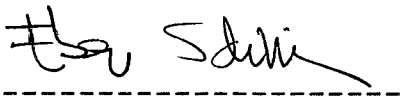
EDITORES GRÁFICOS PARA PROJETO DE CIRCUITOS INTEGRADOS

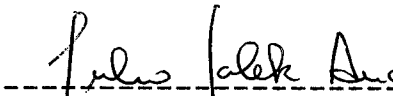
José Antonio dos Santos Borges

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.) EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

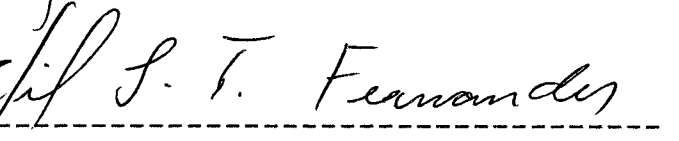
Aprovada por:



Eber Assis Schmitz
(presidente)



Júlio Salek Aude
(NCE/UFRJ)



Edil Severiano Tavares Fernandes
(COPPE/UFRJ)

Rio de Janeiro, RJ - BRASIL
Fevereiro de 1988

BORGES, JOSé ANTONIO DOS SANTOS

Editores gráficos para projeto de Circuitos Integrados (Rio de Janeiro), 1986.

XI, 150 p. 29.7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1988).

Tese - Universidade Federal do Rio de Janeiro, COPPE.

1. CAD para projeto de circuitos integrados

I. COPPE/UFRJ

II. Título (serie)

DEDICATóRIA

Dedico este trabalho à memória dos meus maiores amigos:

Adauto Borges de Moraes

e

Antonio Rodrigues dos Santos

AGRADECIMENTOS

Ao meu orientador e amigo, Eber Assis Schmitz, pela tremenda força que deu a este trabalho, e pelas broncas que deu (com razão) pela falta de determinação de seu orientado para escrever esta tese. Eber é a mola mestra que impulsiona o grupo de circuitos integrados do NCE.

A todos os analistas e programadores que trabalharam comigo durante estes 8 anos de projeto LSI, em especial ao Antonio Anibal Teles, João Sérgio Assis, Julio Tadeu da Silveira e Oswaldo Vernet Pires.

Ao grupo de projetistas de circuitos integrados que foram os maiores sofredores mas também os felizes usuários dos editores gráficos que foram criados pelo grupo de software, em especial Carlo Emmanoel Tolla de Oliveira, Heloísa Teixeira da Silva e Mario Afonso da Silveira Barbosa.

Finalmente à minha esposa, Sonia, por tudo.

AGRADECIMENTOS ESPECIAIS

À S!D Informática S.A. que, através do projeto ESTRA de apoio à pesquisa em computação, forneceu uma parte substancial dos recursos necessários para o desenvolvimento deste trabalho, em especial apoiando o projeto TEDMOS, do Núcleo de Computação Eletrônica da UFRJ.

Resumo da Tese apresentada á COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

EDITORES GRÁFICOS PARA PROJETO DE CIRCUITOS INTEGRADOS

José Antonio dos Santos Borges

Fevereiro de 1988

Orientador: Dr. Eber Assis Schmitz
Programa: Engenharia de Sistemas e Computação

Esta tese é uma condensação do conhecimento teórico e prático para a criação de editores gráficos para circuitos integrados em metodologia "full custom.

São estudadas diversas características operacionais, estruturas de dados e algoritmos de diversos editores, procurando dar uma forma didática a um assunto para o qual não existe bibliografia organizada.

As técnicas aqui apresentadas, foram utilizadas, aprimoradas ou geradas na construção de diversos editores gráficos, alguns deles, puramente experimentais, outros efetivamente utilizados em produção pela equipe de projeto de circuitos integrados do NCE/UFRJ.

Abstract of the thesis presented to COPPE/UFRJ as partial fulfilment of the requirements for the degree of Master of Science (M.Sc.)

GRAPHIC EDITORS FOR THE DESIGN OF INTEGRATED CIRCUITS

José Antonio dos Santos Borges

February, 1988

Chairman: Dr. Eber Assis Schmitz
Department: Systems and Computing Engineering

This thesis is a condensation of the theorical and practical knowledge to create graphic editors for integrated circuits in full custom methodology. Operational characteristics, data structures and main algorithms are studied and organized.

The techniques presented here have been used, improved or generated during the creation of many graphic editors, some of them experimental, others effectively used on production by the IC project group at NCE/UFRJ.

I N D I C E

1. Introdução

1.1 - Apresentação

A indústria de circuitos integrados em grande escala é hoje uma indústria que lida com volumes de vendas extremamente grandes. Estima-se que o valor do mercado mundial de "chips" seja da ordem de 25 bilhões de dólares. A quase totalidade deste valor é constituída por componentes chamados "de catálogo" ou padrões. Estes circuitos são fabricados por grandes empresas americanas e japonesas e vendidos aos fabricantes de equipamentos eletrônicos.

Por outro lado, razões de custo e de segredo industrial estão levando muitos fabricantes de equipamentos eletrônicos a usarem circuitos integrados proprietários em seus equipamentos. Por exemplo, os microcomputadores MacIntosh e IBM/PC-AT usam vários circuitos personalizados.

O projeto de circuitos integrados em grande escala é uma tarefa impossível de ser realizada sem o auxílio de ferramentas computacionais devido à complexidade dos problemas a serem tratados [Silva Fo., 83]. Um circuito integrado razoavelmente complexo pode conter dezenas de milhares de transistores, cada qual envolvendo, por sua vez, um conjunto de 4 a 6 formas geométricas.

É impossível lidar com tantos elementos sem que exista um conjunto adequado de ferramentas computadorizadas que permita a criação, manipulação e verificação de consistência entre esses elementos. Essas ferramentas computadorizadas, entretanto, são um produto muito valioso, caríssimo, muitas vezes segredo industrial do fabricante de circuitos integrados. Algumas vezes, mesmo comprado ou licenciado, somente algumas partes desse software são executadas pelo usuário (a entrada de dados) enquanto que outras (a geração do circuito) são processadas pelo fabricante.

1.2 - O grupo do NCE

Por volta de 1982, o Núcleo de Computação Eletrônica da UFRJ iniciou uma linha de pesquisa na área de projeto de circuitos in-

tegrados. A finalidade maior desta equipe era absorver a tecnologia de projeto de circuitos integrados para aplicação no Brasil.

Logo ao início, a equipe foi dividida em dois grupos: um grupo de projetistas de circuitos e um grupo de programação para produzir ou adaptar programas que seriam utilizados pelos primeiros. Assim, o grupo de projetistas deveria assimilar e desenvolver técnicas de projeto. Com a experiência adquirida, ele poderia realimentar o grupo de software, no sentido de se dispor cada vez mais de ferramentas mais aprimoradas.

Inicialmente o grupo contou com algum apoio estrangeiro para conseguir softwares de grupos de pesquisa nesta área, mas pouco tempo depois, devido ao fato que o projeto de circuitos integrados é uma tecnologia de interesse estratégico, a ajuda cessou, e fomos forcados a desenvolver nossas próprias ferramentas, nossos próprios programas.

Para a equipe, foi excelente, mas tivemos que comecar do zero. Ao longo destes anos, construímos diversos sistemas de software, alguns chips, e realmente adquirimos bastante conhecimento nesse campo, pois tivemos que criar os programas, e não somente usá-los. Em suma, a tecnologia de software para projeto de circuitos integrados foi dominada.

1.3 - O que é este trabalho

Este trabalho é uma condensação do conhecimento necessário para construir editores gráficos para projeto de circuitos integrados em metodologia "full-custom", e que foi obtido pela equipe de software através da construção de diversos editores.

O problema maior do projetista de editores gráficos reside na dificuldade em obter uma documentação sobre como eles são construídos. Existem relativamente poucos artigos que tratam deste assunto na literatura, e os que existem, se dedicam a tratar de um ou outro aspecto algorítmico mais interessante, mas não entram em detalhe sobre a estrutura geral.

Isso é facilmente explicável, uma vez que a maior parte dos editores que existem tem sido produzido pelos próprios fabricantes de circuitos integrados, ou por software houses que não tem maiores interesses em divulgar esta tecnologia. Somente alguns poucos editores, criados em Universidades, tem uma documentação razoável, ou são disponíveis em forma de programa fonte para serem estudados.

À primeira vista, o trabalho de construção de um editor parece relativamente simples, porque as idéias básicas envolvidas são relativamente simples: inserir e remover desenhinhos na tela do computador. Entretanto é difícil encontrar alguém que saiba construir um editor gráfico!

A maior causa: é necessario um conhecimento multi-disciplinar muito grande. Para construir um editor gráfico não são necessarios somente sólidos conhecimentos de processamento gráfico e da disciplina à qual o editor vai ser aplicado, mas também pleno domínio de estruturas de dados (incluindo organização de arquivos e bancos de dados), sistemas operacionais e hardware (periféricos e interfaces), conhecimento razoável de telecomunicações (ligação e controle de terminais), compiladores, álgebra linear, alem de experiência prática com ergonomia.

O que este trabalho tenta fazer então é mostrar os problemas que existem e quais suas soluções. Como os problemas são de natureza diversa, as soluções também são apresentadas de diversas maneiras. Algumas vezes, é a explicação "como se faz". Outras vezes, os algoritmos que resolvem o problema. Outras vezes, as alternativas de projeto.

1.4 - Características gerais do trabalho

O que se pretendeu, ao escrever este trabalho, foi obter um texto didático sobre o assunto.

Entre as diversas abordagens possíveis, optou-se por separar os editores em quatro tipos fundamentais, de complexidade crescente, e estudar os problemas de cada um. Essa divisão permite uma

abordagem mais interessante, pois é possível entender totalmente um editor simples, e por comparações e analogias, ir entendendo as construções mais complexas.

Na análise dos tipos de editores, é seguido sempre o mesmo roteiro: inicialmente uma visão geral do método utilizado pelo editor, uma visão geral de características operacionais, estudo das estruturas de dados e algoritmos principais. Algumas vezes são mostrados aspectos de hardware ou sistema operacional que são relevantes para o estudo de algum problema.

Optou-se também por dar pouca ênfase a estudos de complexidade dos algoritmos mostrados, pois eles, na maior parte das vezes, são executados sobre conjunto de dados relativamente pequeno, e de forma interativa.

1.5 - Principais trabalhos realizados nesta área

Os primeiros editores gráficos para VLSI, foram construídos em fábricas de circuitos integrados ou em software houses especializadas em CAD, e a documentação que se tem são apenas folhetos de propaganda.

O primeiro editor fartamente documentado é o ICARUS [Fairbarn, 78], da Xerox PARC, rodando no microcomputador ALTO, da própria XEROX. Esse editor é importante pois foi o primeiro editor do qual se dispõe uma documentação sobre estrutura de dados e implementação.

Os dois editores mais importantes, especialmente em termos de utilização no âmbito universitário são o CAESAR [Ousterhout, 81], e seu sucessor KIC2 [Keller, 82], ambos construídos em Berkeley, para executar sob ambiente UNIX. Esses editores foram divulgados sob a forma de programa fonte, e utilizados em dezenas de instalações no mundo inteiro. No caso do KIC2, inclusive foram feitas adaptações para executar em microcomputadores PC conectados a terminais gráficos.

Um editor que teve muita importância no nosso trabalho foi o

CIFSYM [Lewis, 81]. O prof. Lewis esteve de visita no NCE, ministrou um curso sobre projeto de circuitos integrados, e nos doou seu editor, que permitia a criação de layouts em terminais alfanuméricos. Um dos editores que construímos, o EDMOS [Borges, 83] foi em parte baseado naquele editor.

Finalmente, um sistema que tem farta documentação, mas ao qual não tivemos acesso, foi o Magic [Ousterhout, 84] que incorpora algumas idéias muito interessantes, especialmente nos aspectos de organização e manipulação dos dados.

Um de nossos trabalhos, o sistema TEDMOS [Borges, 87], para ensino de projeto VLSI, é hoje distribuído em diversos países e possui centenas de usuários. Esse sistema é importante, pois é a demonstração de que já possuímos o "know-how" na construção de sistemas de CAD para projeto de circuitos integrados.

1.6 - Plano da tese

A tese é dividida em 5 partes principais.

O capítulo 2 faz um apanhado geral sobre o projeto de circuitos integrados, e mostra características gerais dos editores gráficos destinados a este tipo de atividade. São mostrados também alguns aspectos da tecnologia de fabricação de circuitos, que são necessarios para entender as características dos editores.

O capítulo 3 mostra um estilo muito simples de editor gráfico, baseado em um hardware particular, no qual se tem acesso à memória de vídeo de uma tela colorida, e que é muito fácil de ser construído. Sao mostradas as funções gerais de um editor gráfico, e a maneira com que um operador interage com ele. Sao estudados a estrutura interna de funcionamento e detalhes técnicos de implementação, incluindo os algoritmos de extração das características geométricas do desenho.

O capítulo 4 estuda um estilo de editor conceitualmente semelhante ao primeiro, no qual a edição é feita sobre a memória do computador, numa matriz e depois movida para a memória de vídeo.

Sao estudados os problemas de utilização de terminais gráficos e da associação da estrutura de dados com o desenho criado no terminal. São também feitas algumas considerações sobre características do software básico necessário e sobre problemas de eficiência para a produção do desenho na tela.

O capítulo 5 mostra um editor no qual os elementos gráficos são armazenados em listas de elementos. São mostrados os aspectos de operação e os métodos de atualização da estrutura de dados. São mostradas algumas alternativas de organização dos dados visando uma maior eficiencia operacional, especialmente nas operações de atualização de tela.

