UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA - CI

GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

RELATÓRIO FINAL - INTRODUÇÃO À MICROELETRÔNICA

LAURA LETÍCIA ARAÚJO OLIVEIRA CAMPOS

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 Introdução

1.1.1 ADDAC

O circuito ADDAC, relatado no presente relatório, representa um somador acumulador de 4 bits. Para efetuar uma análise do sistema, é necessária a criação de um modelo de ouro *(Golden Model)*, para analisar as possíveis situações de entrada e definir como o circuito deve se comportar. O diagrama do circuito pode ser visualizado na Figura 1.1.

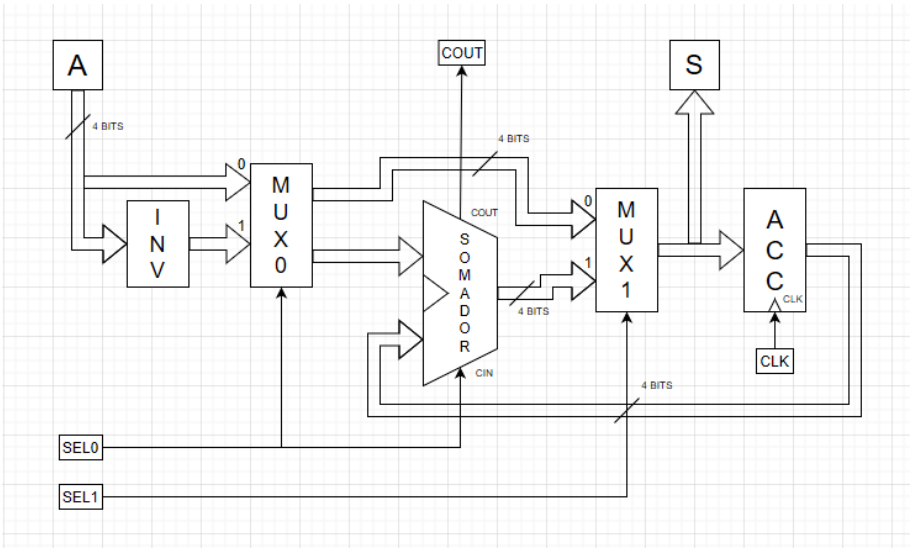

Figura 1.1: Diagrama lógico do ADDAC

Após a criação do diagrama foi possível construir uma tabela verdade para descrever o comportamento do circuito. A análise das linhas permite observar que se trata de um circuito sequencial, ou seja, mudando a entrada a qualquer momento, a saída deve ter uma resposta, independemente do pulso de *clock*. A Tabela 1 expressa a funcionalidade do circuito.

Tabela 1.1: Tabela verdade do ADDAC

Sel 0	Sel 1	Resposta
0	0	S recebe A
0	1	S recebe A + ACC
1	0	S recebe !A
1	1	S recebe ACC - A

1.1.2 Objetivos

Este trabalho tem como objetivo projetar o *layout* de um chip de um somador acumulador de 4 bits, fazendo uso da ferramenta Genlib para declarar, em linguagem C, a conectividade das *standard cells* selecionadas. Também visa criar um modelo comportamental desse bloco em linguagem de descrição de *hardware* em alto nível, assim como realizar a conversão da descrição funcional em descrição comportamental em nível RTL *(Register Transfer Level)* através da ferramenta VASY.

1.2 Metodologia

1.2.1 Graal

Para a realização de grande parte das atividades do presente relatório, utilizou-se a ferramenta gratuita para edição de layouts, Graal. O funcionamento da plataforma é baseado em um conjunto de ferramentas CAD e bibliotecas portáteis para Very-Large-Scale-Integration (VLSI), o que inclui a biblioteca CMOS. O programa efetua a leitura de arquivos do tipo .ap e já possui, internamente, recursos para a verificação de erros, representados pelo comando Druc. Esta ferramenta foi utilizada a partir de um terminal do sistema operacional Linux.

1.2.2 Genlib

A biblioteca Genlib caracteriza um conjunto de funções em C que permite a criação de objetos e utilização de objetos do VLSI. Essa linguagem, permite a descrição dos netlists, assim

como a visualização dos layouts. Para uso da biblioteca, basta incluir o comando “genlib.h” no código em C.

1.2.3 Genpat

A biblioteca Genpat caracteriza um conjunto de funções em C que permite a descrição procedural dos arquivos de entrada padrão para o simulador lógico ASIMUT. Essa linguagem, Para uso da biblioteca, basta incluir o comando “genpat.h” no código em C.

1.2.4 Ferramentas

Além das bibliotecas descritas anteriormente, foi utilizado um conjunto de ferramentas pertencentes ao *Alliance CAD System*, desenvolvido pela Universidade Pierre et Marie Curie, em Paris, França. As funções utilizadas são descritas a seguir.

