VERIFICAÇÃO DE PROJETOS DE CIRCUITOS INTEGRADOS

Antonio Anibal de Souza Teles

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.).

Aprovada por:

YSMAR VIANNA E SILVA FILHO

EBER ASSIS SCHMITZ

PAULO MÁRIO BIANCHI FRANÇA

EDIL SEVERIANO TAVARES FERNANDES

RIO DE JANEIRO, RJ - BRASIL

JULHO DE 1983.

TELES, ANTONIO ANIBAL DE SOUZA

Verificação de Projetos de Circuitos Integrados (Rio de Janeiro) 1983.

VIII, 87 p. 29,7cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas, 1983).

Tese - Universidade Federal do Rio de Janeiro, COPPE.

1. Projeto Assistido por Computador. 2. Circuitos Integrados. I. COPPE/UFRJ. II. Título (série).

A minha esposa e

a meus pais.

A G R A D E C I M E N T O S

Aos colegas JOSÉ ANTONIO DOS SANTOS BORGES, CARLO EMMANOEL TOLIA DE OLIVEIRA e HELOÍSA TEIXEIRA DA SILVA pelas suas sugestões.

A VERA LÚCIA DA COSTA e a CLEBER JOSÉ DE OLIVEIRA RIBEIRO pelos excelentes trabalhos de composição desta tese.

A YSMAR VIANNA E SILVA FILHO pela orientação e pelo apoio.

Aos demais membros da banca que muito me honraram com suas presenças.

R E S U M O

Este trabalho apresenta um conjunto de ferramentas para verificação de projetos de circuitos integrados. Todas foram escritas em Fortran e exigem pequena quantidade de recursos computacionais, o que permite serem utilizadas em computadores de pequeno porte.

Nosso pacote de verificação de projetos faz parte de um trabalho que vem sendo desenvolvido no NCE/UFRJ para apoio ao grupo local de projetos de circuitos integrados e é composto das seguintes ferramentas:

- Simulador funcional
- Verificador de regras de projeto
- Extrator de circuitos
- Verificador estático

Todas as ferramentas listadas estão disponíveis e rodam em um minicomputador PDP-11.

A B S T R A C T

This work presents a set of tools for verifying integrated circuit design. All of them are written in Fortran and do not require large amount of computational resources. These facts allow the tools to be implemented in small size computers.

The design checking package is part of a work been developed at NCE/UFRJ to support our VLSI design group and is composed of the following tools:

- Functional simulator
- Design rule checker
- Circuit extractor
- Static evaluator

All tools listed above are now available, and run on a PDP-11 minicomputer.

ÍNDICE

I - INTRODUÇÃO

Este trabalho tem como objetivo mostrar algumas das principais ferramentas de verificação de projetos de circuitos integrados. Os algoritmos a serem examinados fazem parte de um pacote de verificação por nós desenvolvido.

Resolvemos dividir este trabalho em tópicos, analisando cada ferramenta individualmente. O tópico denominado Projeto de Circuitos Integrados tem por finalidade localizar nosso trabalho no contexto de projetos, sendo lá mostrados os passos necessários à confecção de um circuito integrado e, de forma simplificada, o funcionamento do nosso pacote de verificação. Também são mencionados alguns importantes programas de apoio a projetos que, no entanto, não pertencem ao nosso trabalho de abordagem de ferramentas de verificação.

Cada um dos programas que compõem o pacote é examinado em um tópico, onde é descrito o algoritmo utilizado e, quando possível, alguns métodos alternativos disponíveis na literatura sobre o assunto. As implementações são discutidas localmente em cada tópico.

Finalmente, são sugeridos alguns temas para pesquisa, visando aumentar o nível de automação de projetos de circuitos integrados.

II - OS PROJETOS DE CIRCUITOS INTEGRADOS

2.1 - APRESENTAÇÃO

A denominação circuito integrado, ou simplesmente CI, é usada para referenciar um circuito eletrônico encapsulado em uma única pastilha. Como resultado do aperfeiçoamento de tecnologia de fabricação de CI's é comum encontrarmos pastilhas que contenham circuitos formados por dezenas de milhares de transistores. A expectativa é que, até o fim desta década, o número de transistores em um único circuito denso seja da ordem de milhões.

O emprego de circuitos com elevado fator de integração permite que dispositivos eletrônicos sejam construídos a custos inferiores, com substancial redução de consumo de energia. Dentre as tecnologias de projeto e fabricação existentes destaca-se a NMOS, que é largamente utilizada. Portanto, estaremos nos referindo a esta tecnologia quando, ao longo deste trabalho, mencionarmos o funcionamento, a verificação ou a construção de circuitos integrados.

Como não é nosso objetivo detalhar a construção de um CI, mas somente os procedimentos de verificação de projetos de circuitos integrados, iremos apresentar apenas uma breve descrição das tarefas de um projeto. Maiores detalhes sobre este assunto podem ser encontrados em MEAD [9].

Basicamente, os circuitos são compostos de transistores interligados, e cada transistor funciona como se fosse uma chave com três pinos denominados PORTA (gate), FONTE (source) e DRENO (drain). Quando um sinal lógico "1" é aplicado à porta, existe uma passagem de corrente entre a fonte e o dreno. Neste

caso, diz-se que o transistor está fechado. Caso contrário, não há passagem de corrente e diz-se que o transistor está aberto. No nosso trabalho, os pinos fonte e dreno são intercambiáveis, sendo indiferente qual deles é o fonte e qual deles é o dreno.

O projeto de um circuito integrado consiste em se especificar as máscaras que serão utilizadas pelo laboratório responsável pela fabricação da pastilha. Cada máscara irá definir a geometria de uma das camadas do circuito. Existem cinco camadas onde são formados os transistores e suas interligações, e uma sexta camada, a cobertura, que tem por finalidade proteger o circuito do meio externo. O problema, então, é descrever as máscaras dessas cinco camadas, que são conhecidas como difusão, polisilício, metal, implante e corte. A camada de cobertura não será alvo de considerações.

As camadas de difusão, polisilício e metal são condutoras e podem ser utilizadas em ligações, ou fios. Quando um fio de polisilício cruza um fio de difusão, ocorre a criação de um transistor, onde o fio de polisilício é a porta do dispositivo e o de difusão é a fonte, de um lado da interseção, e o dreno, do outro. O metal pode cruzar com fios das outras duas camadas sem que nada de especial ocorra. Porém, se desejarmos conectar um fio de difusão, ou de polisilício, a um fio de metal, poderemos fazê-lo através de um corte de contato, ou abreviadamente, corte. Também é possível conectar a extremidade de um fio de polisilício à extremidade de um fio de difusão. Usamos para isto um contato especial denominado contato de emenda. A camada de implante é usada para que as características de um transistor sejam alteradas, passando a funcionar como um resistor, chamado de transistor de depleção (depletion mode transistor). Este tipo de

transistor está sempre fechado, independente do sinal que é aplicado à sua porta.

2.2 - A METODOLOGIA MEAD&CONWAY

Esta metodologia enfatiza a importância de arquiteturas apropriadas e o efeito que elas têm no desempenho do sistema. Os principais elementos desta filosofia são: uma planta baixa do circuito definida cuidadosamente, o emprego de estruturas regulares, o perfeito encaixe das células, o escalamento das regras de projeto, modelos de temporização simples e precisos e a abolição do passo de projeto lógico do circuito.

A planta baixa é importante para que se possa avaliar arquiteturas alternativas, determinar um arranjo ótimo dos principais módulos funcionais e resolver problemas básicos de interconexão, como o suprimento de alimentação, de terra e dos sinais do relógio ao longo do circuito. A metodologia considera a fiação e a interconexão dos módulos como problemas básicos, recomendando este planejamento antes do trabalho de implementação dos componentes, a fim de evitar mudanças na arquitetura para sanar problemas geométricos ou de desempenho.

As estruturas regulares são importantes por serem o principal fator na redução do trabalho de projeto, diminuindo o número de elementos a serem detalhados e por permitirem um esquema de testes mais simples, uma vez que torna mais fácil a compreensão do circuito. Blocos complexos podem ser construídos através da repetição de estruturas regulares.

A técnica de encaixe de células simplifica o projeto, pois elimina o trabalho de ligação de uma célula às suas vizi

nhas, permitindo melhor aproveitamento do espaço e levando a construções de melhor desempenho, ao reduzir o comprimento das interconexões.

As regras de projeto são impostas pelos laboratórios de fabricação de semicondutores e, basicamente, referem-se às resoluções dos processos utilizados. A metodologia MEAD&CONWAY cria uma unidade, denominada lambda (λ) que nada mais é que a resolução do processo. Todas as regras geométricas são descritas em termos dessa unidade, que pode variar de laboratório para laboratório, permitindo que o circuito possa sofrer mudanças de escala somente com a alteração do valor de lambda.

Durante a fase de projeto, precisa-se de um modelo de temporização que possa fornecer, de forma precisa, o desempenho de várias alternativas topológicas. Esta metodologia utiliza o modelo tau (τ), que reconhece ser a velocidade de propagação em um nó dependente de sua capacitância e do tempo de transição do transistor que o alimenta. Dessa forma, define-se tau como sendo o tempo necessário para descarga de um nó com um mínimo de capacitância. Qualquer outro tempo é proporcional a tau, que pode ser obtido através de simulação ou de simples medição.

O último elemento da metodologia MEAD&CONWAY é a implementação direta de funções de alto nível, eliminando a tradicional fase de projeto lógico. As funções são descritas através de diagramas de barras, que são fáceis de serem desenhados e simples de serem convertidos em trechos do circuito.

2.3 - CIF - UM FORMATO DE DESCRIÇÃO DE CIRCUITOS

O projeto de construção de um circuito integrado é,

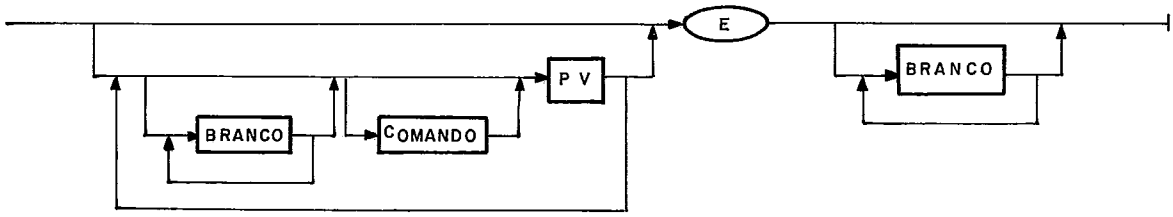
na realidade, um trabalho de especificação da geometria do circuito. É desejável ser possível especificar geometrias de maneira independente dos processos de fabricação e de projeto do circuito. O formato intermediário da Caltech (CIF) é uma linguagem gráfica de baixo nível que atende a essa necessidade, fornecendo uma interface padrão entre projetistas e laboratórios de fabricação de CI's. O emprego da linguagem CIF, como um padrão, oferece as seguintes vantagens:

a) a descrição do circuito é fácil de ser gerada e processada;
b) a linguagem é bem definida, possuindo uma gramática não ambígua;
c) o circuito é descrito de forma compacta, devido à estrutura hierárquica da linguagem;
d) o circuito é guardado na forma de um texto, podendo ser facilmente transportado de uma máquina para outra;
e) o texto é legível tanto por máquinas como por pessoas;
f) permite a criação e o intercâmbio de bibliotecas de células;
g) não depende do processo usado para a fabricação da pastilha.

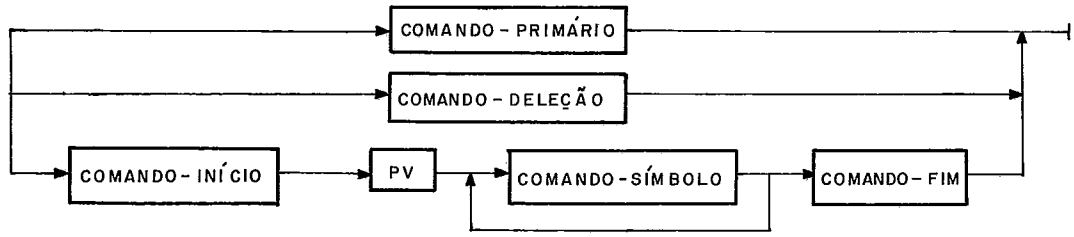
A sintaxe da linguagem CIF é apresentada na forma de diagramas sintáticos na figura (II.1). Alguns detalhes a respeito de seus comandos serão comentados, embora não seja nosso objetivo estudar com profundidade o assunto, que pode ser visto em HON [7].

