UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA - CI

GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

INTRODUÇÃO À MICROELETRÔNICA

WENDSON CARLOS SOUZA DA SILVA

PROFESSOR: DR. ANTÔNIO CARLOS CAVALCANTI

João Pessoa – PB

Março - 2020

UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA - CI

GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

INTRODUÇÃO À MICROELETRÔNICA

WENDSON CARLOS SOUZA DA SILVA

Relatório apresentado na disciplina Introdução a microeletrônica como requisito para obtenção de nota.
Área: Engenharia de Computação
Professor: Dr. Antônio Carlos Cavalcanti

João Pessoa – PB

Março - 2020

LISTA DE FIGURAS

SUMÁRIO

Capítulo 1

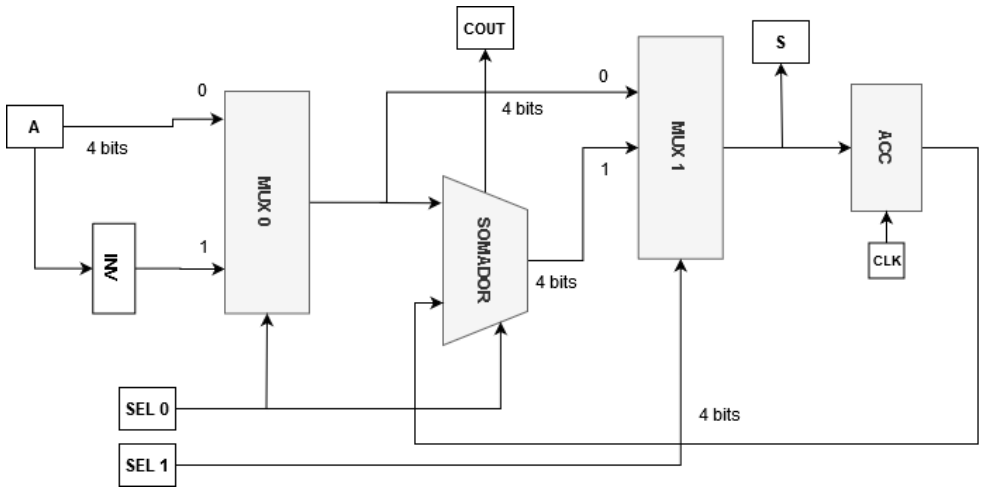
CONCEPÇÃO TOP DOWN DE UM SOMADOR ACUMULADOR DE 4 BITS : DO MODELO EM C AO LAYOUT DO CHIP EM SILÍCIO

1.1 Objetivo

Nesse relatório é apresentado a criação de um somador acumulador de 4 bits. Propõe a análise que vai do modelo em linguagem C até o layout do chip. Para isto é realizado a geração de um arquivo de um modelo de referência para o bloco lógico, a criação de um modelo em VHDL de alto nível, a conversão da descrição funcional em comportamental em nível RTL além de outras coisas. Durante todo o processo são realizadas simulações que validarão o modelo que está sendo implementado. Será utilizado diversas ferramentas do pacote *Alliance* para realizar posicionamento e roteamento automáticos de standard cells, geração das netlist, validação das especificações, geração procedural de netlist etc. Todos esses processos servirão para obtenção do layout final do chip completo.

1.2 Somador Acumulador de 4 bits

Na figura (1) exibe o modelo do somador acumulador de 4 bits composto por multiplex idores, acumulador e somador. Na figura o "S"é gravado no acumulador quando acontece a subida do clock, modificando-se com variações no acumulador. Já o "COUT"é alterado ao passo que as entradas do somador são modificadas.



(a)

Figura 1.1: Modelo do somador acumulador de 4 bits

1.3 Desenvolvimento do layout

O projeto é iniciado com a criação de um arquivo para geração procedural da descrição comportamental de um **somador acumulador de 4 bits** na linguagem de programação *C* para um circuito, inicialmente, sem atraso. Com isso, é realizado uma concepção *topdown* utilizando o comando **alliance-genpat adac_sem_atraso**, gerando assim um arquivo .pat. Em seguida, utiliza-se a ferramenta **vasy**, desenvolvida para migração de circuitos no alliance que transformará o alto nível em álgebra boleana RTL, servirá para realizar a tradução do VHDL rudimentar. Com o comando **vasy -a -I vhd adac adac_vasy** é gerado um arquivo VHDL comportamental em nível RTL (.vbe) visível na figura(2).

adac_vasy.vbe ×

```
2 -- Generated by VASY
3 --
4 ENTITY adac_vasy IS
5 PORT(
6   a     : IN BIT_VECTOR(3 DOWNTO 0);
7   sel0  : IN BIT;
8   sel1  : IN BIT;
9   clk   : IN BIT;
10  s     : OUT BIT_VECTOR(3 DOWNTO 0);
11  c_4   : OUT BIT;
12  vdd   : IN BIT;
13  vss   : IN BIT
14 );
15 END adac_vasy;
16
17 ARCHITECTURE VBE OF adac_vasy IS
18
19  SIGNAL rtlsum_2       : BIT_VECTOR(4 DOWNTO 0);
20  SIGNAL rtlcarry_2     : BIT_VECTOR(4 DOWNTO 0);
21  SIGNAL rtlatom_1      : BIT_VECTOR(4 DOWNTO 0);
22  SIGNAL rtlsum_0       : BIT_VECTOR(4 DOWNTO 0);
23  SIGNAL rtlcarry_0     : BIT_VECTOR(4 DOWNTO 0);
24  SIGNAL rtlexts_2      : BIT_VECTOR(4 DOWNTO 0);
25  SIGNAL rtlexts_1      : BIT_VECTOR(4 DOWNTO 0);
```

Figura 1.2: Arquivo adac_vasy.vbe

É sabido a importância de efetuar testes de simulação ao decorrer do processo de de-

senvolvimento de um chip. Dessa forma, utilizou-se o simulador lógico asimut para identificação de possíveis erros com a linha de comando **asimut -b adac_vasy adac_sem_atraso adac_vasy_res**. A figura (5-a) exibe que não ocorreram erros. A figura (3) exibe as entradas e saída em pulsos digitais com linha de execução **xpat -l adac_vasy_res** e a figura (4) expõe isto em forma de tabela.