1. VASY: é um analizador hierárquico de VHDL *(Very High Speed Integrated Circuits Hardware Description Language)* para síntese. Realiza uma análise semântica de um descritor VHDL RTL no nome do arquivo e identifica com precisão todos os elementos de memória e *buffers tristate*.

2. ASIMUT: é uma ferramenta lógica para simulação de descrições de *hardware*. Compila e carrega uma descrição de *hardware* completa escrita em VHDL, que pode ser estrutural ou comportamental.

3. BOOM: é utilizado para o primeiro passo do processo de síntese. Otimiza a descrição comportamental utilizando a representação das funções lógicas através de um diagrama de decisão binária ordenado.

4. PROOF: é feito para "rodar"uma descrição de fluxo de dados. Também utiliza uma representação das funções lógicas através de um diagrama de decisão binária ordenado que permite provar facilmente a equivalência entre duas descrições comportamentais.

5. BOOG: é um mapeador de uma descrição comportamental para uma biblioteca predefinida de *standard cells*, como a SXLIB.

6. XSCH: é um visualizador gráfico dos esquemáticos gerados.

7. LOON: é uma ferramenta de CAD que permite remover problemas de *fanout* dentro de uma *netlist* e também otimizar o atraso. A *netlist* pode ser hierárquica e achatada, se necessário.

8. OCP: é uma ferramenta de posicionamento automático para *standard cells*.

9. NERO: é um roteador simples para pequenos *designs* acadêmicos. Utiliza o *layout* do GRAAL.

10. COUGAR: é um extrator hierárquico de *layout*. Constrói a *netlist* de interconexões de uma visão simbólica do *layout*.

11. LVX: compara duas *netlists* no nível de porta ou de bloco. O objetivo é comparar a especificação de uma *netlist* com a *netlist* física obtida através do extrator COUGAR.

12. RING: define as posições as quais se deseja aplicar aos *pads* do circuito.

13. X2Y: é um conversor de formato da *netlist*.

1.3 Geração Procedural

Inicialmente, foi criado um arquivo .C com a descrição comportamental de um somador acumulador de 4 bits sem atraso, utilizando a formatação da biblioteca GENPAT. Após compilado, foi realizada uma concepção *top-down* do circuito até o nível RTL, seguida da descrição VHDL em alto nível, através da ferramenta VASY. Ainda foi utilizada a ferramenta ASIMUT para a detecção de possíveis erros, como ilustra a Figura 1.2, e o comando XPAT para visualizar a simulação gráfica do circuito, como ilustrado na Figura 1.3. As fases citadas foram obtidas a partir dos comandos a seguir.

1. alliance-genpat adac_sem_atraso

2. vasy -a -I vhd adac adac_vasy

3. asimut -b adac_vasy adac_sem_atraso adac_vasy_res

4. xpat -l adac_vasy_res

Em seguida, foi realizada uma otimização do circuito, através da ferramenta BOOM e uma posterior detecção de erros utilizando o ASIMUT. Ainda foi utilizado o comando "export", para obter a *netlist* de visualização do arquivo, e as ferramentas PROOF e BOOG, para realizar uma prova formal entre as duas descrições comportamentais do VHDL e mapear as células padrão do circuito, presentes na biblioteca SXLIB, respectivamente. Os comandos descritos são enumerados a seguir.

```
###----- processing pattern 105 : 105 ps -----###
###----- processing pattern 106 : 106 ps -----###
###----- processing pattern 107 : 107 ps -----###
###----- processing pattern 108 : 108 ps -----###
###----- processing pattern 109 : 109 ps -----###
###----- processing pattern 110 : 110 ps -----###
###----- processing pattern 111 : 111 ps -----###
###----- processing pattern 112 : 112 ps -----###
###----- processing pattern 113 : 113 ps -----###
###----- processing pattern 114 : 114 ps -----###
###----- processing pattern 115 : 115 ps -----###
###----- processing pattern 116 : 116 ps -----###
###----- processing pattern 117 : 117 ps -----###
###----- processing pattern 118 : 118 ps -----###
###----- processing pattern 119 : 119 ps -----###
###----- processing pattern 120 : 120 ps -----###
###----- processing pattern 121 : 121 ps -----###
###----- processing pattern 122 : 122 ps -----###
###----- processing pattern 123 : 123 ps -----###
###----- processing pattern 124 : 124 ps -----###
###----- processing pattern 125 : 125 ps -----###
###----- processing pattern 126 : 126 ps -----###
###----- processing pattern 127 : 127 ps -----###
```