Como pode ser notado na sintaxe da linguagem, exis

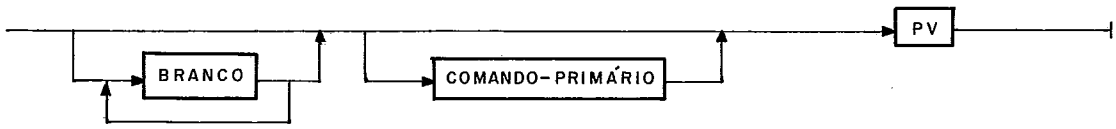
PROGRAMA ::=



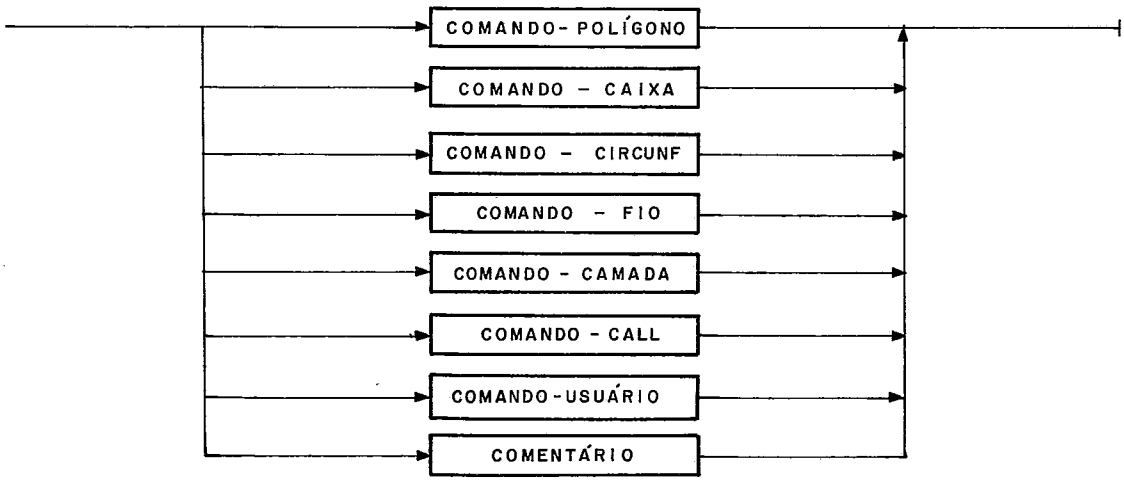
COMANDO ::=



COMANDO - SÍMBOLO ::=

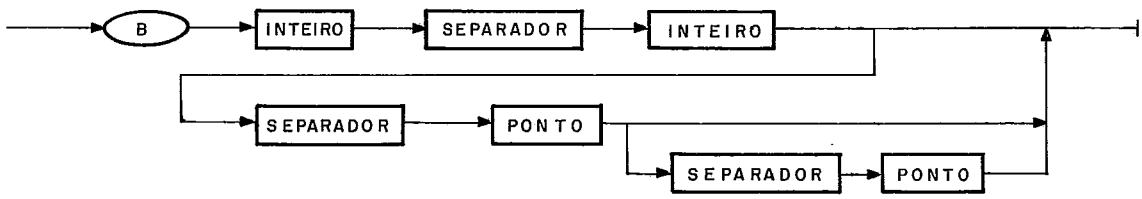


COMANDO - PRIMÁRIO ::=

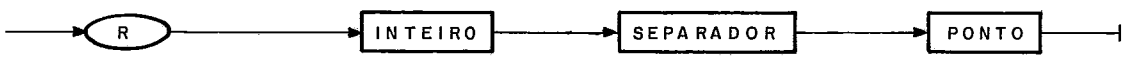


COMANDO - POLÍGONO ::=

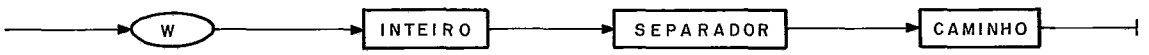
COMANDO - CAIXA ::=



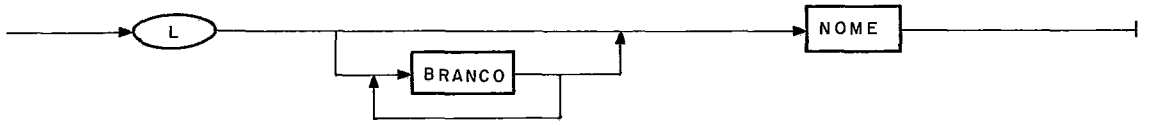
COMANDO - CIRCUNF ::=



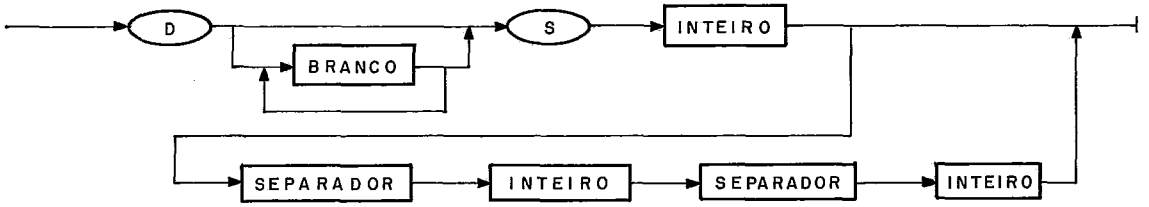
COMANDO- FIO ::=



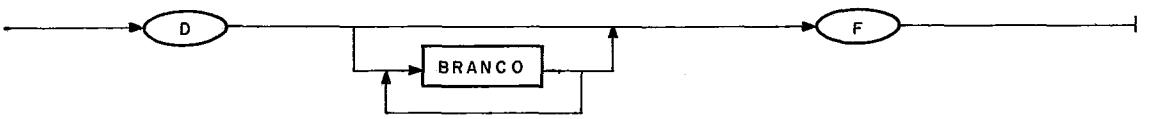
COMANDO- CAMADA ::=



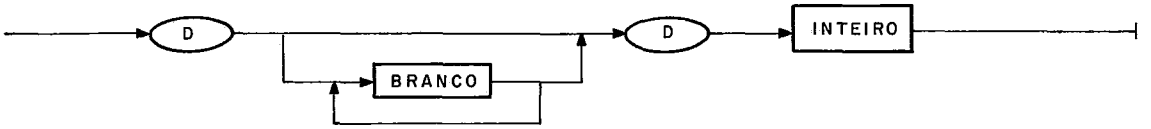
COMANDO- INÍCIO ::=



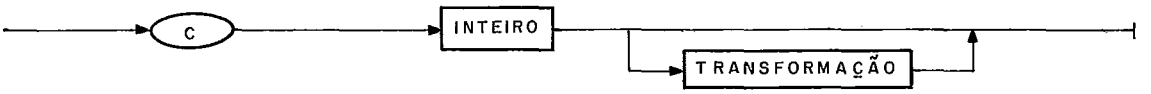
COMANDO- FIM ::=



COMANDO- DELEÇÃO ::=



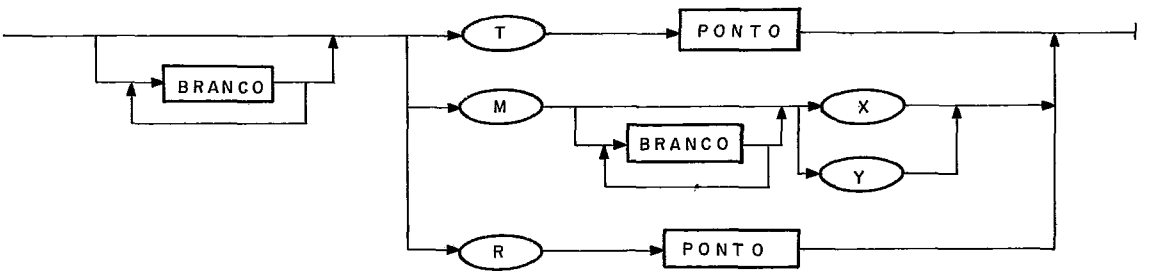
COMANDO- CALL ::=



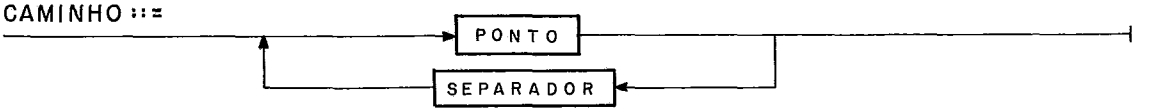
COMANDO- USUÁRIO ::=

COMENTÁRIO ::=

TRANSFORMAÇÃO ::=

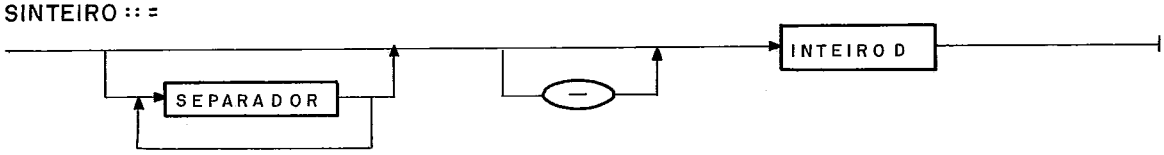


CAMINHO ::=



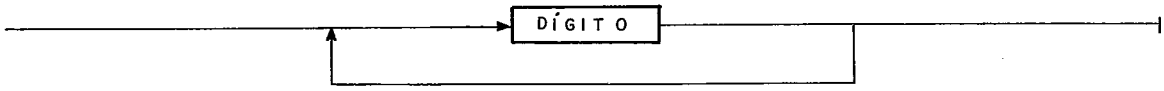
PONTO ::=

SINTEIRO ::=

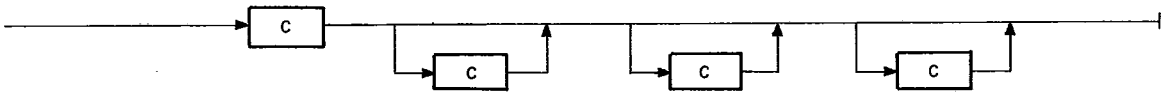


INTEIRO ::=

INTEIROD ::=

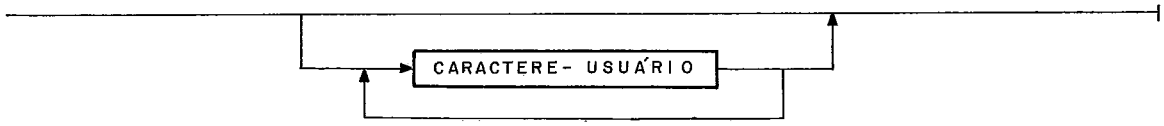


NOME ::=

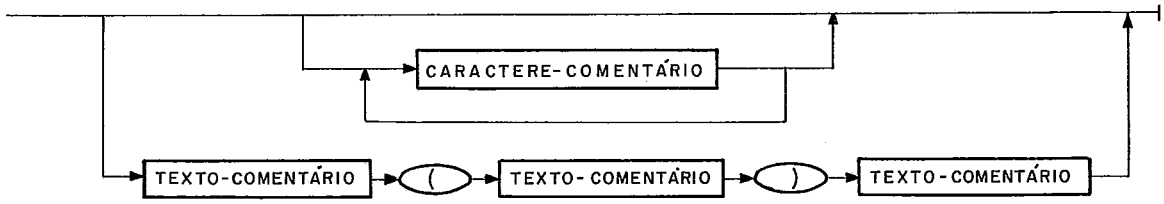


C ::=

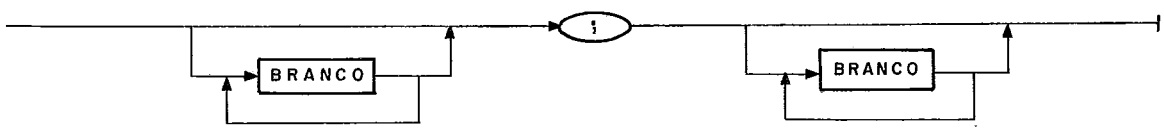
TEXTO- USUÁRIO ::=



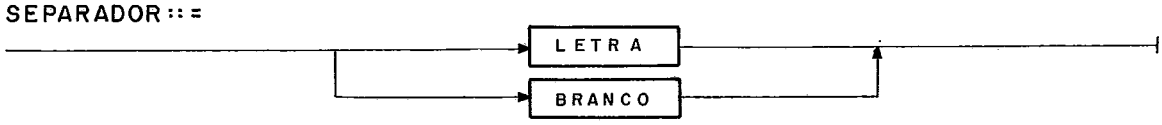
TEXTO- COMENTÁRIO ::=



PV ::=



SEPARADOR ::=



DÍGITO ::=

QUALQUER ALGARISMO ENTRE "0" E "9"

LETRA ::=

QUALQUER LETRA DO ALFABETO, INCLUINDO "K", "W" E "Y"

BRANCO ::=

QUALQUER CARACTERE ASCII, EXCETO "DIGIT", "LETRA", "-", "(",")" E ";"

CARACTERE-USUÁRIO ::=

QUALQUER CARACTERE ASCII, EXCETO " ;"

CARACTERE-COMENTÁRIO ::=

QUALQUER CARACTERE ASCII, EXETO "(" E ")"

FIGURA II.1 - GRAMÁTICA DA LINGUAGEM CIF

tem quatro comandos de descrição de figuras geométricas, que são os comandos de polígonos, caixas, círculos e fios.

No comando de descrição de polígonos são especificados os vértices dessas figuras. Assim sendo, um triângulo cujos vértices são representados pelos pontos (0,0), (0,10) e (15,10) é descrito através do comando

P 0,0 0,10 15,0

e possui a forma ilustrada na figura (II.2).

No comando de descrição de caixa são especificados o ponto central da caixa, ou retângulo, a largura e o comprimento do quadrilátero, além de uma possível inclinação. O comando

B 10 15 20,20 - 10,10

gera um retângulo como o mostrado pela figura (II.3).

Os círculos são especificados através das coordenadas de seus centros e de seus diâmetros. Já os fios, são descritos através de segmentos de retas e de suas larguras. O comando

W 5 0,0 10,0 10,15 20,15

produz o corpo geométrico indicado na figura (II.4).

O comando de camada serve para indicar em qual camada o projetista está trabalhando. Para isso, a interpretação do texto em CIF deve possuir o conceito de camada atual, que pode ser modificada por um comando de camada. Os nomes das camadas obedecem a um padrão onde a tecnologia empregada e um identificador de camada devem aparecer. Em NMOS, existem os seguintes nomes de camadas:

ND - difusão;

NP - polisilício;

NM - metal;

NI - implante;

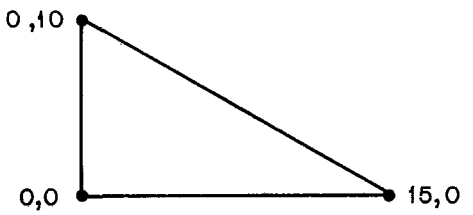


FIGURA II.2 - EXEMPLO DE UM POLÍGONO

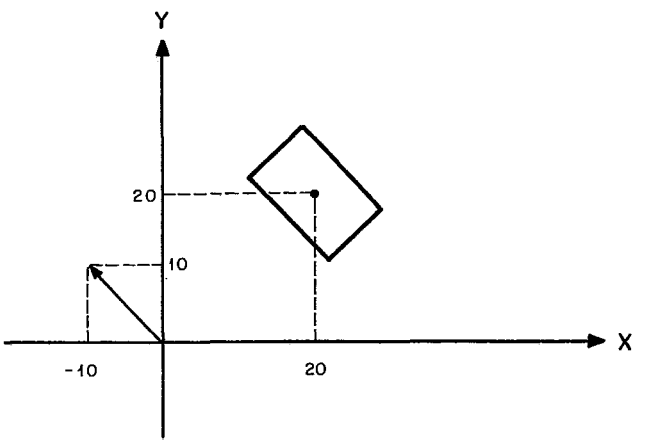


FIGURA II.3 - EXEMPLO DE UMA CAIXA

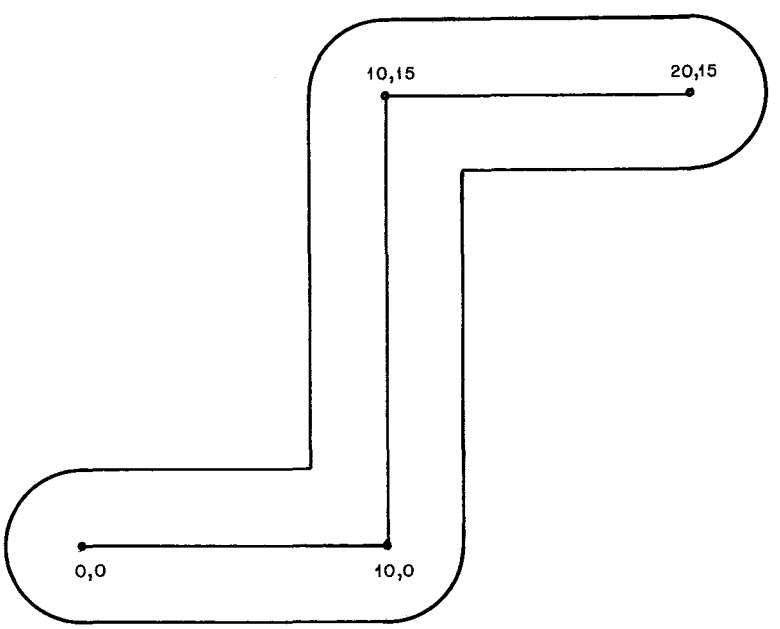


FIGURA II.4 - EXEMPLO DE UM FIO

NC - corte de contato;

NG - janela de cobertura.

Novos nomes de camadas podem ser introduzidos, para especificar outras tecnologias, sem que incompatibilidades sejam introduzidas no formato.

A linguagem CIF permite o uso de comentários. Um texto entre parênteses é encarado como tal e pode estar localizado em qualquer ponto do programa.

Como já foi mencionado anteriormente, a descrição de um circuito pode ser feita de maneira hierárquica. Isto é conseguido com definições e expansões de símbolos, que são equivalentes às "macros" de algumas linguagens de programação. Uma vez definidos, esses símbolos podem ser instanciados. Mais do que isto, pode-se nas suas expansões promover rotações, translações ou espelhamentos. A definição de um símbolo é feita através de um texto delimitado pelos comandos de início e fim de definição de símbolos. No comando de início de definição aparece um rótulo, que vai identificar o símbolo, e um fator de escala, pelo qual serão multiplicados todos os valores dimensionais dele. No interior de uma definição não pode ocorrer a definição de um outro símbolo. Além disso, dois símbolos não podem se referenciar mutuamente. A figura (II.5) ilustra a definição de um símbolo rotulado pelo número 42.

Este símbolo deverá mais tarde ser expandido através de um comando CALL. Na figura (II.6) são mostrados dois comandos de expansão (call) e, graficamente, o efeito que eles produzem quando aplicados ao símbolo 42 definido anteriormente. Note as translações impostas pelas chamadas.

```
DS   42  100  1 ;
(SÍMBOLO # 42. TRANSISTOR  4 x 4)
L N D ;
B  12  4  6,6 ;
L N P ;
B  4  12  8,6
DF ;
```

FIGURA II.5 - EXEMPLO DA DEFINIÇÃO DE UM SÍMBOLO

```
C   42   T  2,4
C   42   T  2,16
```

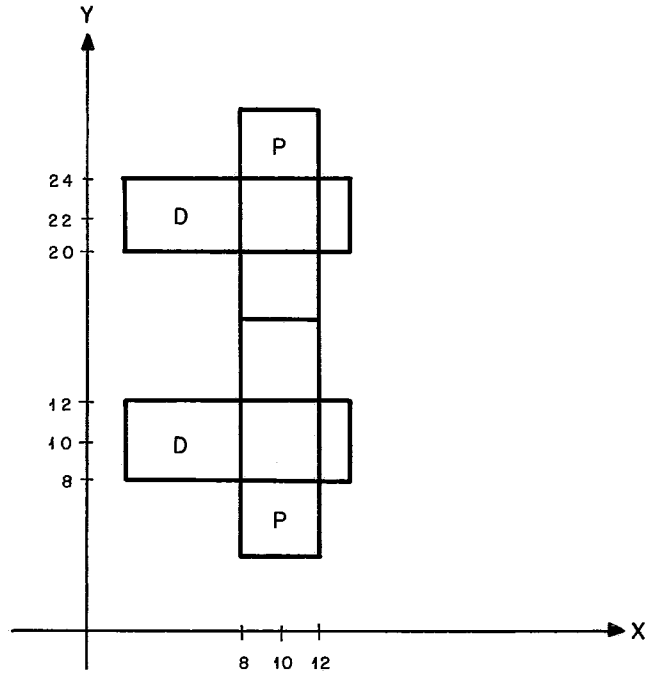


FIGURA II.6 - EXEMPLO DE CHAMADAS DE SÍMBOLOS

Além dos comandos de início e fim de definição de símbolos, existe um comando responsável pela deleção de um símbolo, tornando-o não mais disponível a partir desse ponto. O comando DD 42, iria inibir futuras expansões do símbolo número 42.

Os comandos de usuário representam uma forma de criação de comandos com significado e uso próprios para uma determinada instalação. No NCE, a extensão de usuário número 0 é utilizada para dar nomes a determinados pontos do circuito. Desta forma, os comandos de usuário devem ser ignorados quando se tratar de um circuito vindo de outra instalação.

Apesar da linguagem CIF permitir a construção de grande variedade de figuras geométricas, é comum utilizar-se preferencialmente caixas em detrimento dos demais corpos. Além disso, as caixas são dispostas horizontal ou verticalmente, numa geometria ortogonal. Esta última limitação é conhecida como padrão MANHATTAN.

2.4 - FERRAMENTAS DE APOIO A PROJETOS DE CI's

Um projeto de circuito integrado consiste em se especificar um texto em CIF que descreva a geometria desejada. Esta tarefa, porém, envolve vários passos, pois é praticamente impossível escrever um texto grande nesta linguagem sem que se visualize a geometria do circuito, assim como, garantir que o texto gerado esteja correto. As ferramentas de apoio a projetos de CI's têm por finalidade validar modelos, facilitar a confecção de textos em CIF, verificar a forma dos textos e fazer algumas análises da geometria, da lógica e das características elétricas dos circuitos. Algumas ferramentas mais utilizadas serão apresen

tadas.

A simulação funcional deve ser empregada na fase inicial de um projeto. Uma vez definidos os blocos lógicos que compõem o circuito, é interessante verificar a interconexão entre esses blocos e analisar os atrasos por eles causados. Esta simulação, de alto nível, irá detectar erros cujos acertos geralmente implicam em grandes alterações no circuito. Um exemplo de um simulador funcional é o programa SIMF, desenvolvido para auxiliar projetos no NCE.

A confecção de um texto em CIF é uma tarefa árdua e sujeita a erros. Ela pode, no entanto, ser atenuada com o auxílio de um editor gráfico, onde o projetista fornece uma representação geométrica do circuito e o editor é capaz de converter o desenho num texto CIF equivalente. A grosso modo, os editores podem ser divididos em duas categorias: os editores de células e os de circuitos.

Um editor é dito de célula se ele não é capaz de manipular grandes áreas do circuito. Trabalhar com um editor de células envolve dividir o circuito em pequenos trechos, chamados de células, e editar cada trecho individualmente. Na prática, essa divisão em pequenos pedaços é desejável, uma vez que é frequente a repetição de certos trechos em um circuito. São exemplos de editores de células o programa CIFSYM, de autoria do professor Daniel W. Lewis da Universidade de Santa Clara (EUA), e o editor EDMOS, desenvolvido no NCE/UFRJ pelo colega José Antonio dos Santos Borges. Nestes editores são encontrados comandos que permitem a leitura ou a gravação de células, facilidades de edição gráfica, como desenhar ou apagar seções de uma camada e impres

são da imagem rastreada da célula em edição.

Um editor de circuitos é uma ferramenta bem mais sofisticada, operando sobre um banco de dados que contém células do circuito em edição. Este tipo de programa tem a capacidade de materializar uma célula em um certo local do circuito e possui facilidades de construir as ligações entre as diversas células. Também é desejável que um editor desta categoria possua comandos que permitam a rotação e o espelhamento de células, quando da materialização delas.

Completando o pacote de edição, existem programas que têm por finalidade desenhar o circuito obtido em equipamentos gráficos como vídeos, impressoras ou plotadoras. No NCE foram desenvolvidos programas com esta finalidade, utilizando como periféricos de saída um vídeo gráfico colorido e uma plotadora autônoma.

Uma vez construída uma célula, ou um circuito, deve-se passar à fase de verificação do mesmo. Existem programas responsáveis pela realização de testes sobre a geometria, sobre a lógica e sobre as características elétricas de um circuito.

O verificador de regras de projeto é utilizado para testar a geometria de um circuito. Existem algumas regras que limitam a largura mínima de um fio e a separação entre dois fios em uma determinada camada. Além disso, existem regras que definem geometricamente a formação de transistores e cortes de contato. A violação dessas regras pode causar a não condutibilidade, curtos circuitos, induções de corrente ou o aparecimento de transistores que não funcionam corretamente.

Um circuito isento de erros geométricos deve ser fornecido a um verificador estático. Pode-se aqui traçar uma analo

gia entre estas ferramentas e um compilador de uma linguagem de programação. O verificador de regras desempenha uma função comparável à de um analisador sintático enquanto que o verificador estático se assemelha à fase de análise estática da semântica. Para um verificador estático, existem dois tipos de dados: os transistores e os nós, ou fios, de um circuito. Ele realiza testes como a formação lógica de transistores e verifica a conectividade dos nós, assinalando possíveis curtos circuitos ou falta de alimentação em determinados nós. O verificador estático opera sobre o chamado circuito extraído, ou seja, um arquivo que possui informações sintéticas sobre os transistores e os nós. O simulador funcional, o verificador de regras de projeto, o extrator de circuitos e o verificador estático serão motivo de um estudo mais aprofundado nesta tese.