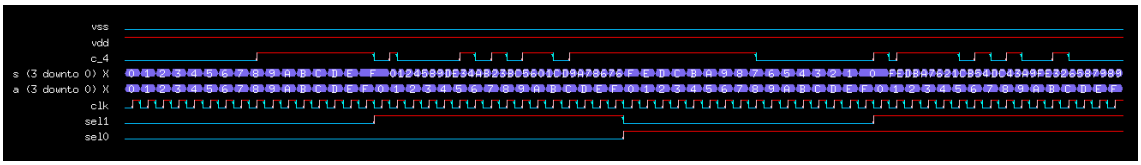


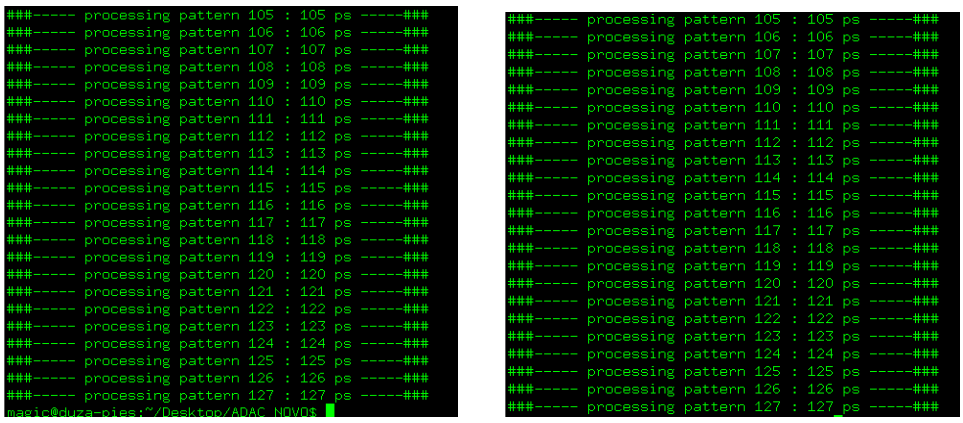
Figura 1.3: Visualização dos pulsos digitais

adac_vasy_res.pat

```
21 -- Pattern description :
22
23 --                                          s  s  c  a      s      c  v
   v
24 --                                          e  e  l                _  d
   s
25 --                                          l  l  k                4  d
   s
26 --                                          0
   1
27
28 <          0 ps>copia_0                   : 0  0  0  0000  ?0000  ?0  1
   0  ;
29 <          1 ps>                          : 0  0  1  0000  ?0000  ?0  1
   0  ;
30 <          2 ps>                          : 0  0  0  0001  ?0001  ?0  1
   0  ;
31 <          3 ps>                          : 0  0  1  0001  ?0001  ?0  1
   0  ;
32 <          4 ps>                          : 0  0  0  0010  ?0010  ?0  1
   0  ;
33 <          5 ps>                          : 0  0  1  0010  ?0010  ?0  1
   0  ;
```

Figura 1.4: Arquivo adac_vasy_res.pat

O processo de criação do chip prossegue com a otimização da descrição comportamental aplicando a ferramenta *boom*, um otimizador boleado que facilitará este procedimento. É inserido um arquivo no formato *.vbe* de entrada na linha **boom -l 3 -d 50% -i 100 adac_vasy adac_vasy_boom_3_50_100**. Nota-se a diminuição discrepante nas equações boleana em relação ao modelo inicial. Como citado anteriormente, será aplicado o simulador lógico asumit ao longo de todo o processo, agora para o novo arquivo gerado com o comando **asimut -b adac_vasy_boom_3_50_100 adac_sem_atraso adac_vasy_boom_3_50_100_res**. Nota-se que a simulação ainda não apresenta nenhum erro como pode ser visto na figura (5-b). Após isso, faz-se a prova formal entre duas descrições comportamentais em VHDL do Alliance como pode ser visto na figura (6), com um resultado positivo, que implica que o comando *boom* não alterou logicamente a descrição RTL.