Figura 1.2: Simulação do ASIMUT para o ADDAC sem atraso

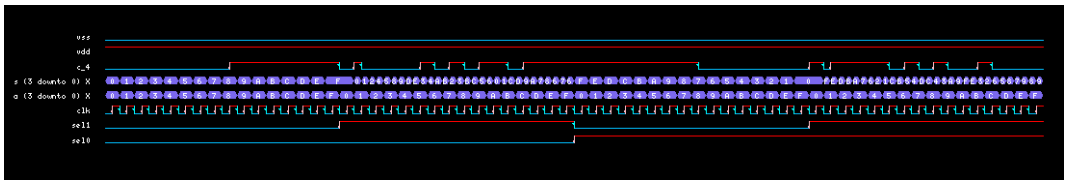

Figura 1.3: Simulação gráfica do circuito (.pat)

1. boom -l 3 -d 50% -i 100 adac_vasy adac_vasy_boom_3_50_100
2. asimut adac_vasy_boom_3_50_100 adac_sem_atraso adac_vasy_boom_3_50_100_res
3. proof -a -d adac_vasy adac_vasy_boom_3_50_100
4. export MBK_OUT_LO=vst
5. boog adac_vasy_boom_3_50_100 adac_vasy_boom_3_50_100_boog_2

As figuras 1.3 e 1.4 dizem respeito aos resultados dos comandos PROOF e BOOG, respectivamente. A primeira obtem a prova formal de que a ferramenta BOOM não alterou logicamente a descrição RTL e a segunda aponta para parâmetros importantes para a otimização do circuito, como a sua área equivalente, identificada como 92000λ.

Posteriormente, foi utilizada a ferramenta XSCH para visualizar o esquemático do ADDAC desenvolvido até o momento. Na figura 1.6, que ilustra o resultado do comando, as linhas vermelhas indicam o caminho crítico, ou seja, o caminho mais longo entre as entradas e a saída.

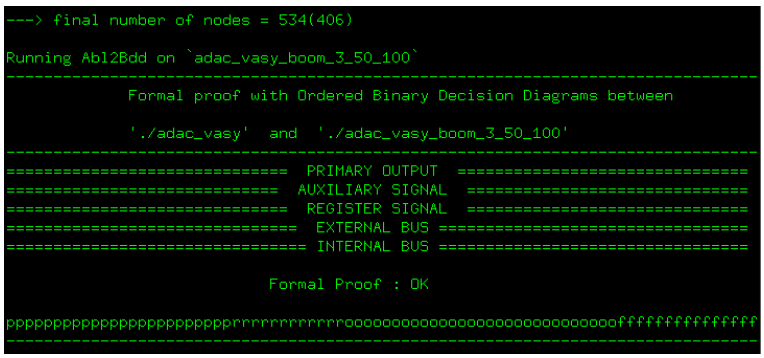

Figura 1.4: Simulação do PROOF para o circuito

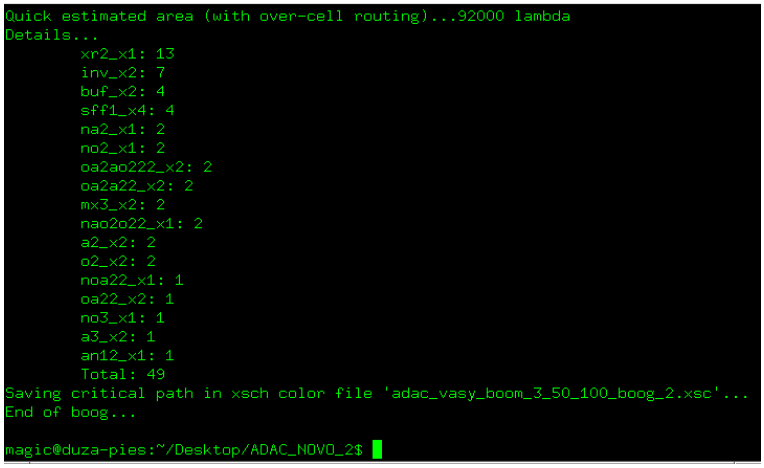

Figura 1.5: Simulação do BOOG para o ADDAC sem atraso

Ainda foram executadas duas vezes o comando referente à ferramenta ASIMUT, para simulação da *netlist* com zero *delay* e com o atraso em cada porta lógica. Os comandos são listados abaixo.

1. xsch -l adac_vasy_boom_3_50_100_boog_2
2. asimut -zd adac_vasy_boom_3_50_100_boog_2
   adac_sem_atraso adac_vasy_boom_3_50_100_boog_2_res
3. asimut adac_vasy_boom_3_50_100_boog_2
   adac_sem_atraso adac_vasy_boom_3_50_100_boog_2_res

A etapa seguinte teve como ponto de partida a criação de um arquivo .C para um somador acumulador de 4 bits com atraso, a partir da edição do código anterior. Foi adicionado um *delay* de 13000 entre os comandos AFFECT para entrada e saída. Também foi utilizada ferramenta ASIMUT para a detecção de erros do novo circuito, agora com o atraso adicionado, como é ilustrado na Figura 1.7.

