem, o que alteram, por que a utilizamos), a descrição do comando destas que foram utilizadas durante o processo via terminal e, em seguida, uma série de imagens que também serão analisadas, tanto no que se refere ao que ela mostra, como uma breve explicação do que pode estar implícito nelas.

#### 2 Construção do ADAC sem atraso

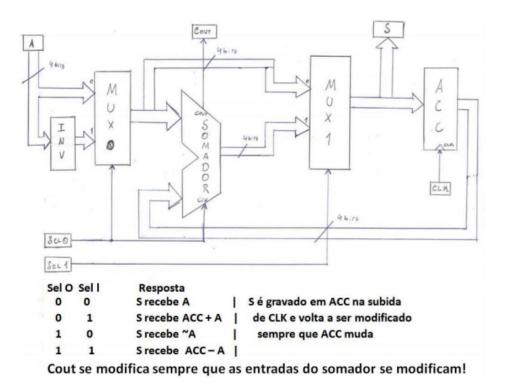


Figura 1: Desenho e respostas do projeto final.

Primeiramente, analisamos a estrutura do código disponibilizado pelo professor, identificando as funcionalidades do mesmo e se corresponde com a descrição, em forma procedural com C, utilizando a biblioteca genpat, ao projeto identificado na figura 1. Após isso, trabalharemos na linha de comando de forma que consigamos gerar o arquivo genpat equivalente à descrição que foi feita no arquivo C anterior.

Em seguida, ainda com arquivos disponibilizados pelo professor, pegamos o arquivo adac.vhd, que seria um tipo de "VHDL de alto nível" e não um RTL. Este não tem tradução imediata para um modelo de silício, portanto, deve ser transformado por meio de uma compilação para que consigamos chegar num modelo RTL e, assim, chegarmos ao silício. No nosso, caso utilizaremos a ferramenta Vasy para transformar este arquivo adac.vhd em um modelo próximo ao de silício.

Alliance-genpat adac\_sem\_atraso

Vasy -a -I vhd adac adac\_vasy

Asimut -b adac\_vasy adac\_sem\_atraso adac\_vasy\_res

Ao fazermos a conversão, utilizando o Vasy, conseguimos perceber diferenças entre uma descrição e outra. Enquanto o adac.vhd conseguia descrever sinais, entradas e saídas de formas mais sucintas e organizadas, o adac\_vasy.vbe é limitado nas entradas e saídas para valores 0, 1 e Z e, também, existe a necessidade de criar sinais intermediários para descrever todo o comportamento do circuito, além do uso de lógica booleana na descrição do arquivo.

Após a análise entre os dois arquivos, utilizamos (e continuaremos com ela durante o relatório) a ferramenta Asimut (a simulation tool). Como queremos analisar o comportamento, utilizamos a instrução "-b" (behavior) na linha do terminal, que pode ser vista acima. Depois, continuando na sintaxe do Asimut, passamos o nome do nosso arquivo.vbe, em seguida, o arquivo.pat feito pelo genpat. Por fim, um arquivo resultante da simulação feita pelo Asimut. A operação de simulação pode ser conferida abaixo:

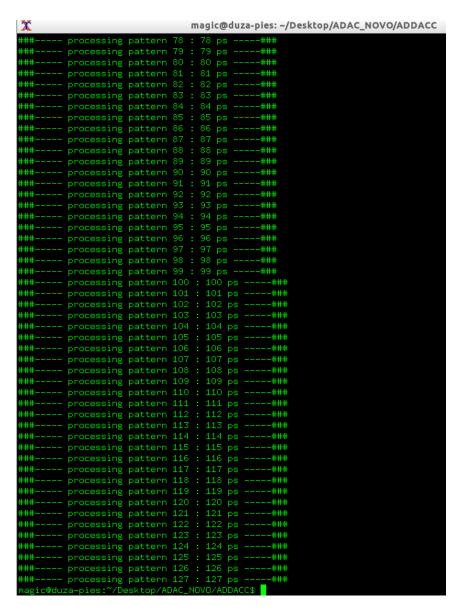


Figura 2: Resultado da simulação do adac\_vasy\_res.

É importante dizer que, se o Asimut (como aconteceu durante a execução do projeto) não retornar algum erro, significa que a especificação em VHDL é equivalente à implementação feita no arquivo C e, qualquer erro que seja encontrado durante a operação do Asimut deve ser corrigido diretamente do arquivo de mais alto nível, no nosso caso o arquivo C.

Para confirmarmos as operações do circuito ao longo do tempo, utilizaremos a ferramenta Xpat, disponibilizada pelo Alliance. O resultado será o que está exposto na figura 3 e, por fim, nosso papel é analisar e confirmar, por meio de formas de onda, se o comportamento do projeto está condizente à sua especificação.

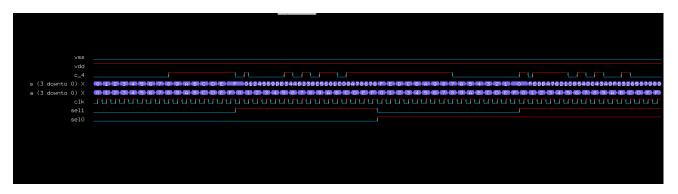


Figura 3: Modelo de visualização das entradas e saídas em formato de tabela.