Outro aspecto importante nesta fase é a simulação lógica e elétrica do circuito. Ambas utilizam o circuito extraído. Para a simulação lógica, um transistor não passa de uma chave que pode ou não permitir a passagem de corrente entre seus extremos. Ele é utilizado unicamente para verificar o comportamento lógico do circuito. São exemplos desta categoria de simulação os programas MOSSIM e RSIM.

O simulador elétrico utiliza informações sobre as características elétricas dos transistores, considerando atrasos e a capacidade destes dispositivos de poderem alimentar outros transistores. O mais conhecido simulador elétrico é o programa SPICE.

Uma vez editado, verificado e simulado um circuito, podemos enviar o seu texto CIF a um laboratório para a confecção da pastilha que, ao retornar, necessita ainda ser testada. Para

facilitar o trabalho de teste do circuito, existe a possibilidade de se utilizar equipamentos específicos para testes de CI's. O programa que fizer a interface do testador com o projetista deve ser capaz de emular um simulador, permitindo que os resultados aí obtidos possam ser comparados com as informações fornecidas pela simulação.

2.5 - A ESTAÇÃO DO NCE DE APOIO A PROJETOS

A atuação do Núcleo de Computação Eletrônica da UFRJ na área de confecção de circuitos integrados iniciou-se em fins de 1981. Com o objetivo de dominar a tecnologia de projetos de CI's, deu-se início a um trabalho visando desenvolver as seguintes atividades:

a) criação de um laboratório de projeto de sistemas digitais em VLSI;
b) formação de recursos humanos;
c) desenvolvimento completo de projetos em VLSI.

Para que o objetivo fosse cumprido, duas linhas de trabalho foram ativadas. Uma responsável pela construção de um conjunto mínimo de ferramentas de apoio e a outra, pelo desenvolvimento de um projeto piloto, que é a integração da lógica de acesso à rede local do NCE.

Atualmente, a estação de apoio a projetos de CI's utiliza os seguintes equipamentos:

a) minicomputador PDP 11/70 da DEC;
b) microcomputador SDE-40 da EBC;
c) vídeo gráfico colorido, desenvolvido pelo NCE;
d) plotadora autônoma CALCOMP.

Quanto ao conjunto de programas utilizados na estação destacam-se:

a) editor CIFSYM;

b) gerador automático de PLA's;

c) plotador de células CIFPLOT;

d) editor EDMOS, que utiliza o vídeo gráfico;

e) gerador de mapa de pixel's;

f) verificador de regras de projeto;

g) traçador de erros geométricos;

h) extrator de circuitos;

i) verificador estático;

j) simulador funcional;

ℓ) simulador lógico (em conclusão).

Na figura (II.7) é apresentado um fluxo que ilustra a utilização das nossas ferramentas de edição e de verificação durante um projeto. Além dos programas, pode-se notar as várias formas de se representar um mesmo circuito.

Maiores detalhes do projeto de integração da lógica de acesso à rede local podem ser vistos em SILVA[11] .

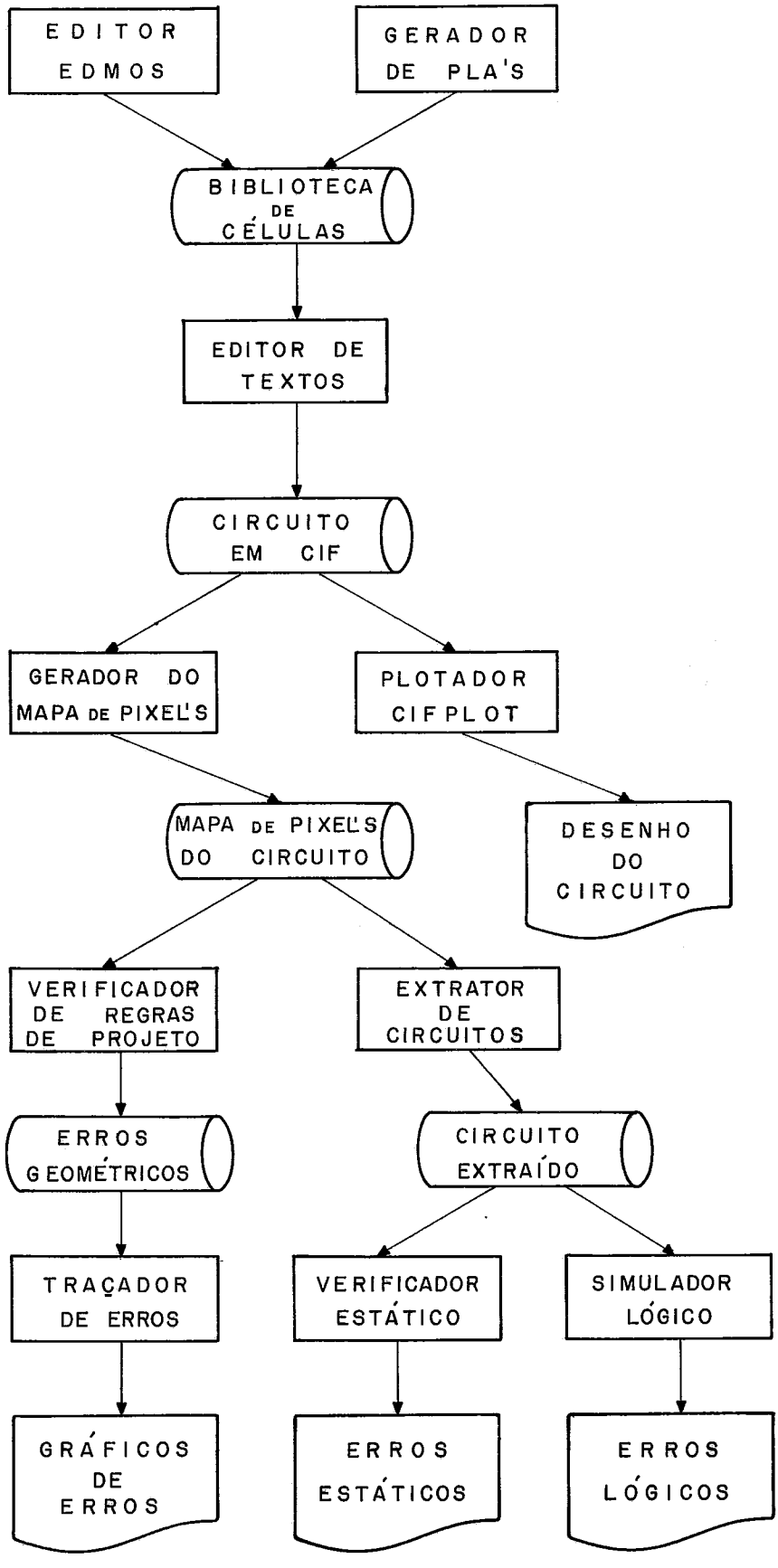


FIGURA II.7- FLUXO DA EDIÇÃO E VERIFICAÇÃO DE UM CIRCUITO

III - SIMULADOR FUNCIONAL

3.1 - APRESENTAÇÃO

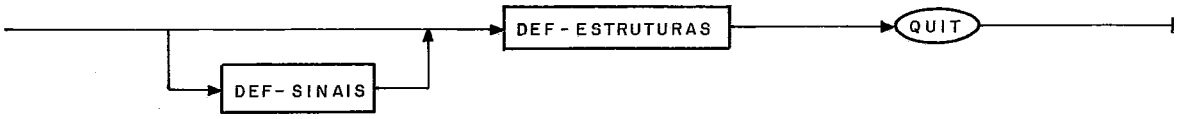
A simulação funcional serve para validar um projeto a nível dos blocos lógicos que o formam. Através dela pode-se simular a interação entre os blocos e analisar, do ponto de vista lógico, o comportamento do circuito. O simulador funcional que desenvolvemos recebe como entrada uma descrição do modelo a ser simulado, escrito em uma linguagem criada para este propósito. Essa descrição é compilada e um código a ser interpretado pelo simulador é criado. O simulador é conversacional, aceitando comandos que possuem sintaxe semelhante aos comandos do simulador lógico MOSSIM, que é descrito em BRYANT[4] e BRYANT[5].

3.2 - A DEFINIÇÃO DO CIRCUITO

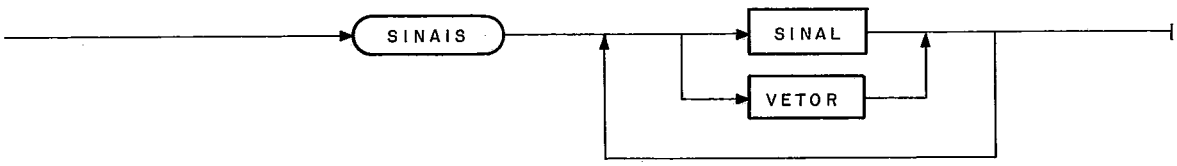
O circuito, descrito na linguagem de definição de circuitos, é composto por dois tipos de estruturas: as matrizes de lógica programada, PLA's, onde a lógica de controle do circuito é descrita e os registros, onde são armazenadas informações. Existe na linguagem a possibilidade de definição de sinais. Um sinal pode ser declarado explicitamente, introduzido pela declaração SINAIS, ou implicitamente, ao ser referenciado pela primeira vez no texto. A declaração SINAIS permite a introdução de vetores de sinais o que, como veremos adiante, facilita as operações de carga e leitura de registros.

Podemos analisar a gramática da linguagem de definição de circuitos, que é mostrada na forma de diagramas sintáticos na figura (III.1). A declaração SINAIS define sinais e veto

CIRCUITO ::=

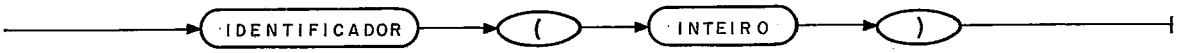


DEF-SINAIS ::=

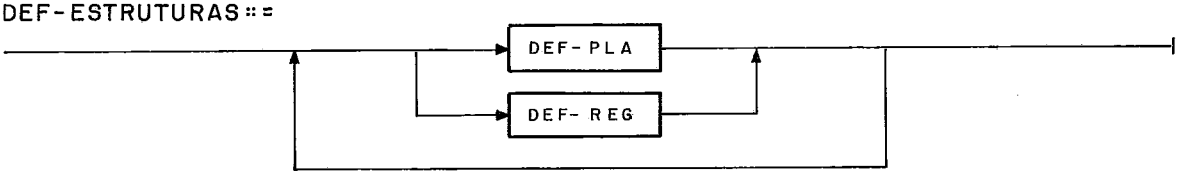


SINAL ::=

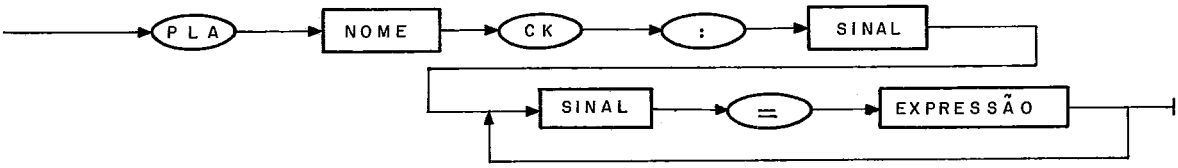
VETOR ::=



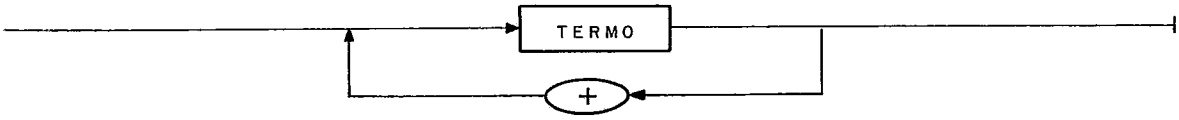
DEF-ESTRUTURAS ::=



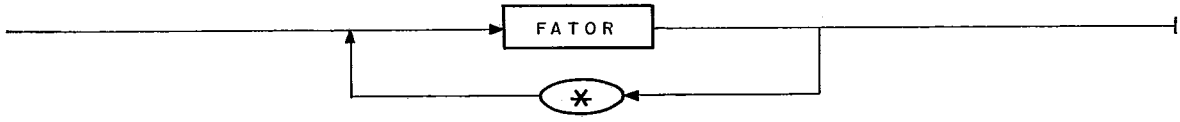
DEF-PLA ::=



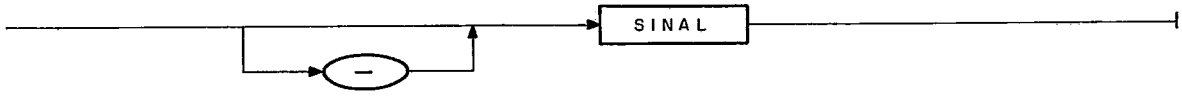
EXPRESSÃO ::=



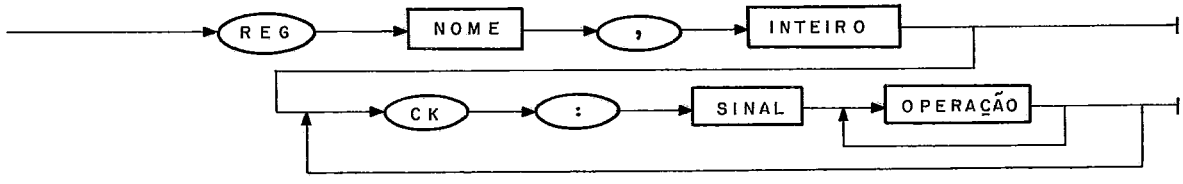
TERMO ::=



FATOR ::=



DEF-REG ::=



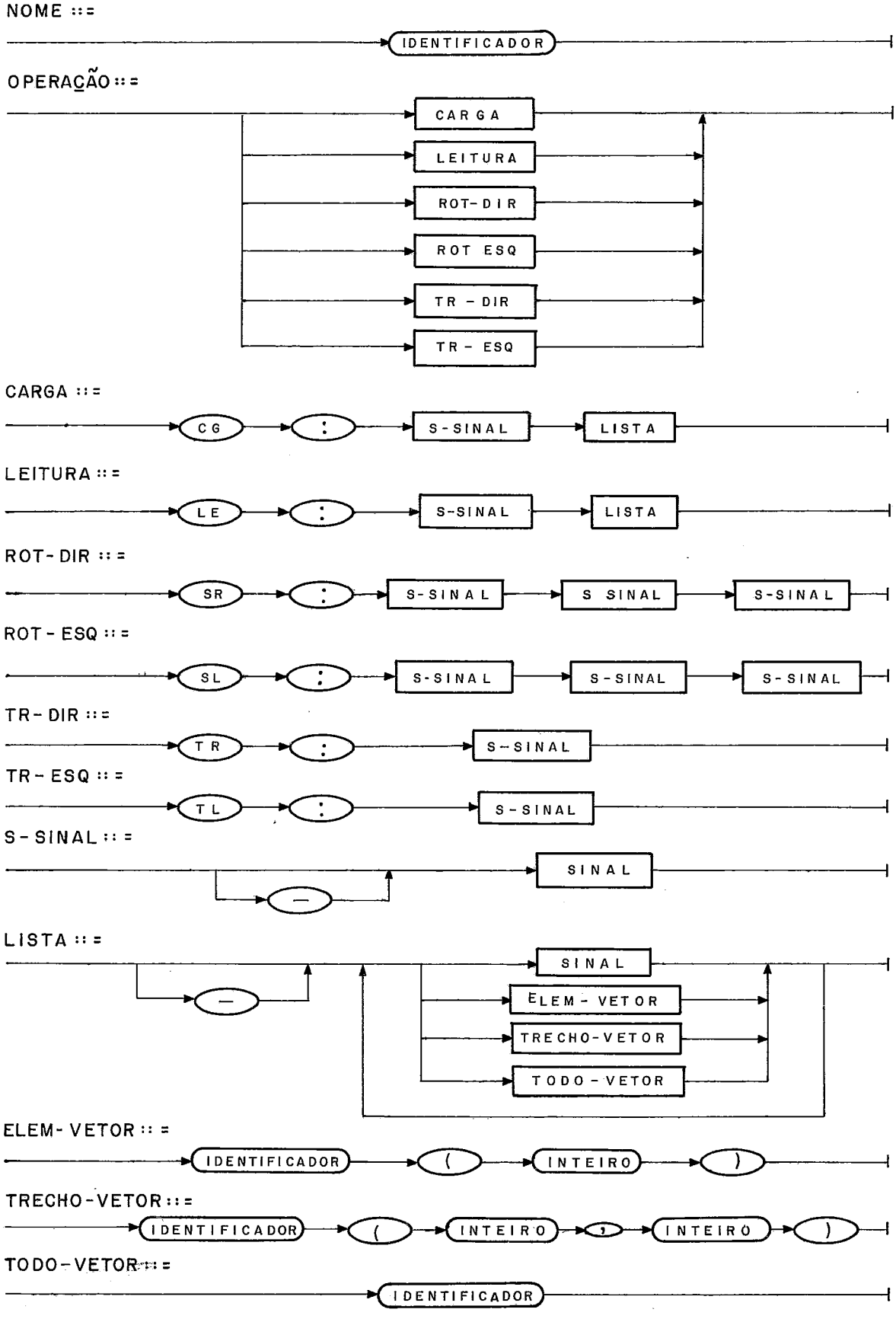


FIGURA III.1 - GRAMÁTICA DA LINGUAGEM DE DEFINIÇÃO DE CIRCUITOS

res de sinais, onde a dimensão do vetor é especificada. A posição 3 de um vetor X pode ser referenciada como X(3) ou como X3, uma vez que a análise cria N sinais quando da definição de um vetor de tamanho N. Os sinais simples, ou não vetorados, não necessitam ser declarados, sendo automaticamente introduzidos quando da primeira referência a cada um deles.

As PLA's agem como elementos de controle e lá são gerados os valores de sinais através de expressões lógicas. O sinal "+" denota o operador "ou", o "*" representa o "e" e o sinal "-" refere-se à operação complemento. Como de costume, os dois primeiros operadores são binários e o último, unário. Como também é o normalmente usado, o complemento tem maior prioridade que a interseção, que é a mais prioritária que a união. A compilação de um PLA irá gerar um código em que as expressões são representadas através de notações polonesas. A sequência CK: < sinal > irá denotar, tanto nas PLA's quanto nos registros, que o trecho que segue só será executado se o sinal, no momento da simulação, estiver ativo. Desta forma é implementado um esquema de fases no simulador.

Um registro é modelado segundo o padrão de registro completo da tecnologia NMOS. Sobre ele são válidas as operações de carga, leitura, rotação à esquerda e rotação à direita. Um registro, nesta tecnologia, é implementado como um duplo "buffer", sendo necessárias duas operações de ação interna, onde cada uma delas copia de um "buffer" para o outro. Nas figuras (III.2) e (III.3) apresentamos a forma esquemática e o modelo de um registro por nós usado. As transferências internas estão representadas pelas operações TRR e TRL. As rotações por SHR e SHL. A carga por CG e a leitura por LE.