O capítulo 6 introduz a abordagem hierárquica de edição, que é indispensável para a edição de grandes circuitos. São abordados os aspectos operacionais em editores hierárquicos e variantes da apresentação gráfica hierarquico. Apresenta-se alguns fundamentos de álgebra linear úteis para o tratamento de hierarquia. A estrutura de dados é estudada e são mostradas algumas alternativas para codificação e armazenamento da hierarquia.

Finalmente, nas conclusões, são mostradas algumas considerações sobre os resultados obtidos e as tendências que se apresentam neste tipo de trabalho.

Nos capítulos 3, 4 e 6, a título de ilustração, são apresentados editores que foram construídos por nós utilizando as técnicas mostradas no capítulo, e que são efetivamente utilizados no NCE pela equipe de projeto de circuitos integrados.

1.7 - Advertência ao leitor

Os capítulos deste trabalho não são estanques, ou seja, o trabalho foi construído com cada capítulo apoiado nos capítulos anteriores. Desta maneira, é praticamente impossível le-lo de forma não sequencial, pelo menos da primeira vez. Aconselha-se para uma agradável leitura, na primeira vez, pular os algoritmos e as descrições técnicas detalhadas.

2 - Conceitos básicos

2.1 - Fases do projeto de circuito integrado

O projeto de um circuito integrado é uma tarefa complexa e que envolve muitos passos. De forma resumida, um projeto pode seguir os seguintes passos [Schmitz, 85]:

a) especificação funcional:

Neste passo se define o circuito integrado em termos de suas características de entrada e saida: os pinos e sua função, a potência dissipada, as relações lógicas e temporais dos sinais.

b) detalhamento logico:

Quando a definição funcional esta pronta, o próximo passo consiste em fazer o detalhamento dos circuitos internos em termos de portas lógicas.

c) especificação da rede de transistores:

O nível de refinamento seguinte é a criação de uma rede de transistores equivalente à rede lógica.

d) layout:

Neste passo, cada um dos transistores é dimensionado, alocado e interconectado com o resto do circuito. Esta é a fase mais demorada do projeto.

e) verificações geométricas do circuito.

Quando o layout está concluido é submetido a uma verificação das dimensões e espaçamento dos elementos.

À medida que o projeto vai sendo executado são realizadas simulações em cada uma das etapas, para garantir sua correção, antes que cada nova etapa seja iniciada. A simulação normalmente exige que os dados estejam organizados de uma maneira adequada, e para isso é necessário algum tipo de tradução ou compilação.

Na verdade, o projeto não segue os passos acima de uma forma estritamente sequencial. Algumas etapas podem ser realizadas em

paralelo e, dependendo dos resultados de alguma etapa, os passos anteriores podem ter que ser reexecutados.

O nível de complexidade de cada etapa exige que se usem ferramentas computadorizadas para auxiliar nas diversas tarefas (CAD). Cada uma destas etapas, genericamente falando, necessita de programas para:

. edição - entrada e modificação de informações
. verificação - teste de correção de construção
. tradução - conversão de formatos ou linguagens
. simulação - teste dinâmico do modelo do sistema

2.2 - Estratégias de layout e ferramentas de CAD

A medida que os projetos aumentam de complexidade, o problema de layout aumenta muito. Atualmente são utilizadas diversas estratégias para layout do circuito, cada uma delas procurando diminuir o tempo do projeto e a redução do custo.

Para obter este resultado, é usado algum nível de padronização. Os exemplos mais comuns de padronização são mostrados na tabela a seguir [Ohtsuki, 86]:

PADRONIZAÇÃO	ITEM PADRONIZADO	ESTILO DE PROJETO
Elemento	Regras de projeto	Full custom
Transistor	Simbolização de transistores, via furos, conexões, etc...	Layout Simbólico
Célula	Tamanho da célula e fiação	Standard Cell
Blocos	Lugar dos terminais dos blocos e fiação	Building Blocks
Células e blocos	Colocação de células e blocos	Gate array
Fiação	Colocação dos transistores	PLA, ROM

A escolha do nível de padronização a utilizar se traduz num balanceamento área de silício x custo x desempenho: quanto mais alto o nível de padronização, maior é a area gasta em silício e menor é o desempenho, mas menor também são o custo e tempo do projeto.

Independentemente de qualquer outra consideração, o objetivo final do projeto de um circuito integrado é a obtenção das máscaras para fabricação. Em última instância recai-se sempre num problema gráfico: organizar os elementos do projeto em um desenho. Este trabalho quase sempre é suportado por um programa de edição gráfica, que tem a função de permitir ao projetista posicionar e interligar os elementos do seu projeto dentro da area de silício.

A edição gráfica é uma tarefa muito demorada, e eventualmente o tempo necessário pode ser diminuído através da utilização de elementos previamente criados (bibliotecas de células) e ferramentas automáticas de alocação e interligação [Barbosa, 87].

2.3 - Interface com a fábrica do circuito: do layout para as máscaras.

Vamos agora dar alguns detalhes adicionais sobre tecnologia de circuitos integrados que serão importantes para o entendimento geral deste trabalho.

Como foi dito, o layout é o desenho dos elementos componentes do circuito. Um layout consta de elementos geométricos (polígonos) que são dispostos de tal forma a implementar uma certa função. Um circuito integrado é consituído de diversas camadas (algo entre 5 e 20). Cada polígono é associado a uma camada.

Os layouts podem ser criados à mão (desenhados em papel), ou atraves de um editor gráfico. Depois de criados é necessário fazer a tradução da geometria dos polígonos (posição e forma) para um formato legível pelo computador. Existem alguns formatos padronizados, dos quais os mais conhecido são CIF (Caltech Intermediate Format) [Mead, 80] e GDS-2 [Elliott, 86].

Para assegurar que a tradução foi feita corretamente (e também para documentação), é comum transcrever para papel ou acetato, atraves de uma plotadora ou impressora gráfica, algumas (ou todas) as camadas do layout. Nesta transcrição, são utilizados padrões para diferenciar as diversas camadas (por exemplo, cor ou hachuras).

Existem diversas tecnologias para realizar a impressão do desenho em silício. Nas tecnologias mais conhecidas cria-se, para cada camada, um negativo fotográfico numa escala conveniente (10 ou 20 vezes o tamanho real), conhecido como retículo. O retículo pode ser obtido atraves de diversos métodos, por exemplo:

.com fotoplotadora a laser:

Este equipamento opera com um cilindro sobre o qual se coloca uma película fotográfica. O cilindro fica continuamente girando e um laser vai produzindo de forma rastreada (linha a linha), a reprodução do desenho. A foto-plotadora a laser é controlada por fitas magnéticas contendo a imagem rastreada em linhas (bit-map) de cada camada do layout.

. com fotoplotadoras MANN (fabricante do equipamento):

Nestes equipamentos um feixe de luz de abertura variável vai expondo retângulos sobre um filme. Estes retângulos podem ser desenhados aleatoriamente, embora a máquina tenha um método mais eficiente de trabalhar (geralmente manter fixo o tamanho dos retângulos e processar sequencialmente na direção X). A fotoplotadora é comandada por uma fita magnética contendo a lista dos retângulos a desenhar.

. outras formas:

Em alguns casos usam-se feixes de eletrons ou raios X para imprimir o negativo fotográfico.

Estes processos são executados normalmente por empresas

que recebem a descrição do layout em um formato textual (CIF ou GDS-2) e geram o retículo. Muitas vezes essas empresas são sub-contratadas pelo próprio fabricante do circuito integrado (fundidora de silício) que dessa forma pode receber o layout a produzir diretamente de quem o projetou.

Uma vez criados os retículos, são geradas as máscaras para cada camada. A máscara é um negativo (ou positivo) fotográfico, produzido sobre uma placa de vidro, que será usado no processo de fabricação. Este negativo contém o retículo repetido dezenas ou centenas de vezes, e é gerado a partir de um equipamento chamado foto-repetidora (step-and-repeat).

Cada máscara fotográfica é finalmente usada entao num processo de micro-fotolitografia para produzir a impressão em silício e metal, dos polígonos do layout. Uma referência muito completa sobre essas tecnicas é [Elliott, 86].

2.4 - Características do processo de edição no estilo "full custom"

Vamos tratar aqui da produção do layout num processo full-custom, que é aquele em que o projetista manuseia todos os detalhes de construção e montagem de um layout.

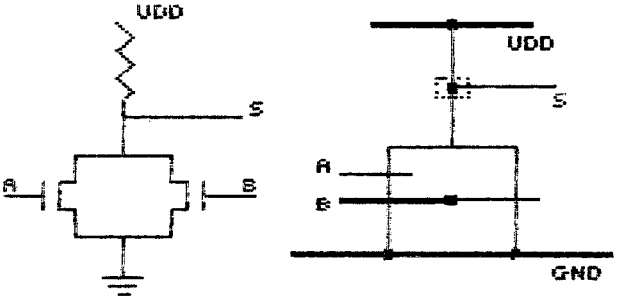
As abordagens para a criação do layout, dependendo se o projeto é grande ou pequeno, podem ser diferentes.

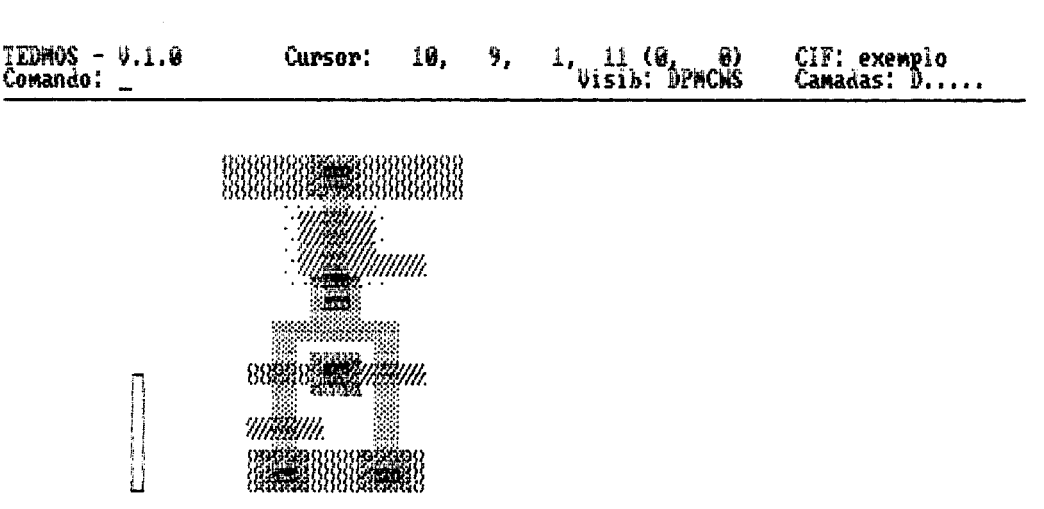
Num projeto pequeno (composto de dezenas de elementos), faz-se um esboço em papel, a nível dos componentes do circuito (por exemplo transistores e capacitores). O passo seguinte é desenhar estes elementos com o formato e escala conveniente, como na figura 2.1. Este processo pode ser feito de forma manual ou através de um editor de layout. Deve ser feita também a transcrição do layout para o formato intermediario (de forma manual ou automaticamente pelo editor).

Num projeto grande, é também necessário fazer a divisão do sistema em blocos funcionais, que vão ser desenhados independen-

temente e depois agrupados. Este processo é equivalente à construção de um programa grande de computador. Estes problemas serão vistos com mais detalhe no cap. 6.

O desenho do layout é uma tarefa extremamente sujeita a erros. Para se ter uma idéia da complexidade de um projeto, vamos imaginar, por exemplo, um microprocessador de 16 bits contendo cerca de 50.000 transistores. Cada um destes transistores usa aproximadamente 10 formas geométricas. Em consequência disso, os 500.000 elementos deste circuito devem estar perfeitamente dimensionados e posicionados. A transcrição para a linguagem de intercâmbio (CIF ou GDS-2) também deve ser realizada sem erros.





Arranjo de transistores, esboco de layout e layout final

figura 2.1

Daí surge a necessidade de utilizar programas de computador para auxílio à construção do desenho e sua verificação. Destes programas um dos mais importantes é o editor de layout, que é o

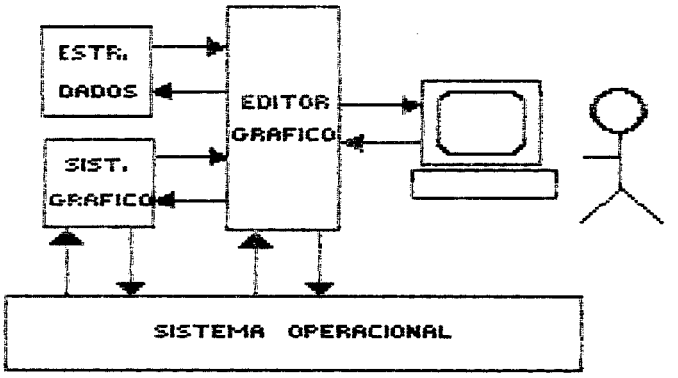
objeto deste trabalho.

2.5 - O que é um editor de layout

Essencialmente um editor de layout é um programa de computador para auxílio a desenho. Existem duas maneiras básicas de especificar um desenho num editor:

- proceduralmente, onde o desenho é descrito atraves de uma linguagem gráfica [Unicamp, LPG].

- gráficamente, onde o projetista interage com um terminal gráfico e produz o desenho, entrando as informações gráficas através de equipamentos de entrada tradicionais, por exemplo, teclado, mouse ou tablete (figura 2.2).