Após a análise utilizando o Xpat, utilizaremos a funcionalidade Boom, para que possamos otimizar o circuito quanto à lógica booleana aplicada em nível RTL, que fizemos com o Vasy. É descrito, durante o comando no terminal, a porcentagem em relação à área e a velocidade (no nosso caso, escolhemos 50%) e o número de iterações que serão feitas (no nosso caso, serão 100 iterações).

Após isso, para continuarmos verificando a equivalência deste modelo modificado com a descrição do arquivo em C, voltamos a utilizar o Asimut, e esse resultado pode ser conferido na figura 4. Conseguido uma simulação sem erros, prosseguimos.

Boom -l 3 -d 50% -i 100 adac\_vasy adac\_vasy\_boom\_3\_50\_100
Asimut -b adac\_vasy\_boom\_3\_50\_100 adac\_sem\_atraso
adac\_vasy\_boom\_3\_50\_100\_res

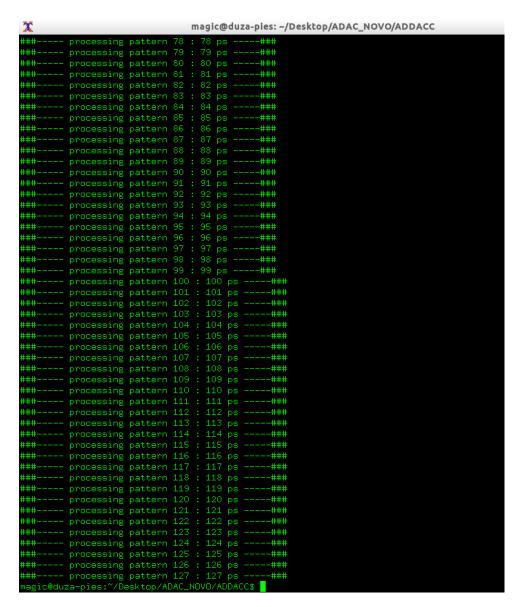


Figura 4: Resultado da simulação do adac\_vasy\_boom\_3\_50\_100\_res.

Em seguida, utilizaremos o Proof, que é utilizado para provar, formalmente, se duas descrições comportamentais VHDL são equivalentes, utilizando as regras matemáticas com relação à equivalência no conceito da lógica booleana. Dentro do Proof, teremos duas representações dos códigos em formato de árvore binária, portanto, se chegamos à mesma árvore (ou árvores equivalentes), teremos duas descrições equivalentes.

```
Environment ==========
1BK_WORK_LIB
                  .:/usr/alliance/cells/sxlib:/usr/alliance/cells/dp_sxlib:
MBK_CATA_LIB
/alliance/cells/rf2lib:/usr/alliance/cells/ramlib:/usr/alliance/cells/romlib:/usr/
lance/cells/padlib
First VHDL file = adac_vasy.vbe
Second VHDL file = adac_vasy_boom_3_50_100.vbe
The common auxiliary signals are kept
irrors are displayed
Compiling 'adac_vasy' ...
Compiling 'adac_vasy_boom_3_50_100' ...
ooking for the common auxiliary signals :
 \rightarrow final number of nodes = 534(406)
        Formal proof with Ordered Binary Decision Diagrams between
         './adac_vasy' and './adac_vasy_boom_3_50_100'
    Formal Proof : OK
```

Figura 5: Resultado da avaliação do adac\_vasy\_boom\_3\_50\_100 utilizando a ferramenta Proof.

Agora, utilizaremos a ferramenta Boog, que mapeia a descrição comportamental na biblioteca sxlib, de forma que consigamos chegar, a partir de um arquivo RTL, em um modelo sintetizável.

Export MBK\_OUT\_LO=vst

Boog adac\_vasy\_boom\_3\_50\_100 adac\_vasy\_boom\_3\_50\_100\_boog\_2

```
Preparing file 'adac_vasy_boom_3_50_100.vbe'...

Capacitances on file 'adac_vasy_boom_3_50_100.vbe'...

Unflattening file 'adac_vasy_boom_3_50_100.vbe'...

Mapping file 'adac_vasy_boom_3_50_100.vbe'...

Saving file 'adac_vasy_boom_3_50_100.vbe'...

Quick estimated critical path (no warranty)...2614 ps from 'acumulador 0' to 's 2'

Quick estimated area (with over-cell routing)...91750 lambda

Details...

xr2_x1: 13

inv_x2: 7

buf_x2: 4

a2_x2: 4

sff1_x4: 4

oa2ao222_x2: 3

na2_x1: 2

oa2a22_x2: 2

mx3_x2: 2

nao2c22_x1: 2

o2_x2: 2

nao2c22_x1: 1

an12_x1: 1

no2_x1: 1

Total: 48

Saving critical path in xsch color file 'adac_vasy_boom_3_50_100_boog_2.xsc'...

magic@duza-pies:~/Desktop/ADAC_NOVO/ADDACC$
```

Figura 6: Utilizando a ferramenta Boog.