Em uma operação de carga, ou leitura, devem ser espe

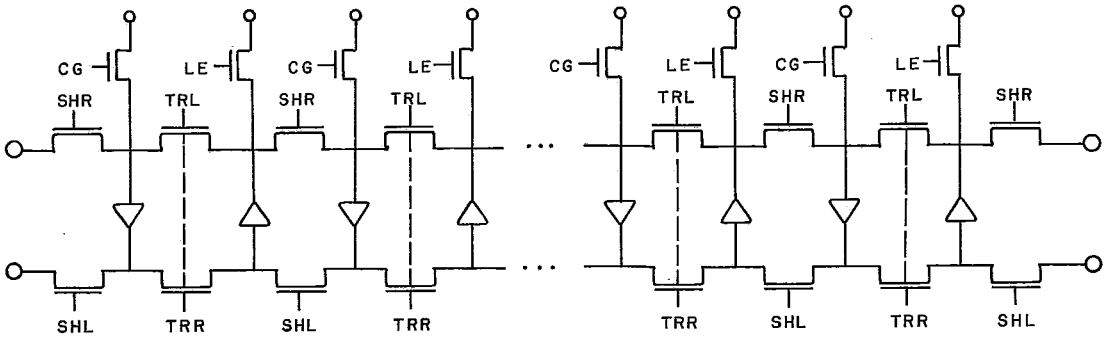


FIGURA III.2- FORMA ESQUEMÁTICA DE UM REGISTRO

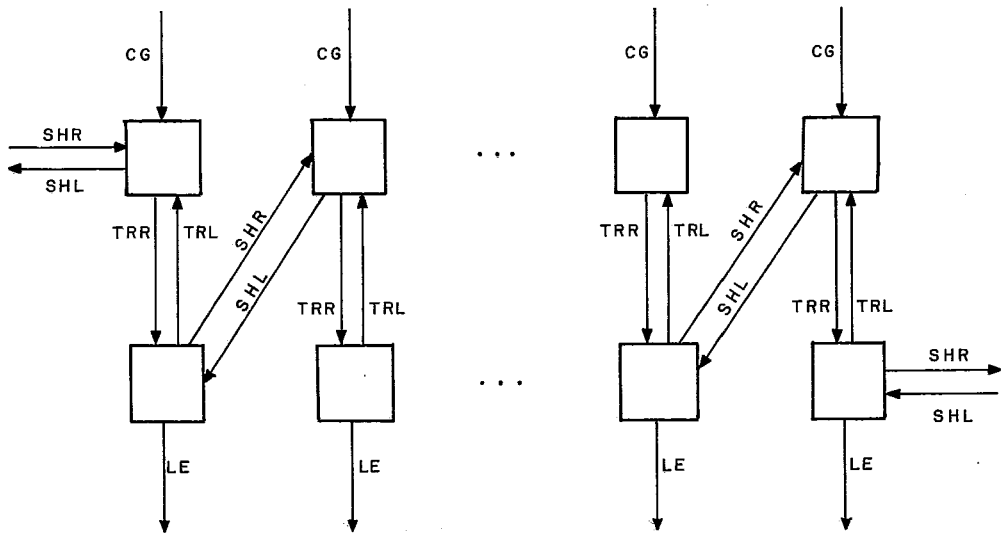


FIGURA III.3 - MODELO DE UM REGISTRO NO SIMULADOR FUNCIONAL

```
REG   R,8
   CK :  FASE 1
      CG:  CARGA, BARRA(0-7)
      SR:  DIREITA, ENTRA, SAI
      TL:  TRESQ
   CK :  FASE 2
      LE:  LEITURA, BARRA(0-7)
      SL:  ESQUERDA, SAI, ENTRA
      TR:  TRDIR
```

FIGURA III.4- EXEMPLO DA DEFINIÇÃO DE UM REGISTRO

cificados um sinal comandando a ação e as entradas, ou saídas, paralelas, uma para cada bit. Como pode ocorrer de se desejar carregar, ou ler, apenas alguns bits do registro, é permitido usar um 0 (zero) no lugar do nome do sinal correspondente a sua entrada, ou saída, indicando que aquele bit não deve ser carregado, ou lido. É aqui permitido o emprego de vetores, ou de trechos de vetores de sinais, para simplificar a tarefa de descrição.

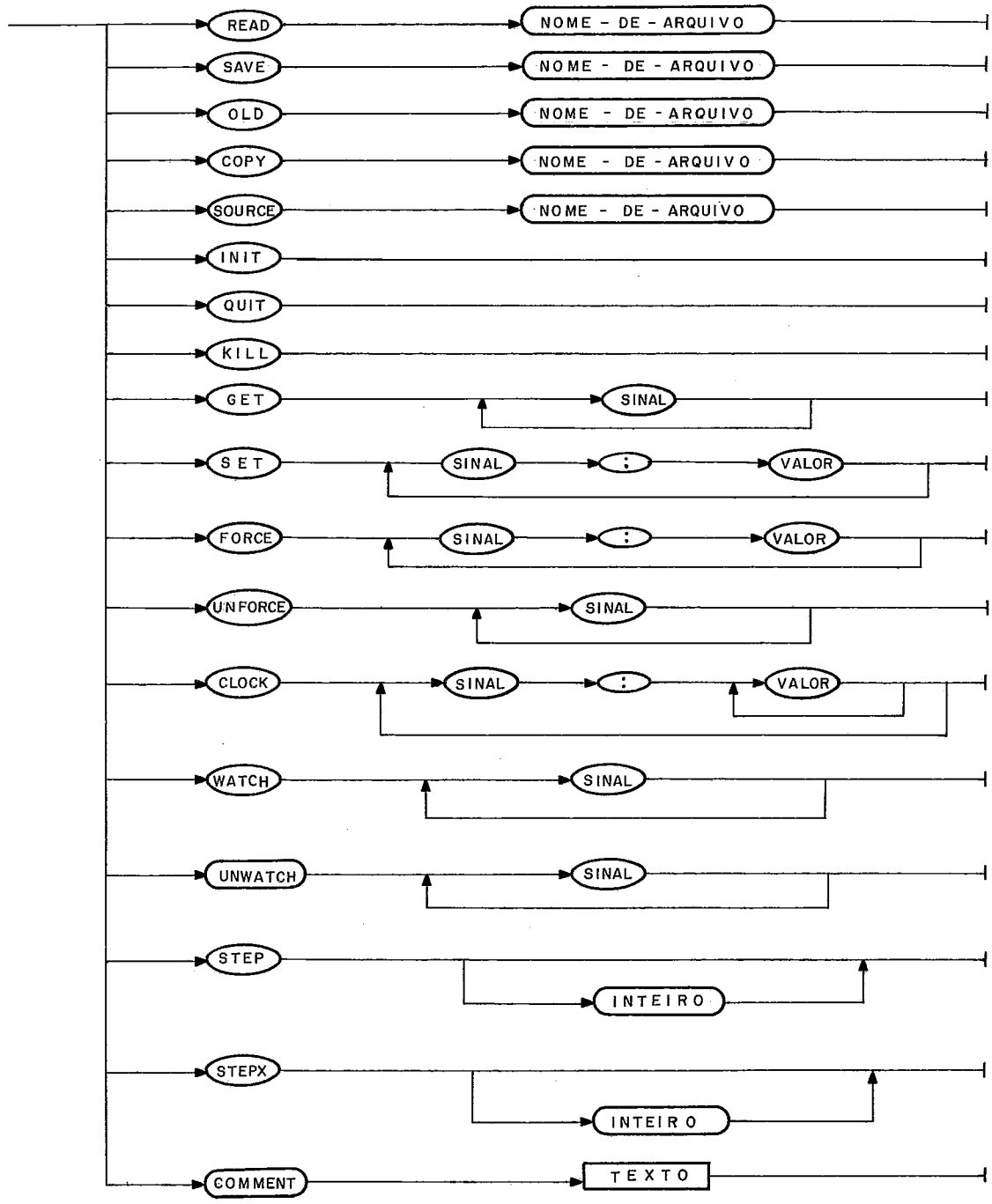
Em uma operação de rotação deve-se fornecer o nome de um sinal que comande o deslocamento seguido de uma entrada e de uma saída serial do registro para essa operação. Já nas transferências internas, apenas os nomes dos sinais que as qualificam devem ser mencionados, uma vez que a ação é bem definida. A figura (III.4) ilustra a descrição de um registro de 8 bits, onde todas as operações possíveis são utilizadas.

3.3 - OS COMANDOS DE SIMULAÇÃO

A manipulação do simulador funcional é feita através de um terminal. Os comandos da simulação são quase idênticos aos do simulador MOSSIM, que está sendo programado no NCE, de modo que poupa ao usuário o aprendizado de mais uma linguagem. Do MOSSIM não foram utilizados os comandos DEFINE, FIND, FLUSH, CYCLE, PHASE e SWITCH. Ao conjunto original, foram adicionados três comandos: OLD, SAVE e STEPX, este último uma variação do comando STEP. A sintaxe dos comandos é mostrada na figura (III.5) e o funcionamento de cada comando é mostrado a seguir.

O comando READ < arquivo > causa a leitura e a compilação do arquivo que contém a definição do circuito. O código gerado fica na memória.

COMANDO : : =



TEXTO : : =

QUALQUER SEQUÊNCIA DE CARACTERES ATÉ O FIM DA LINHA

FIGURA III.5 - GRAMÁTICA DA LINGUAGEM DE COMANDOS

O comando SAVE < arquivo > faz com que sejam copiados para um arquivo o código resultado da compilação, as tabelas do simulador e algumas variáveis por ele usadas. Poderíamos chamar esta operação de salvamento do contexto.

O comando OLD < arquivo > executa procedimento contrário ao que é realizado pelo comando SAVE,restaurando o contexto.

O comando COPY < arquivo > indica ao simulador que se deseja produzir, a partir desse ponto, uma auditoria da simulação. No arquivo são gravados todos os comandos que vierem a ser fornecidos, bem como todas as saídas produzidas pelo simulador, que continuam sendo enviadas ao vídeo do terminal.

O comando SOURCE < arquivo > permite que um certo trecho de comandos possa ser obtido de um arquivo em disco, possibilitando uma indireção de comandos ao simulador. Procedimentos de inicialização e testes podem estar contidos em arquivos independentes, evitando que o usuário tenha que os teclar em simulações distintas.

O comando INIT causa uma reinicialização da simulação, destruindo todas as tabelas e atribuindo "0" a todos os sinais. O código do circuito permanece inalterado.

Os comandos QUIT e KILL têm por função encerrar a simulação. Se, no entanto, um QUIT for encontrado em um arquivo de comandos ativado por um SOURCE, a ação tomada será a de retorno ao terminal para recebimento de novos comandos.

O comando GET < sinal > ... permite que se obtenha os valores dos sinais nele especificados.

O comando SET < sinal > : < valor > ... permite que se altere estados de sinais.

O comando FORCE <sinal > : < valor > ... atua como um SET duradouro, inibindo futuras alterações dos sinais forçados.

O comando UNFORCE <sinal > ... faz com que os sinais lá relacionados não sejam mais forçados.

O comando CLOCK <sinal > : <sequência> ... permite que se especifique sequências de valores que os sinais irão receber, passo a passo. Os sinais não serão alterados pela simulação e as sequências serão expandidas circularmente ao longo do processo.

O comando WATCH < sinal > ... faz com que ao fim de cada passo de simulação sejam mostrados os estados correntes desses sinais.

O comando UNWATCH <sinal > ... faz com que os sinais não mais sejam mostrados ao fim de cada passo.

O comando STEP <n > faz com que N passos sejam simulados. Se N for omitido, o simulador assume que se deseja executar um passo. Ao fim de cada passo simulado são mostrados os estados dos sinais especificados por comandos WATCH.

O comando STEPX <n> é semelhante ao anterior, sendo que os sinais só são mostrados após o término do último passo.

Comentários podem ser introduzidos através da cláusula COMMENT. Todo o restante da linha não será considerado. São muito úteis quando se estiver utilizando um arquivo de comandos ou quando se desejar escrever observações no arquivo de auditoria da simulação.

3.4 - A SIMULAÇÃO

O simulador funcional opera por passos. As fases do relógio são simuladas através de comandos CLOCK's. É comum encontrarmos circuitos que funcionam com ciclos de 2 fases, não superpostas. Isto pode ser simulado com o auxílio de dois sinais. FASE1 e FASE2 por exemplo, aos quais fornecemos sequências através de CLOCK's. A sequência é de responsabilidade do usuário pois o simulador, para não ficar restrito a este tipo de abordagem, não atribui a esses sinais algum tipo especial de tratamento. Poderíamos então implementar o esquema de duas fases citado através de

CLOCK FASE1:10 FASE2:01.

A cada passo de simulação um trecho de uma PLA ou de um registro só é considerado se o estado do sinal especificado em sua cláusula CK valer "1". No esquema de duas fases não superpostas, devemos usar exclusivamente os dois sinais que representam as fases, FASE1 e FASE2 anteriores, nas cláusulas CK das estruturas. Se uma operação é não vinculada ao relógio, ela deve ocorrer tanto em FASE1 quanto em FASE2. Uma alternativa seria utilizar um outro sinal, por exemplo FASE3, e forçá-lo sempre como ativo, para controlar as operações que são independentes do relógio, que podem ser realizadas em qualquer fase.

Uma vez simulado um passo, o simulador guarda os estados dos sinais e volta a simular o passo. Este procedimento é realizado até que não ocorram mudanças nos estados dos sinais, atingindo o circuito sua estabilidade. Se o simulador não conseguir alcançar esta situação em N rodadas, onde N é o número de sinais, ele encerra a execução do passo e avisa ao usuário que o

circuito não foi estabilizado, listando os sinais que mudaram de estado na última rodada. O valor N foi determinado supondo que, no pior caso, apenas um sinal é definitivamente estabiliza so em cada avaliação.

3.5 - EXEMPLO DA SIMULAÇÃO FUNCIONAL

Apresentamos no apêndice I a simulação de um circui to serializador. Ele possui quatro ligações com o meio externo: uma barra, de 8 bits, onde o padrão é depositado, um sinal coman dado a carga da informação que está na barra, a saída serial e um sinal de estado, indicando que a saída está disponível. Para o funcionamento correto do circuito, o sinal de carga deve perma necer ativo por um ciclo completo ou, de acordo como esquema de duas fases não superpostas adotado, durante dois passos de simu lação. O estado pronto estará ativo durante uma única fase.

Tentamos neste exemplo utilizar o maior número pos sível de comandos do simulador, assim como, introduzir boa quan tidade de comentários a fim de facilitar o entendimento da simu lação. A saída que apresentamos foi obtida ao listarmos a audito ria da simulação, produzida por um comando COPY.

IV - VERIFICADOR DE REGRAS DE PROJETO

4.1 - AS REGRAS DE PROJETO

Uma das mais importantes características do atual estágio na construção de pastilhas é o fato dos projetos não serem dependentes dos processos de fabricação. Ou seja, existe uma clara separação entre o processamento realizado durante o fabrico da pastilha e o esforço de projeto necessário para a criação dos padrões a serem implementados. Esta separação exige que o projetista possua uma definição precisa das capacidades do processo utilizado pelo laboratório a que ele vai enviar as especificações do seu projeto. Essa definição abrange aspectos geométricos, como o conhecimento da resolução do processo a ser utilizado, o que obriga o projetista a cumprir determinadas normas, conhecidas como regras de projeto. Estas normas estabelecem os menores valores possíveis para larguras, separações, extensões e sobras de objetos geométricos do circuito.

Com o desenvolvimento da tecnologia de construção de pastilhas, esses valores mínimos tendem a ser cada vez menores, o que pode obrigar a um projeto ter de ser reconsiderado devido a uma mudança no processo de fabricação. Em MEAD[9], uma solução é proposta com a finalidade de sanar o problema causado pela mudança de escala em um processo. Ele cria uma unidade de cumprimento relativa denominada lambda (λ) e sugere que os projetistas desenvolvam seus circuitos utilizando esta unidade. Basicamente, lambda pode ser visto como o erro máximo de posicionamento que pode ser cometido na confecção de uma camada. Assim sendo, o maior desvio que pode ser causado entre duas camadas é dois lambdas, em relação às especificações originais.

As regras de projeto são decorrentes da própria definição de lambda. Para preservar a topologia dos elementos de uma camada, é exigida uma largura mínima de dois lambdas, valor idêntico à separação mínima entre dois fios. Estas regras, entretanto, não podem ser aplicadas a todas as camadas por razões diretamente relacionadas ao processo de fabricação.

No processo NMOS, existem três tipos de camadas condutoras: a de difusão, a de polisilício e a de metal. Além delas, existe uma máscara correspondente a aberturas de janelas de contato, ou cortes de contato. A camada de polisilício, que normalmente é a mais bem controlada, obedece às regras básicas de largura e separação de dois lambdas. Na difusão, esta regra vale para a largura mas uma separação maior é necessária para prevenir condução de corrente através de regiões de depleção (ou sangria) quando a tensão for elevada. Uma separação de três lambdas é exigida entre dois fios de difusão. A camada de metal, que das três é a última a ser posicionada, cobrindo regiões rugosas causadas pelo depósito das anteriores, é geralmente utilizada como elo da ligação entre os elementos do circuito. Seus fios devem possuir uma largura mínima de três lambdas assim como uma separação entre eles de mesma magnitude. Fios de polisilício e difusão que não se juntem para a formação de contatos ou transistores devem possuir uma separação mínima de um lambda. Embora, devido a um desalinhamento, o polisilício possa sobrepor-se à difusão, nenhum problema é causado além de um estreitamento no fio de difusão.

Quando o polisilício cruza uma linha de difusão um transistor é criado, onde a largura e o comprimento mínimo de seu canal são regidos, respectivamente, pelas larguras mínimas dos fios de difusão e polisilício que o formam. Para que um tran

sistor funcione corretamente é necessário que o polisilício cubra completamente o fio de difusão. Como o erro máximo aceitável entre duas camadas é dois lambdas, é exigido que o fio de polisilício ultrapasse a região do transistor em, pelo menos, esses dois lambdas. Analogamente, o fio de difusão deve ultrapassar em dois lambdas o transistor por ele formado.

A outra interação existente entre camadas refere-se aos contatos. Algumas regras garantem a formação de contatos perfeitos. Elementos na camada de corte especificam janelas que tornam expostos o polisilício, se ali existir, ou a difusão, se ela existir mas não o polisilício, de tal maneira que o metal entre em contato com a área de silício. Contatos são feitos usando cortes de, no mínimo, dois X dois lambdas sobre uma região de difusão, ou polisilício, que deve exceder a região do corte, assim como o metal, pelo menos um lambda em todos os sentidos. A fim de prevenir que um corte para a difusão acidentalmente provoque um contato com um fio de polisilício que passe por perto, é exigida uma separação mínima de dois lambdas entre esse fio e o corte. Os contatos vistos até agora podem ser chamados de contatos simples. Além destes, existe o contato de emenda que tem por finalidade unir um fio de difusão a um outro de polisilício. Um contato de emenda é feito removendo-se o isolante, formando um contato do tipo polisilício-corte-metal,e extendendo-se o corte sobre a difusão, formando um retângulo de dois X quatro lambdas na região da emenda. Desta forma teremos dois X dois lambdas sobre o polisilício e dois X dois lambdas sobre a difusão. Com a finalidade de se obter uma construção mais compacta, o fio de polisilício deve sobrepor-se ao de difusão em um lambda na região da emenda. Uma outra regra indica que um corte expondo a difusão deve estar separado de um tran

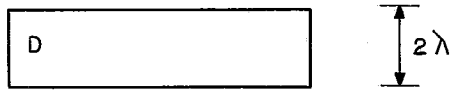
sistor de pelo menos dois lambdas, a fim de prevenir um possível curto circuito.