1. alliance-genpat adac_com_atraso

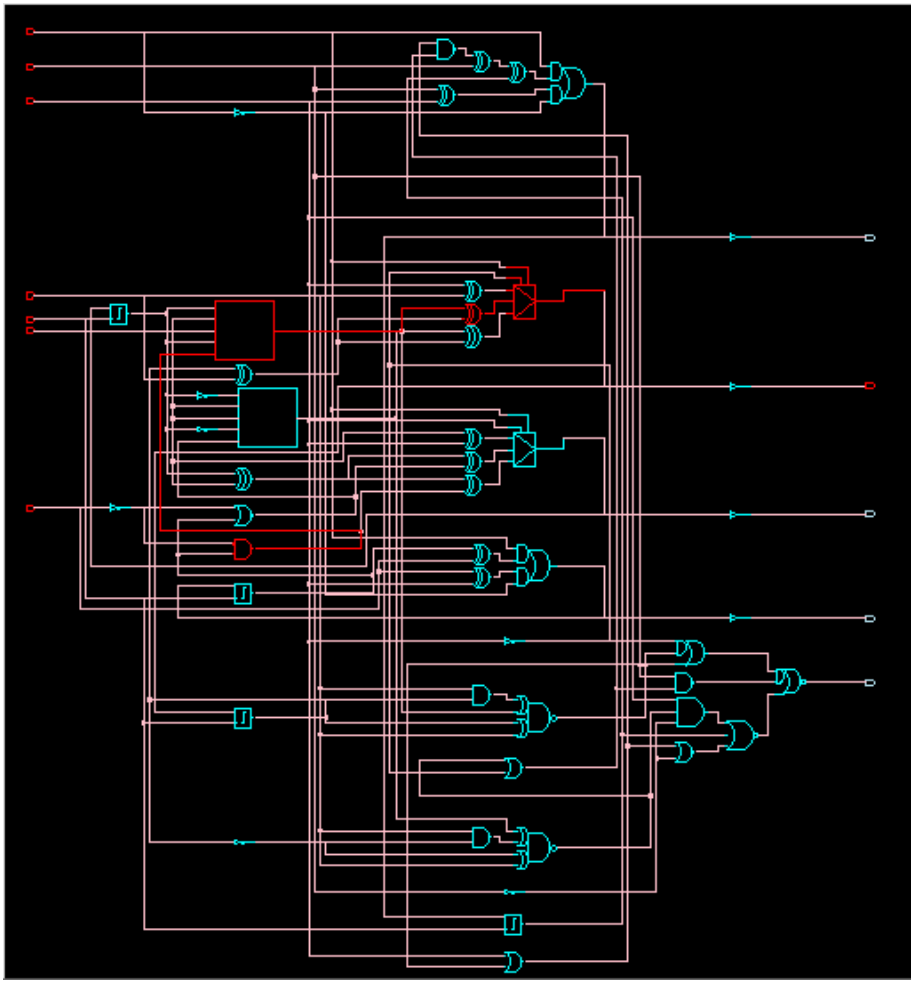

Figura 1.6: Esquemático do circuito

2. asimut adac_vasy_boom_3_50_100_boog_2
   adac_sem_atraso_13000 adac_vasy_boom_3_50_100_boog_2_13000_res

Após isso, foi utilizada a ferramenta LOON para realizar uma análise do caminho feito pela solução anterior do XSCH e propor uma versão otimizada. Foram utilizadas as ferramentas XSCH, para ilustrar graficamente o novo esquemático do circuito, e ASIMUT, para a detecção de erros. A otimização realizada pela ferramenta LOON e o esquemático gerado, podem ser vistos nas Figuras 1.8 e 1.9, respectivamente.

1. loon -m 4 adac_vasy_boom_3_50_100_boog_2
   adac_vasy_boom_3_50_100_boog_2_loon_4

2. xsch -l adac_vasy_boom_3_50_100_boog_2_loon_4

3. 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

Figura 1.7: Simulação do ASIMUT para o ADDAC com atraso

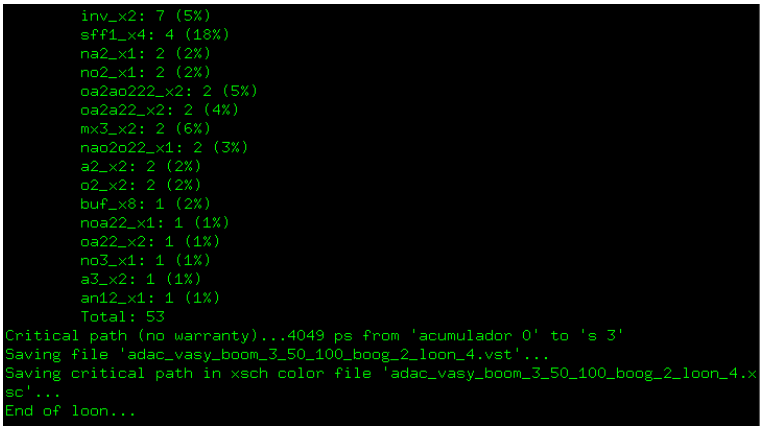

Figura 1.8: Simulação do LOON para o ADDAC com atraso

Em seguida, foram utilizadas as ferramentas OCP e NERO para realizar o posicionamento das *standard cells* do núcleo (core) e o seu roteamento, respectivamente, como ilustrado nas Figuras 1.10 e 1.11. Os comandos necessários são listados a seguir.