Ao finalizarmos o mapeamento, podemos utilizar a ferramenta Xsch para enxergarmos o circuito resultante. Esta visualização pode ser acompanhada abaixo, na figura 7:

Xsch-l'adac vasy boom 3 50 100 boog 2'&

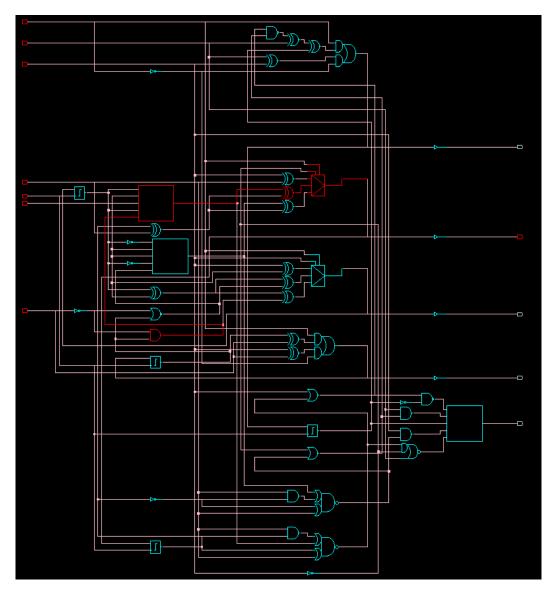


Figura 7: Modelo esquemático do adac\_vasy\_boom\_3\_50\_100\_boog\_2.

Vale salientar que as representações em vermelho acima significam que o caminho percorrido nestas é mais longo entre a entrada e a saída do circuito e, como foi mencionado anteriormente, como utilizamos a biblioteca sxlib para a construção do modelo, as componentes podem ser vistas com a função identify, dentro do Xsch.

Prosseguindo, utilizaremos o Asimut, novamente, para verificar se continuamos dentro da proposta inicial:

Asimut –zd adac\_vasy\_boom\_3\_50\_100\_boog\_2 adac\_sem\_atraso adac\_vasy\_boom\_3\_50\_100\_boog\_2\_res

```
magic@duza-pies: ~/Desktop/ADAC_NOVO/ADDACC
        processing pattern 79
                                         ---###
        processing
       processing pattern 81 : 81 ps --
        processing pattern 82 : 82 ps
        processing pattern 83
                              : 83 ps
        processing pattern 84
                                84 ps
       processing pattern 85
        processing pattern 86 : 86 ps
        processing pattern 87
        processing
       processing pattern 89
                              : 89 ps
                                           -###
     -- processing pattern 90 : 90 ps
                                          --###
       processing pattern 91
                              : 91 ps
       processing pattern 92
                                92 ps
       processing pattern 93
     -- processing pattern 94 : 94 ps
                                          --###
   ---- processing pattern 95
                                95 ps
   ---- processing pattern 96
       processing pattern 97
                                97 ps
                                           -###
    --- processing pattern 98 : 98 ps -
       processing pattern 99 : 99 ps
   ---- processing pattern 100 : 100 ps ----###
       processing pattern 101
                               : 101 ps
   ---- processing pattern 102
   ---- processing pattern 103
                               : 103 ps
##---- processing pattern 104
                                 104 ps
    --- processing pattern 105
                               : 105 ps
                                             -###
   ---- processing pattern 106
                                 107 ps
##---- processing pattern 108
                                 108 ps
   ---- processing pattern 109
                                 109 ps
###---- processing pattern 110 : 110 ps
   ---- processing pattern 111
##---- processing pattern 112
   --- processing pattern 113
                                : 113 ps
                                             -###
   ---- processing pattern 114 : 114 ps
   ---- processing pattern 115
                               : 115 ps
###---- processing pattern 116
                                 116 ps
                                 117 ps
###---- processing pattern 117
##---- processing pattern 118
   ---- processing pattern 119 : 119 ps
##---- processing pattern
                               : 121 ps
   ---- processing pattern 121
   ---- processing pattern 122
                               : 122 ps
    --- processing pattern 123
                               : 123 ps
                                 124 ps
        processing pattern 124
   ---- processing pattern 125
       processing pattern 126
      - processing pattern 127 : 127 ps --
agic@duza-pies:~/Desktop/ADAC_NOVO/ADDACC$
```

Figura 8: Resultado da simulação do adac\_vasy\_boom\_3\_50\_100\_boog\_2\_res com o comando -zd.

Como podemos ver na figura 8, o comando Asimut foi executado com sucesso. Foi necessário utilizar a extensão -zd para que a simulação da netlist não houvesse delay. Para analisarmos a simulação com delay, retiramos esta extensão e observamos a execução do Asimut abaixo:

```
linking ...

executing ...

###---- processing pattern 0 : 0 ps ----###

Error 113: expected value differs from the simulation's result on `s 3`

Error 113: expected value differs from the simulation's result on `s 2`

Error 113: expected value differs from the simulation's result on `s 1`

Error 113: expected value differs from the simulation's result on `s 0`

Error 113: expected value differs from the simulation's result on `c_4`

###---- processing pattern 1 : 1 ps ----###

Error 113: expected value differs from the simulation's result on `s 3`

Error 113: expected value differs from the simulation's result on `s 2`

Error 113: expected value differs from the simulation's result on `s 1`

Error 113: expected value differs from the simulation's result on `s 0`

Error 113: expected value differs from the simulation's result on `c_4`

magic@duza-pies:~/Desktop/ADAC_NOVO/ADDACC$
```

Figura 9: Resultado da simulação do adac\_vasy\_boom\_3\_50\_100\_boog\_2\_res sem o comando -zd.

Ao analisar a figura 9, percebemos que a simulação com delay gerou erros, que faz sentido, já que a descrição inicial em C não indica atrasos na sua sintaxe, portanto, é necessário que o Asimut retornasse erros nesta execução, validando o código fonte original.