Uma outra camada também comumente empregada por laboratórios de fabricação é a de implante, responsável pela criação de transistores de depleção, ou seja, aqueles que exigem tensões negativas em suas portas, em relação aos respectivos drenos, para permanecerem abertos, sendo empregados como resistores. Em MEAD[9] é exigido que a região de implante exceda ao transistor em um e meio lambda, em todos os sentidos. Já LYON[8], apresentando resultados mais apurados, informa que o implante deve formar um retângulo, cuja largura é dois lambdas maior que a do fio de difusão e cujo comprimento é quatro lambdas maior que a largura do fio de polisilício, com centro no ponto médio do transistor. Admite também que boa parte dos projetistas usam, e continuarão usando, uma região que exceda o transistor de dois lambdas, em todos os sentidos. Vemos em MEAD[9] que além dessa exigência, é necessária uma separação de um e meio lambda entre uma região com implante e um transistor comum, ou seja, sem implante.

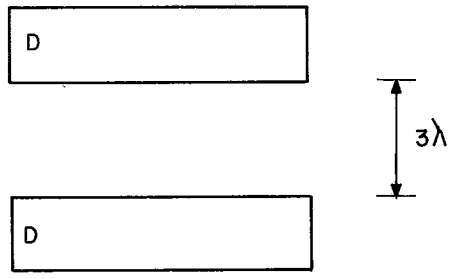
As regras apresentadas na simplificação de MEAD[9] estão graficamente ilustradas na figura (IV.1).

4.2 - MÉTODOS DE VERIFICAÇÃO DE REGRAS

O método dos polígonos, para verificação de regras de projeto, trata cada máscara como se fosse uma coleção de polígonos e utiliza uma álgebra para manipulação e combinação das camadas. Nesta álgebra existem as operações de expansão e contração de polígonos de uma camada e operações tradicionais como a união, interseção e complemento para a combinação de camadas ou a criação de pseudocamadas. Uma camada transistor, por exemplo ,

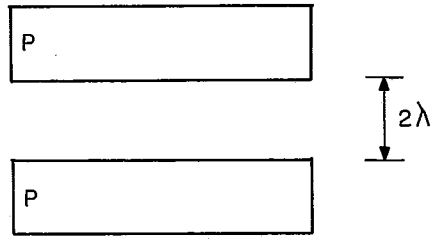


(a) LARGURA MÍNIMA EM DIFUSÃO



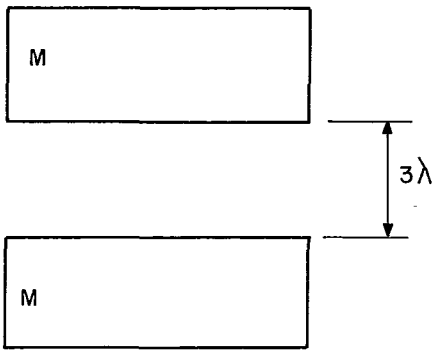
(b) SEPARAÇÃO MÍNIMA EM DIFUSÃO

(c) LARGURA MÍNIMA EM POLISILÍCIO

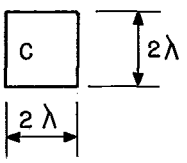


(d) SEPARAÇÃO MÍNIMA EM POLISILÍCIO

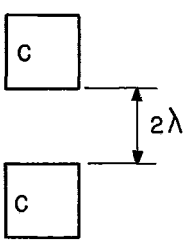
(e) LARGURA MÍNIMA EM METAL



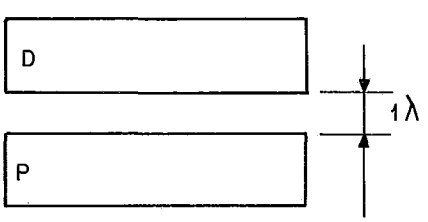
(f) SEPARAÇÃO MÍNIMA EM METAL



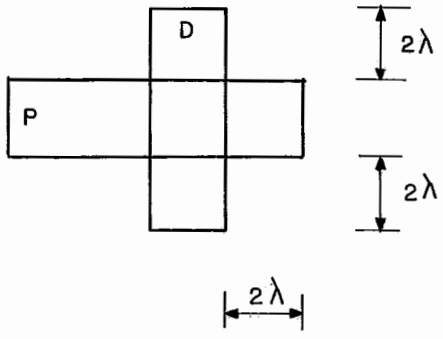
(g) LARGURA MÍNIMA EM CORTE



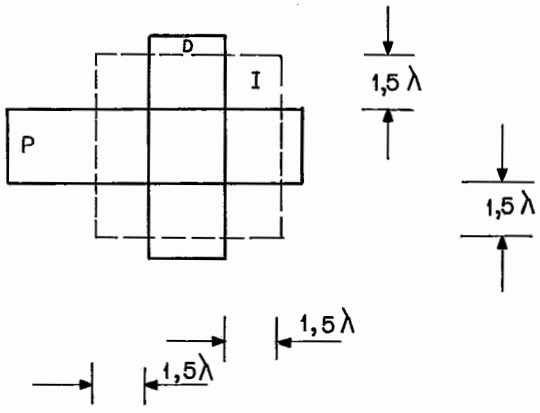
(h) SEPARAÇÃO MÍNIMA ENTRE CORTES



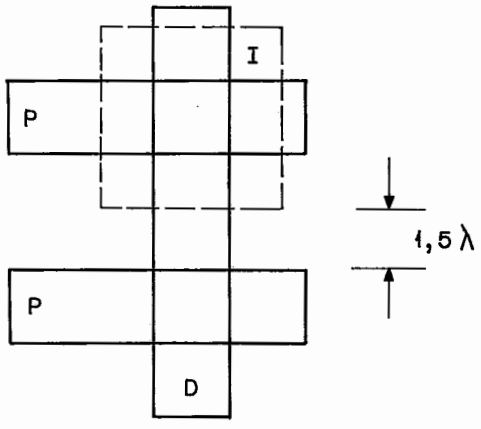
(i) SEPARAÇÃO MÍNIMA ENTRE DIFUSÃO E POLISILÍCIO



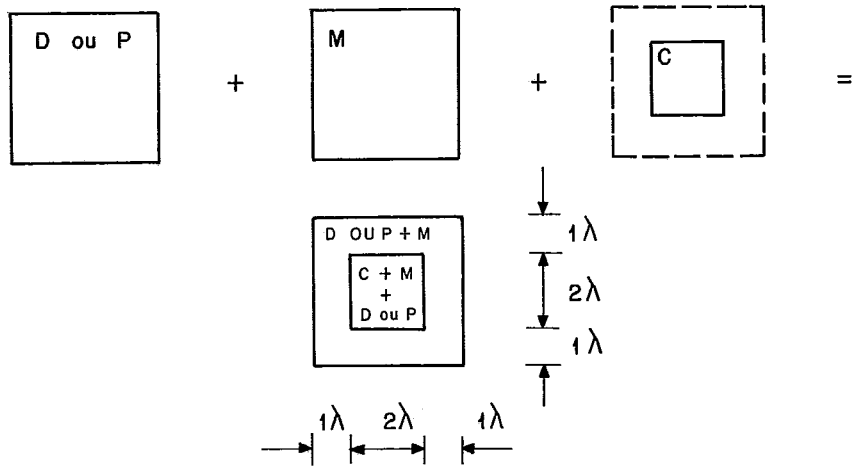
(j) FORMAÇÃO DE UM TRANSISTOR



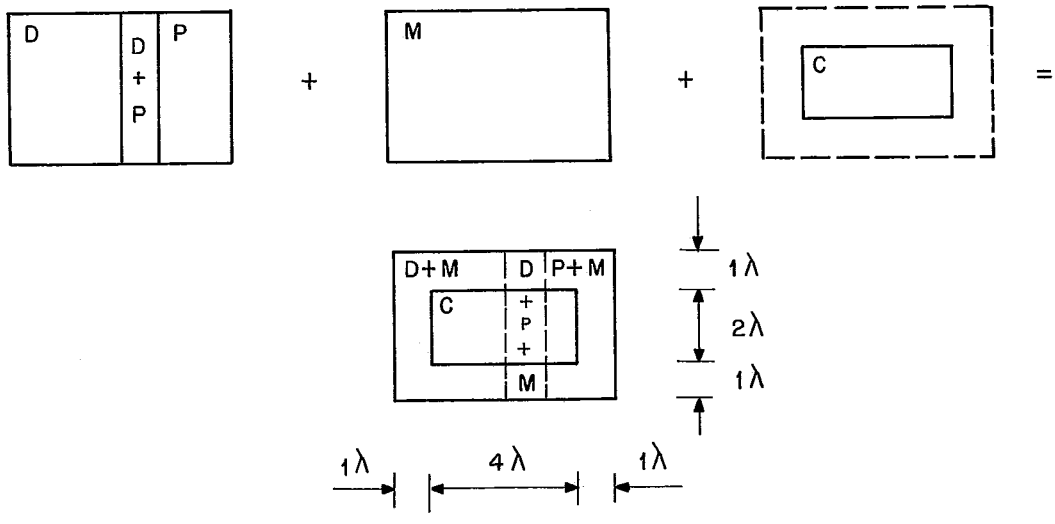
(l) IMPLANTE EM UM TRANSISTOR



(m) SEPARAÇÃO ENTRE IMPLANTE E TRANSISTOR COMUM



(n) CORTE SIMPLES



(o) CONTATO DE EMENDA

FIGURA IV.1 – REGRAS DE PROJETO

é criada através da interseção das camadas de difusão e polisilício. As regras de projeto são expressas em função dessas operações, que podem ser vistas como comandos. Testes de espacejamento são feitos expandindo-se os polígonos de um valor igual a metade do valor definido para a separação entre os fios dessa camada. Após a expansão, procura-se por interseções de polígonos. Cada interseção corresponde a um erro de separação, No pior caso, o número de comparações necessárias é proporcional ao quadrado do número de polígonos. Este método pode ser visto em YAMIN[13].

O método da varredura é apresentado detalhadamente em BAKER[2] e de um modo sucinto em BAKER[3]. Ele baseia-se no fato de todas as regras serem locais, pois especificam somente larguras e separações mínimas, que nunca superam três lambda. Segundo Baker , existe suficiente informação em uma janela de tamanho quatro X quatro lambdas que varra o circuito, representado através de uma mapa de PIXEL's, onde pixel é um jargão gráfico denotando um elemento do desenho (picture element). Neste método, os testes são realizados ao se comparar a configuração existente na janela que percorre o circuito, ou em parte dela, com padrões previamente definidos, que indicam ser a configuração correta ou inválida.

De modo geral, uma mesma célula é várias vezes repetida em um circuito. Isto significa que um verificador de regras de projeto irá realizar a mesma sequência de testes toda vez que encontrar esta célula. Em ROWSON[10] encontra-se uma proposta de projetos hierárquicos, onde a solução apresentada é a antiga política DIVIDIR PARA CONQUISTAR. Esta metodologia nos diz que, ao possuirmos um problema complexo, devemos dividí-lo em subproblemas, cada subproblema em outros menores, até obtermos uma série

de subproblemas, onde cada um possua uma solução trivial.

Aproveitando esse trabalho, WHITNEY[12] sugere a construção de um filtro que iniba a aparição de trechos repetidos mas que não anule a interligação existente entre esses trechos e o resto do circuito. No caso de Whitney, o filtro recebe como entrada a descrição em CIF do circuito e, após a filtragem, gera uma outra descrição do circuito onde os trechos repetidos foram retirados. Este novo circuito é então submetido a um verificador.

LYRA é um verificador em que as regras são especificadas através de exigências que devem ser satisfeitas em certos "cantos" do circuito. Entende-se por cantos não só os vértices que estejam presentes em uma determinada camada mas também aqueles provenientes da combinação de duas ou mais camadas do circuito. Cada regra é formada por duas partes: na primeira, é definida a configuração onde a regra deve ser aplicada e, na segunda, aparecem as exigências que devem ser atendidas. A configuração é descrita através da presença de camadas e cada exigência especifica uma área retangular, adjacente ao canto, na qual certas combinações de camadas são obrigatórias ou proibidas. Este método é descrito em ARNOLD[1].

O método dos autômatos finitos bidimensionais explora a semelhança existente entre a verificação de regras de projeto e o reconhecimento sintático de uma linguagem de programação. O autômato finito bidimensional é definido como sendo uma quádrupla, $M = (S, \Sigma, \delta, s_0)$, onde S é o conjunto dos n estados do autômato; Σ é o alfabeto de entrada, onde cada símbolo é da forma $a_{i,j} \in \{d, p, m, i, c, g\}$, representando o pixel situado na linha "i", coluna "j", da imagem rastreada; $s_0 \in S$ é o símbolo inicial do início de cada coluna e δ é a função de transição

que define o estado $s_{i,j}$ da seguinte maneira:

$$s_{i,j} = \delta (s_{i-1,j}, s_{i,j-1}, a_{i,j})$$

Os estados $s_{i-1,j}$ e $s_{i,j-1}$ são respectivamente chamados de estado horizontal (s_H) e estado vertical (s_V) do estado $s_{i,j}$. A cada transição do autômato está associada uma saída que pode valer "E", denotando um erro, ou ser nula, correspondendo a configurações válidas do projeto. Além disso, define-se que $s_{0,j} = s_{i,0} = 0$, para todo i e j. Note que i corresponde à variação no eixo horizontal e j à variação no eixo vertical. Maiores detalhes podem ser vistos em EUSTACE[8].

4.3 - IMPLEMENTAÇÃO DE UM VERIFICADOR DE REGRAS

No primeiro semestre de 1982 demos início à implementação de um verificador de regras de projeto. O método escolhido foi o da varredura, proposto por Baker. A escolha do método foi devida principalmente à pequena exigência de recursos computacionais e à necessidade de se obter rapidamente tal programa. Uma primeira versão foi escrita na linguagem FORTRAN e utiliza o minicomputador PDP 11/70 instalado no NCE/UFRJ. A escolha da linguagem permite que o verificador seja facilmente transportado para outro equipamento, pois tencionamos também utilizá-lo em um microcomputador SDE-40, dotado de vídeo gráfico colorido. É bom lembrar que métodos mais elaborados como o dos cantos ou o dos autômatos finitos bidimensionais foram apresentados publicamente em junho desse mesmo ano e, portanto, não tínhamos conhecimento de tais métodos quando iniciamos o trabalho.

O nosso verificador recebe como entrada a imagem rastreada do circuito. Possui uma janela que caminha de cima para baixo e da esquerda para a direita. Os eixos X e Y são posicionados na forma habitual. Essa janela tem dimensões quatro X quatro lambdas. Em alguns testes, este tamanho é excessivo e utilizamos uma subjanela, de tamanho apropriado, localizada na parte inferior esquerda da janela principal. A cada erro encontrado, escrevemos uma mensagem num arquivo, composta de um código do erro e das coordenadas do canto superior esquerdo da janela quatro X quatro que percorre o circuito. Devido às características do método, acrescentamos regiões em branco à esquerda, acima, à direita e abaixo da imagem, que funcionam como uma moldura e permitem que se possa realizar testes nas bordas do circuito.

As larguras e as separações mínimas de dois lambdas podem ser averiguadas através de uma janela três X três, enquanto que as de três lambdas necessitam de janelas quatro x quatro. Baker sugere que para cada janela possível deve existir a ela associada a informação correta ou incorreta, de tal maneira que uma vez identificada uma configuração, pode-se consultar uma tabela, para ser mais preciso um vetor de bits, e saber se ela é válida ou não. Ele também sugere a utilização de um algoritmo simples para a montagem de tal tabela. Nós preferimos introduzir o algoritmo no verificador pois o tempo de identificar a janela é o mesmo de validá-la.

Para uma janela três X três, num teste de largura e utilizando somente os bits da camada em questão, o algoritmo funciona da seguinte maneira: se a posição central da janela valer "1", calcule a alternação (A) da janela; se A for maior que dois, um erro foi encontrado. Resta-nos portanto descrever o cálculo

da alternação. Ele consiste em ir percorrendo todas as posições da janela, exceto a central, num único sentido, o horário, por exemplo. A primeira posição visitada será a última a ser considerada. Toda vez que ocorrer uma variação de "0" para "1", ou de "1" para "0", incrementa-se o valor da alternação, que inicialmente é nulo. A figura (IV.2) ilustra o cálculo da alternação para algumas janelas, indicando serem elas corretas ou erradas. Neste exemplo, os quadros que possuem "0" na posição central são considerados corretos pois a largura do fio está sendo considerada. Os quadros cujas alternações (A) são inferiores a três também estão corretos. Já as janelas cujas alternações são superiores a dois, correspondem à configurações inválidas e, portanto, a erros de projeto.

Comparando a validade das janelas fornecida pelo algoritmo com o resultado obtido manualmente, analisando janela a janela, notamos que existem algumas diferenças. Elas, entretanto, não chegam a comprometer o método, pois se referem a trechos mortos, não conectados a nenhuma outra parte do circuito, ou a erros que serão detectados pelo verificador quando da análise de uma outra janela.

Os testes de separação de dois lambdas são realizados usando o mesmo método, apenas trocando o "1" pelo "0" no algoritmo, pois a separação pode ser vista como a largura da ausência. Os testes de largura e separação de três lambdas utilizam janelas maiores e este algoritmo necessita de uma pequena modificação para verificar esses casos. No lugar de considerarmos a posição central com "1", ou quando da separação com "0", utilizaremos as quatro posições centrais com "1", ou todas com "0". Estes testes só devem ser aplicados após terem sido passados os testes

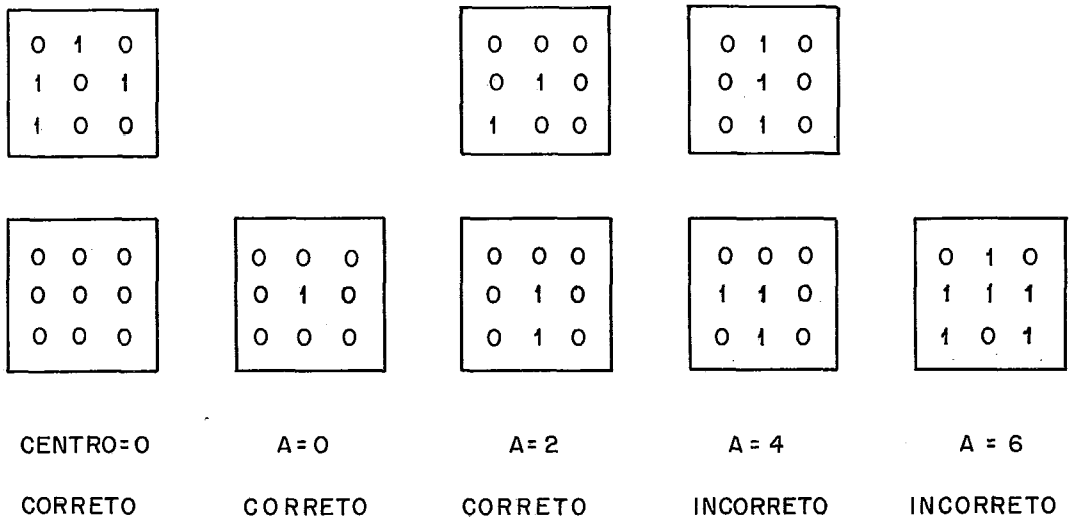
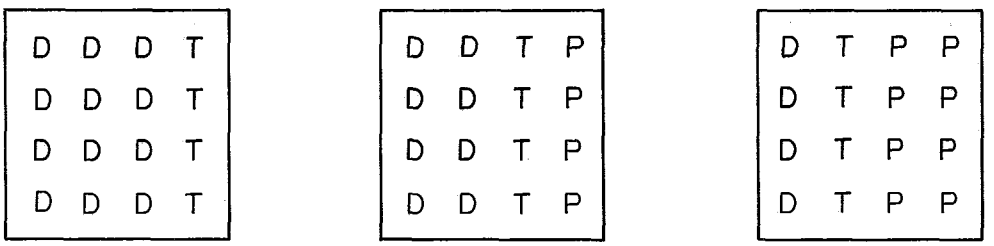


FIGURA IV.2 - EXEMPLO DO CÁLCULO DA ALTERNAÇÃO



D= DIFUSÃO
P= POLISILÍCIO
T= DIFUSÃO + POLISILÍCIO

FIGURA IV.3 - CONTATO DE EMENDA

com janelas três x três nessa camada. Em outras palavras, a separação entre dois fios de difusão é de, no mínimo, três lambdas. Ao ser fixada uma janela quatro x quatro, deve-se primeiro testar a separação de dois lambdas para depois realizar o teste de separação de três lambdas na difusão.