Ambiente interativo de edição
figura 2.2

Neste trabalho, estamos interessados basicamente no estilo de edição gráfica interativa, que é a característica de todos os editores modernos.

Existem alguns elementos que diferenciam um editor de outro. Os aspectos mais importantes são aqueles que o usuário vê, ou seja, aqueles com os quais um operador que vai usar o programa tem realmente que interagir. O editor deve atender às necessidades do usuário, e neste aspecto se destacam:

a) o conjunto de funções disponíveis

O editor deve ter um conjunto de funções que seja adequado à grande maioria de usos. Por outro lado, a existência de um número muito grande de funções torna o editor um "elefante branco", cujo uso é muito difícil. A situação adequada é aquela em que exista um conjunto de funções simples, que, operadas em sequencia, permitam atender a todas as necessidades mais complexas do usuário.

Um conjunto mínimo de funções que estes editores interativos oferecem são [Borges, 83]:

Pintura e apagamento de elementos de desenho
Movimentação, ajuste e cópia de trechos de layout
Escalas de edição e seleção de áreas de interesse
Visibilidade parcial de camadas
Armazenamento

b) a ernogomia do sistema

O usuário normalmente passa horas, dias ou mesmo semanas executando uma tarefa de edição. Assim, o editor deve ser muito confortável para o usuário, o que implica em fatores que não são muito fáceis de conseguir, especialmente porque cada pessoa opera de forma diferente.

- o usuário deve poder ativar rapidamente os comandos, com um mínimo de esforço
- o editor deve executar rapidamente as tarefas mais comuns
- deve existir um padrão na ativação de todos os comandos
- a execução de todos os comandos, em todas as situações, deve ser realizada num tempo razoável.

E' impossível construir um "editor perfeito". Existe uma série de objetivos que são conflitantes. Por exemplo,

- a facilidade de uso versus o número de funções;
- o tempo de resposta versus a complexidade exigida;
- a qualidade de apresentação do layout versus os custos do

equipamento gráfico e computacional exigido.

Felizmente (ou infelizmente) para o projetista de editores gráficos, os usuários se acostumam com tudo. Uma vez o usuário treinado, qualquer conjunto razoável de comandos no final das contas, está bem... Mais que isso, um usuário treinado num certo editor, ao aprender a operar um novo editor, normalmente acha este novo mais complicado de operar do que aquele com que ele estava habituado...

c) a adaptabilidade a diferentes tecnologias.

A tecnologia de circuitos integrados evolui com uma rapidez impressionante. Um editor gráfico deve estar preparado para esta evolução. Não adianta, por exemplo, (já aconteceu conosco) criar um editor para uma única tecnologia. A durabilidade dele não será maior que alguns meses.

Alguns deste fatores tecnológicos que tem maior importância são o número de camadas, o tamanho máximo do layout, a representação das camadas na tela e as regras geométricas.

d) adaptabilidade a outras ferramentas de CAD

O layout produzido pelo editor normalmente será submetido a ferramentas posteriores, como:

. verificador de regras geométricas
. extrator de circuitos ,
. simuladores elétricos e lógicos
. avaliadores estáticos e dinâmicos
. comandos do sistema operacional do computador

O editor deve estar preparado de duas maneiras para atender a isto. Primeiramente, a saída do editor deve ser facilmente conversível para um formato que estes programas possam entender. Em segundo lugar, o editor deve poder chamar diretamente algumas dessas ferramentas, e com elas interagir (especialmente com aquelas mais usadas), dando ao usuário um ambiente de trabalho mais con-

fortável.

2.6 - Características dos equipamentos influenciando o editor

Para o usuário, a parte física do editor é composto da tela e do meio de interação. Assim, os principais fatores que determinam a qualidade de apresentação são:

a) a qualidade física do vídeo, que é determinada por fatores como resolução (número de pontos por unidade de área), número de cores simultâneas, presença ou ausência de cintilamento, tamanho da tela, etc...

b) a disponibilidade de equipamentos de entrada adequados: teclado, mouse, tablete, digitalizador, light-pen, etc...

c) o tempo de resposta e pintura na tela, que são funções da velocidade do processador e formas de conexão da estação de trabalho com o processador.

Para o projetista do editor, entretanto, o sistema pode ter muitos componentes mais. De forma geral, essas sao as perguntas que tem que ser respondidas durante a definição do editor:

a) Características do sistema gráfico:
   - Será um mainframe conectado a um terminal gráfico burro (sem processamento local) ?
   - Ou será uma estação de trabalho com um processador autônomo capaz de realizar todo (ou parte do) processo de edição ?
   - Ou será um microcomputador dedicado ?

b) Características do terminal gráfico:
   - Terá planos de memória para armazenar a imagem com as cores, manipuláveis independentemente ?
   - Terá palhetas de cor fixa ou variável ?
   - Terá processador gráfico dedicado para algumas funções específicas ?
   - Estará ligado através de interface serial, paralela ou DMA?

. Será lento ou rápido na execução de que funções basicas ?

c) Características do software gráfico basico:

. O software basico gráfico disponível é adequado para o editor (quase sempre não é) ?

. Existe um padrão de interface gráfica que compatibilize todos os terminais ?

. Até que ponto a interface gráfica baixa o desempenho do sistema ?

. A interligação de que equipamentos de entrada é suportada ?

2.7 - Elementos fundamentais dos editores gráficos para circuitos integrados

Os editores manuseiam simultaneamente 4 elementos principais: a representação gráfica na tela, a representação interna das informações (estrutura de dados), o armazenamento permanente (arquivos, banco de dados) e os meios de entrada e interação (equipamentos interativos, como teclado e mouse, menus de vários tipos e cursores).

O controle simultâneo de todos estes elementos não é uma operação trivial, dada a restrição de que se deseja dar o melhor desempenho ao editor, tanto do ponto de vista do usuário, quanto do ponto de vista do ambiente (computador + sistema gráfico) no qual ele executa.

A representação gráfica é a maneira atraves da qual o usuário vê os elementos do circuito. Esta representação está diretamente influenciada pelas características do terminal gráfico utilizado, não so' no que tange às suas potencialidades gráficas (o número de cores disponíveis, o número de pontos na tela e o conjunto de funções que o seu firmware pode realizar) como o desempenho exigido do editor.

A estrutura de dados é a forma que o editor usa para manipular as informações do layout. Como o editor é interativo, o tempo de resposta de cada função é diretamente proporcional à facilidade de acesso às informações que ela precisa. Portanto, o acesso às

informações deve ser o mais eficiente possível.

Estas informações são especialmente volumosas, a tal ponto que frequentemente não cabem na memoria principal do computador, sendo o editor obrigado a operar com algum esquema de uso de disco, que mantenha fora da memória principal os dados que não estão momentaneamente sendo utilizados.

Além disso, se o editor é operado num ambiente multiprogramado, uma estrutura de dados eficiente pode diminuir muito o tempo de resposta, caso se garanta que a execução das funções seja menor que a fatia de tempo dada pelo sistema operacional a cada processo.

Os dados manipulados pelo editor devem ser armazenados de forma permanente, de modo a garantir uma série de requisitos: robustez do armazenamento, rapidez de carga, possibilidade de backups, transportabilidade, etc... O uso de um sistema de banco de dados para este armazenamento muitas vezes é conveniente, pois permite além da confiabilidade, o aumento da facilidade de integração de todas as etapas do projeto. Apesar disso, se deve pesar muito bem os aspectos de queda de desempenho envolvidos nesta escolha.

O desenho é produzido com o controle do operador através de um ou mais meios de entrada de dados gráficos. Os principais equipamentos são o Teclado, o Mouse e o Tablete. Associados a estes equipamentos estão os elementos de software para interação: cursores, menus, ícones, prompts, perguntas, etc... Estes elementos devem se organizar de tal forma a produzir o maior conforto e rapidez na ativação das ações do editor.

Concluindo, esses 4 elementos se apresentam em muitas formas e com aspectos complexos de interação. Cada editor gráfico que conhecemos os utiliza de uma forma diferente.

Vamos portanto fazer neste trabalho um estudo que permita delimitar as situações onde cada método é ou deve ser utilizado, estudando-se tanto vantagens quanto restrições. O conhecimento uti-

lizado foi adquirido pelo autor na construção de (pelo menos) cinco editores gráficos para circuitos integrados no NCE/UFRJ.

3. Editores de memória de tela

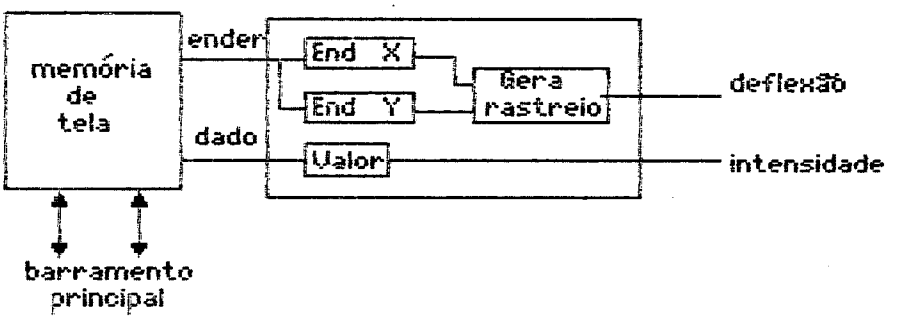
Os editores de memória de tela são o tipo mais simples de editor gráfico para circuitos integrados, em termos de implementação. Eles usam um dos modelos que dá mais conforto ao usuário, pela simplicidade de operação e pela associação com operações de pintura, tesoura e cola. Tem entretanto a desvantagem de exigir que o hardware tenha certas características especiais, que só são encontradas em alguns tipos de estações de trabalho.

3.1 - Características do hardware necessário

Nos equipamentos que são capazes de suportar este estilo de editor existe uma tela gráfica que é uma matriz de "l" linhas e "c" colunas (figura 3.1). Cada um dos elementos desta matriz (que chamamos pixel) é diretamente controlado pelo estado um elemento correspondente de uma matriz na memória do computador, chamada mapa de bits (bitmap). Desta forma, para pintar um ponto na tela, simplesmente se altera a posição correspondente na matriz da memória, segundo a fórmula

endmem = linha * num_cols_por_linha + coluna

Esta memória de vídeo, em alguns equipamentos pode ser acessada como uma memória comum, e em outros equipamentos indiretamente através de instruções de entrada e saida [Silva, 82].



hardware de um vídeo gráfico

figura 3.1

Podemos através destes equipamentos, criar duas primitivas de desenho, que podem ser suportadas diretamente por hardware ou implementadas como rotinas de software:

cor = LEPONTO (linha, coluna)
ESCREVEPONTO (linha, coluna, cor)

O monitor de vídeo a cores, normalmente trabalha num padrão de cores chamado RGB ("red, green, blue"). Do equipamento computacional saem tres sinais analógicos, que são obtidos a partir dos valores da memória, utilizando uma função eletrônica, conhecida por LUT (ou "vídeo lookup table"), que associa um valor na memória 'a cor que será apresentada na tela.

Nos sistemas que são usados para projeto de circuitos integrados, um elemento da memória de vídeo frequentemente contém 8 bits, e a LUT permite que seja escolhida para cada uma das 256 possibilidades que um pixel pode ter, uma dentre 4096 cores.

Obs.: Um pixel não é necessariamente um elemento quadrado, ou seja uma área presumivelmente quadrada, com 20 x 20 pixels possivelmente será vista como um retângulo na tela..

Sobre a tela pode ser posicionado um cursor gráfico, que dependendo do hardware pode ser apresentado de várias formas. Alguns tipos de cursores são mostrados na figura 3.2. Os cursores podem ser movidos na tela (ou seja, apagados e redesenhados em nova posição) sem destruir o conteúdo das posições onde eles são colocados.

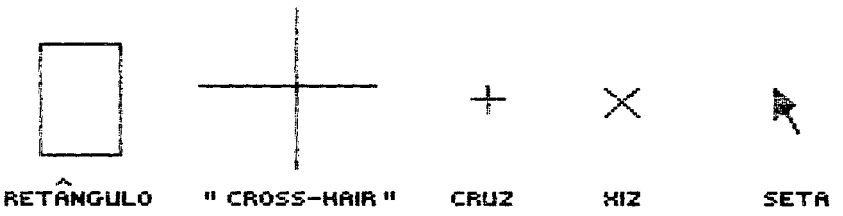
Os cursores muitas vezes podem ser criados por hardware, mas em geral são gerados utilizando-se um dos seguintes métodos:

a) salva-se o conteúdo das posições da memória tela correspondentes 'as posições que o cursor vai ocupar. Desenha-se o cursor. Para apagar o cursor, reescreve-se o conteúdo salvo.

b) o cursor pode ser desenhado também fazendo-se um "ou-exclusivo" das posições da memória com um valor que represente uma cor conve-

niente (por exemplo, branco). Assim, nas posições da tela onde não há nada desenhado, aparece esta cor. Nas posições onde há algo desenhado aparece uma nova cor que é resultado do "ou exclusivo" da cor anterior com a da cor do cursor. Para apagar o cursor, basta desenhar novamente o cursor, pois o segundo "ou exclusivo" anula o efeito do primeiro ou seja

A xor B = C ==> C xor B = A



vários tipos de cursores usados em editores para projeto de circuitos integrados
figura 3.2

Os cursor mais adequado para o projeto de circuitos integrados é o retangular, pois a maior parte das operações que são realizadas referenciam áreas retangulares ou realizam algum tipo de alinhamento. Entretanto, em algumas situações, o uso de um outro tipo de cursor pode ser interessante (embora a funcionalidade destes outros cursores possa muitas vezes ser simulada pelo cursor retangular). Na falta de disponibilidade de um cursor retangular, o editor simula a existência dele através de algum artifício, como por exemplo a marcação de pontos na tela, ou a utilização de um reticulado.