Para fazermos a simulação com atraso, esta deve estar explicitada no código C, que será a próxima construção deste relatório.

#### 3 Construção do ADAC com atraso

Continuando a construção de nosso projeto, utilizamos o arquivo adac\_com\_atraso.c, onde, pelo que o nome diz, é a versão com atraso do circuito que fizemos anteriormente. Este atraso foi feito dentro do arquivo C por meio de uma variável auxiliar descrita no código como "atraso" e o seu valor foi definida, para este código, um atraso de 13 mil.

Este valor deve ser verificado como um valor plausível durante a posterior execução do Asimut, após o uso do genpat, como fizemos na construção do ADAC sem atraso, anteriormente. Se retornar algum erro durante a execução do Asimut, pode ser que o valor de atraso escolhido não seja maior (ou menor) suficiente para a descrição feita no arquivo C (como aconteceu durante a aula do Asimut não funcionar para um atraso de 10 mil ou 15 mil, chegando, então, no atraso de 13 mil, onde não houveram erros na execução).

Alliance-genpat adac\_com\_atraso

Asimut adac\_vasy\_boom\_3\_50\_100\_boog\_2 adac\_com\_atraso\_13000
adac\_vasy\_boom\_3\_50\_100\_boog\_2\_13000\_res

```
magic@duza-pies: ~/Desktop/ADAC_NOVO/ADDACC
processing pattern
processing pattern
                       : 2704000 ps
processing pattern 208
processing pattern
processing pattern 211
processing pattern 212
processing pattern
processing pattern
                                         -###
processing pattern 215
                         2808000 ps
processing pattern 217
processing pattern 218
                         2834000 ps
processing pattern 219
processing pattern
processing pattern 221
                         2873000 ps
                                         -###
processing pattern 222
processing pattern
                         2899000 ps
processing pattern
processing pattern
processing pattern 228
processing pattern 229
processing pattern
processing pattern 231
processing pattern 232
processing pattern
                         3042000 ps
processing pattern 234
                                     ----###
processing pattern 235
processing pattern 237
                         3081000 ps ----###
processing pattern 238
processing pattern
                         3133000 ps
processing pattern 241
processing pattern 242
processing pattern
                         3172000 ps ----###
processing pattern 244
processing pattern 245
processing pattern 246
                         3198000 ps
processing pattern 248
processing pattern
processing pattern 251
processing pattern
processing pattern
                         3302000 ps
processing pattern 254
                                        --###
                                         -###
```

Figura 10: Resultado da simulação do adac\_vasy\_boom\_3\_50\_100\_boog\_2\_13000\_res.

Após a verificação da integridade, utilizaremos o Loon. Esta ferramenta serve para otimizar os caminhos críticos mencionados quando comentamos sobre a figura 7, de forma que estes caminhos não estejam ou extrapolem o limite de tempo necessário. Vale dizer que, pelo nosso circuito ser relativamente pequeno, o uso do Loon pode piorar o desempenho do mesmo.

```
Assaing file 'adac_vasy_boom_3_50_100_boog_2_vst'...
Reading file 'dasc_vasy_boom_3_50_100_boog_2_vst'...
Reading lib '/usr/alliance/cells/sklib'...

Clapacitances on file 'adac_vasy_boom_3_50_100_boog_2_vst'...
Delays on file 'adac_vasy_boom_3_50_100_boog_2_vst'...4009 ps
frace of file 'adac_vasy_boom_3_50_100_boog_2_vst'...100 ps
frace of file 'adac_vasy_boom_3_50_100_boog_2_vst'...100 ps
frace of file 'adac_vasy_boom_3_50_100_boog_2_vst'...100 ps
frace of file 'adac_vasy_boom_3_50_100_boog_2_vst'...133 ps
frace of file 'adac_vasy_boom_3_50_100_boog_2_vst'...133 ps
frace of file 'adac_vasy_boom_3_50_100_boog_2_vst'...133 ps
Inserting buffers on critical path for file 'adac_vasy_boom_3_50_100_boog_2_loon_4.vst'...4049 ps
Improving RC on critical path for file 'adac_vasy_boom_3_50_100_boog_2_loon_4.vst'...
### Improving IN C on file 'adac_vasy_boom_3_50_100_boog_2_loon_4.vst'...
### IN Coality IN Coalit
```

Figura 11: Resultado da otimização de componentes utilizando o Loon.

Pela figura 11, conseguimos perceber o aumento na quantidade de componentes do circuito com relação ao que era anteriormente, para tentar resolver esses problemas de caminho destacados. Essa alteração no circuito pode ser vista, a seguir, na figura 12:

Xsch-l'adac vasy boom 3 50 100 boog 2 loon 4'&

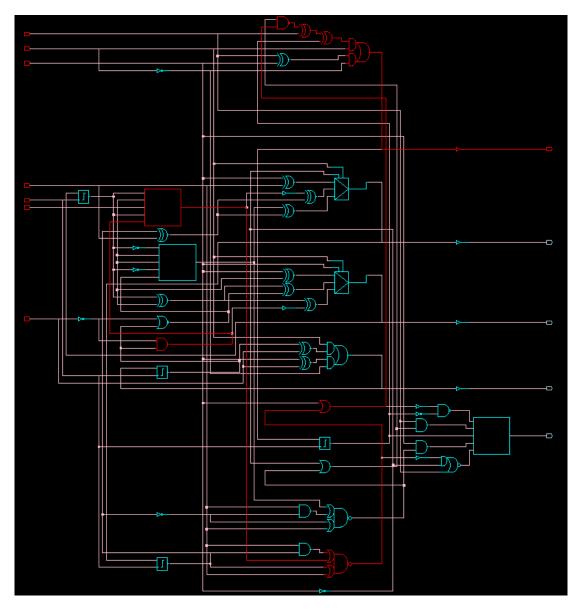


Figura 12: Modelo esquemático do adac\_vasy\_boom\_3\_50\_100\_boog\_2\_loon\_4.