Com o algoritmo mostrado são realizados pelo verificador os seguintes testes:

a) Largura-2 na camada de polisilício;

b) Separação-2 na camada de polisilício;

c) Largura-2 na camada de difusão;

d) Separação-2 na camada de difusão;

e) Separação -3 na camada de difusão;

f) Largura-2 na camada de metal;

g) Largura-3 na camada de metal;

h) Separação-2 na camada de metal;

i) Separação-3 na camada de metal;

j) Largura-2 nos cortes de contato;

L) Separação-2 entre cortes de contato.

A segunda classe a ser analisada é a dos cortes de contato. Existem dois tipos: os simples e os de emenda. Para verificar os cortes simples, basta procurarmos uma janela quatro X quatro onde as quatro posições centrais possuam corte. A janela deve estar completamente preenchida com metal e também toda preenchida com difusão, se a ligação for difusão-metal, ou com polisilício, sendo neste caso a ligação polisilício-metal. Nesta janela, as camadas de difusão e polisilício não podem se interceptar.

Os contatos de emenda são analisados separadamente.

Pela regra, deveríamos usar uma janela quatro X seis. O mesmo efeito é conseguido se utilizarmos três janelas quatro X quatro e considerarmos cada uma delas individualmente. A figura (IV.3) mostra a aparência dessas janelas. Se um corte não corresponder a um corte simples deve-se então verificar se ele é um contato de emenda. Ao realizarmos os testes de contatos de emenda, devemos não só comparar as janelas com os padrões mostrados mas também com os obtidos através de rotações deles. Utilizaremos, desta forma, 12 padrões.

A separação entre um corte para a difusão e um transistor é testada criando-se uma pseudocamada que seja a união das camadas de polisilício e corte. Verifica-se então se elementos dessa pseudocamada possuem separação mínima de dois lambdas entre eles, utilizando uma janela três X três.

Os testes com transistores completam a verificação. O fio de polisilício deve ultrapassar o de difusão em dois lambdas. O de difusão também deve ultrapassar o de polisilício em dois lambdas. Deve existir uma separação entre eles de um lambda se nenhum transistor for formado. A figura (IV.4) mostra um fio de polisilício posicionado perpendicularmente a um de difusão. Inicialmente existe uma separação entre eles de um lambda. O fio de polisilício é então deslocado, lambda a lambda, até a formação correta de um transistor. O mesmo resultado seria conseguido se mantivéssemos preso o fio de polisilício e deslocássemos o de difusão.

O caso (V) pode ser descoberto considerando a pseudo camada polisilício e não difusão e procurando um objeto com largura de um lambda. O (II) pode ser considerado utilizando-se janelas dois X dois e confrontando-se com padrões que facilmente

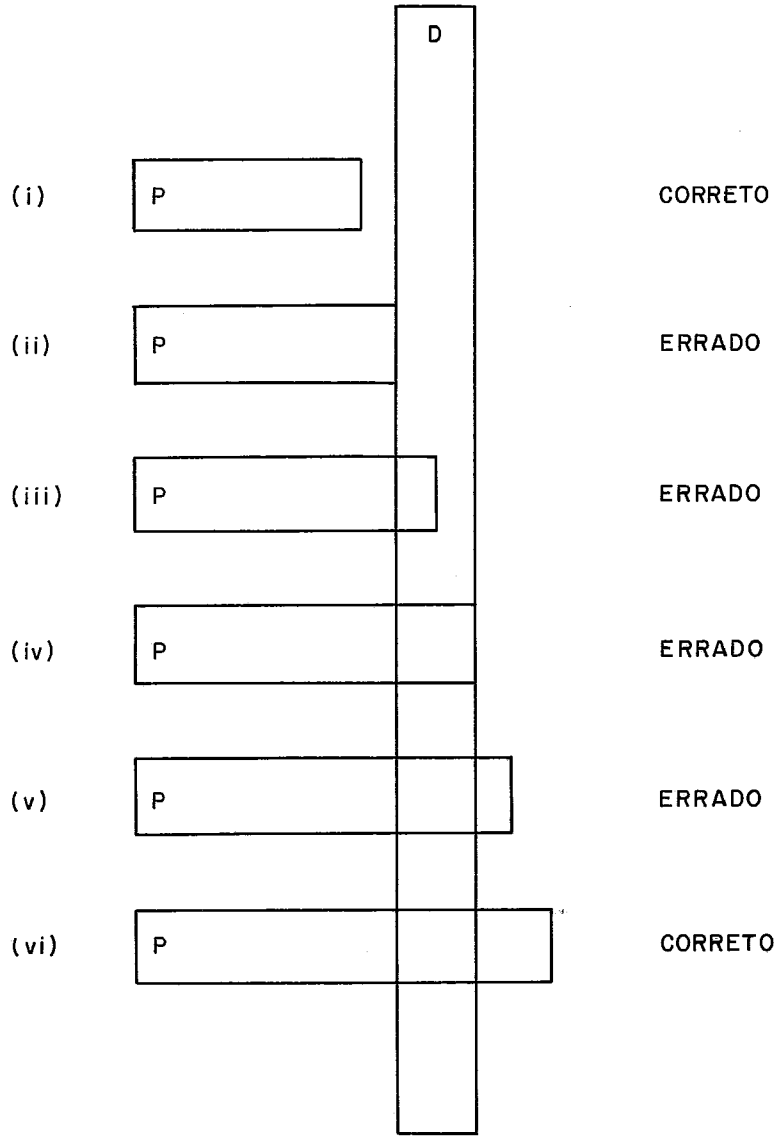


FIGURA IV.4 - DIFUSÃO x POLISILÍCIO

podem ser construídos, devido ao pequeno número de casos possíveis de janelas dois X dois. Os casos (III) e (IV) são detectados com a ajuda de uma janela dois X três, onde as duas posições inferiores esquerdas contenham tanto difusão quanto polisilício. Também devemos considerar as configurações que resultem de um espelhamento ou rotação desses casos.

O último teste a ser realizado refere-se à zona de implante. Como não possuimos informações que nos permitam testar meio lambda, fazemos testes com um lambda e consideramos correto o implante se o teste for satisfeito. Se o centro de uma janela três X três contiver difusão, polisilício e implante então toda a janela deve conter implante. A separação entre um transistor com implante e outro comum é feita de maneira semelhante. Se o centro de uma janela três X três contiver difusão e polisilício, mas não contiver implante, não deve haver implante em toda a janela.

O verificador por nós implementado gasta, em média, cerca de 8 ms de processador por cada lambda quadrado do circuito. Um circuito de 1000 X 1000 lambdas, considerado pequeno, levaria pouco mais de 2 horas para ser verificado. Circuitos típicos são da ordem de 3000 X 3000 lambdas. Como o tempo de verificação é proporcional à área do circuito, o programa gastaria cerca de 20 horas para realizar a análise. Embora esse tempo pudesse ser diminuído utilizando técnicas como reescrever os trechos críticos em assembler, notamos estar ele de acordo com as estimativas a que tivemos acesso, em que o tempo é cerca de quatro vezes menor com a utilização de equipamentos mais velozes como o DEC-10 e o VAX-11/780, ambos comercializados pela DIGITAL.

4.4 - LOCALIZAÇÃO DOS ERROS

O nosso verificador produz um arquivo contendo os erros por ele detectados. Para cada erro, é inserido um registro com duas informações: um código de erro e as coordenadas da janela principal do algoritmo na qual a violação foi registrada. O código refere-se ao tipo de teste que localizou o erro. A codificação é feita da seguinte forma:

A - Largura na camada de difusão;

B - Separação na camada de difusão;

C - Largura na camada de polisilício;

D - Separação na camada de polisilício;

E - Largura na camada de metal;

F - Separação na camada de metal;

G - Largura na camada de corte;

H - Separação na camada de corte;

I - Formação de um corte de contato;

J - Formação de um transistor;

L - Formação de transistores com implante;

M - Separação entre um corte para a difusão e um transistor.

Uma vez terminada a execução do verificador, basta listar o arquivo para saber que regras foram violadas e onde. A visualização dos erros, entretanto, não é das melhores. Para sanar este problema, foi desenvolvido um programa, denominado DRCMAP, com a finalidade de facilitar a localização dos erros. Ele traça um "gráfico" onde os eixos X e Y são posicionados e coloca, na coordenada correspondente a uma violação, o seu códi

go. Se mais de um código tiver de ser escrito no mesmo ponto, ele substitui os códigos por "*". Além disso, permite que se estipule janelas de visibilidade, através de valores mínimos e máximos para X e Y, e fatores de escala, possibilitando que cada unidade gráfica represente uma certa área do circuito.

4.5 - EXEMPLO DA VERIFICAÇÃO GEOMÉTRICA

No apêndice II é apresentado um exemplo de verificação de uma célula que fornece um sinal, e o seu inverso, que é o cálculo da função "e" de oito entradas. São apresentados o seu texto em CIF, uma representação gráfica obtida com o auxílio do editor CIFSYM, a saída do verificador e o gráfico produzido pelo programa DRCMAP, usando fator de escala 1, e uma janela que cobre todo o circuito.

V - EXTRATOR DE CIRCUITOS

5.1 - APRESENTAÇÃO

Vimos até agora duas formas de representar um circuito: através de um texto em CIF ou através de um mapa de pixel's, que nada mais é que uma materialização do texto. O primeiro formato é utilizado pelos editores e pelos laboratórios de fabricação e o segundo, pelo verificador de regras de projeto. Nenhuma das duas representações pode ser manipuladas eficientemente por uma classe muito importante de ferramentas, que é a formada por verificadores estáticos e simuladores lógicos e elétricos. Embora seja possível descrever um circuito através de uma notação, fornecida pelo projetista, que atenda às necessidades desses processos, seria impossível garantir que essa descrição realmente correspondesse ao circuito implementado.

O extrator é responsável pela conversão de um circuito na forma de mapa de pixel's em uma representação que pode ser fornecida aos simuladores e ao verificador estático. Nesta representação, um circuito é composto de transistores e fios, ou nós do circuito. Fazendo analogia com um processo de compilação, esta fase corresponde à geração da árvore sintática abstrata, onde as informações irrelevantes são eliminadas e as que permanecem enviadas à fase de análise da semântica.

5.2 - DESCRIÇÃO DO MÉTODO

Uma proposta de funcionamento de um extrator de circuitos pode ser encontrada em BAKER [2]. O extrator utiliza o mesmo

mapa de pixel's que é fornecido ao verificador de regras de projeto por ele descrito. Da mesma forma que o verificador, este programa necessita guardar uma pequena quantidade de linhas do mapa na memória. Para ser mais preciso, apenas uma linha do mapa e uma matriz, da largura de uma linha, onde são guardadas as informações sobre as camadas que são analisadas.

Este extrator está baseado em um algoritmo denominado LAGOAS E ILHAS, cujo funcionamento é bastante simples. Imagine uma lagoa contendo um certo número de ilhas. Esta topologia pode ser representada através de uma matriz bidimensional, como o mapa de pixel's, onde cada elemento só pode assumir dois valores: "0" ou "1". Cada "0" da matriz refere-se a uma região unitária de água e cada "1" a uma certa área unitária de ilha. O algoritmo percorre a matriz, linha a linha, sendo capaz de reconhecer regiões contínuas de terra e atribuir um código a cada ilha.

Uma vez fixado um elemento da matriz, o algoritmo utiliza três informações: a existência de terra no ponto, acima e à esquerda dele. A primeira informação é obtida no acesso ao elemento. As duas outras, com o auxílio de um vetor que é construído a medida que as linhas são percorridas, indicando a existência de terra nos pontos acima e à esquerda do que se está analisando. A figura (V.1) ilustra as três informações que são consideradas pelo algoritmo.

Esta é a idéia geral. Mas para que ele seja capaz de identificar áreas contíguas e rotular regiões, é necessário que possua não somente a informação de existência de terra acima e à esquerda, mas também os rótulos atribuídos a essas regiões. Neste caso, o rótulo "0" indica a inexistência de terra.

Uma vez fixado um elemento, e de posse das três informa

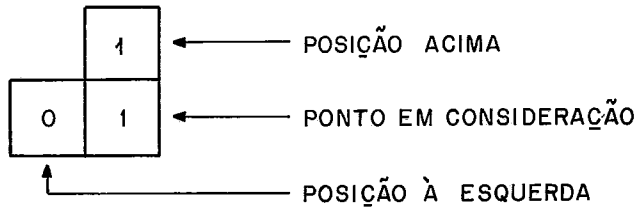


FIGURA V.1 - INFORMAÇÕES UTILIZADAS PELO ALGORITMO

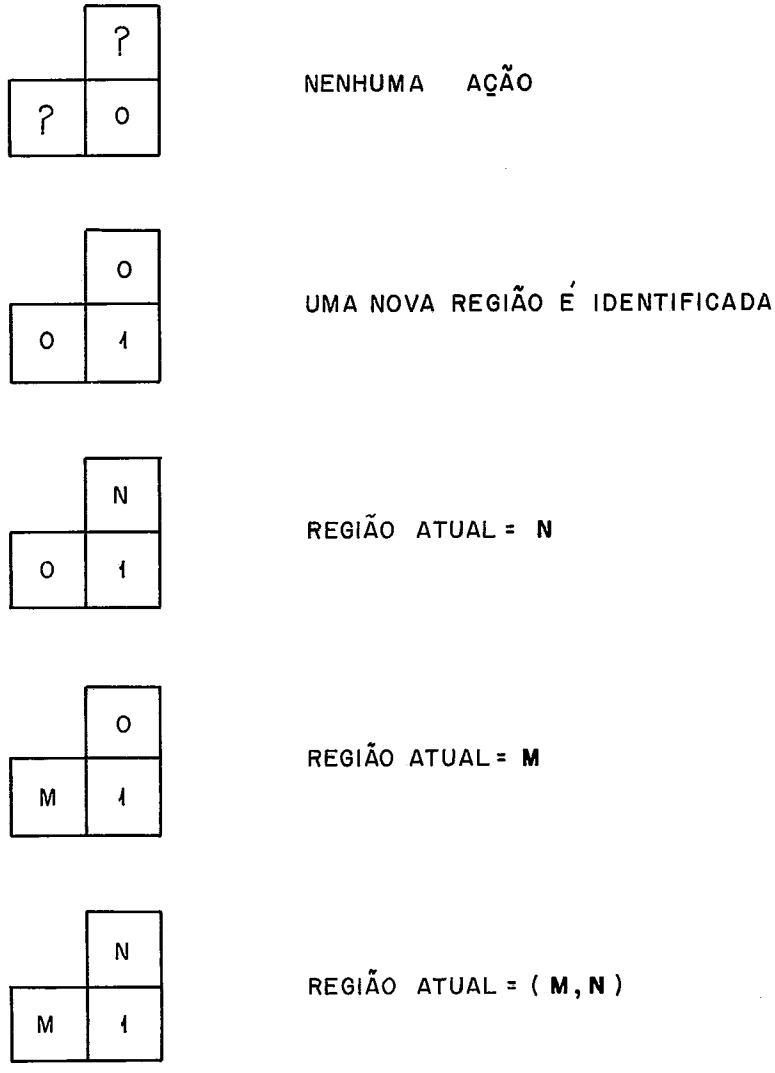


FIGURA V.2 - AÇÕES REALIZADAS PELO ALGORITMO

ções, algumas ações são tomadas. Se o elemento da matriz contiver "0", nenhuma ação é exigida. Se contiver "1", alguma ação deve ser executada, dependendo das outras duas informações. Se não houver terra tanto acima quanto à esquerda, o canto superior esquerdo de uma nova região foi encontrado. Atribui-se um rótulo numérico a esta região. Se houver terra acima mas não à esquerda, uma aresta vertical foi encontrada. O código do ponto em análise será o mesmo do ponto superior a ele. Já se não houver terra acima mas existir à esquerda, encontrou-se uma aresta horizontal. Neste caso o código será o mesmo do código do ponto à esquerda. Finalmente, se existir terra acima e à esquerda, as três regiões são contíguas e devem possuir o mesmo código. Quando este ponto for o vértice de um joelho, pode acontecer dos códigos das regiões acima e à esquerda serem diferentes. Neste caso, atribui-se um deles ao ponto em questão e faz-se uma equivalência entre as duas regiões. Ao fim do processo um único código será atribuído a essas regiões. A figura (V.2) ilustra as ações que são realizadas pelo algoritmo.

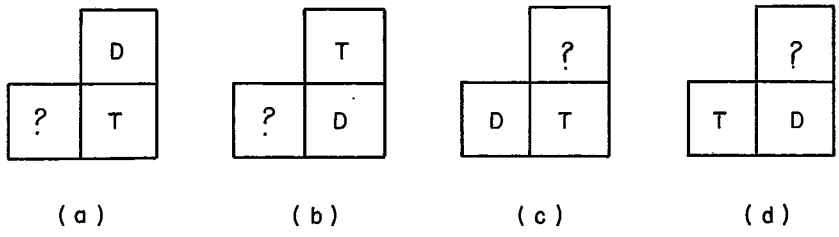
O extrator de circuitos utiliza o algoritmo de LAGOAS E ILHAS em quatro camadas: polisilício (P), metal (M), difusão mas não polisilício (D) e transistor (T), ou seja, difusão e polisilício. Com isto, ele é capaz de reconhecer regiões nessas quatro camadas e, embora não sendo suficiente para a extração de transistores e nós, constitui uma boa base para que esse objetivo seja alcançado.

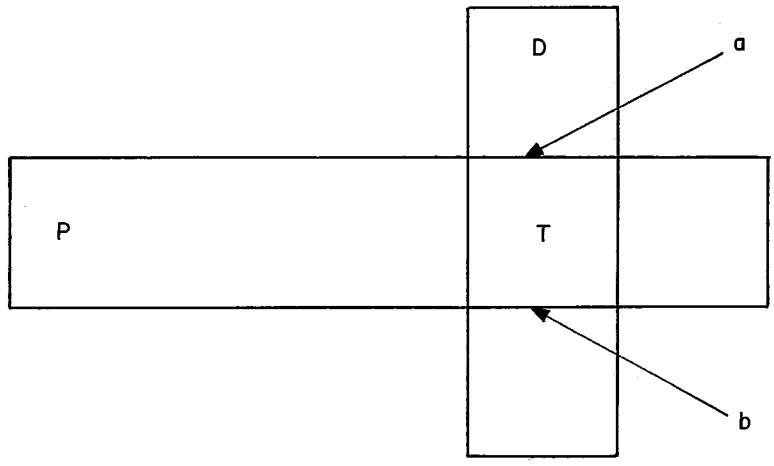
Os cortes de contato devem ser considerados pelo extrator. A função de um contato é permitir a passagem de corrente entre um fio da camada de metal e outro da camada de polisilício, ou de difusão. Quando o ponto em análise contiver um corte, de

ve-se estabelecer uma equivalência entre a região do metal e a região do polisilício, ou a da difusão. Quando se proceder a renumeração dos códigos, estas duas regiões receberão o mesmo código, passando a representar uma única região.