3.2 - Apresentação dos dados na tela

A edição de um layout é feita sobre uma grade fictícia. Cada posicao desta grade é chamado lambda. Um lambda é representado por um conjunto retangular de pixels na tela. A unidade mínima de edição é portanto uma área de um lambda quadrado.

Obs.: Este conceito é diferente do conceito de lambda que é usa-

do em algumas tecnologias de construção de circuitos integrados, especialmente Mead & Conway. Nestas, a unidade lambda é usada como unidade de referência base para construção dos fios e transistores, e é possível a existência de elementos de tamanho não múltiplo de lambda (exemplo: implantação iônica NMOS).

A tela do computador apresenta um pedaço do layout. O projetista move um cursor retangular sobre a tela e, nos pontos desejados, manda o editor preencher este retângulo com um padrão colorido que representa uma ou mais máscaras de fabricação.

Desta forma a edição é vista como um processo de colagem [Borges, 87]. A tela pode ser imaginada como um quadro onde o projetista pode colar polígonos coloridos que, justapostos, formarão os padrões geométricos presentes nas máscaras do circuito integrado.

No layout de circuitos integrados é importante que cada combinação de camadas presentes num dado pixel seja representada por uma cor diferente, que permita ao projetista identificar quais os elementos presentes naquele ponto. Em geral, também, todas as combinações são, se não válidas, pelo menos permitidas. Desta forma, o número de cores que é necessário usar é igual a 2 elevado ao número de camadas..

é muito comum também que se use, especialmente para algumas camadas de superposição (como implantação iônica ou poço), ao invés de uma área totalmente pintada, o perímetro desta área. é possível também usar o artifício, em vídeos com poucas cores, de fazer uma combinação (hachura) das cores existentes, simulando desta forma um vídeo com mais cores.

O uso das hachuras ou perímetros, entretanto, complicam o processamento do editor, pois como se verá adiante, a estrutura de dados que é utilizada no processo de edição é o próprio bitmap. Por exemplo, para extrair do desenho as informações geométricas (como ocorre na conversão do layout para linguagem CIF), estes artifícios criam muita dificuldade.

3.3 - Funções do editor

3.3.1 - pintura e apagamento

A grande vantagem dos editores de memória de tela, é a fácil analogia que o usuário faz com o mesmo trabalho realizado de forma manual, em que são usados o lápis de cor, para pintar áreas, borracha para apagar, tesoura e cola para mover, esticar, e encolher, e copias xerox para replicar partes.

As funções mais simples, e também mais utilizadas são:

- mover e/ou alterar as dimensões do cursor para selecionar uma área do desenho.
- selecionar uma camada
- encher esta área com a cor referente àquela camada.

Na eventualidade de um erro, o usuário deve poder apagar um determinado trecho para redesenhar. Este apagamento deve poder ser realizado tanto sobre todas como sobre algumas camadas, uma vez que num desenho intrincado, um dos erros mais comuns é selecionar a camada errada e pintá-la. Ter que redesenhar uma área, completamente, muitas vezes é um trabalho muito custoso para o projetista. Em muitos casos também é desejável a existência de uma função "desfazer a última operação" (undo).

3.3.2 - operações com áreas

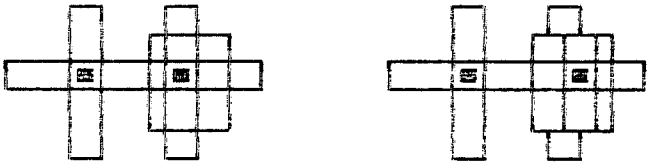
As operações com áreas são elementos que dão um grande potencial a um editor, no sentido de que permitem o posicionamento conveniente dos elementos, além de facilitar a criação de trechos similares do desenho (uma vez que os layouts têm quase sempre elementos repetidos).

As principais operações são:

- cópia - mover o conteúdo de um trecho do layout para outro.

. movimentação - identico a cópia, mas apagando o trecho original.
. ladrilhagem - formação de uma matriz, em que os elementos são repetidos no sentido x e y.
. ajuste - movimentação de um em um lambda, para acerto.

Destas opções, a mais usada é o ajuste, especialmente na fase final da criação de um layout, para compressão dos espaços vazios, ou abertura de espaço para inclusão de novos elementos. Normalmente a operação de ajuste não quebra os elementos da borda da região a ajustar, mas sim os estica, como mostrado na figura 3.3:



ajuste de áreas
figura 3.3

Como a matriz de edição está disponível diretamente, estas operações são trivialmente implementadas.

3.3.3 - Visibilidade parcial

Como o desenho é normalmente extremamente denso e complexo, é conveniente poder visualizar apenas parte das camadas. Para conseguir este efeito, pode-se agir de duas maneiras:

a) guardar a parte do desenho que se deseja ver com visibilidade parcial numa memória auxiliar e depois remover as camadas indesejadas. Após a visualização, recupera-se o layout original. Esta abordagem tem o inconveniente de não permitir que sejam feitas alterações no layout enquanto em visibilidade parcial.

b) fazer a reprogramação da LUT, como será descrito em 3.5.5.

3.3.4 - Anotações no desenho e colocação de nomes de nós

Durante o processo de criação de um circuito integrado é comum dar-se nome a pontos do layout (nós). Esses nomes são necessários para outras ferramentas de CAD, em especial, a simulação do circuito. Além dos nomes dos nós, é ainda interessante permitir que o usuário introduza anotações para documentação do circuito.

Esses nomes devem aparecer no vídeo de uma forma destacada, porém devem poder ser removidos para facilitar as tarefas de edição. No caso de uma movimentação e ajuste de áreas, esses nomes devem acompanhar a área movida (mas normalmente não numa operação de cópia).

3.3.5 - Impressão e plotagem

O desenho que está sendo criado deve poder ser copiado para impressora ou plotter, como será descrito em 3.6.

3.3.6 - Armazenamento permanente e comunicação com outros programas

Uma vez criado o layout ele deve ser armazenado, visando edições futuras ou a passagem para outras ferramentas de CAD. A forma de mapa de bits é muito simples de ser processada, porém inconveniente para armazenamento permanente, por ser muito extensa.

Usa-se para armazenamento permanente um formato mais compacto, dos quais o padrão mais conhecido é a linguagem intermediária CIF (Caltech Intermediate Format), no qual o layout é descrito em termos de seus elementos gráficos:

- camadas (layer)
- áreas retangulares (boxes)
- áreas circulares (round flash)
- polígonos (polygon)

Um layout descrito em CIF é um texto alfanumérico que pode ser processado por um analisador sintático muito simples [Hon, 80].

Para armazenamento temporário, outros formatos não padronizados podem ser utilizados visando maior economia de disco (por exemplo, armazenamento das informações em binário).

3.4 - Processos de entrada e interação

Durante o processo de edição, o usuário está continuamente executando a sequência: move cursor, ativa função, move cursor, ativa função, ... Vamos ver alguns tópicos relevantes destes dois procedimentos..

3.4.1 - movimentação do cursor

O cursor é movido utilizando-se um equipamento de entrada, em geral, teclado, mouse ou tablete.

No caso de movimentação por teclado, associa-se uma tecla (ou um conjunto de teclas) a cada um dos movimentos básicos (sobe, desce, esquerda, direita, amplia e reduz). Para facilitar a realização de movimentos grandes do cursor, pode-se usar uma tecla para selecionar a velocidade de movimentação.

obs.: Existem alguns sistemas que usam características especiais do teclado para produzir um movimento com aceleração crescente quando o dedo é mantido sobre a tecla de movimentação.

No caso do mouse, este, ao ser movimentado, envia ao computador a distância andada e a posição de cada um de seus botões (apertado ou solto). O editor deve usar estas informações para mover ou ampliar o cursor. A ampliação geralmente é realizada quando ao mover, um dos botões permanece apertado. Um outro botão indica que o cursor chegou ao seu lugar destino.

A rotina de tratamento do mouse, apesar de simples, deve levar em consideração alguns aspectos (muitas vezes previstos no "driver" fornecido pelo fabricante):

. debouncing dos botões
. histerese (eliminar tremidos da mão na movimentação ou

posicionamento)
- escalamento de movimentação (associação conveniente do passo da mão com o passo do cursor.
- tratamento de aceleração: um movimento rápido da mão realiza um salto grande do cursor.

No caso do tablete, o procedimento é similar ao do mouse, exceto que pelo fato do tablete informar a posição absoluta, o movimento pode ser mais rápido. Para posicionamentos finos, muitas vezes utilizam-se também duas escalas diferentes na conversão das coordenadas do tablete para a tela..

3.4.2 - Ativação de funções

Cada editor opera de uma forma diferente com relação à forma com que as funções são ativadas. Essas formas dependem essencialmente do software básico disponível. Os processos mais usados são os seguintes:

a) Comandos abreviados

Reserva-se algumas linhas do vídeo para comunicação alfanumérica. Os comandos são escritos pelo usuário, geralmente de forma abreviada nesta área. O editor pode ajudar o usuário solicitando parâmetros e fornecendo dicas sobre os comandos.

b) Popup menus

Uma vez atingida a posição do mouse, é desenhado um menu com opções na tela. O usuário move um cursor (que normalmente não é o mesmo retangular) e escolhe a opção desejada neste menu. Caso essa função tenha parâmetros, esses também são escolhidos com novos menus que se superpõem aos outros menus anteriores.

Na implementação dos popup menus podem ser usadas duas alternativas: ou é reservada uma área da tela para os menus, no caso mais comum, à direita do desenho, o que diminui ligeiramente a área disponível para edição, ou se desenha os menus diretamente sobre o desenho. Neste último caso, o software básico apenas

salva o conteúdo daquela região do vídeo na memória do computador, e depois de escolhida a função, restaura aquelas posições.

c) comandos rápidos

Alguns comandos são executados muito frequentemente. E' o caso por exemplo de pintar áreas e selecionar camadas. Neste caso, o editor deve ter um tratamento diferenciado para estas funções. Uma opção é usar algum botão da maquina ou do periférico de localização, ou alguma tecla especial.

O editor tem a obrigação de ser muito rápido tanto na ativação quanto na execução destes comandos rápidos. Além disso, a ativação destes comandos deve ser ergonomicamente estudada para permitir que a passagem do processo de movimentação e o de ativação destes comandos utilize o menor esforço muscular possível.

d) feed backs ao usuário

O sistema deve sempre dar automaticamente informações sobre:

. coordenadas do cursor
. camadas selecionadas
. camadas visíveis.

Além disso, o sistema deve informar ao usuário através de "prompts" padronizados o estágio da execução dos comandos. Os comandos muito demorados devem ter algum indicador visual que indique a porcentagem do comando que ainda falta realizar.

e) sistemas com 2 telas

Muitos equipamentos possuem duas telas, uma gráfica, outra alfanumérica. Esta tela alfanumérica é muito util para o operador, pois a tela gráfica pode ser totalmente utilizada para edição, e podem ser mantidas muitas informações adicionais, além das citadas no item d, como a lista de opções disponíveis a cada momento, informações sobre o layout, comunicação com o sistema operacional, etc...

3.5 - Mecanismos básicos de funcionamento

3.5.1 - Posicionamento do cursor e pintura de uma área

O processo de edição basicamente é realizado através do posicionamento do cursor sobre uma área seguida da especificação de uma função a ser executada. Essa área é frequentemente retangular, dadas as características do projeto de circuitos integrados (criação de fios e transistores)..

Caso o cursor seja retangular, além da movimentação pode-se também executar a ampliação ou redução do cursor. Caso não seja, a especificação da área é feita posicionando o cursor em dois extremos opostos da área retangular.

A operação mais trivial a executar é a pintura desta área com uma certa cor (que é associada a uma camada do layout). Numa implementação simples, podemos supor que cada bit de um pixel seja responsável pela representação de uma camada, por exemplo, o bit 0, estaria associado a difusão, o bit 1 a polissilicio, o bit 2 a metal, e assim por diante. Para preencher a região do cursor com polissilicio, por exemplo, o que o editor gráfico deveria fazer é simplesmente ligar todos os bits 1 da área interna ao cursor.

3.5.2 - Programação da tabela de cores de hardware (LUT)

Ao realizar a pintura de uma área, aparecem no vídeo novas cores nas posições editadas. As cores devem ser semelhantes a uma das convenções utilizadas na literatura (verde para difusão, vermelho para polissilicio, azul para metal, etc...)..