(a) (b)

Figura 1.5: Simulação com a ferramenta asimut

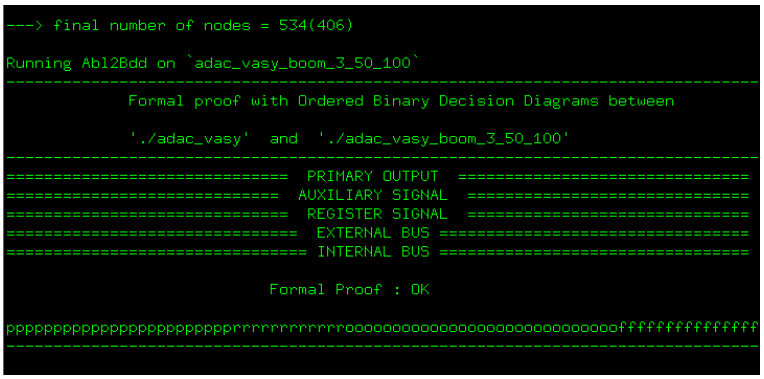


Figura 1.6: Prova formal com a ferramenta proof

Nesta etapa do projeto será realizado uma sintetização do circuito a partir da descrição lógica otimizada com mapeamento automático sobre a biblioteca *sxlib*. Inicia-se verificando as possibilidades de otimização com a ferramenta *boog*. Os parâmetros importantes são a área do circuito e o tempo para percorrer o caminho crítico. Na figura (7) é possível visualizar os dados, com um tempo de 2677 (ps) e com um tamanho de 124.750 (λ). Para gerar esses dados foram utilizados os seguintes comandos: **export MBK_OUT_LO=vst** e **boog adac_vasy_boom_3_50_100 adac_vasy_boom_3_50_100_boog_2**.

Com a execução do comando **xsch -ladac_vasy_boom_3_50_100_boog_2** é criado o modelo esquemático, visto na figura (8). Este demostra o caminho mais longe entre a entrada do circuito e a saída. Nas próximos passos serão operados técnicas que possam melhorar essa percurso. A simulação da netlist com zero de atraso ou delay em cada porta lógica é feita a partir da linha de comando a seguir **asimut -zd adac_vasy_boom_3_50_100_boog_2 adac_sem_atraso adac_vasy_boom_3_50_100_boog_2_res**

```
Saving file 'adac_vasy_boom_3_50_100_boog_2.vst'...
Quick estimated critical path (no warranty)...2677 ps from 'acumulador 0' to 's
2'
Quick estimated area (with over-cell routing)...124750 lambda
Details...
        inv_x2: 15
        xr2_x1: 11
        o2_x2: 7
        na3_x1: 6
        na4_x1: 6
        na2_x1: 5
        buf_x2: 4
        sff1_x4: 4
        mx3_x2: 3
        ao22_x2: 2
        no2_x1: 2
        oa2a22_x2: 2
        a2_x2: 2
        oa2ao222_x2: 1
        on12_x1: 1
        nxr2_x1: 1
        noa22_x1: 1
        no3_x1: 1
        o4_x2: 1
        oa22_x2: 1
        an12_x1: 1
        o3_x2: 1
        Total: 78
Saving critical path in xsch color file 'adac_vasy_boom_3_50_100_boog_2.xsc'...
End of boog...
```

Figura 1.7: Otimização com ferramenta boog

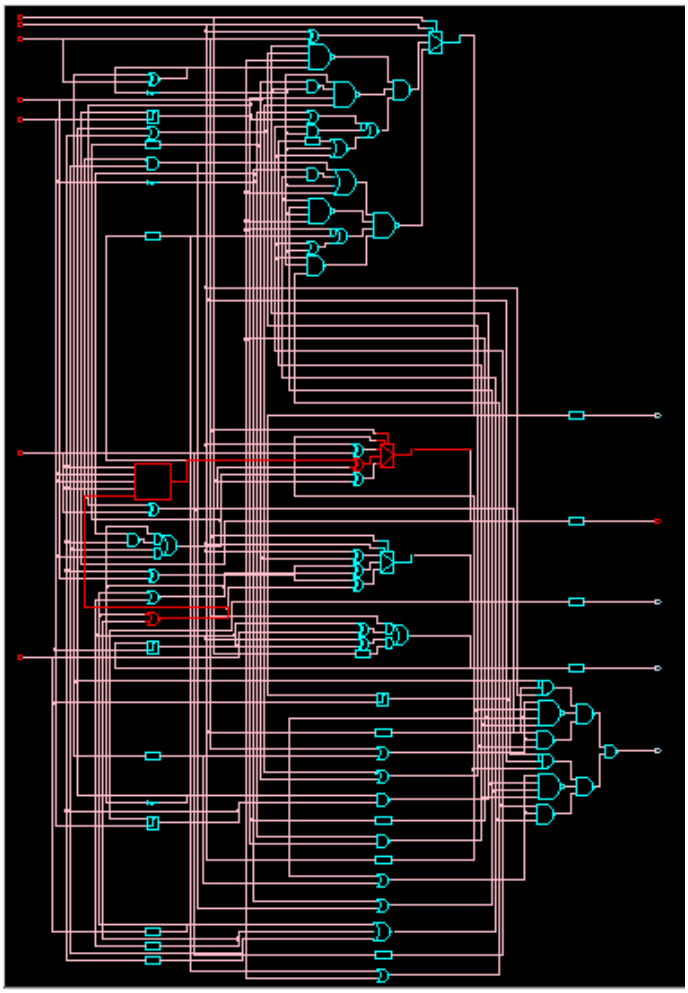
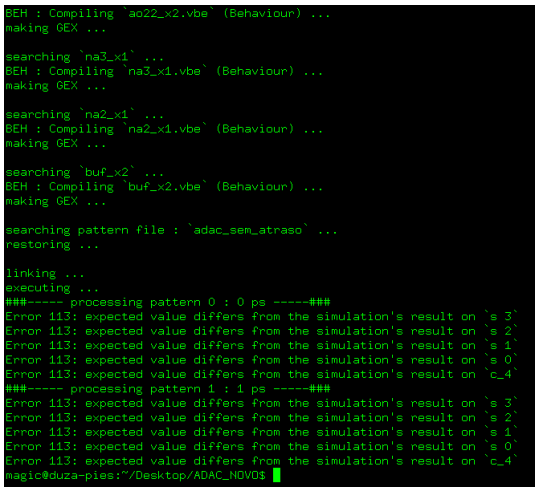