Os transistores são identificados através das bordas. Algumas de suas arestas possuem informações valiosas: os códigos da difusão, do polisilício e do transistor. Como nem todas as arestas possuem a informação do código da difusão, só consideramos as que possuem este dado. O extrator reconhece pedaços de transistores que são armazenados em um arquivo. Cada registro desse arquivo possui as seguintes informações: códigos das regiões T, P e D e a existência, ou não, de implante no transistor. A figura (V.3) ilustra os casos em que pedaços de transistores são reconhecidos.

Após o término do algoritmo, os códigos são renumerados de modo que as regiões equivalentes possuam um mesmo código. Os pedaços de transistores devem ser atualizados após a mudança dos códigos. O arquivo de transistores é então ordenado usando como chave o campo código do transistor. Todos os pedaços de um mesmo transistor são agrupados pela ordenação e uma simples leitura deste arquivo traz todas as informações que necessitamos. Deve-se ter o cuidado de verificar se o código do polisilício é o mesmo para todos os pedaços e de se construir uma lista com os diferentes códigos de difusão. Se para um transistor existir um único código de difusão na lista, encontramos um transistor degenerado e não o incluímos no arquivo de saída do extrator. Se a lista possuir dois códigos, um transistor foi encontrado e um registro é inserido no arquivo de saída. Se mais de dois aparecerem, trata-se de um transistor incomum que, embora teoricamente possível, deve ser considerado errado e uma mensagem é





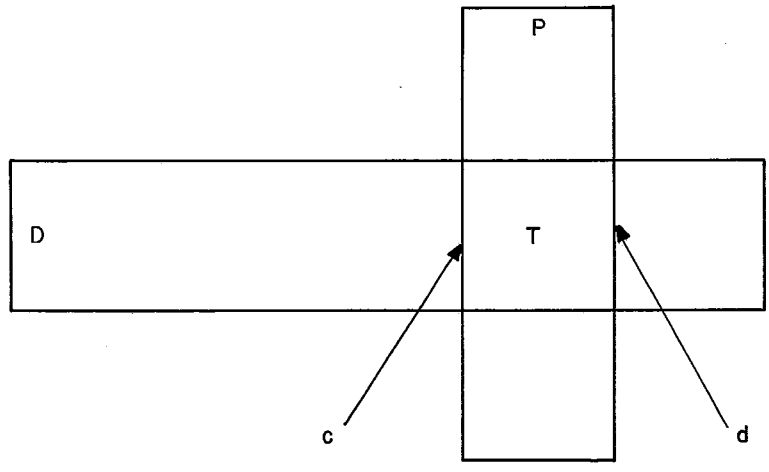


FIGURA V.3- BORDAS DE TRANSISTORES

enviada reportando o fato.

5.3 - IMPLEMENTAÇÃO DE UM EXTRATOR

Foi desenvolvido um extrator de circuitos, utilizando o método da varredura anteriormente descrito, para uso na estação de apoio a projetos de VLSI do NCE. Mais uma vez utilizamos o minicomputador PDP 11/70 e a linguagem FORTRAN. Algumas alterações foram realizadas em relação ao método original. Estas alterações e alguns detalhes da implementação são agora mostrados.

O método proposto por Baker utiliza apenas um mapa de pixel's e por isto não comporta a identificação de nós batizados que, na fase de geração do mapa, são depositados em um arquivo. O nosso extrator localiza esses nós, descobre o código de suas regiões e os deposita em um arquivo temporário. Estas informações são úteis para os procedimentos de verificação estática e simulação lógica.

Os códigos das regiões acima e à esquerda do ponto são guardados em quatro vetores, um para cada camada, construídos durante a varredura de cada linha do mapa. O código da região à esquerda do ponto atual, coluna J do mapa, é encontrado na posição J-1 do vetor que memoriza essa camada, enquanto que o código da região acima desse ponto pode ser obtido na posição J do mesmo vetor. Uma vez definido o código do ponto atual, ele é depositado na posição J e irá servir tanto para a informação da região à esquerda do próximo ponto como da região acima do seu ponto inferior.

A equivalência entre regiões é uma relação reflexiva , simétrica e transitiva. Toda vez que uma região de código N é

identificada, temos que N R N. Para cada região iremos possuir uma lista de equivalências construída de tal modo que EQUIV(N) <= N. Desta forma, faz-se a equivalência entre duas regiões, N e M, seguindo as listas de equivalência de N e M até, em cada lista, ter-se EQUIV(T) = T. Supondo que a lista de equivalência de N leve a uma região T1 e a de M, a uma região T2, faz-se EQUIV(T2) = T1, se T1 < T2. Caso contrário, faz-se EQUIV(T1) = T2. A equivalência entre regiões foi implementada em um arquivo. A informação contida no registro de ordem N é uma região equivalente a N.

Uma vez executado o algoritmo de LAGOAS E ILHAS, o arquivo de equivalência é processado. Os arquivos contendo os pedaços de transistores e os nós são atualizados, utilizando-se os códigos das regiões equivalentes. Após esta atualização, os arquivos estão prontos para a segunda parte da extração, a qual poderíamos chamar de aglutinação. Como no equipamento por nós usado não é permitido invocar o "sort" do sistema no interior de um programa, optamos pela ordenação externa e quebramos o extrator em dois programas: o primeiro é responsável pela extração propriamente dita e o segundo, pela aglutinação das partes.

A segunda parte do extrator recebe os dois arquivos produzidos pela primeira parte, onde o de pedaços de transistores foi ordenado pelo código da região do transistor e o de nós, pelo código da região que o nó rotula. Este programa junta todos os pedaços, que foram agrupados pela ordenação, formando um único registro para cada dispositivo e o inserindo em um arquivo de símbolos. Os nós identificados na primeira parte são transportados para esse arquivo que, portanto, possui dois tipos de registros, cujos formatos são mostrados a seguir.

Campos de um registro de transistor:

a) Tipo do registro ('T');

b) Tipo do transistor ('E' ou 'D');

c) Número do transistor;

d) Código da região da porta;

e) Código da região da fonte;

f) Código da região do dreno;

g) Largura do canal;

h) Comprimento do canal.

Campos de um registro de nó:

a) Tipo do registro ('N');

b) Tipo do nó ('I' ou 'N');

c) Código da região do nó;

d) Nome do nó.

5.4 - EXEMPLO DA EXTRAÇÃO

A fim de ilustrar a extração de circuitos, executamos o nosso extrator fornecendo-lhe a mesma célula materializada submetida ao verificador de regras de projeto. O resultado é apresentado no apêndice II. Note que por convenção, um nome de nó no texto CIF precedido pelo caracter " # " refere-se à um nó de entrada. Durante a geração do mapa de pixel's, os tipos dos nós foram estipulados a partir dessa convenção. Cada nó não batizado pelo projetista recebe do extrator um nome padrão, que é da forma NO #XXXXX, onde XXXXX é o código de sua região.

VI - VERIFICADOR ESTÁTICO

6.1 - APRESENTAÇÃO

Um verificador estático tem por finalidade localizar, de forma rápida, alguns erros que só poderiam ser detectados através de simulação, que é um processo mais complexo e mais demorado. Utilizando o circuito extraído, o verificador pode diagnosticar transistores mal formados, trechos do circuito isolados dos demais, nós com possíveis valores lógicos indefinidos, além de nós interligados indevidamente.

6.2 - DESCRIÇÃO DO MÉTODO

A verificação que será apresentada baseia-se na descrição de BAKER [2] . Alguns testes foram, no entanto, incorporados e outros retirados da solução original. Preferimos apresentar a solução final, pois a achamos melhor estruturada, dentro do nosso contexto de verificação de projetos.

O verificador estático recebe um arquivo contendo informações sobre os transistores e os nós do circuito. Dentre os nós que pertencem a esse arquivo de símbolos, existem alguns que serão muito importantes no processo de verificação: aqueles cujos nomes são fornecidos pelo projetista, em particular, os nós denominados VDD e GND, alimentação e terra do circuito, e os demais nós de entrada. É suposto que o VDD refere-se ao valor lógico "1" e o GND ao valor "0". Um nó de entrada pode, num determinado instante valer "1" ou "0". Os demais nós irão se comportar de acordo com as propagações dos sinais no circuito.

Os testes que são realizados sobre os nós podem ser divididos em três categorias:

a) dois nós com mesmo nome devem referenciar a mesma região;

b) dois nós com nomes diferentes devem referenciar regiões diferentes;

c) um nó, que não seja de entrada, deve poder assumir tanto "1" quanto "0".

Como abordaremos os detalhes de implementação posteriormente, não vamos nos deter aos dois primeiros casos acima. A terceira categoria de testes pode ser realizada após a propagação dos sinais VDD e GND. Supondo que cada transistor possa possuir, em um determinado instante, o valor "1" em sua porta, podemos encarar os transistores fechados como operadores relacionais. Chamando de X ao código do nó conectado à sua fonte e de Y ao código do nó conectado ao seu dreno, podemos representar um transistor pela notação X R Y. Como não fazemos distinção entre a fonte e o dreno, temos que R é simétrica, ou seja, X R Y → Y R X. A propagação do VDD, ou do GND, pode ser feita através de um fechamento transitivo de R, ou seja, VDD R X e X R Y → VDD R Y. Realizando a propagação do VDD e anotando todos os nós relacionados a ele, e fazendo o mesmo para o GND, precisamos então procurar os nós que não sejam de entrada e não estejam relacionados com o VDD e com o GND.

Os testes sobre os transistores são realizados de acordo com seu tipo:

a) transistor normal:

a.1) três códigos de região distintos em seus pinos;

b) transistores de depleção;

b.1) PULL UP:

b.1.1) a porta e o dreno devem possuir o mesmo código de região;

b.1.2) a fonte e o VDD devem possuir o mesmo código de região;

b.2) SUPERBUFFER:

b.2.1) o dreno e o VDD devem possuir o mesmo código de região;

b.2.2) a porta e a fonte não podem possuir o mesmo código de região;

b.3) YELLOW TRANSISTOR:

b.3.1) a fonte e o VDD não podem possuir o mesmo código de região;

b.3.2) o dreno e o VDD não podem possuir o mesmo código de região;

b.3.3) analisá-lo como um transistor normal que possua VDD em sua porta.

Os testes aplicáveis aos transistores devem ocorrer após a propagação de VDD e GND e a realização dos testes sobre os nós.

6.3 - IMPLEMENTAÇÃO DO VERIFICADOR ESTÁTICO

O verificador estático, como os demais programas por nós implementados, foi escrito na linguagem FORTRAN, utilizando o computador PDP-11/70. Devido à necessidade de ordenações, o verificador foi dividido em duas partes: na primeira, alguns arquivos são construídos a partir do circuito extraído. A segunda

parte executa após as ordenações e realiza os testes citados anteriormente. Para que seja possível explicar o funcionamento do verificador, vamos mostrar os formatos de alguns arquivos por ele utilizado e como estes são manipulados.

Ao ler o circuito extraído, o verificador preenche três arquivos: dois relativos a transistores, TR0 e TR3, e um relativo aos nós, NO0, com os seguintes formatos:

Arquivo TR0:

a) código da região da fonte (dreno);

b) código da região do dreno (fonte).

Arquivo NO0:

a) nome do nó;

b) código da região do nó;

c) tipo do nó.

Arquivo TR3:

a) número do transistor;

b) código da região da porta;

c) código da região da fonte;

d) código da região do dreno;

e) tipo do transistor.

Para cada transistor lido no circuito extraído, um registro é inserido em TR3 e dois registros são adicionados a TR0. O arquivo TR0 será utilizado para indicar a relação entre os nós. Assim sendo, inserimos as informações fonte R dreno e dreno R fonte. Para cada nó lido, criamos um novo registro em NO0 com as informações desse nó. Ao término da leitura do circuito, ordenamos TR0 ascendentemente pelos campos "a" e "b" e NO0, também

em ordem crescente, pelo campo "a". Como resultado das ordenações são produzidos os arquivos TR2 e NO1.

A segunda parte faz a verificação propriamente dita. Para isso, ela utiliza um quarto arquivo com o seguinte formato:

Arquivo TR1:

a) código da região;

b) ponteiro para o início da região em TR2;

c) ponteiro para o fim da região em TR2;

d) nome do nó;

e) VDD propagado;

f) GND propagado;

g) tipo do nó.

O verificador cria o arquivo TR1 a partir de TR2. Para cada região encontrada em TR2 ele insere um registro em TR1 que possui ponteiros para o primeiro e para o último registro dessa região em TR2. Ambos são de acesso direto permitindo que se obtenha rapidamente as regiões relacionadas através da relação fonte R dreno. Uma vez esgotado o arquivo de transistores, passa-se ao arquivo de nós. TR1 tem seu campo "nome do nó" preenchido a partir de NO1. Os testes "a" e "b", aplicáveis aos nós, são realizados durante a leitura deste arquivo. O primeiro, com o auxílio de uma área que guarda o conteúdo do último nó lido anteriormente em NO1. Se o nome do nó recentemente lido for igual ao que estiver na área auxiliar, as duas regiões devem possuir o mesmo código, já que é incorreto utilizar o mesmo nome para referenciar regiões distintas. O ítem "b" é feito quando da atualização de TR1. Ao se preencher o campo "nome do nó", verifica-se se algum outro nome foi atribuído a essa região, o que caracteriza um possível erro e, portanto, deve ser reportado.

Uma vez processado o arquivo N01, realiza-se a propagação do VDD e do GND. Durante a leitura do arquivo de nós os códigos desses dois nós são memorizados. O procedimento que faz as propagações é simples. Procura todas as regiões relacionadas ao VDD em TR2 e marca-as como visitadas. Para cada uma delas, procura as regiões que estão relacionadas a elas e que ainda não foram visitadas, e assim sucessivamente. Feita a propagação do VDD, repete-se o algoritmo para o GND. Após as propagações, percorre-se TR1 sequencialmente a procura de nós que não estejam relacionados tanto ao VDD quanto ao GND e que não sejam de entrada, realizando o ítem "c" dos testes sobre os nós.

Os testes sobre os transistores utilizam os arquivos TR1 e TR3. Para cada transistor, verifica-se seu tipo, que pode ser normal ou depleção. Se for de depleção, testa-se se corresponde a um "pull" up", a um "superbuffer" ou a um "yellow transistor". Estes transistores diferem entre si de acordo com suas propriedades elétricas e podem ser vistos em MEAD [9] . Se o transistor pertencer à última categoria, convertêmo-lo em um normal e atribuímos o código do VDD à sua porta. Se o transistor em análise não for de depleção, considerando os convertidos, verificamos se ele está de acordo com a regra de formação descrita no ítem "a.1". Qualquer erro encontrado é reportado no arquivo relatório do verificador.

6.4 - EXEMPLO DA VERIFICAÇÃO ESTÁTICA

No apêndice II é mostrado o relatório produzido pela execução do verificador estático ao utilizar o circuito extraído do exemplo do capítulo anterior. Os erros foram causados pela

não inclusão de alguns nós no conjunto dos nós de entrada.

VII - CONCLUSÕES E SUGESTÕES PARA PESQUISA

O nosso pacote de verificação de projetos de circuitos integrados é o que podemos chamar de ferramental mínimo necessário para este tipo de verificação. Ele se comporta como um sistema e acha-se totalmente implementado, escrito em Fortran e cada programa utiliza no máximo 64 Kb de memória. Esta primeira versão utiliza um minicomputador PDP-11/70 e as ferramentas foram dimensionadas para suportarem circuitos inteiros, com a complexidade dos que estão sendo desenvolvidos no NCE.

Como os projetos de circuitos integrados são desenvolvidos de forma hierárquica, célula a célula, o nosso pacote pode ser facilmente redimensionado para executar em microcomputadores que possuam menor quantidade de memória, possibilitando a obtenção de estações de trabalho de baixo custo.

Com a experiência adquirida no desenvolvimento de ferramentas de apoio e no contato com o grupo de projeto do NCE, parece-nos interessante propor a confecção de procedimentos que possibilitem automatizar ainda mais a tarefa de projetar circuitos integrados. Estamos apresentando as sugestões em uma ordem que nos parece ser um grau crescente de complexidade.

Ao longo das descrições das ferramentas, mostramos uma certa analogia entre os processos de verificação e de compilação. Assim sendo, a verificação de um circuito pode ser feita através da compilação de um texto bidimensional, como um mapa de pixel's. Tal compilador poderia possuir um conjunto de opções, que seriam ativadas pelo usuário, de modo que fosse possível realizar apenas parte da análise, como verificar somente as regras de projeto. Este programa provavelmente necessitaria

de uma grande quantidade de recursos computacionais, dificultando sua implementação em máquinas de pequeno porte, como a que usamos.

As duas outras sugestões referem-se à tarefa de edição de células e circuitos. Se um circuito pudesse ser criado isento de erros, toda a fase de verificação seria suprimida. No lugar de utilizarmos editores, poderíamos descrever o circuito através de um texto em uma linguagem de médio nível, que possuísse comandos estruturados e facilidades no manuseio de tipos, como em Pascal, mas que permitisse alguma intervenção de projetista na geração do código, com a indicação do posicionamento relativo dos blocos no circuito. O programa objeto poderia ser um texto em CIF ou uma imagem rastreada do circuito.

A última sugestão seria o projeto de um compilador de silício completo. Enquanto a ferramenta anterior pode ser encarada como um compilador de uma linguagem de médio nível, esta poderia ser vista como um compilador de uma linguagem de alto nível, onde o código gerado não pode ser controlado pelo usuário. O programa receberia uma descrição do circuito e geraria um objeto após possíveis otimizações do código. Embora estas duas últimas ferramentas produzissem, a princípio, circuitos maiores e menos eficientes que os projetados manualmente, elas permitiriam confecções em curto espaço de tempo que poderiam ao menos servir de base para a construção de novos circuitos.

VIII - REFERÊNCIAS BIBLIOGRÁFICAS

1. ARNOLD, Michael H.; OUSTERHOUT, John K. - LYRA: A New Approach to Geometric Layout Rule Checking, Proceedings of the Nineteenth Design Automation Conference, USA, 530-536, 1982.

2. BAKER, Clark - Artwork Analysis Tools for VLSI Circuits. MIT/LCS/TR-239, USA, Massachusetts Institute of Technology, 1980.

3. BAKER, Clark; TERMAN, Chris - Tools for Verifying Integrated Circuit Designs, Lambda, USA, 1(4): 22-30, 1980.

4. BRYANT, Randal E. - An Algorithm for MOS Logic Simulation, Lambda, USA, 1(4): 46-53, 1980.

5. BRYANT, Randal E. - A Logic-Level Simulator for MOS LSI. User's Manual. VLSI Memo 80-21, USA, Massachusetts Institute of Technology, 1980.