É possível ver o posicionamento dos buffers colocados nas saídas, além do aumento dos componentes. E, para mantermos a prática, utilizaremos, novamente, a ferramenta Asimut para que possamos confirmar se a mudança na estrutura feita pelo Loon afetou a integridade do projeto:

Asimut adac\_vasy\_boom\_3\_50\_100\_boog\_2\_loon\_4 adac\_com\_atraso\_13000 adac\_vasy\_boom\_3\_50\_100\_boog\_2\_loon\_4\_13000\_res

X			n a	nic@duz	a-ni	es: ~/Desktop/ADAC_NOVO/ADDACC
				-	•	,
### processing						
### processing						
### processing				2704000		
### processing				2717000		###
### processing						###
### processing						###
### processing	pattern 2					###
### processing	pattern 2	213		2769000	ps	###
### processing	pattern 2	214		2782000	ps	###
### processing	pattern 2	215		2795000	ps	###
### processing	pattern 2	216		2808000	ps	###
### processing	pattern 2	217		2821000	ps	###
### processing	pattern 2	218		2834000	ps	###
### processing	pattern 2	219		2847000	ps	###
### processing	pattern 2	220		2860000	ps	###
### processing	pattern 2	221		2873000	ps	###
### processing	pattern 2	222		2886000	ps	###
### processing	pattern 2	223		2899000	ps	###
### processing	pattern 2	224		2912000	ps	###
### processing	pattern 2	225		2925000	ps	###
### processing	pattern 2	226		2938000	ps	###
### processing	pattern 2	227		2951000	ps	###
### processing	pattern 2	228		2964000		###
### processing				2977000	ps	###
### processing	pattern 2	230		2990000	ps	###
### processing	pattern 2	231		3003000	ps	###
### processing	pattern 2	232		3016000	ps	###
### processing	pattern 2	233		3029000	ps	###
### processing	pattern 2	234		3042000		
### processing	pattern 2	235		3055000	ps	###
### processing				3068000	ps	###
### processing	pattern 2	237				###
### processing	pattern 2	238		3094000	ps	###
### processing	pattern 2	239				###
### processing	pattern 2	240		3120000		###
### processing	pattern 2	241		3133000	ps	###
### processing				3146000		
### processing				3159000		
### processing				3172000		
### processing				3185000		###
### processing				3198000		###
### processing				3211000		###
### processing				3224000		###
### processing				3237000		###
### processing				3250000		###
### processing				3263000		
### processing						###
### processing						###
### processing						
### processing						###
magic@duza-pies:~/D	esktop/ADA	₹C_N	U۱	/U/ADDAC(	-\$	

Figura 13: Resultado da simulação do adac\_vasy\_boom\_3\_50\_100\_boog\_2\_loon\_4\_13000\_res.

Para que possamos ajudar a otimizar o roteamento e o posicionamento dos pads, no produto final, utilizaremos o arquivo adac.ioc, formato industrial que descreve os pinos de entrada e saída. Antes de fazer o ocp, ferramenta utilizada para posicionamento de standard cells utilizado em relatórios anteriores, fazemos o comando cp, para transferir o arquivo resultante de várias aplicações para um arquivo com nome menos extenso.

Diferentemente das utilizações anteriores, o comando ocp, neste caso, precisou de alguns parâmetros a mais para a construção do adac core. Nestes parâmetros, temos o número de rows (linhas) que foi definida com 6 (número testado por um monitor antigo da disciplina,

já que números diferentes geravam erros e, por algum motivo, 6 linhas fazia com que o comando funcionasse).

Além disso, como foi mencionado a pouco, passamos o arquivo adac.ioc como parâmetro do ocp para que o mesmo identifique as posições descritas nele, com intuito de melhorar a construção dos pads posteriormente.

Cp adac\_vasy\_boom\_3\_50\_100\_boog\_2\_loon\_4.vst adac\_core.vst

Alliance-ocp -rows 6 -ring -ioc adac adac\_core adac\_core\_posicionado

Graal -l 'adac core posicionado'&

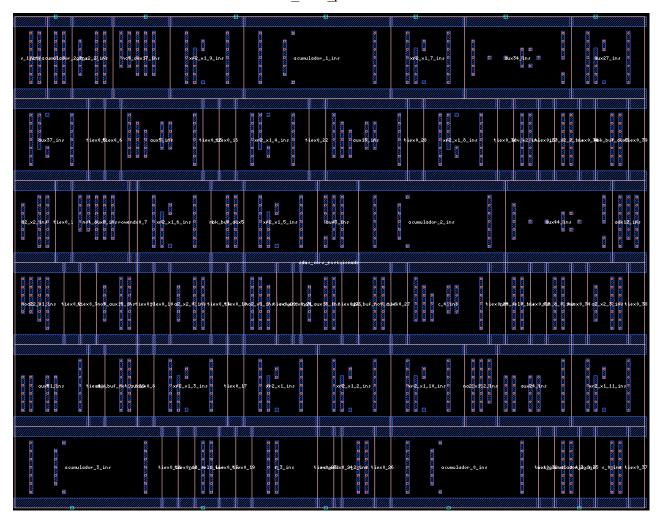


Figura 14: Modelo em standard cells do adac\_core\_posicionado.