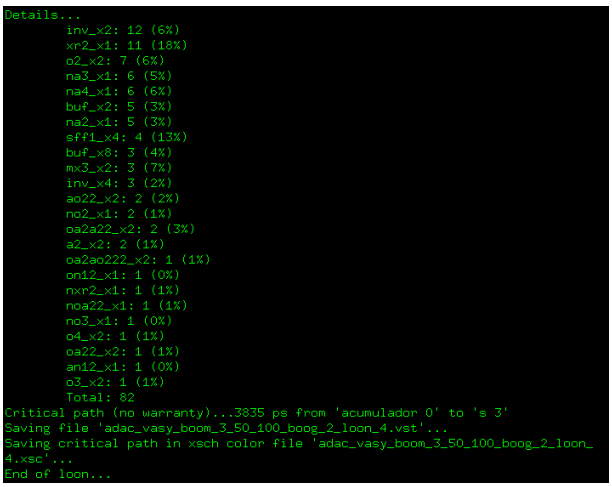
Figura 1.8: Primeiro modelo esquemático

Até esse momento, estávamos lidando com um arquivo que produzia uma simulação sem atraso, o que provoca erros já há modificação nas entradas, contudo há insuficiência de tempo para alterar a saída. Este erro é explicitado na figura (9-a). Como solução para esse empasse,

é preciso criar um arquivo com tempo de atraso modificando apenas a entrada em um vetor e a apenas a saída no outro. Na linha de execução **asimut adac_vasy_boom_3_50_100_boog_2 adac_com_atraso_13000 adac_vasy_boom_3_50_100_boog_2_13000_res**, foi testada com um tempo de atraso de 13000 e se obtive êxito.



(a)



(b)

Figura 1.9: Resultados de simulação

A ferramenta *loon* propõe uma solução com melhoria na velocidade e espaços ocupados pelas células, mas como estamos trabalhando com um circuito pequeno, é possível que piore o desempenho. Para aplicar essa ferramenta utiliza-se os comandos abaixo e o resultado pode ser visto na figura (10). Na figura (9-b) fica evidente o aperfeiçoamento em relação a figura (7).

- **loon -m 4 adac_vasy_boom_3_50_100_boog_2 adac_vasy_boom_3_50_100_boog_2_loon_4**
- **xsch -l adac_vasy_boom_3_50_100_boog_2_loon_4**
- **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**

A definição do posicionamento dos conectores ao redor do núcleo e posicionamentos dos *standard cells* do núcleo são feitas pelo *OCP* com os comandos abaixo. Isto produz um arquivo .ioc que estabelece a ordem dos conectores. Já a figura (11) corresponde ao ADAC posicionado

- **cp adac_vasy_boom_3_50_100_boog_2_loon_4.vst adac_core.vst**
- **alliance-ocp -rows 6 -ioc adac adac_core adac_core_posicionado**
- **graal -l 'adac_core_posicionado'**