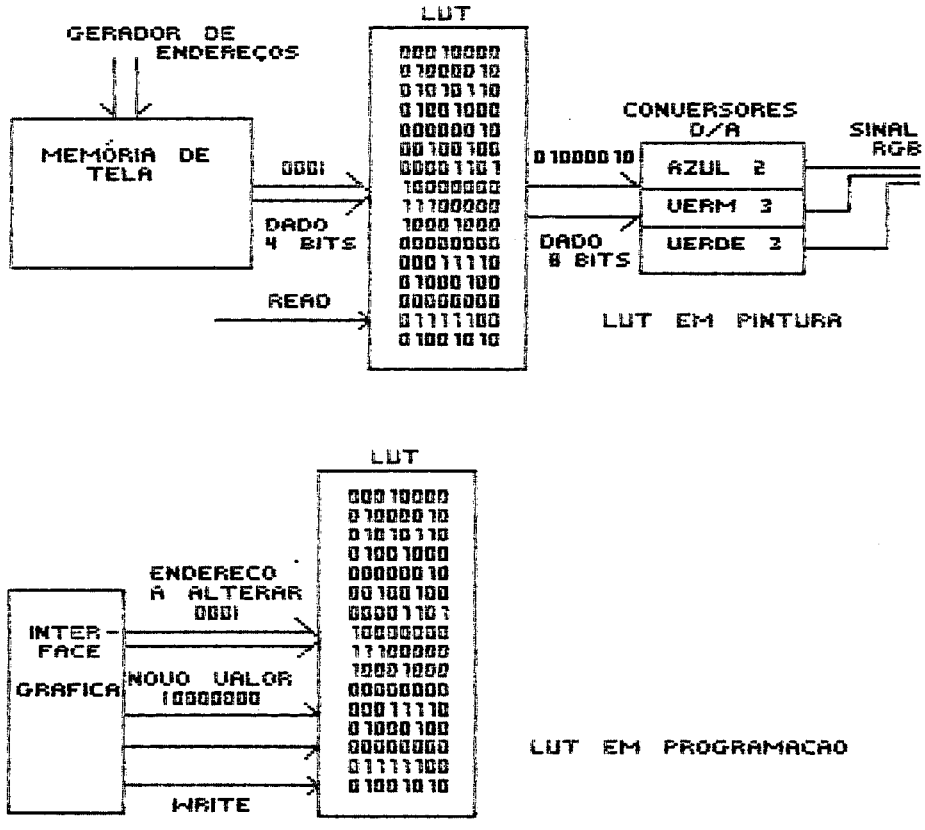


tabela de traducao de cores (LUT)

figura 3.4

A associação entre o conteúdo da memória da tela e a cor que aparece no vídeo é realizada por uma tabela chamada LUT (vídeo lookup table), que é implementada em hardware como uma memória de alta velocidade cujos endereçamento é feito pela saida da memória de vídeo e cuja saida alimenta diretamente os conversores analógico-digitais que produzem as tensões aceitas pelo monitor de vídeo.

Essa LUT deve ser carregada com esses valores convenientes para produzir cores que a literatura (ou a prática) indicam como adequadas. Por exemplo,

. carrega-se a posição 0 da LUT (equivalente a todos os bits em zero) com uma configuração que dê a cor preta, representando a ausência de camadas.
. a posição 1 da LUT com a configuração da camada de difusão

(verde)

. a posição 2 com a configuração de poli (vermelho)

. a posição 3 com a configuração que simule uma mistura de difusão com poli (talvez um vermelho esverdeado).

....

. a posição 7 com mistura de difusão, poli e metal (vermelho misturado com verde e com azul)

... e assim por diante para todas as combinações possíveis de camadas (se 7 camadas, 2 elevado a 7, 128 combinações).

Essa carga da LUT, é aparentemente uma coisa simples, mas é um problema para o implementador: o usuário está acostumado a usar lápis de cor, que é um sistema de geração de cores subtrativa, enquanto o vídeo é um emissor de luz e portanto um sistema aditivo. Num sistema subtrativo, azul mais vermelho dá roxo. Num sistema aditivo dá magenta.

O implementador tem duas hipóteses, ou descobre as configurações por tentativa e erro, ou usa um mapa de tradução aditivo-subtrativo [Foley, 83]. A primeira hipótese, funciona melhor (pelo menos o usuário fica mais satisfeito, especialmente se ele ajudar a escolher).

3.5.3 - Opções para tratamento do mapa de bits

O mapa de bits, neste estilo de editor, é a própria estrutura de dados, ou seja, durante o processo de edição, o programa apenas precisa alterar o "bitmap" e controlar a movimentação do cursor. No caso simples acima, em que a memória de vídeo associa cada pixel (8 bits) a um lambda, haveria um limite de 8 camadas na tecnologia.

Eventualmente seria possível usar mais de 1 pixel (por exemplo 4, dispostos dois em cima e dois em baixo) para representar um lambda, o que aumentaria muito o número máximo de camadas, mas diminuiria a area maxima do layout a editar. A criação da LUT, também seria muito mais complicada.

Obs.: Neste caso, para tradução de uma configuração de um lambda na tela para o equivalente em camadas, uma tabela de acesso direto, indexada pela configuração não seria adequada, uma vez que com 2 pixels apenas, esta tabela teria já 65536 posições, o que é um tamanho muito grande. Seria, talvez, mais prático utilizar algum método particular para a criação da LUT, por exemplo, algum tipo de hash, a fim de obter esta tradução. Por essas razões, esta implementação de vários pixels por lambda poucas vezes é usada.

3.5.4 - fator de escala

Trabalhar em fator de escala maior que 1, também é simples. Se o hardware do equipamento gráfico possuir opção de Zoom, isto é trivial, pois o programa continua vendo um pixel como um lambda. Caso o hardware não possua essa opção, usa-se uma replicação de pixels:

um lambda em escala 1: um pixel
um lambda em escala 2: dois por dois pixels (quadradinho)
um lambda em escala 3: tres por tres pixels

Naturalmente, cai a capacidade de armazenamento da imagem quando se trabalha desta última maneira, mas pode-se ter na tela simultaneamente partes do desenho em várias escalas. A edição em escala menor que 1 é impossível pois perde-se informações, embora, eventualmente este tipo de escala pode ser realizado, por exemplo para visualização global (mas não para alteração).

3.5.5 - Visibilidade

A visualização de somente algumas camadas pode ser conseguida de uma maneira não muito trivial. E' necessário reprogramar a LUT remapeando as cores, da seguinte forma:

Seja C uma configuração binária em que os bits referentes a todas as camadas visíveis estejam ligados e todos os outros desligados. Para todas as posições da LUT, calculamos o valor de E, o "e lógico" entre o endereço da posição e C. A seguir, substituímos a configuração desta posição da LUT pela configuração da posição E.

Por exemplo, vamos imaginar uma LUT programada da seguinte forma:

posição	em binário	conteúdo
0	00000000	0000
1	00000001	1013
2	00000010	3427
3	00000011	7858
4	00000100	8765
5	00000101	1753
6	00000110	3798
7	00000111	7786
	etc....	

Para promover a invisibilidade da camada 2 (referente ao bit 1, o segundo menos significativo) reprogramamos todas as posições que em binário tem o bit 1 ligado, de forma a que cada posição seja equivalente aquela sem este bit. A posição 7, em binário 00000111 ficaria equivalente à posição 5 (00000101, desligamos o bit 1 de 00000111). A nova tabela do exemplo ficaria com

posição	em binário	conteúdo	
0	00000000	0000	
1	00000001	1013	
2	00000010	0000	(equivalente a 00000000)
3	00000011	1013	(equivalente a 00000001)
4	00000100	8765	
5	00000101	1753	
6	00000110	8765	(equivalente a 00000100)
7	00000111	1753	(equivalente a 00000101)
	etc....		

O algoritmo de reprogramação da LUT para visibilidade é o seguinte::

```
seja PADRAOLUT [0..MAXLUT] o padrão de cada posição da LUT
     INVIS o conjunto das camadas invisíveis
     BITCAM o bit associado a uma determinada camada
     MÁSCARA := 1111111111....

para toda camada C pertencente a INVIS
     MASCARA := MASCARA xor BITCAM[C]
para toda posição P da LUT
     LUT[P] := PADRAOLUT [P and MASCARA]
```

3.5.6 - Manipulação dos nomes dos nos e dos comentários

A anotação do nome dos nós e de textos de comentário sobre o layout que está sendo produzido causa um problema mais ou menos complicado: é necessaria a utilização de um bit a mais de cada posição do mapa de bits, e que é utilizado para escrita destes nomes. Os nomes são desenhados nesta "pseudo camada" utilizando-se um gerador de caracteres que tanto pode ser o do próprio terminal (se ele tiver a opção de escrever caracteres em somente um dos bits) ou desenhado pelo editor, utilizando uma rotina geradora de caracteres.

Para processamento desses nomes, o editor deve guardá-los numa tabela ou lista, que é processada de forma mais ou menos independente da geometria. E' importante notar que algumas operações, como por exemplo ajuste de áreas, não só alteram esta lista como também promovem a modificação da posição do nome no mapa de bits.

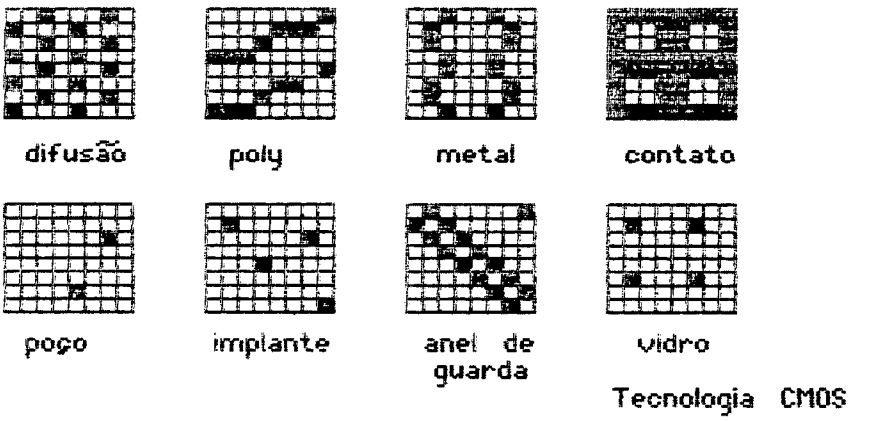
No caso do procedimento de impressão, o impressor pode produzir estes nomes de várias maneiras. A mais usual é usar o próprio gerador de caracteres da impressora, imprimindo na forma não gráfica. Outra forma é criar um padrão de hachura e imprimir gráficamente da mesma forma que uma camada qualquer.

3.6 - Impressão gráfica

As impressoras gráficas matriciais são equipamentos que se adaptam muito bem a estes editores. Nestes equipamentos, existem normalmente um conjunto vertical de agulhas (no mínimo 8), que são movimentadas horizontalmente numa linha, podendo ser ativadas em pontos determinados, e marcando o papel nesta posição. O papel é tracionado verticalmente e pode andar no mínimo o equivalente á distancia de uma agulha (em algumas impressoras menos que isso).

Para comandar essas impressoras é muito simples, basta enviar a sequência de pontos a imprimir em cada linha. E' comum que se enviem 8 linhas de cada vez, ou seja sequências de bytes cada um deles contendo a configuração das 8 agulhas.

Para copiar o desenho da tela para a impressora é necessário definir um padrão de hachuras para cada combinação de camadas da tela (ou seja para cada cor). Esta escolha não é muito difícil: define-se um padrão básico para cada camada isolada (ou seja para os valores potência de dois: 1,2,4,8,16,32...) da memória. As outras cores são obtidas como um "ou" destas configurações. Por exemplo a configuração 5 (difusão (bit 0, ou seja valor 1) + metal (bit 2, valor 4)) é obtida pela combinação ("ou") da hachura da configuração 1 a com a configuração 4. O padrão que é utilizado no NCE/UFRJ é mostrado na figura 3.5.



padrões de hachura utilizados no NCE/UFRJ

figura 3.5

A conversão para impressora é realizada criando-se na memória do computador uma matriz de bits diretamente associados às posições das agulhas no papel. Preenche-se esta matriz com os padrões de hachuras descritos acima. A matriz, depois de totalmente criada, é usada para criar sequências de ativação das agulhas da impressora.

A impressão a cores pode ser feita com uma impressora matricial policromática ou com uma impressora a jato de tinta. O padrão de hachura, neste caso deve ser outro: define-se um quadrado contendo uma área que tenha um número de pontos maior ou igual ão número de camadas mais importantes: por exemplo, em tecnologia NMOS, as camadas mais importantes são tres: difusão, polissilicio e metal. Pode-se usar, assim uma matriz de 2 por 2 -> 4 pontos. Cada um destes pontos vai ser colorido com uma camada. As camadas menos usadas são criadas usando-se um padrão de hachura semelhante ao processo monocromático.

Pode-se ainda utilizar um processo um pouco mais complexo de cor, se utilizarmos uma impressora matricial monocromática comum. Basta que se disponha de fitas carbonadas em cores para a impressora e se realizem diversas impressões sobre o mesmo papel, desenhando apenas uma camada de cada vez, com a fita correspondente. Este processo, embora muito mais trabalhoso produz um resultado final com a qualidade comparável com o obtido em uma impressora matricial a cores.

é possível também produzir uma cópia em plotter do desenho. Neste caso, é necessário extrair do desenho os retângulos ou os polígonos componentes para desenho no plotter, através do algoritmo mostrado em 3.7. A plotagem pode também ser realizada com facilidade a partir do arquivo de armazenamento permanente, uma vez que ele já contém os retângulos ou polígonos extraidos.

A saída em plotter tem o inconveniente de ser mais demorada do que a saída em impressora, especialmente no caso de se desejar preencher os retângulos com algum padrão de hachura. Por essa razão, aliado ao fato de que a qualidade das impressoras está se tornando cada vez melhor, existe uma tendência a não se usar mais

plotter para copiar desenhos de circuitos integrados.

3.7- Algoritmos para extração de geometria para armazenamento permanente

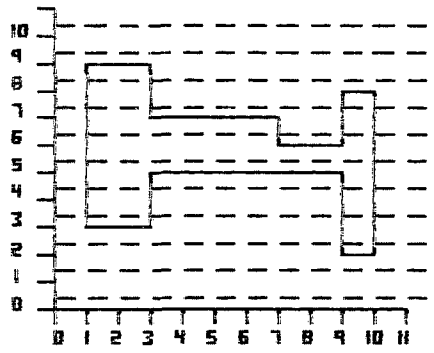
O próprio mapa de bits é um meio de armazenamento conveniente e usado diretamente em algumas ferramentas de CAD, como alguns programas de DRC ou extração de circuitos. Entretanto o tamanho do mapa de bits é muito grande (por exemplo, uma simples célula de 300 x 300 lambdas ocupa 90.000 bytes de memória !).