Novamente com uma ferramenta já utilizada na disciplina, faremos o uso do Nero para criação do roteamento do circuito. Vejam que, na figura 15, o Nero se permitiu o uso de alumínio 2 na vertical e horizontal nos extremos do circuito, por saber que está passando por apenas alumínio 1 em vdd e vss, apesar de diferir das regras de desenho, a aplicação faz sentido, nesta situação específica.

# Nero –p adac\_core\_posicionado adac\_core adac\_core\_roteado Graal –l 'adac\_core\_roteado'&

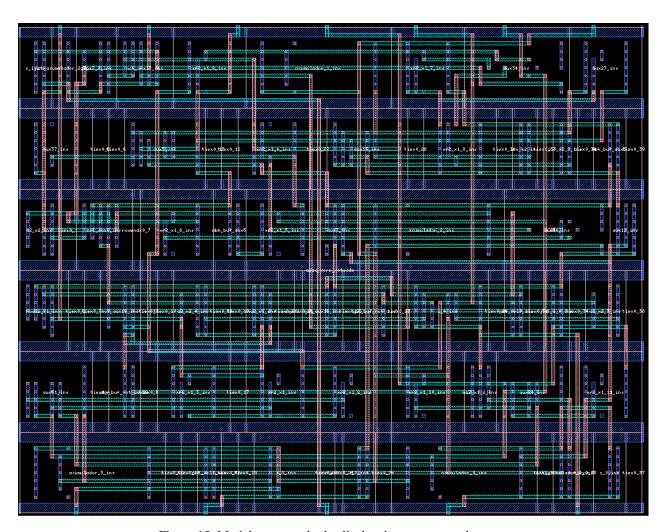


Figura 15: Modelo em standard cells do adac\_core\_roteado.

Em seguida, utilizamos o Cougar, também utilizado no passado. Neste caso, não utilizamos as extensões -t, -ar ou -ac, já que o uso destes causaria uma extração de muita capacitância ou estouraria as resistências, que não é de nosso interesse.

Após a extração, utilizaremos o LVX (layout versus extracted), para comparar as netlists e ver se o modelo de silício bateu com a netlist de entrada. Este procedimento pode ser visto na figura 16, logo abaixo:

Export MBK\_OUT\_LO=al

Cougar adac\_core\_roteado adac\_core\_roteado\_extra

Figura 16: Resultado da comparação entre as netlists utilizando o LVX.

Novamente, utilizaremos o Asimut para simular o modelo roteado:

Export MBK\_IN\_LO=al

Asimut adac\_core\_roteado\_extra adac\_com\_atraso\_13000 adac\_core\_roteado\_extra\_13000\_res

```
magic@duza-pies: ~/Desktop/ADAC_NOVO/ADDACC
        processing pattern
        processing pattern
       processing pattern 208
                               : 2704000 ps
       processing pattern 209
                               : 2717000 ps ----###
       processing pattern 210
                                2730000 ps
                                2743000 ps --
       processing pattern 211
       processing pattern 213
                                2782000 ps
       processing pattern 214
                               : 2795000 ps ----###
       processing pattern 215
                               : 2808000 ps
       processing pattern 216
                                2821000 ps
       processing pattern 217
       processing pattern 218
                                2834000 ps ---
                                2847000 ps ----###
       processing pattern
                                2873000 ps
       processing pattern 221
                                2886000 ps ----###
       processing pattern 222
       processing pattern
       processing pattern 224
                                2912000 ps --
                                2925000 ps ----###
       processing pattern 225
       processing pattern 226
                                2938000 ps
       processing pattern 227
##---- processing pattern 228
                                2964000 ps ----###
                                2977000 ps ----###
       processing pattern 230
                                3003000 ps --
       processing pattern 231
                                3016000 ps ----###
       processing pattern
                                3042000 ps -
       processing pattern 234
                                3055000 ps ----###
       processing pattern 235
       processing pattern 236
       processing pattern 237
                                3081000 ps ----###
                                3094000 ps ----###
       processing pattern 238
       processing pattern 239
                                3107000 ps
       processing pattern 240
       processing pattern 241
                                3133000 ps ----###
                                3146000 ps ----###
   ---- processing pattern 242
                                3159000 ps ----###
       processing pattern 243
       processing pattern 244
                                3172000 ps ---
                                3185000 ps ----###
       processing pattern
                                3211000 ps
       processing pattern 247
       processing pattern 248
                               : 3224000 ps ---
                                3237000 ps
       processing pattern 249
                                 3250000 ps ----###
       processing pattern 250
       processing pattern 251
                                3263000 ps ----###
       processing pattern
       processing pattern 254
```

Figura 17: Simulação do adac\_core\_roteado\_extra\_13000\_res.

Em seguida, trabalharemos com outro arquivo disponibilizado pelo professor, o adac\_chip.c, assim, geraremos a netlist do chip (core + pads) a partir dele, por meio do genlib.