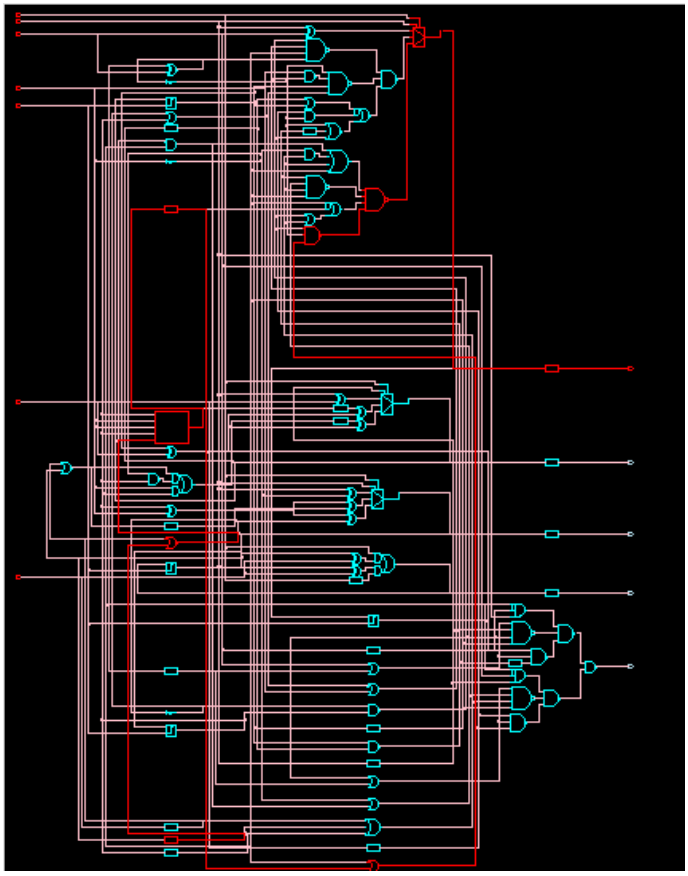


Figura 1.10: Segundo modelo esquemático

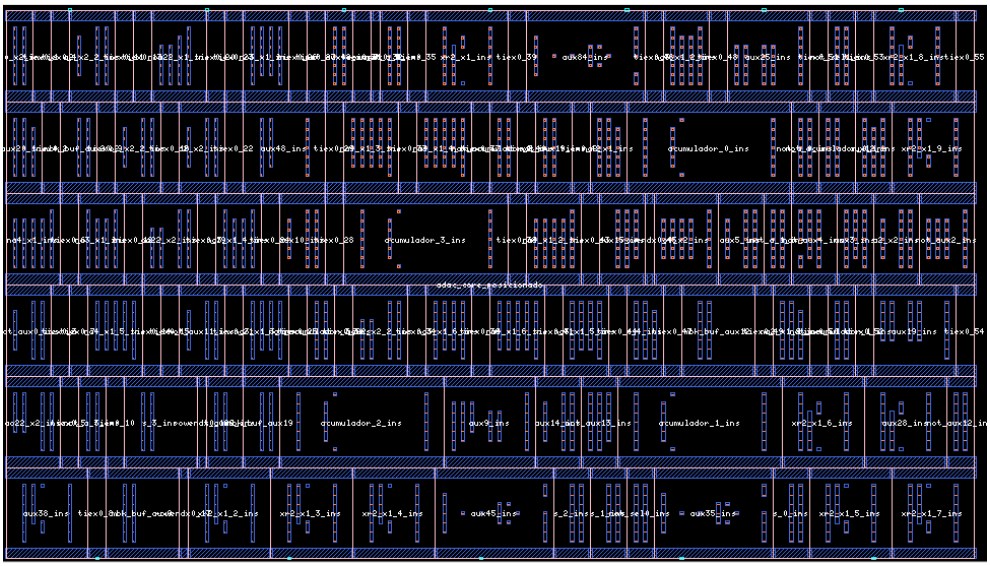
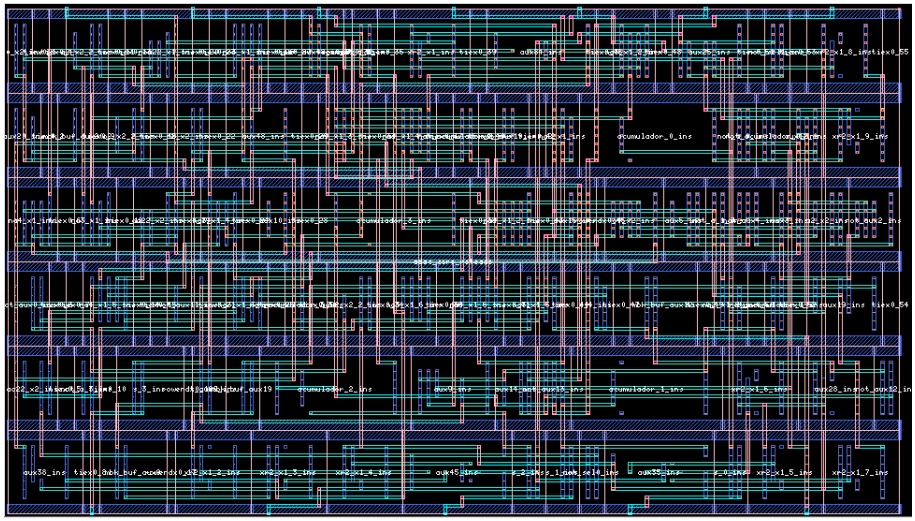


Figura 1.11: Posicionamento do circuito

O *Nero* é aplicado para realizar o roteamento do circuito usando o layout do *Graal* com o comando **nero -p adac_core_posicionado adac_core adac_core_roteado** . A figura (12) exibe o resultado desse processo. Em seguida é feito a construção da netlist dos interconectores a partir do layout. A estrutura de dados **export MBK_OUT_LO=al** descreve a netlist

e o **cougar adac_core_roteado adac_core_roteado_extra**. Já o *lvx* compara a netlist extraída do roteamento com a sintetizada com o comando **lvx al vst adac_core_roteado_extra adac_core**. É realizado uma simulação da netlist extraída do layout roteado com o modelo de referência com atraso usando **asimut adac_core_roteado_extra adac_com_atraso_13000 adac_core_roteado_extra_13000_res** visto na figura (13).



(a)

Figura 1.12: Roteamento do circuito

```
###----- processing pattern 230 : 2990000 ps -----###
###----- processing pattern 231 : 3003000 ps -----###
###----- processing pattern 232 : 3016000 ps -----###
###----- processing pattern 233 : 3029000 ps -----###
###----- processing pattern 234 : 3042000 ps -----###
###----- processing pattern 235 : 3055000 ps -----###
###----- processing pattern 236 : 3068000 ps -----###
###----- processing pattern 237 : 3081000 ps -----###
###----- processing pattern 238 : 3094000 ps -----###
###----- processing pattern 239 : 3107000 ps -----###
###----- processing pattern 240 : 3120000 ps -----###
###----- processing pattern 241 : 3133000 ps -----###
###----- processing pattern 242 : 3146000 ps -----###
###----- processing pattern 243 : 3159000 ps -----###
###----- processing pattern 244 : 3172000 ps -----###
###----- processing pattern 245 : 3185000 ps -----###
###----- processing pattern 246 : 3198000 ps -----###
###----- processing pattern 247 : 3211000 ps -----###
###----- processing pattern 248 : 3224000 ps -----###
###----- processing pattern 249 : 3237000 ps -----###
###----- processing pattern 250 : 3250000 ps -----###
###----- processing pattern 251 : 3263000 ps -----###
###----- processing pattern 252 : 3276000 ps -----###
###----- processing pattern 253 : 3289000 ps -----###
###----- processing pattern 254 : 3302000 ps -----###
###----- processing pattern 255 : 3315000 ps -----###
magic@duza-pies:~/Desktop/ADAC_NOVO$
```