1. cp adac_vasy_boom_3_50_100_boog_2_loon_4.vst adac_core.vst
2. alliance-ocp -rows 6 -ioc adac adac_core adac_core_posicionado
3. graal -l 'adac_core_posicionado'
4. nero -p adac_core_posicionado adac_core adac_core_roteado
5. graal -l 'adac_core_roteado'

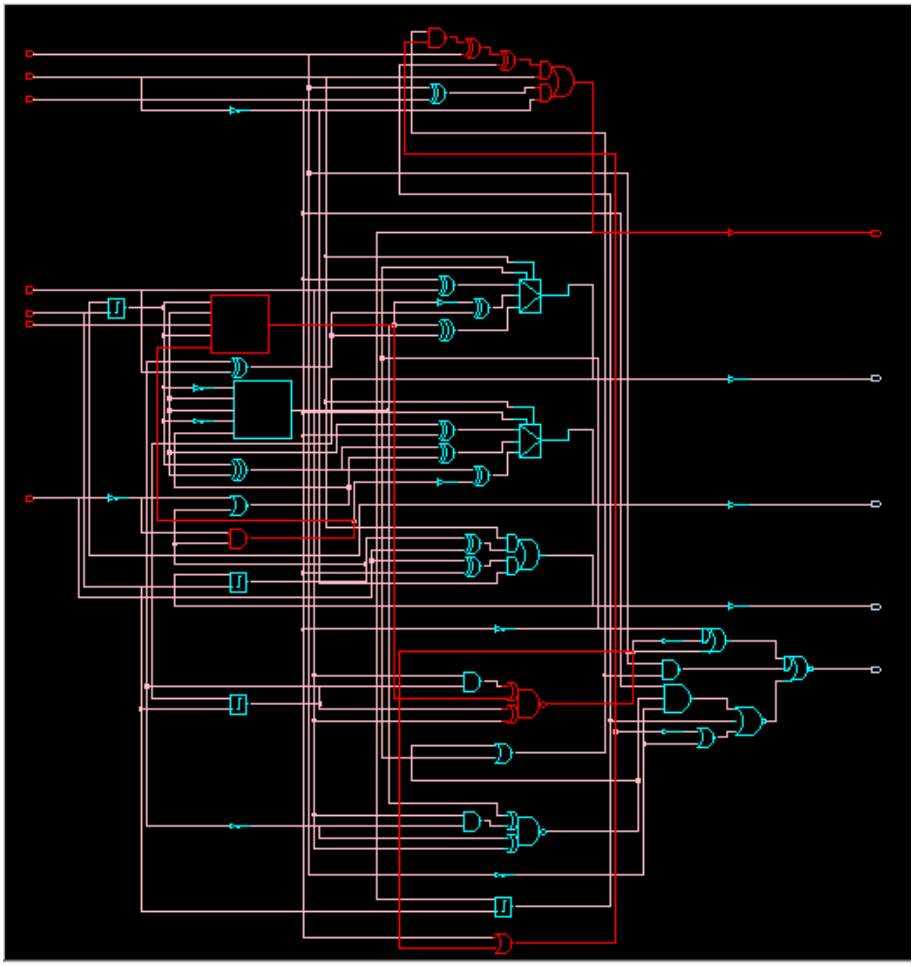

Figura 1.9: Esquemático do circuito otimizado

Seguidamemnte, foi utilizada a ferramenta COUGAR para realizar a extração da *netlist* de interconexões, através da estrutura de dados MBK. Também foi utilizada a ferramenta LVX para realizar o posicionamento e roteamento através da comparação das *netlists*, como mostrado na Figura 1.12, e, em seguida, a ferramenta ASIMUT para a verificação do funcionamento correto da descrição comportamental para a versão *al* simulada.