Por essa razão é conveniente que o circuito seja armazenado de uma forma mais compacta. O processo mais usado é extrair do mapa de bits as suas informações geométricas, ou seja, descobrir todos os polígonos ou todos os retângulos que formam cada uma das camadas.

O seguinte algoritmo extrai os retângulos que compoem o layout [Borges, 83].

Para cada camada do layout
1. inicializa uma lista de retângulos, que vai conter as coordenadas da base e sua altura
2. Para cada linha do mapa de bits
   2.1 obter o inicio e fim de todos os segmentos daquela camada
   2.2 para cada segmento obtido
      2.2.1 verificar a presença dele na lista de segmentos
      2.2.2 se existe,
            incrementa a altura do retângulo
         senão
            inclui novo retângulo na lista
   2.3 cada elemento da lista que não foi atualizado em 2.2 é removido da lista e produzido na saida, por exemplo em CIF.

A figura 3.6 ilustra o processo.



Y	X1, X2, DY	SAÍDA
0	---	---
1	---	---
2	9,10,1	---
3	1,3,1 9,10,2	---
4	1,3,2 9,10,3	---
5	1,10,1	1,3,2 9,10,3
6	1,7,1 9,10,1	1,10,1
7	1,3,1 9,10,2	1,7,1
8	1,3,2	9,10,2
9	---	1,3,2
10	---	---

processo de extração de retângulos

figura 3.6

3.8 - Deficiencias deste método

Pelo fato de utilizar a própria memória de tela para produzir e armazenar as informações do layout, existem algumas problemas intrínsecos deste estilo de editor.

a) Processamento de figuras não ortogonais

Este estilo de editor é muito adequado para um estilo de projeto "full-custom" conhecido por "geometria Manhattan", ou seja todas os retângulos componentes do desenho são posicionados de forma ortogonal.

Na criação de layouts compactos, entretanto, usam-se retângulos com outras inclinações, em especial 45 graus. Estes editores não seriam adequados para qualquer angulo, mas suportariam parcialmente o desenho a 45 graus. E' importante notar que a distância entre dois pontos a 45 graus seria raiz de 2 vezes maior do que na horizontal ou vertical.

Para tecnologias não MOS, bipolar por exemplo, é frequente o uso de círculos e coroas circulares. Por aproximação, este estilo de editor também pode ser utilizado. O algoritmo abaixo, que foi baseado no algoritmo de Brensenhan para aproximação de circunfe-

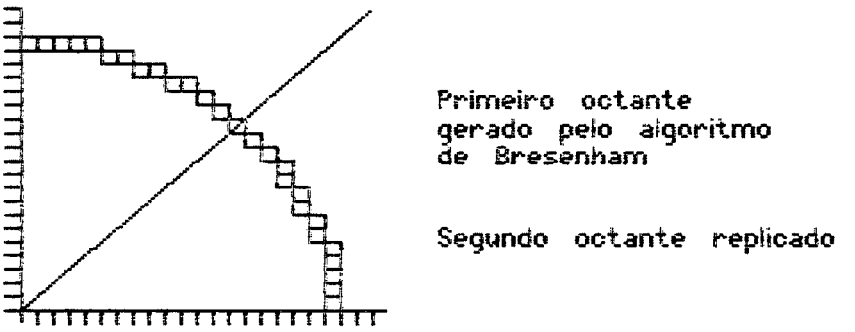
rencias [Foley, 82], pode ser usado para selecionar os pontos internos a um círculo. Uma extensão simples do algoritmo pode ser usado para desenho de coroas circulares.

O corpo do algoritmo calcula os pontos de um octante da circunferencia, como mostrado na figura 3.7. Como a circunferencia é simétrica, podem ser calculados os outros pontos.

```
rotina círculo (x,y,raio))

    1.  x = 0
        y = raio
        d = 3 - 2*raio
    2.  enquanto x < y
            2.1  enche (x,y,xc,yc)
            2.2  se d < 0 então
                     d = d + 4 * x + 6
                 senão
                     d = d + 4 * (x-y) + 10
                     y = y - 1
            2.3  x = x + 1
```

O procedimento "enche" liga os pontos correspondentes na horizontal. xc e yc são os pontos do centro do círculo.

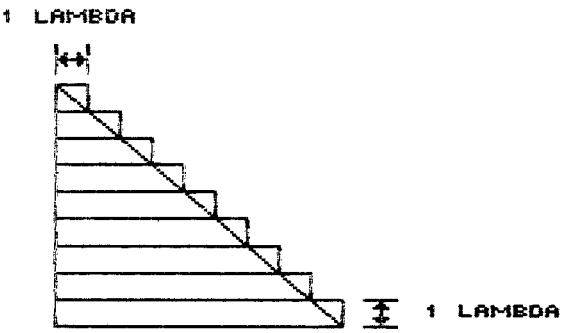
```
rotina enche (x,y, xc, yc)

    1.  liga os pontos entre  xc-x,yc+y e xc+x,yc+y
    2.  liga os pontos entre  xc-x,yc-y e xc+x,yc-y
    3.  liga os pontos entre  xc+y,yc+x e xc-y,yc+x
    4.  liga os pontos entre  xc+y,yc-x e xc-y,yc-x
```



tracado de circunferencia

figura 3.7

Tanto a geração de figuras a 45 graus quanto do círculo não são um problema serio, porém os algoritmos de extração não são capazes de interpreta-los de forma inteligente. Por exemplo, um triângulo com um lado a 45 graus seria convertido na sequência de retângulos mostrados na figura 3.8. O mesmo raciocínio vale para círculos e coroas circulares.



conversão de figuras não ortogonais para retângulos

figura 3.8

b) Impossibilidade de criação de layouts hierárquicoss

Como foi dito no cap. 2, um layout completo pode ser composto

de várias partes, e eventualmente cada parte composta de outras subpartes. Isto nos leva á idéia de criar o layout em partes que possam ser unidas, tanto por justaposição quanto por hierarquia.

A maior deficiencia deste estilo de edição é que não se pode criar um layout de forma hierárquica, ou seja, pode-se colocar num layout um bloco já criado, mas uma vez realizada esta operação, o bloco perde sua individualidade.

Este problema será visto com mais detalhe no cap. 6.

3.9 - O editor EDMOS

Foi construído por nós, em 1982, um editor gráfico destinado ao projeto de circuitos integrados em tecnologia NMOS, utilizando a metodologia descrita neste capítulo.

Este editor executava em um microcomputador EBC-SDE/42 da Empresa Brasileira de Computadores, que dispunha de uma interface videográfica com 32 cores e capacidade de piscagem de pontos, num total de 6 bits por ponto na tela. Esse computador utilizava o sistema operacional CP/M e o sistema foi construído em Fortran. O software básico de controle da interface, muito pequeno, foi também construído por nós, em assembler do microprocessador Z-80.

Entre as principais facilidades do EDMOS, havia a criação e apagamento de retângulos coloridos, ajuste, cópia, movimentação e replicação de trechos, espelhamento e rotação de áreas, e armazenamento em disco. Posteriormente foi incluída uma função de impressão em impressora matricial.

O EDMOS não era capaz de editar layouts maiores que a tela (240 x 400 lambdas em escala 1, 120 x 200 em escala 2, e assim por diante), mas era capaz de ter na tela, simultanteamente, desenhos em várias escalas, proporcionando por exemplo, operações de ampliação de áreas para visualização mais detalhada de trechos.

Como o computador possuia duas telas, uma gráfica e outra alfanumérica, os comandos eram dados na tela alfanumérica, abrevia-

dos por duas letras, ficando reservada a tela gráfica unicamente para mostrar o desenho e um cursor retangular, que também era comandado (posição e dimensões) pelo teclado.

O tempo de projeto da primeira versão EDMOS foi extremamente curto (pouco mais de 1 mes, 1 projetista), o que demonstra claramente a simplicidade da metodologia. O editor foi construído em cerca de 1500 linhas de programa incluindo linhas de comentário.

3.10 - Conclusão

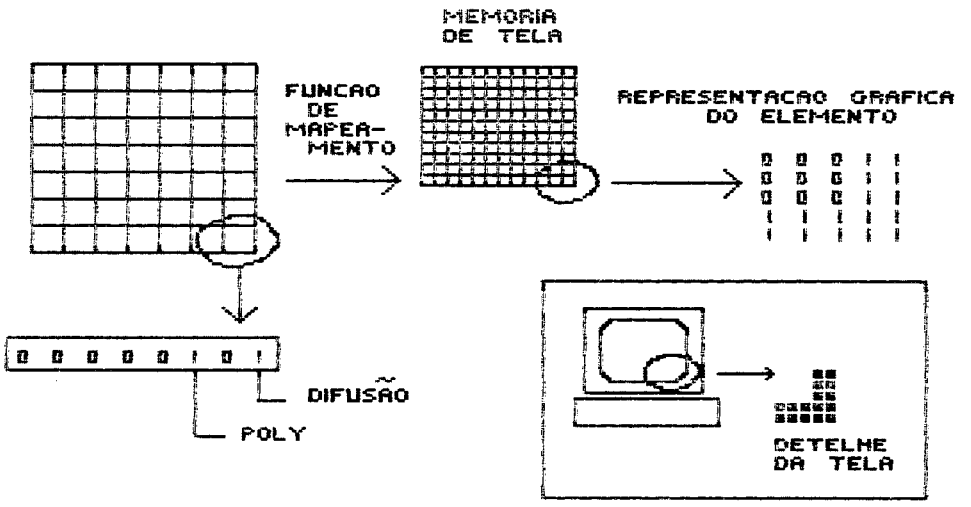
Analisamos neste capítulo o modelo de editor de memória de tela, abordando detalhes de implementação e algoritmos. Este estilo de editor tem muitas vantagens, especialmente os aspectos de ergonomia. Pelo fato de exigir características especiais no hardware, tem seu uso limitado, embora a capacidade dos modernos microcomputadores seja muito próxima aos requisitos exigidos.

4 - Editores não hierárquicos matriciais

Os editores não hierárquicos matriciais formam o caso mais geral dos editores bitmap. Nestes editores, o mapa de bits fica representado na memória do computador e a imagem é apresentada num terminal à parte. Isso minimiza muito a dependência de aspectos particulares do hardware, e permite que um editor deste estilo possa ser executado praticamente em qualquer computador, desde que a ele esteja conectado um terminal ou vídeo gráfico.

4.1 - Estrutura geral

O mapa de bits é representado internamente por uma estrutura chamada matriz de edição. Cada elemento desta matriz é um vetor de bits contendo tantos bits quantas forem as camadas da tecnologia a editar. Desta forma não existe mais a restrição de número de camadas, pois a implementação pode usar tantos bits quantos sejam necessários (normalmente um número de bits que seja múltiplo do tamanho da palavra do computador, por questão de eficiência).



Associação entre a matriz de edição e a imagem da tela

figura 4.1

Numa parte da tela é feita a representação visual desta matriz. Cada elemento da matriz é mapeado num conjunto de pontos da tela, com uma aparência aproximadamente quadrada. A função de mapeamento entre cada elemento possível e sua representação na tela

é fornecida normalmente através de um arquivo de configurações alterável pelo operador.

A figura 4.1 apresenta um esquema simples que pode ser utilizado num sistema baseado em microcomputador, em que se tenha acesso, por programa, à memória do vídeo gráfico do computador.

A associação entre um elemento do bitmap e a representação na tela pode assumir muitas formas, o que depende do número de pontos e número de cores do vídeo. Podem ser usados padrões coloridos, hachuras, pontilhados [Borges, 87], e existem até mesmo implementações que utilizam somente caracteres alfanuméricos para esta representação [Lewis, 81].

Como a tela é relativamente pequena, na verdade é apresentado no vídeo apenas parte da matriz de edição, e o editor deve prover a seleção da área a ser mostrada e facilidades de fator de escala (seleção do número de pixels da tela que representam um lambda quadrado).

4.2 - Operação do editor

As operações, neste estilo de editor, são ativadas de forma idêntica ao descrito no capítulo anterior. Um cursor é movimentado até a posição desejada e é pedida uma ação ao editor. A diferença está na forma com que o editor realiza esta ação:

a) altera a matriz de edição, na memória
b) desenha na tela a área alterada, fazendo a associação de cores ou padronagens, como descrito em 4.1.

As funções são essencialmente as mesmas descritas no capítulo anterior, exceto que o editor deve poder operar em fatores de escala ou seja, associar um lambda quadrado do layout a um conjunto retangular de pixels de tamanho variável. Na tela podem estar representados um ou mais trechos selecionado do layout (janelas de edição).

O usuário então, pode "passear" dentro do layout (operação

conhecida como "pan") e alterar a escala de edição para ver os trechos do desenho com maior ou menor ampliação. Esta operação deve ser extremamente dinâmica, ou seja, a seleção da área e da escala de edição, e o desenho desta área na tela devem consumir o menor tempo possível.

Os fatores de escala não são necessariamente inteiros nem maiores que 1. Assim, por exemplo, é possível ter-se escalas reduzidas, para permitir ver todo layout em uma área pequena de tela.

Outra característica indispensável é a visibilidade. Em alguns casos, esta operação pode ser realizada por hardware, através da LUT, como descrito no capítulo 3, especialmente quando a associação da matriz de edição com a tela é simples, ou seja, cada configuração possível de um lambda é desenhado em uma única cor da tela. Entretanto, o mais comum é que a operação de visibilidade implique no redesenho total da área desejada (que eventualmente pode ser apenas parte do desenho, por questão de economia de tempo).