(a)

Figura 1.13: Simulação do layout simulado

Escrevendo as seguintes linhas de comando gera-se a netlist do núcleo mais os pads e a visualização deste em um esquemático pode ser vista na figura (14). Indo em Tool -> Hierarchy e clicando sobre o núcleo é possível ver o conteúdo do core expresso na figura (15)

- **export MBK_IN_LO=vst**
- **export MBK_OUT_LO=vst**
- **genlib adac_chip**
- **xsch -l adac_chip**

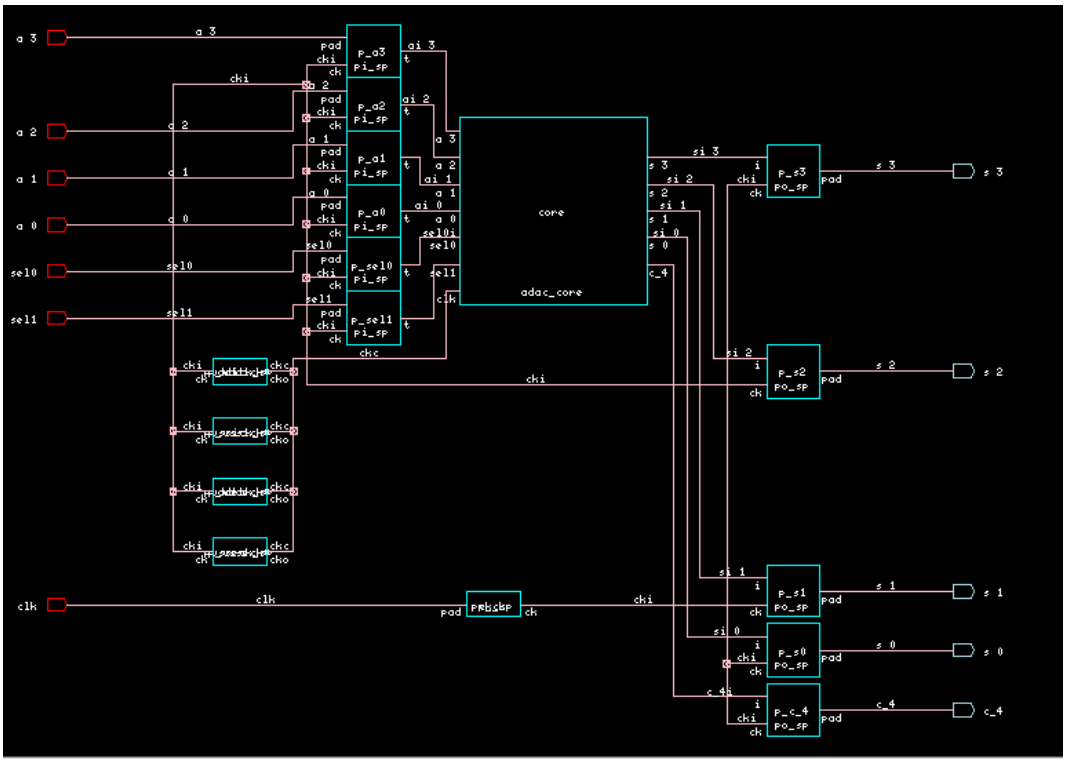
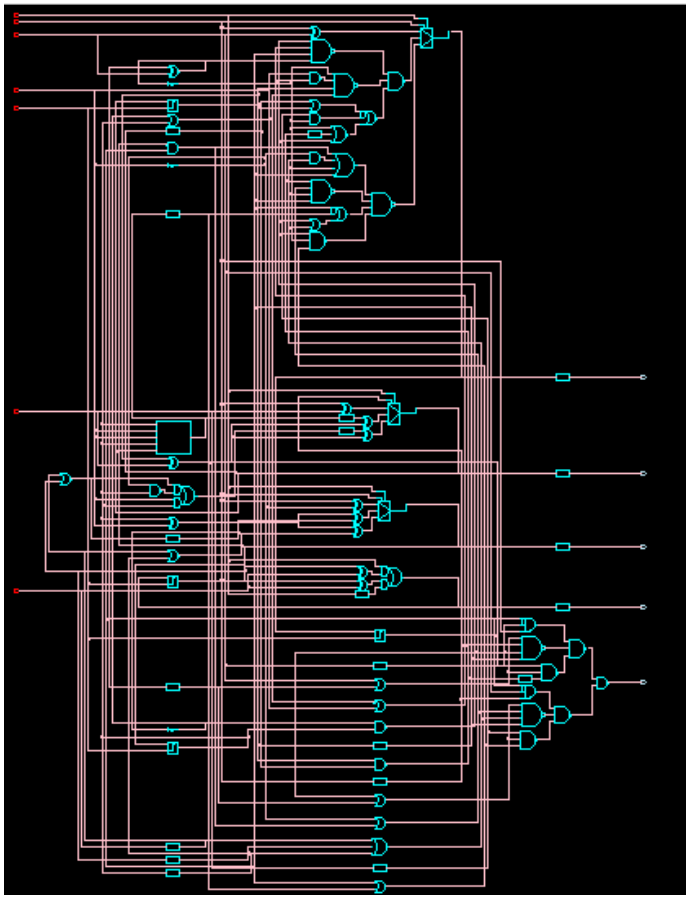