1. export MBK_OUT_LO=al
2. cougar adac_core_roteado adac_core_roteado_extra
3. lvx al vst adac_core_roteado_extra adac_core
4. export MBK_IN_LO=al
5. asimut adac_core_roteado_extra adac_com_atraso_13000
   adac_core_roteado_extra_13000_res

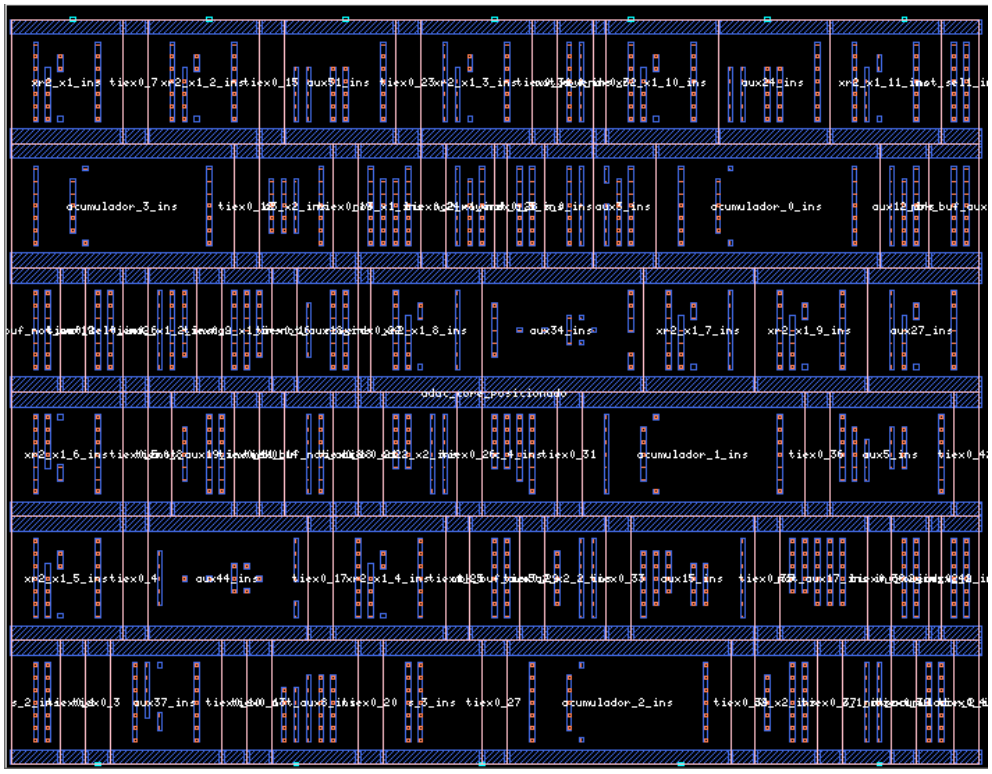

Figura 1.10: Somador com acumulador de 4 bits posicionado

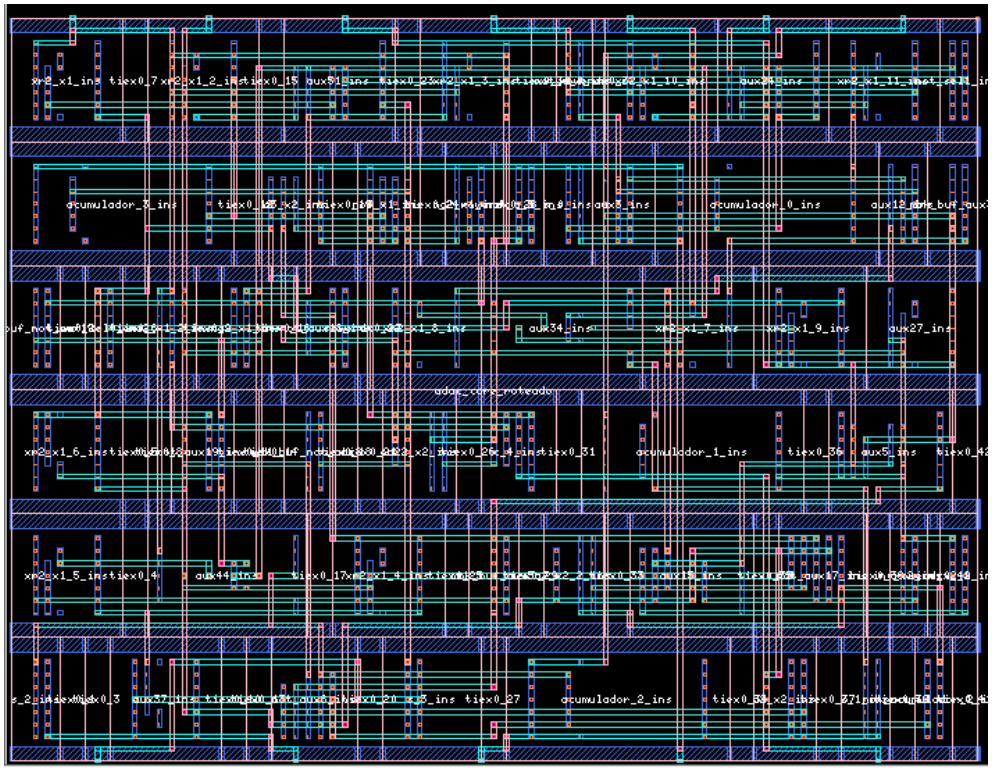

Figura 1.11: Somador com acumulador de 4 bits roteado

Após a realização dos comandos descritos anteriormente, está finalizado o projeto do núcleo roteado. Para dar prosseguimento, será feita a extração das *netlists* no formato .vst, através da