6. EUSTACE, R. Alan; MUKHOPADHYAY, Amar - A Deterministic Finite Automaton Approach to Design Rule Checking for VLSI, Proceedings of the Nineteenth Design Automation Conference, USA, 712-717, 1982.

7. HON, Robert W.; SEQUIN, Carlo H. - A Guide to LSI Implementation, USA, XEROX- Palo Alto Center, 1979.

8. LYON, Richard- Simplified Design Rules for VLSI Layouts, Lambda, USA, 2(1): 54-59, 1981.

9. MEAD, Carver; CONWAY, Lyn - Introduction to VLSI Systems, USA, Addison-Wesley Publishing Company, 1980.

10. ROWSON, J. - Understarding Hierarchical Design, USA, Tese de Ph.D., California Institute of Technology, 1980.

11. SILVA Filho, Ysmar Vianna et al. - Projetos de circuitos Integrados em VLSI. Relatório Técnico 0382, Rio de Janeiro, NCE/UFRJ, 1982.

12. WHITNEY, Telle - A Hierarchical Design-Rule Checking Algorithm, Lambda, USA, 2(1): 40-43, 1981.

13. YAMIN, M. - A Computer Program for Integrated Circuit Mask Design Checkout, Bell System Technical Journal, USA, 51(7), 1972.

APENDICE I

Neste anexo é apresentado o resultado da simulação funcional aplicada a um circuito exemplo. A listagem foi obtida ao listarmos o arquivo de auditoria da simulação.

```
SOURCE EXEMP
COMMENT  ==  COMPILACAO DA DEFINICAO DO CIRCUITO  ==
READ EXEMP
%
% DECLARACAO DE SINAIS EXTERNOS E DE FASE
%
  SINAIS
    FASE1, FASE2, CGEXT, BARRA (8);
%
% DECLARACAO DA PLA QUE GERA SINAIS DE CONTROLE
%
  PLA CONTROLE
    CK: FASE1
        CARGA  = CGEXT;
        ZERA   = CGEXT;
        PRONTO = DESLOCA;
        NOP    = -CGEXT * -DESLOCA;
    CK: FASE2
        DESLOCA = CGEXT + DESLOCA * -C2
        PRONTO  = FASE1;
%
% DECLARACAO DA PLA QUE CONTA O NUMERO DE BITS SERIALIZADOS
%
  PLA CONTADOR
    CK: FASE1
        T0 = -Tx0;
        C0 = Tx0;
        T1 = Tx1 * -C0  +  -TX1 * C0;
        C1 = TX1 * C0;
        T2 = Tx2 * -C1  +  -TX2 * C1;
        C2 = Tx2 * C1;
    CK: FASE2
        TX0 = T0 * -ZERA;
        TX1 = T1 * -ZERA;
        TX2 = T2 * -ZERA;
%
% DECLARACAO DO REGISTRO DE DESLOCAMENTO
% OBS: A SAIDA SERIAL E O SINAL "SAIDA"
%
  REG R, 8
    CK: FASE1
        CG: CARGA, BARRA
        SR: DESLOCA, NADA, SAIDA
        TL: NOP
    CK: FASE2
        TR: FASE2;
QUIT
*** FIM DA COMPILACAO DAS DEFINICOES ***
COMMENT  ==  FASES DO RELOGIO  ==
CLOCK FASE1:10  FASE2:01
COMMENT  ==  NUMERO A SER SERIALIZADO NA BARRA  ==
SET BARRA0:1  BARRA1:0  BARRA2:1  BARRA3:0
SET BARRA4:1  BARRA5:0  BARRA6:1  BARRA7:0
COMMENT  ==  SINAIS A SEREM MONITORADOS  ==
WATCH FASE1 FASE2 PRONTO
COMMENT  ==  RODO 2 PASSOS SEM FAZER NADA  ==
STEP 2
===== INICIO DO PASSO      1 =====
```

```
FASE1     = 1
FASE2     = 0
PRONTO    = 0
***** FIM DO PASSO      1 *****
***** INICIO DO PASSO     2 *****
FASE1     = 0
FASE2     = 1
PRONTO    = 0
***** FIM DO PASSO      2 *****
COMMENT   ==  MANDO LER A BARRA E RODO 2 PASSOS  ==
FORCE CGEXT:1
STEP 2
***** INICIO DO PASSO     3 *****
FASE1     = 1
FASE2     = 0
PRONTO    = 0
***** FIM DO PASSO      3 *****
***** INICIO DO PASSO     4 *****
FASE1     = 0
FASE2     = 1
PRONTO    = 0
***** FIM DO PASSO      4 *****
COMMENT   ==  DESLIGO CGEXT E RODO 16 PASSOS  ==
UNFORCE CGEXT
SET CGEXT:0
WATCH SAIDA
STEP 16
***** INICIO DO PASSO     5 *****
FASE1     = 1
FASE2     = 0
PRONTO    = 1
SAIDA     = 1
***** FIM DO PASSO      5 *****
***** INICIO DO PASSO     6 *****
FASE1     = 0
FASE2     = 1
PRONTO    = 0
SAIDA     = 1
***** FIM DO PASSO      6 *****
***** INICIO DO PASSO     7 *****
FASE1     = 1
FASE2     = 0
PRONTO    = 1
SAIDA     = 0
***** FIM DO PASSO      7 *****
***** INICIO DO PASSO     8 *****
FASE1     = 0
FASE2     = 1
PRONTO    = 0
SAIDA     = 0
***** FIM DO PASSO      8 *****
***** INICIO DO PASSO     9 *****
FASE1     = 1
FASE2     = 0
PRONTO    = 1
SAIDA     = 1
***** FIM DO PASSO      9 *****
***** INICIO DO PASSO    10 *****
```

```
FASE1     = 0
FASE2     = 1
PRONTO    = 0
SAIDA     = 1
***** FIM DO PASSO     10 *****
***** INICIO DO PASSO     11 *****
FASE1     = 1
FASE2     = 0
PRONTO    = 1
SAIDA     = 0
***** FIM DO PASSO     11 *****
***** INICIO DO PASSO     12 *****
FASE1     = 0
FASE2     = 1
PRONTO    = 0
SAIDA     = 0
***** FIM DO PASSO     12 *****
***** INICIO DO PASSO     13 *****
FASE1     = 1
FASE2     = 0
PRONTO    = 1
SAIDA     = 1
***** FIM DO PASSO     13 *****
***** INICIO DO PASSO     14 *****
FASE1     = 0
FASE2     = 1
PRONTO    = 0
SAIDA     = 1
***** FIM DO PASSO     14 *****
***** INICIO DO PASSO     15 *****
FASE1     = 1
FASE2     = 0
PRONTO    = 1
SAIDA     = 0
***** FIM DO PASSO     15 *****
***** INICIO DO PASSO     16 *****
FASE1     = 0
FASE2     = 1
PRONTO    = 0
SAIDA     = 0
***** FIM DO PASSO     16 *****
***** INICIO DO PASSO     17 *****
FASE1     = 1
FASE2     = 0
PRONTO    = 1
SAIDA     = 1
***** FIM DO PASSO     17 *****
***** INICIO DO PASSO     18 *****
FASE1     = 0
FASE2     = 1
PRONTO    = 0
SAIDA     = 1
***** FIM DO PASSO     18 *****
***** INICIO DO PASSO     19 *****
FASE1     = 1
FASE2     = 0
PRONTO    = 1
SAIDA     = 0
```

```
***** FIM DO PASSO     19 *****
***** INICIO DO PASSO     20 *****
FASE1     = 0
FASE2     = 1
PRONTO    = 0
SAIDA     = 0
***** FIM DO PASSO     20 *****
COMMENT  ==  RODO MAIS 2 PASSOS E TERMINO  ==
UNWATCH SAIDA
STEP 2
***** INICIO DO PASSO     21 *****
FASE1     = 1
FASE2     = 0
PRONTO    = 0
***** FIM DO PASSO     21 *****
***** INICIO DO PASSO     22 *****
FASE1     = 0
FASE2     = 1
PRONTO    = 0
***** FIM DO PASSO     22 *****
KILL
```

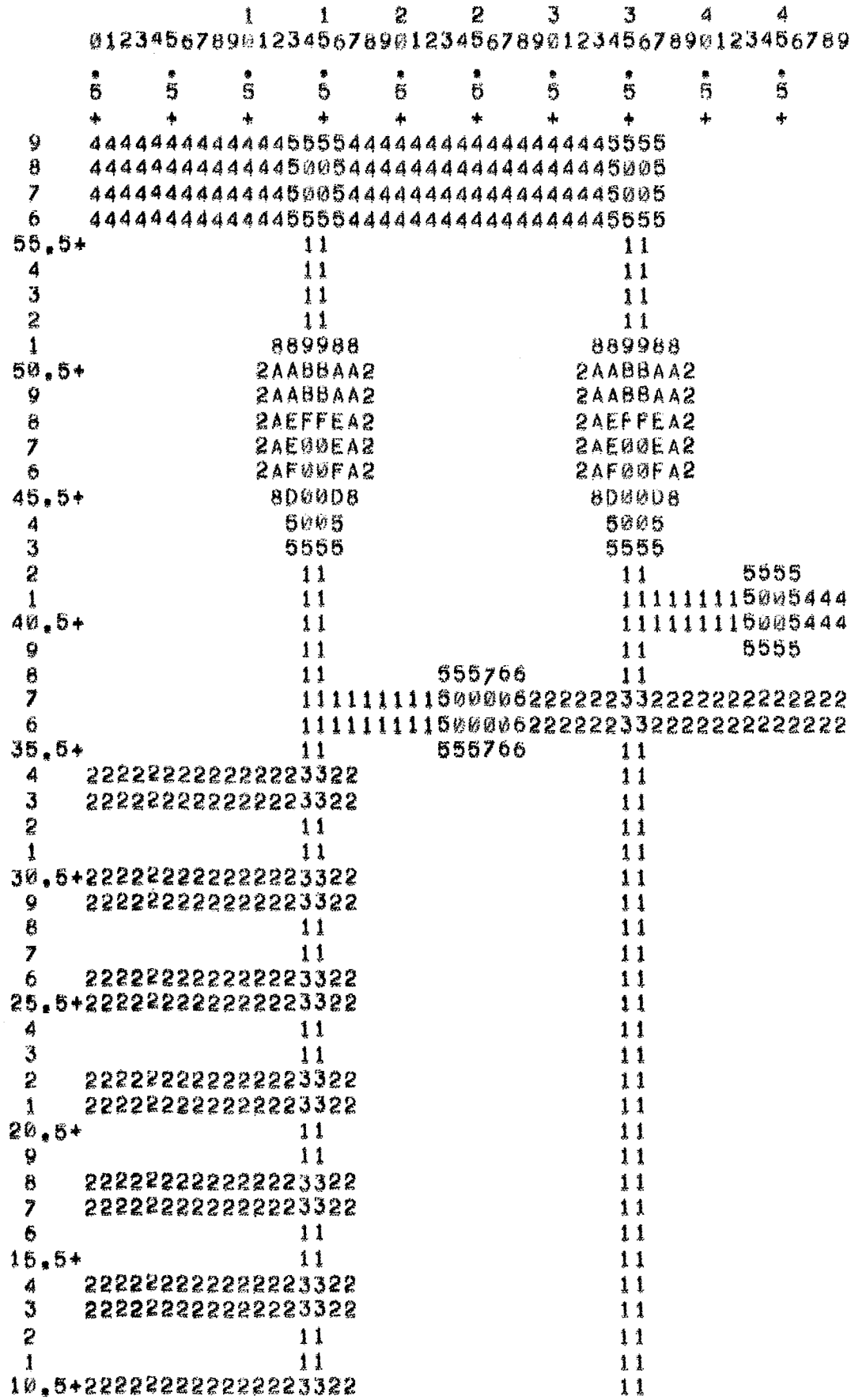
APÊNDICE II

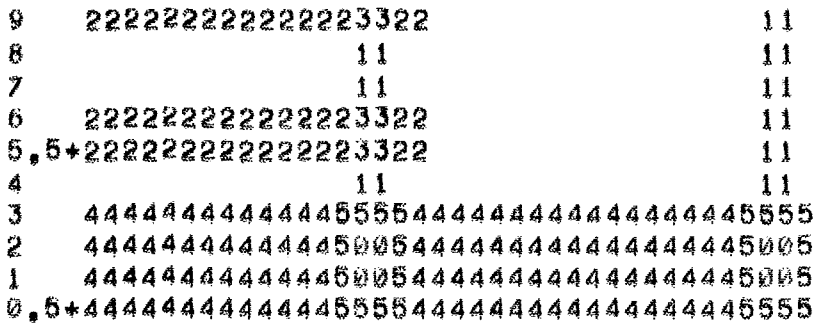
São aqui apresentadas várias listagens obtidas a partir de arquivos em disco. Todas referem-se ao mesmo circuito que foi submetido ao nosso pacote de verificação. São elas, na ordem em que aparecem:

- Texto em CIF do circuito.
- Representação gráfica do circuito gerada pelo editor CIFSYM.
- Saída do verificador de regras do projeto.
- Saída do programa DRCMAP.
- Saída do extrator de circuitos.
- Saída do verificador estático.

```
(NAME: EXEMP                    SIZE:  50 X  60);
DS100 25 1;
LND;    B    40    40   150    20;
        B    40    40   360    20;
        B    20   320   150   200;
        B    20   360   360   220;
        B    40    10   250   355;
        B   130    20   205   370;
        B    20    50   150   405;
        B    40    10   250   385;
        B    40    10   450   395;
        B   120    20   410   410;
        B    20    10   360   425;
        B    40    10   450   425;
        B    40    40   150   450;
        B    40    40   360   450;
        B    20    90   150   515;
        B    20    90   360   515;
        B    40    40   150   580;
        B    40    40   360   580;
LNP;    B   180    20    90    60;
        B   180    20    90   100;
        B   180    20    90   140;
        B   180    20    90   180;
        B   180    20    90   220;
        B   180    20    90   260;
        B   180    20    90   300;
        B   180    20    90   340;
        B    30    10   275   355;
        B   240    20   380   370;
        B    30    10   275   385;
        B    80    50   150   485;
        B    80    50   360   485;
LNM;    B   380    40   190    20;
        B    60    40   260   370;
        B    40    10   450   395;
        B    70    20   465   410;
        B    40    10   450   425;
        B    40    60   150   460;
        B    40    60   360   460;
        B   380    40   190   580;
LNI;    B    70    80   150   485;
        B    70    80   360   485;
LNC;    B    20    20   150    20;
        B    20    20   360    20;
        B    40    20   260   370;
        B    20    20   450   410;
        B    20    40   150   460;
        B    20    40   360   460;
        B    20    20   150   580;
        B    20    20   360   580;
0                       0   590 NM N #VDD          ;
0                     490   370 NP N SAINV          ;
0                     490   410 NM N SAI            ;
0                       0   340 NP N ENT1           ;
0                       0   300 NP N #ENT2          ;
0                       0   260 NP N ENT3           ;
0                       0   220 NP N #ENT4          ;
```

```
0                       0   180 NP N ENT5          ;
0                       0   140 NP N #ENT6         ;
0                       0   100 NP N ENT7          ;
0                       0    60 NP N #ENT8         ;
0                       0    30 NM N #GND          ;
DF;
C 100;
E;
```




9 2222222222223322 11
8 11 11
7 11 11
6 2222222222223322 11
5,5+2222222222223322 11
4 11 11
3 4444444444445555444444444444444445555
2 4444444444445005444444444444444445005
1 4444444444445005444444444444444445005
0,5+4444444444445555444444444444444445555

```
EXEMP
X:        0        50    Y:        0        60    M:37
ERRO I NA POSICAO       13,     49
ERRO I NA POSICAO       34,     49
ERRO I NA POSICAO       13,     48
ERRO I NA POSICAO       34,     48
ERRO B NA POSICAO       37,     45
ERRO E NA POSICAO       44,     43
ERRO E NA POSICAO       45,     43
ERRO E NA POSICAO       46,     43
```

```
MAPA PARA VERIFICACAO DE REGRAS DE PROJETO
LAYOUT: EXEMP      AREA: X:        0       50  Y:        0       60
JANELA:  X:        0       50  Y:        0       60  REDUCAO: 1/  1

      0000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000
      0000000000000000000000000000000000000000000000000000000000
      0000000000111111111122222222223333333333444444444455555555
      0123456789012345678901234567890123456789012345678901234567
    +-----------------------------------------------------------+
 60 |                                                           |
 59 |                                                           |
 58 |                                                           |
 57 |                                                           |
 56 |                                                           |
 55 |                                                           |
 54 |                                                           |
 53 |                                                           |
 52 |                                                           |
 51 |                                                           |
 50 |                                                           |
 49 |            I                    I                         |
 48 |            I                    I                         |
 47 |                                                           |
 46 |                                                           |
 45 |                                 B                         |
 44 |                                                           |
 43 |                                             EEE           |
 42 |                                                           |
 41 |                                                           |
 40 |                                                           |
 39 |                                                           |
 38 |                                                           |
 37 |                                                           |
 36 |                                                           |
 35 |                                                           |
 34 |                                                           |
 33 |                                                           |
 32 |                                                           |
 31 |                                                           |
 30 |                                                           |
 29 |                                                           |
 28 |                                                           |
 27 |                                                           |
 26 |                                                           |
 25 |                                                           |
 24 |                                                           |
 23 |                                                           |
 22 |                                                           |
 21 |                                                           |
 20 |                                                           |
 19 |                                                           |
 18 |                                                           |
 17 |                                                           |
 16 |                                                           |
 15 |                                                           |
 14 |                                                           |
 13 |                                                           |
```


12
11
10
9
8
7
6
5
4
3
2
1
0

```
TD       1       2       1       2       5       2
TD       2       3       1       3       5       2
TE       3       2       3       4       2       2
TE       4       5       2       6       2       2
TE       5       7       6       8       2       2
TE       6       9       8      10       2       2
TE       7      11      10      12       2       2
TE       8      13      12      14       2       2
TE       9      15      14      16       2       2
TE      10      17      16      18       2       2
TE      11      19      18       4       2       2
NIVDD                    1
NNSAINV                  2
NNSAI                    3
NIGND                    4
NNENT1                   5
NNNO#00006               6
NIENT2                   7
NNNO#00008               8
NNENT3                   9
NNNO#00010              10
NIENT4                  11
NNNO#00012              12
NNENT5                  13
NNNO#00014              14
NIENT6                  15
NNNO#00016              16
NNENT7                  17
NNNO#00018              18
NIENT8                  19
```

```
ERRO 6 - NO ENT1          NAO PODE ASSUMIR VALOR LOGICO 1
ERRO 6 - NO ENT1          NAO PODE ASSUMIR VALOR LOGICO 0
ERRO 6 - NO ENT3          NAO PODE ASSUMIR VALOR LOGICO 1
ERRO 6 - NO ENT3          NAO PODE ASSUMIR VALOR LOGICO 0
ERRO 6 - NO ENT6          NAO PODE ASSUMIR VALOR LOGICO 1
ERRO 6 - NO ENT5          NAO PODE ASSUMIR VALOR LOGICO 0
ERRO 6 - NO ENT7          NAO PODE ASSUMIR VALOR LOGICO 1
ERRO 6 - NO ENT7          NAO PODE ASSUMIR VALOR LOGICO 0
```