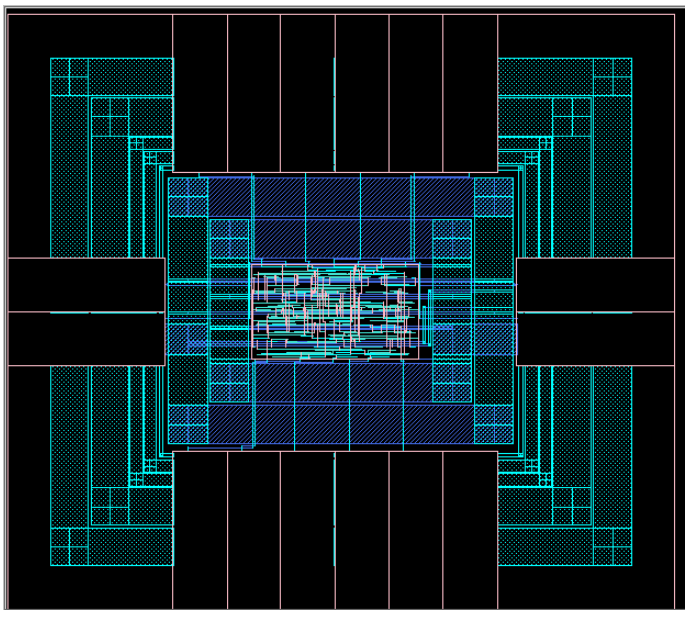
Figura 1.14: Modelo esquemático do chip

Verifica-se que não houve alteração na funcionalidade do ADAC após a conexão dos pads com linha **asimut adac_chip adac_com_atraso_13000 adac_chip_res_13000**. A figura (16) e (17) mostra a visão layout do chip ADAC no Graal.



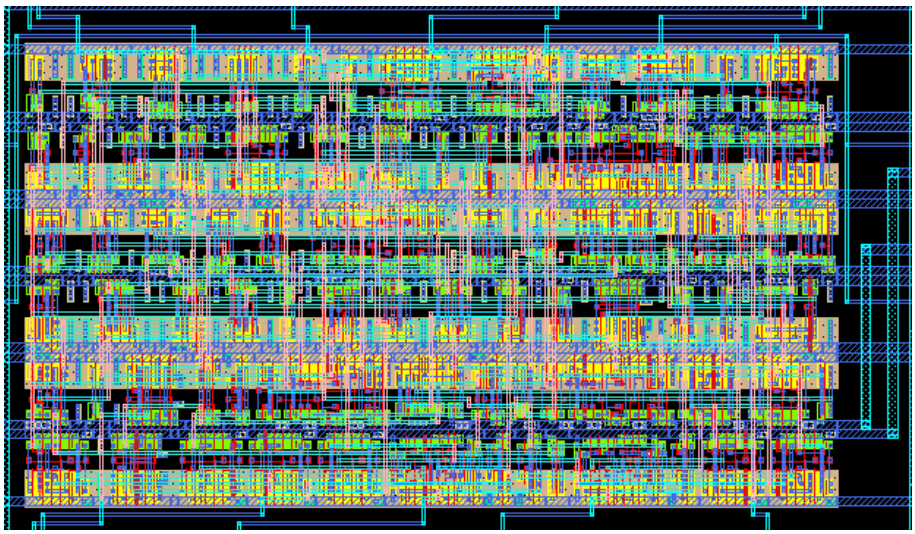
(a)

Figura 1.15: Modelo esquemático dentro do núcleo



(a)

Figura 1.16: Layout do chip ADAC



(a)

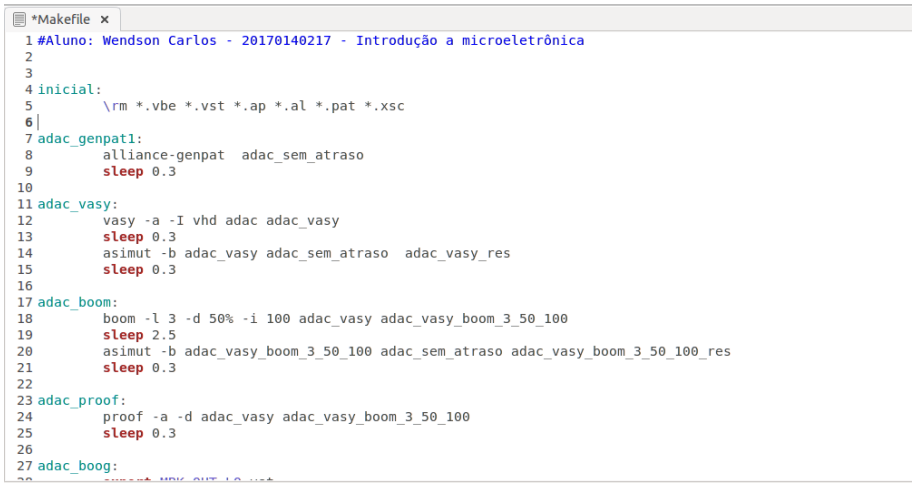
Figura 1.17: Layout do núcleo do chip ADAC

Com a ferramenta *ring* cria-se automaticamente o layout com o core roteado e os pads usando a linha de comando **ring adac_chip adac_chip**. Com os comandos abaixo conseguimos provar que a netlist extraída do chip é igual as anteriores. Com a ferramenta *x2y*, o projeto é convertido para outros formatos, usando **x2y vst al adac_core adac_core**, na qual cria-se uma cópia de adac_core.al. Utilizando o comando **asimut adac_chip_ex adac_com_atraso_13000 adac_chip_ex_res_13000** prova-se que o chip se comporta como as especificações iniciais propostas ao projeto.