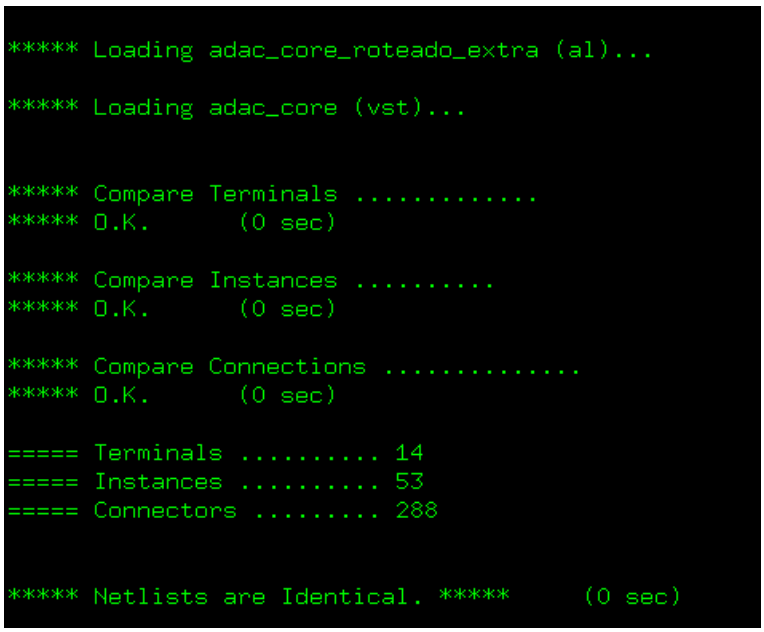

Figura 1.12: Simulação do LVX

estrutura MBK e a geração do esquemático do circuito através da ferramenta XSCH, que pode ser visto na Figura 1.13. Também será realizada uma simulação através da ferramenta ASIMUT.

1. export MBK_IN_LO=vst
2. export MBK_OUT_LO=vst
3. genlib adac_chip
4. xsch -l adac_chip
5. asimut adac_chip adac_com_atraso_13000 adac_chip_res_13000

Por fim, são utilizadas as ferramenta RING, para a criação automática do *layout* com *cores* e *pads*, como mostrado da figura 1.14, e X2Y, para a conversão de tipos diferentes de arquivos para o mesmo tipo de objetos. Ainda foi realizada uma simulação com a ferramenta ASIMUT para a certificação de que o projeto do chip foi concluído com êxito. A seguir são enumerados os comandos finais.