Export MBK\_IN\_LO=vst

Export MBK\_OUT\_LO=vst

Genlib adac\_chip

Xsch -l 'adac\_chip'&

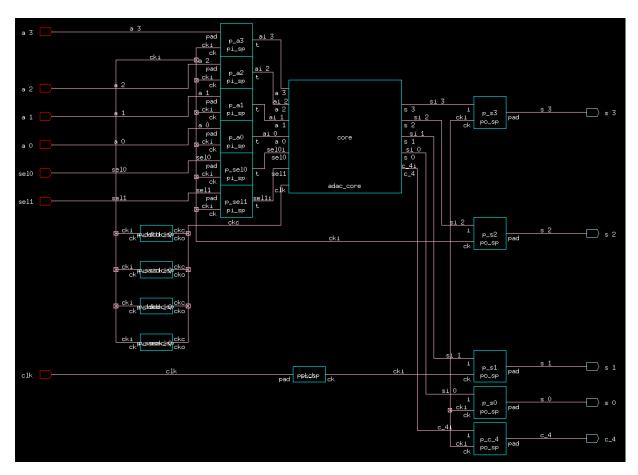


Figura 18: Formato esquemático do chip.

Na figura 18, podemos ver a adição dos pads descritos no adac\_chip e, se utilizarmos o "Go down" dentro do Xsch, no core, poderemos enxergar o mesmo circuito da figura 14. Após analisar a representação, voltamos a abrir o adac core roteado, que pode ser visto, novamente, na figura 19, e o salvaremos como adac\_core, para podermos passar para a ferramenta Ring, que será explicado posteriormente.

Da mesma forma, se não salvarmos como, dentro do Xsch, teríamos que alterar a primeira linha, editando o arquivo. De qualquer forma, o critério é de quem está construindo, então, existem essas duas formas para prepararmos o uso do Ring.

Graal -l 'adac core roteado'&

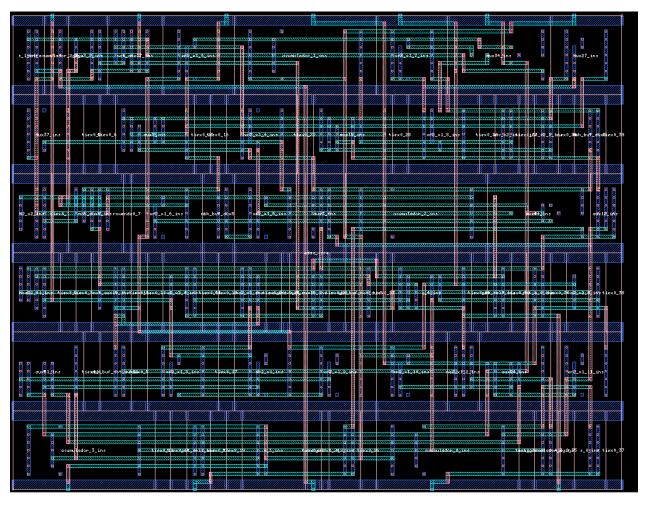


Figura 19: Modelo em standard cells do adac\_core\_roteado, agora chamado adac\_core.

Quanto à ferramenta Ring, ela, basicamente, irá criar o layout final do chip, após o core estar roteado e os pads inseridos. Se o trabalho foi feito de forma correta, o Ring retornará a mensagem "lucky, no error" juntamente do que o mesmo faz durante a execução. Esta situação pode ser verificada na figura 20, logo abaixo:

Ring adac\_chip adac\_chip

```
agic@duza-pies:~/Desktop/ADAC_NOVO/ADDACC$ ring adac_chip adac_chip
                  0000000
                                                     00000 0
                         00
                                000
                                                         00
                    00
                                                   00
                                                 @@
                                     000 000
                    00
                          00
                                                 00
                              0000
                                      000
                                                 @@
                    00000
                                            @@
                                                @@
                                                        00000
                                                        0 00
                    @@
                         00
                                @@
                                      @@
                                            @@
                                                 @@
                                                         @@
                    @@
                                @@
                                      @@
                                            @@
                                                 @@
                                                          @@
                    00
                          @@
                                            @@
                                      00
                                                  00
                                                         00
                                                     0000
                  00000
                          000 000000 0000
                                            0000
                                PAD ring router
                  Alliance CAD System 5.0,
                                                      ring 5.0
                  Copyright (c) 1991-2020,
                  E-mail
                                : alliance-users@asim.lip6.fr
      o reading netlists, layout views of core and pads.
      o reading file of parameters, including the placements of pads.
      o making equipotential list.
      o filling data internal structures.
      o reading the connectors positions of the pads.
      o routing deportation of connectors.
      o routing equipotentials.
      o compressing channels.
      o saving in MBK data structure.
agic@duza-pies:~/Desktop/ADAC_NOVO/ADDACC$
```

Figura 20: Criação do layout final com core roteado e pads.