- **export MBK_OUT_LO=vst**
- **cougar adac_chip adac_chip_ex**
- **lvx al vst adac_chip_ex adac_chip**

1.4 Criação do Makefile

Neste relatório os arquivos **Makefile** e **execute.sh** são criados com o intuito facilitar a compilações dos comandos citados anteriormente. É importante mencionar que todos os comandos do projeto do ADAC foram executados manualmente a fim de analisar e compreender o processo de desenvolvimento do chip e o makefile criado em último lugar. Esse arquivo é composto de labels categorizadas de acordo com as ferramentas do Alliance tática empregada para, caso acontece algum erro, seja possível executar os blocos de comandos separadamente. A figura (18) exibe um trecho do código.

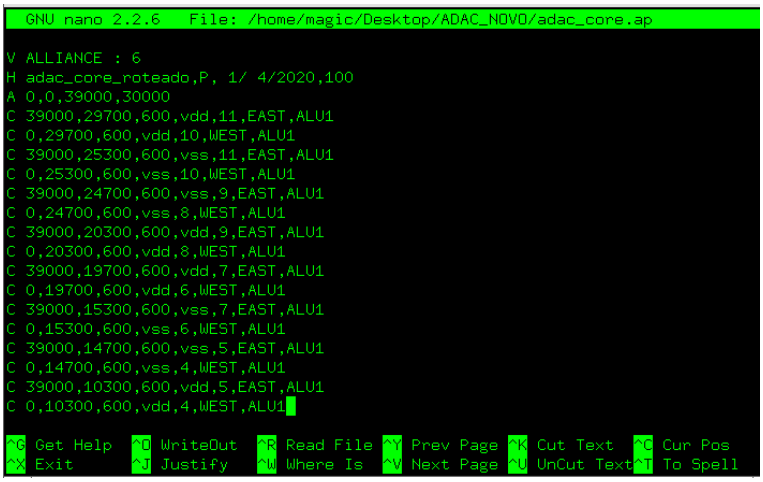


```
*Makefile ×
#Aluno: Wendson Carlos - 20170140217 - Introdução a microeletrônica


inicial:
        \rm *.vbe *.vst *.ap *.al *.pat *.xsc

adac_genpat1:
        alliance-genpat  adac_sem_atraso
        sleep 0.3

adac_vasy:
        vasy -a -I vhd adac adac_vasy
        sleep 0.3
        asimut -b adac_vasy adac_sem_atraso  adac_vasy_res
        sleep 0.3

adac_boom:
        boom -l 3 -d 50% -i 100 adac_vasy adac_vasy_boom_3_50_100
        sleep 2.5
        asimut -b adac_vasy_boom_3_50_100 adac_sem_atraso adac_vasy_boom_3_50_100_res
        sleep 0.3

adac_proof:
        proof -a -d adac_vasy adac_vasy_boom_3_50_100
        sleep 0.3

adac_boog:
```

(a)

Figura 1.18: Arquivo Makefile

Em primeiro lugar, execute o terminal na pasta que contém os arquivos do ADAC. Digite o comando **chmod 777 execute.sh** que dará permissão para leitura, escrita e alteração ao .sh(Bash Shell Script File) no diretório. Em seguida execute com **./execute.sh** e após isso o processo se iniciará. Durante a execução aparecerá na tela esse script como na figura (19). Essa label foi criada para que possível alterar o arquivo "adac_core_roteado"para "adac_core". Depois dá modificação, deve-se aperta **Ctrl X** para sair, **Y** para confirmar as modificações e **Enter** para dá continuidade ao processo. Ao final será gerado todos os arquivos iguais ao descritos durante o relatório. Para confirmar que o makefile deu certo, é executado a linha de comando **graal -l 'adac_chip'**



```
  GNU nano 2.2.6   File: /home/magic/Desktop/ADAC_NOVO/adac_core.ap

V ALLIANCE : 6
H adac_core_roteado,P, 1/ 4/2020,100
A 0,0,39000,30000
C 39000,29700,600,vdd,11,EAST,ALU1
C 0,29700,600,vdd,10,WEST,ALU1
C 39000,25300,600,vss,11,EAST,ALU1
C 0,25300,600,vss,10,WEST,ALU1
C 39000,24700,600,vss,9,EAST,ALU1
C 0,24700,600,vss,8,WEST,ALU1
C 39000,20300,600,vdd,9,EAST,ALU1
C 0,20300,600,vdd,8,WEST,ALU1
C 39000,19700,600,vdd,7,EAST,ALU1
C 0,19700,600,vdd,6,WEST,ALU1
C 39000,15300,600,vss,7,EAST,ALU1
C 0,15300,600,vss,6,WEST,ALU1
C 39000,14700,600,vss,5,EAST,ALU1
C 0,14700,600,vss,4,WEST,ALU1
C 39000,10300,600,vdd,5,EAST,ALU1
C 0,10300,600,vdd,4,WEST,ALU1

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

(a)

Figura 1.19: Comando adac_change