4.3 - Métodos para a obtenção da saída gráfica

Conforme já foi dito, os algoritmos de processamento, que neste estilo de editor, envolvem unicamente alterações no mapa de bits, são quase triviais. A dificuldade, consiste em codificar e desenhar na tela do terminal estas alterações.

Esse procedimento, entretanto, pode ser organizado, e agrupado em um software básico gráfico, de implementação pouco complexa. Os problemas que são resolvidos por esse software são os seguintes:

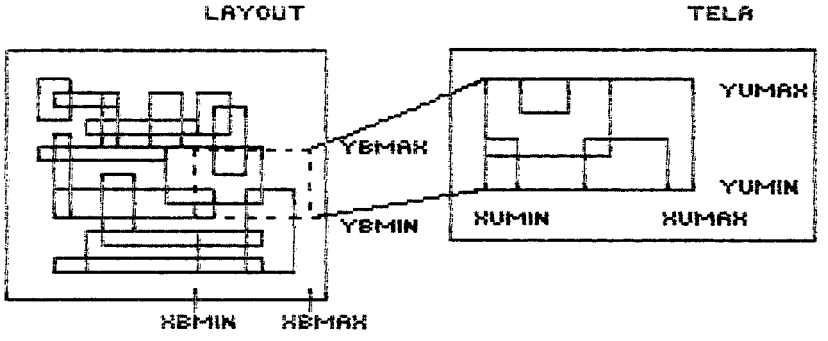
4.3.1 - Mapeamento do mapa de bits na tela

O problema aqui consiste em, dado um elemento no mapa de bits, descobrir qual o trecho da tela que ele ocupa.

A idéia consiste simplesmente em utilizar o conceito de "ja-

nela-viuporte" (window-viewport) descrito em [Newman, 78]. através deste método, define-se a região do mapa de bits que se deseja visualizar (janela), e uma viuporte ou seja, uma parte da tela que vai recebe-la de forma codificada. Uma regra de três simples fornece a correspondencia.

Imaginemos, por exemplo, a região compreendida entre os pontos (xbmin, ybmin) e (xbmax, ybmax), sendo visualizados na tela nos pontos da região compreendida entre (xvmin, yvmin) e (xvmax, yvmax).



Transformação janela-viuporte
figura 4.2

Um ponto (xb,yb) está relacionado com o ponto (xv, yv), através das seguintes relações.

$$\frac{(xbmax-xbmin+1)}{(xvmax-xvmin+1)} = \frac{(xb-xbmin)}{(xv-xvmin)}$$

$$\frac{(ybmax-ybmin+1)}{(yvmax-yvmin+1)} = \frac{(yb-ybmin)}{(yv-yvmin)}$$

Obs.:

Os "+1" da fórmula são devido ao fato de serem coordenadas inteiras, e as coordenadas máximas consideradas de forma inclusiva.

Muitas vezes, o usuário não fornece exatamente esses valores, e sim, uma constante conhecida por "fator de escala" ou "fator de ampliação" (E) que é utilizado na fórmula da seguinte maneira:

$$\frac{(xb - xbmin)}{(xv - xvmin)} = E * F1$$

$$\frac{(yb - ybmin)}{(yv - yvmin)} = E * F2$$

onde F1 e F2 são o número de pixels horizontais e verticais utilizados na representação de um lambda quadrado de layout.

4.3.2 - Representação na tela

Existem várias maneiras através das quais se pode representar os elementos do layout na tela, e cada uma delas tem diversas implicações. Além disso, várias delas podem ser utilizadas por um mesmo editor, na representação de diferentes camadas.

Mostramos a seguir os diversos métodos que podem ser usados para desenhar uma área que tenha sofrido alguma alteração.

a) criação através de padrões absolutos

Neste caso, cada combinação de camadas que constitui um elemento do mapa de bits está representada por um padrão absoluto, seja de cores, ou pontos, ou hachuras, ou outro qualquer. Neste caso, o padrão de cada combinação deve ser movido para a região destino da tela, tomando-se o cuidado de replica-lo para ocupar a região de E x F1 por E x F2 pixels na tela. Estes padrões devem estar contidos numa tabela de acesso direto.

Este é o estilo mais fácil, porém só é aplicável quando se

tem acesso rápido à memória de vídeo, caso contrário, o tempo de desenho fica absurdamente grande.

b) pintura de camadas com sobreposição por prioridade temporal

O software comanda o desenho das camadas presentes à região, em ordem inversa a uma prioridade preestabelecida. Assim, cada camada ao ser desenhada superpõe (substitui) as camadas anteriores que ocupam mesmas posições na imagem.

A grande vantagem deste método é a possibilidade do aproveitamento de funções do terminal, em especial, o desenho de retângulos cheios, ou o preenchimento de áreas ou polígonos com padronagens. Desta maneira o redesenho de uma área pode ser bastante rápido, pois o que tem que ser transmitido para o terminal, para cada camada, é simplesmente:

. a cor ou padronagem da camada
. a lista de polígonos que a compõem.

Entretanto, a lista de polígonos deve ser extraída do desenho e isso é um procedimento que, embora não consuma tempo de comunicação, consome tempo do processador. Normalmente, por simplicidade de programação, os polígonos são decompostos em retângulos, que são extraídos através de um algoritmo semelhante ao visto no cap. 3 para criação de arquivos de armazenamento.

c) pintura de camadas através de sobreposição combinacional

Caso o terminal de vídeo possua facilidade de inibição de bits na escrita de dados na memória de vídeo (write bit mask), é possível associar algumas das camadas a um dos bits da memória de vídeo. Neste caso, para criar um polígono em uma camada, inibe-se os bits da memória de vídeo que não são referentes a ela, e cria-se os polígonos diretamente, como descrito no item b.

Com isso, somado a uma programação conveniente da LUT, conforme foi explicado no cap. 3, pode-se obter um efeito de combinação de cores adequado, e bastante rapidez.

Este método tem as mesmas vantagens, e utiliza os mesmos algoritmos do método anterior.

d) representação de áreas pela envoltória

Na criação de layouts em tecnologias com muitas camadas, é comum que algumas das camadas tenham larga extensão, superpondo grandes áreas de layout, por exemplo, as regiões de poço e implante em tecnologia CMOS. A representação dessas áreas através de cores ou padrões tende a sobrecarregar desnecessariamente o desenho. Neste caso é comum que estas áreas sejam representadas apenas pela sua envoltória.

A envoltória pode ser representada na tela através de uma linha poligonal contínua, tracejada, pontilhada, etc. Este traçado é muito rápido, e geralmente realizado com comandos de traçado de reta previstos no próprio terminal. Para calcular essas envoltórias pode-se usar os algoritmos mostrados em 4.4.

e) representação de áreas através de hachuras

Trata-se aqui de preencher as áreas relativas a uma camada com um padrão de hachuras ou pontilhados. Estes preenchimentos podem ser criados através de sobreposição simples ou combinacional.

O problema desta representação é que as hachuras ou pontilhados devem ser enviados ao terminal gráfico através de primitivas que o terminal seja capaz de processar: retas ou pontos. Uma área relativamente pequena possuirá centenas ou mesmo milhares desses elementos. Assim, o tempo que será gasto será muito grande.

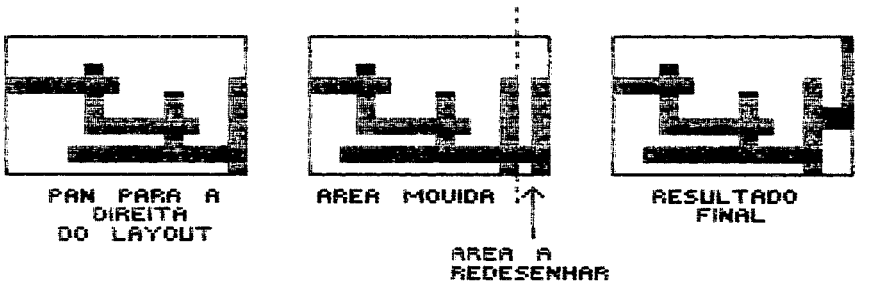
Por essa razão, só é conveniente usar essas representações quando o terminal gráfico já tem funções para fazer o preenchimento de polígonos através de inteligência local.

4.3.3 - Pan e zoom

Como a tela do terminal (ou a viewport selecionada) é pequena, o usuário vê-se freqüentemente com a necessidade de alterar a janela de visualização (pan) ou o fator de escala (zoom).

No caso de pan, podem ser usadas duas metodologias:

1) redesenha-se toda a viewport
2) move-se a imagem na memória de tela, redesenhando o que for necessário, como na figura 4.3.



Movimentação de dados na tela em PAN
figura 4.3

O segundo método é muito mais eficiente, mas envolve a existência de funções de movimentação de áreas no terminal gráfico.

Já no caso de zoom, também podem ser usados dois métodos:

1) redesenhar toda a viewport
2) usar facilidades de replicação de pixels (zoom) do próprio terminal.

O segundo método, embora muito rápido, tem uma seria restrição: as padronagens mudam (o terminal, fazendo replicação de pixels, "engorda" as padronagens). Por essa razão, só se usa em situações muito particulares, especialmente quando ao inves de padronagens, para representar as camadas, se utilizam cores puras.

4.4 - Algoritmos para extração de polígonos

Na maior parte das vezes, a extração de retângulos, através do algoritmo mostrado no capítulo 3 é plenamente suficiente para as operações de desenho. Entretanto há ocasiões, em que se deseja obter diretamente os polígonos, ou pelo menos, os pontos da envoltória dos polígonos. Para isso são utilizados alguns algoritmos especiais.

4.4.1.Obtenção trivial dos pontos da envoltória

Caso só se deseje obter uma lista de pontos da envoltória, referente a uma determinada camada, pode-se seguir o seguinte método:

```
1. Para todos os elementos da área desejada
     1.1 se o elemento possui a camada então
           1.1.1 se todos os seus vizinhos também tem
                 então   é um ponto do interior
                 senão   é um ponto da envoltória.
```

Em geometrias ortogonais ("Manhattan"), é necessário apenas testar os vizinhos de cima, baixo, esquerda e direita, caso contrário, todos os 8 vizinhos.

4.4.2. Varredura a 45 graus

Uma outra alternativa simples para layouts Mannhatan é varrer o layout a 45 graus, usando um método de paridade ("estou dentro, estou fora"):

```
1. para cada linha de varredura
     1.1  repete até o fim da linha
            1.1.1 varre até achar um ponto que possua a camada
            1.1.2 adiciona aquele ponto à lista de envoltória
```

1.1.3 varre até achar um ponto que não possua a camada

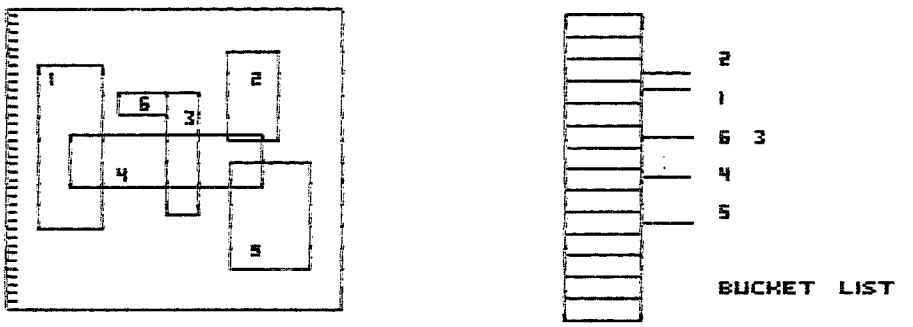
1.1.4 adiciona o ponto imediatamante anterior à lista

A varredura a 45 graus possui uma pequena falha, uma vez que deixa de detectar os pontos marcados com um x:

Esta falha pode ser corrigida acrescentando o seguinte teste adicional: sempre que se tiver a sequência: "alguns pontos dentro seguido de somente um ponto fora, depois mais pontos dentro", verificar também os pontos a -45 e 135 graus deste ponto fora. Caso algum desses pontos contenha a camada, também pertence à envoltória.

Como os terminais gráficos possuem comandos de traçado de linhas retas, por questão de eficiência, é conveniente que eles sejam utilizados para o traçado da envoltória, ao invés de pontos.

Para a transformação da lista de pontos, podem-se usar estruturas conhecida como "bucket array", uma para as linhas no sentido x e outra para as linhas y, como mostrado na figura 4.4. Cada elemento do vetor aponta para uma lista de retas (tecnologia ortogonal) que estão naquela coordenada.



"bucket array"
figura 4.4

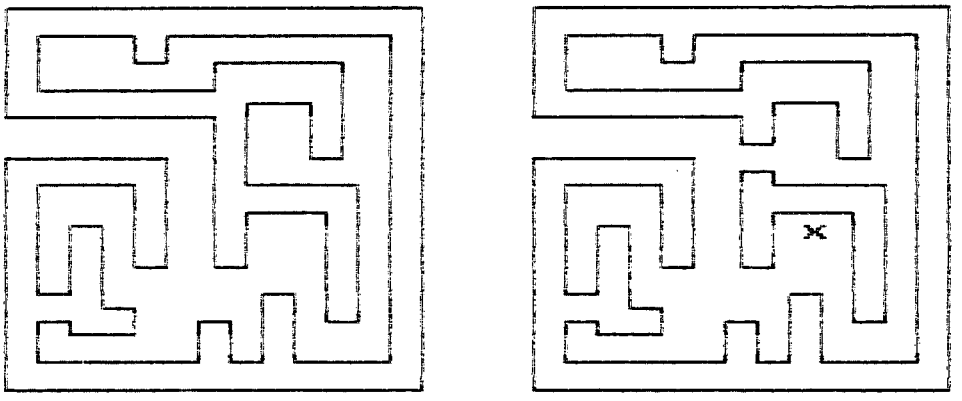
De cada reta, basta guardar o ponto de início e o tamanho. Ao fim da varredura, as retas com tamanho 1 são descartadas, e as outras enviadas ao terminal gráfico.