Abaixo, podemos ver o modelo, em standard cells, do formato final do nosso chip, com seu core no meio e os pads nas extremidades:

Graal -l 'adac chip'&

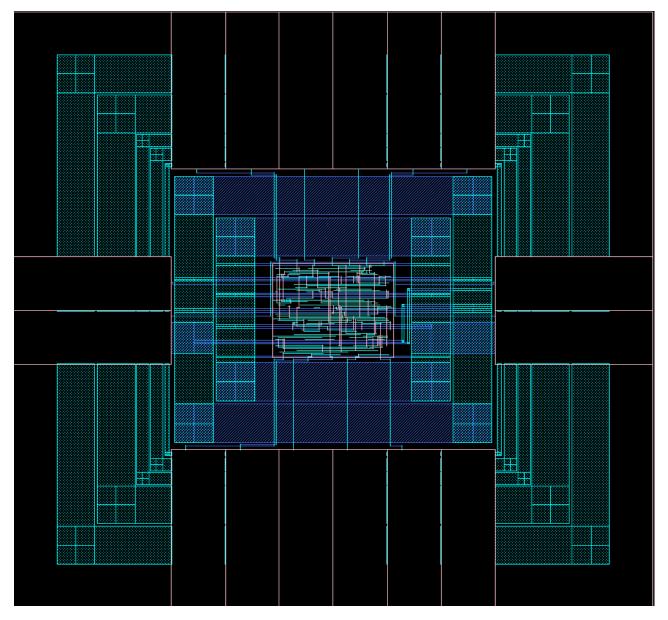


Figura 21: Modelo em standard cells do adac\_chip.

Para finalizar a construção do nosso chip, faremos as últimas simulações e comparações com uso das ferramentas, para garantir que está tudo dentro dos conformes. Primeiramente, extraímos o chip utilizando o Cougar e, em seguida, utilizamos, novamente, o LVX, para compararmos as netlists:

Export MBK\_OUT\_LO=al

Cougar adac\_chip adac\_chip\_ex

Lvx al vst adac\_chip\_ex adac\_chip

```
**** Loading adac_chip_ex (al)...
***** Loading adac_chip (vst)...
***** Compare Terminals ......
**** O.K.
               (0 sec)
«жжж Compare Instances ......
**** O.K.
              (0 sec)
○**** Compare Connections .......
×××× □.K.
               (0 sec)
==== Terminals ...... 16
:==== Instances ...... 17
:==== Connectors ...... 137
***** Netlists are Identical. ****
                                      (0 sec)
agic@duza-pies:~/Desktop/ADAC_NOVO/ADDACC$
```

Figura 22: Resultado da compração entre as netlists utilizando o LVX.

Agora, utilizaremos a ferramenta x2y para, basicamente, criar uma cópia chamada adac\_core.al a partir do adac\_core.vst. O uso desta ferramenta foi necessário para manter todas as netlists, na hora da simulação, em uma extensão compatível. Como começamos no uso de arquivos.al, usamos o x2y para fazer a simulação hierárquica, novamente, utilizando o Asimut, cuja execução pode ser vista abaixo:

Export MBK\_IN\_LO=al

X2y vst al adac\_core adac\_core

Asimut adac\_chip\_ex adac\_com\_atraso\_13000 adac\_chip\_ex\_res\_13000

X		ma	aic@duz	a-pi	es: ~/Desktop/ADAC_NOVO/ADDACC
	nottonn				
### processing ### processing					
### processing			2704000		###
### processing				ps	###
### processing				ps	###
### processing				ps	###
### processing				ps	###
### processing				ps	###
### processing			2782000	ps	###
### processing			2795000	ps	###
### processing			2808000	ps	###
### processing			2821000	ps	###
### processing			2834000	ps	###
### processing			2847000	ps	###
### processing			2860000	ps	###
### processing			2873000	ps	###
### processing			2886000	ps	###
### processing			2899000	ps	###
### processing			2912000	ps	###
### processing			2925000	ps	###
### processing			2938000	ps	###
### processing			2951000	ps	###
### processing			2964000	ps	###
### processing			2977000	ps	###
### processing			2990000	ps	###
### processing			3003000	ps	###
### processing			3016000	ps	###
### processing			3029000	ps	###
### processing			3042000	ps	###
### processing			3055000	ps	###
### processing			3068000	ps	###
### processing			3081000	ps	###
### processing			3094000	ps	###
### processing			3107000	ps	###
### processing			3120000	ps	###
### processing			3133000	ps	###
### processing			3146000	ps	###
### processing			3159000	ps	###
### processing			3172000	ps	###
### processing			3185000	ps	###
### processing			3198000	ps	###
### processing			3211000	ps	###
### processing			3224000	ps	###
### processing			3237000	ps	###
### processing			3250000	ps	###
### processing			3263000	ps	###
### processing				ps	###
### processing				ps	###
### processing					###
### processing					###
magic@duza-pies:~/De					

Figura 23: Simulação final do adac\_chip\_ex\_res\_13000.

Finalmente, com a última verificação do Asimut feita sem erros, criamos, portanto, o nosso chip adac com sucesso, respeitando o passo a passo descrito pelo professor.

### 4 Conclusão

Este projeto, além de complementar os assuntos e ferramentas introduzidos durante do andamento da disciplina, serviu para, também, nos introduzir ao conceito de verificação

funcional hierárquica, assunto que será aprofundado de uma forma maior durante a disciplina de Concepção Estruturada de Circuitos Integrados.

Desta forma, a construção deste chip servirá muito bem como porta de entrada para o que veremos futuramente, agora com uma bagagem significativa quanto aos conceitos da microeletrônica que foram utilizados na construção. Conseguimos testemunhar e aplicar, com êxito, as ferramentas disponibilizadas a nós, de forma que estamos bem introduzidos à construção de chips que poderão ser aplicados em ambientes de serviços.