1. ring adac_chip adac_chip
2. graal -l 'adac_chip'
3. export MBK_OUT_LO=al

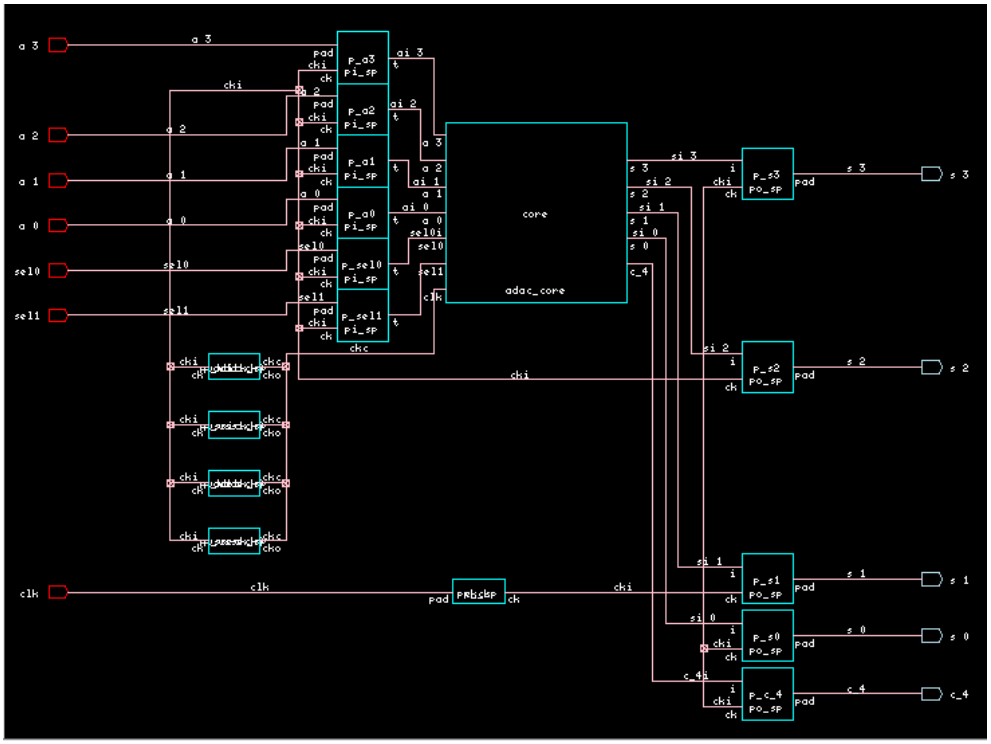

Figura 1.13: Esquemático do chip do ADDAC

4. cougar adac_chip adac_chip_ex
5. lvx al vst adac_chip_ex adac_chip
6. export MBK_OUT_LO=al
7. x2y vst al adac_core adac_core
8. asimut adac_chip_ex adac_com_atraso_13000 adac_chip_res_13000

1.4 *Makefile*

Após a confecção do *layout* do chip do somador acumulador de 4 bits, foi criado um arquivo *makefile* para automatizar o processo de construção de aplicações chamando o compilador e executando testes. O arquivo é um mapa do projeto indicando todos arquivos que estão envolvidos e como eles devem ser compilados. O código *make*, assim como todos os arquivos gerados pelas ferramentas do *Alliance CAD System*, encontra-se em anexo.

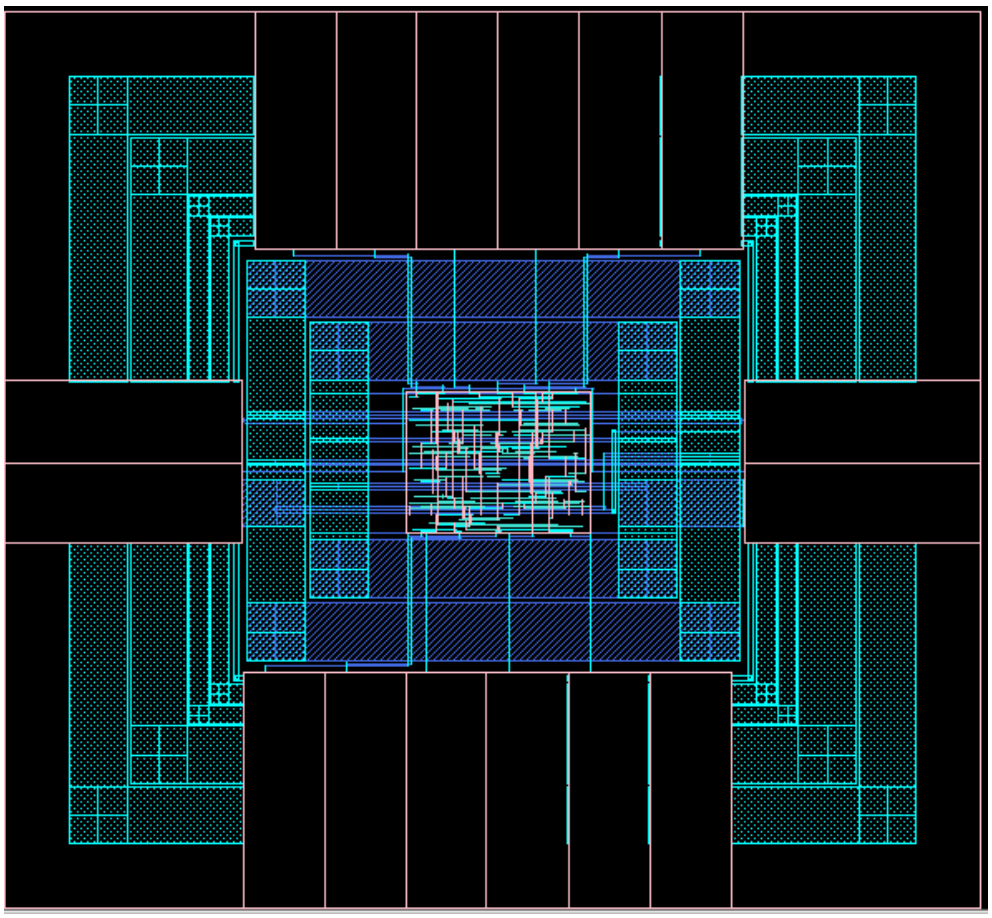

Figura 1.14: Chip do ADDAC finalizado

1.5 Considerações Finais

Diante do exposto, pode-se concluir que o *layout* do chip confeccionado ficou de acordo com as especificações de funcionalidade descritas no diagrama do projeto, sem a presença de erros, como foi afirmado através da última execução da ferramenta ASIMUT. Por fim, a atividade colaborou para ampliar os conhecimentos sobre microeletrônica e fixar os conceitos aprendidos ao longo da disciplina.

REFERÊNCIAS

GUIA ADAC. Disponível em: <https://sigaa.ufpb.br/sigaa/portais/discente/beta/discente.jsf>. Acesso em 29 mar. 2020.