Caso se tenha restrições de memória para armazenar esta estrutura, é possível ainda à medida que a varredura vai sendo executada, eliminar da lista as retas com tamanho 1 que estão pelo menos a uma distância de dois pontos em diagonal, acima da linha de varredura.

4.4.3 - Algoritmo do "percurso de labirinto"

A idéia aqui é aproveitar a idéia geral do algoritmo de Tremeau [Dudeney, 17] para percurso de labirintos simplesmente conectados (vide figura 4.5).

Um labirinto simplesmente conectado é um tipo de labirinto em que nenhuma das paredes está separada do resto do labirinto. Então, se ficarmos sempre tocando a parede com a mão direita (ou esquerda), e seguirmos sempre encostados à parede (mesmo em corredores sem saída), deveremos percorrer todas as paredes do labirinto.



Um labirinto simplesmente conectado

Figura 4.5

O algoritmo que propomos funciona da seguinte maneira:

1) faz-se uma varredura geral num sentido qualquer, por exemplo todas as linhas de cima para baixo, procurando uma parede, ou seja, uma sequência de dois pontos um sem e outro com a camada. O ponto com a camada, certamente pertence a uma envoltória.

2) Aplica-se o algoritmo de Tremeau a partir deste ponto.

3) Os pontos dessa envoltória são marcados.

4) Continua-se a varredura, no item 1, a partir do ponto onde se havia parado.

A grande vantagem deste algoritmo é a eficiência, e o fato de que não é necessário nenhum procedimento inicial para gerar linhas a partir da lista de pontos da envoltória. A direção de percurso do algoritmo fornece o ângulo, e o número de passos na mesma direção fornece o tamanho da linha.

Para a execução do algoritmo, entretanto, existe a necessidade de mais um bit para fazer as marcações dos pontos de envoltória já traçados, o que eventualmente pode ser problemático.

Além do algoritmo de Tremeau existem na mesma referência [Dudeney, 17] outros algoritmos de percurso de outros tipos de labirinto que poderiam ser aplicados, porém são mais mais difíceis de programar.

4.5 - Software gráfico de suporte

A construção de um editor gráfico pode ser muito facilitada caso seja organizado um conjunto de rotinas básicas, que serão utilizadas na implementação das funções.

Não existe um padrão para estas rotinas, mas existe um conjunto delas que a maioria dos editores que consultei ou construí possui. O conjunto mencionado diz respeito unicamente a metodologias ortogonais.

a) Define a área da viewport da tela (x1, y1, x2, y2, escala)

Através dessa primitiva se especifica a diagonal da região da tela que irá receber o desenho, e a escala utilizada.

b) Define área do bitmap (x,y,ang)

Através dessa primitiva é especificada a região do bit map que será mapeada em uma região da tela (viewport). Em alguns editores é possível também fornecer um ângulo de apresentação, múltiplo de 90 graus. O tamanho da área é automaticamente definido pela primitiva a).

c) Define forma de pintar (vetpadrões, vetmodo)

Através desta primitiva, é fornecida ao software básico os padrões utilizados para pintura da área do vídeo. O vetor de padrões especifica uma lista de padronagens a utilizar e o vetmodo seleciona para cada combinação de camadas o modo através do qual será feita a pintura:

- através do padrão de hachuras
- por superposição temporal de cores
- por superposição combinacional de cores
- pela envoltória

d) Apaga trecho da viuporte (x1,y1, x2,y2)

Esta primitiva serve para limpar a área da viuporte referente ao trecho do layout especificado

e) Redesenha área (x1,y1,x2,y2)

Esta primitiva serve para desenhar na viuporte um trecho do desenho.

f) Move áreas da tela (xorig1, yorig1, xorig2, yorig1,
xdest, ydest)

Duas operações muito frequentes em editores gráficos são o ajuste de áreas e o "pan" (passeio sobre o desenho). Para tornar estas funções mais rápidas, pode-se utilizar a inteligência do terminal para mover áreas da tela.

g) Mova/Leia posição do cursor (x, y, dx, dy)

Estas duas primitivas permitem mover ou saber a posição do cursor (geralmente retangular). A movimentação do cursor pode ser controlada diretamente pelo editor ou pelo terminal.

f) Texto (x,y,ampl,ang)

Escrever um texto na viuporte. As coordenadas são fornecidas em termos do layout e não da viuporte. A ampliacao é geralmente um valor inteiro, e o angulo, multiplo de 90 graus.

g) Retângulo direto (x,y,dx,dy)

Alguns editores utilizam, para as camadas mais comuns do layout, uma cor que é mapeada diretamente em um dos bits do terminal. Assim, para as operações de criação e remoção de retângulos com aquelas camadas também são usadas primitivas do software básico.

4.6 - Problemas de eficiência para a produção do desenho na tela

Criar o desenho na tela é, na verdade, a maior dificuldade do editor, devido ao fato de que funções relativamente simples alteram uma grande quantidade de pontos na tela. A adição de um retângulo de 100 x 100 lambdas, por exemplo, se imaginarmos a associação de cada lambda quadrado com 2 x 2 pixels na tela, implica

na alteração de 40.000 posições da tela.

A alteração da tela, por outro lado, deve ser feita da maneira mais rápida possível, para atender aos requisitos de interação. Desta maneira, o editor deve ser cuidadosamente preparado para extrair do terminal ou vídeo gráfico o máximo de desempenho.

Quando a memória do vídeo é diretamente acessível pelo computador, como no caso de um vídeo gráfico ou de um microcomputador com tela gráfica, o problema de tempo é relativamente pequeno. Entretanto, quando a produção do desenho é feita num terminal gráfico, são necessários diversos cuidados para que o desempenho seja aceitável.

Os terminais gráficos normalmente podem ser ligados ao computador de duas maneiras: através de uma linha serial ou através de um canal de acesso direto à memória (DMA). No caso de comunicação serial, a taxa máxima através da qual se pode ligar um terminal é 19200 bits/segundo (aproximadamente 2000 bytes por segundo), ou seja, para enviar para o terminal cada pixel de uma tela contendo 1000 x 1000 pontos seriam gastos aproximadamente 500 segundos !

No caso de comunicação via canal DMA, embora a comunicação seja muito mais rápida (por exemplo, todo o conteudo de uma tela poderia ser enviado em cerca de 5 segundos), ocorrem diversos outros problemas. Primeiramente, o terminal não deve estar distante do computador, na prática, 20 metros no máximo. Em segundo lugar, as interfaces DMA utilizam o barramento do computador, ou seja, interferem muito no desempenho geral da máquina, o que é importante, por exemplo, em ambientes multiprogramados.

Entretanto, raramente se controla diretamente a memória de tela do terminal. Os terminais aceitam comandos mais complexos para desenhar elementos na tela, o que é feito de forma muito mais rápida do que se o conteudo da tela fosse enviado do computador. Alguns desses comandos que são úteis para este estilo de editor seguem-se:

. desenho de retas e retângulos

. preenchimento de áreas retangulares com cores ou padrões
. máscara de escrita, o que torna possível pintar áreas de forma aditiva, ou seja, superpor ou apagar uma nova cor a outras já existentes.
. movimentação de trechos da memória de vídeo
. pan e zoom utilizando a inteligência (ou o hardware) do terminal
. manipulação de um cursor retangular

Um problema adicional é que cada terminal possui um conjunto diferente de comandos, o que torna difícil a programação de editores portáteis.

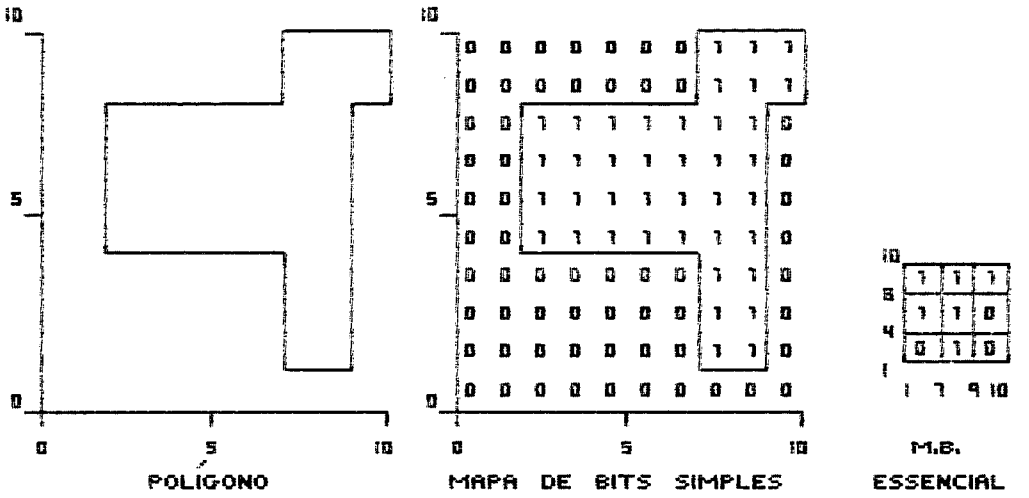
4.7 - Alternativas de armazenamento

O armazenamento matricial apresenta diversas vantagens, especialmente nos aspectos de simplicidade algorítmica. Entretanto existem alguns serios problemas devido à quantidade de memória que é necessária, e ao tempo de CPU necessário para a execução desses algoritmos.

Para executar a edição completa de um chip de 1 cm2 de área, com resolução de 1 mícron, seriam necessárias 10 ** 8 bits por camada. Devido a isso os editores gráficos que operam com mapa de bits quase sempre são restritos a edição de células, da ordem de 500 x 500 lambda no máximo.

Podem-se utilizar algumas alternativas de armazenamento que reduzem significativamente a quantidade de memória.

A primeira alternativa é uma técnica conhecida como "mapa de bits essencial" [Yoshida, 85] . Uma lista de todas as coordenadas x e y que são início ou fim de retângulo é gerada, e o desenho é dividido em regiões delimitadas por essas coordenadas. O número de bits é significativamente reduzido, embora a complexidade dos algoritmos cresca bastante [Losleben, 79].

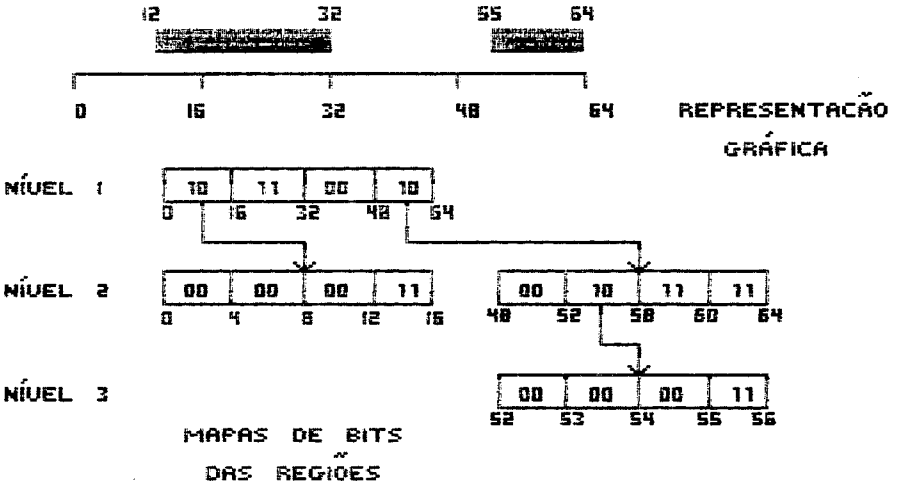


Mapa de Bits Essencial

figura 4.6

Uma outra alternativa é utilizar um esquema de mapa de bits hierárquico [Yoshida, 86]. O mesmo esquema, é executado para cada camada e para cada cada linha horizontal de varredura. Para utilizar a mesma nomenclatura do autor, chamemos de região transparente quando a camada envolvida na representação não está presente e opaca quando está (figura 4.7)

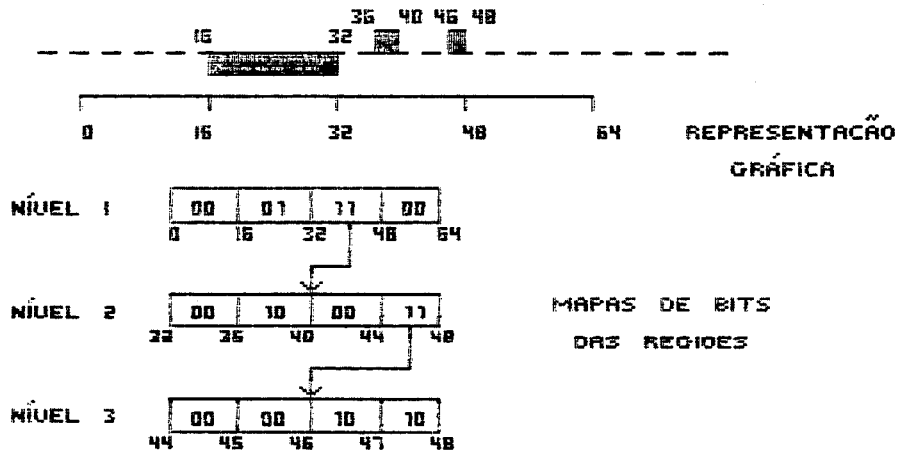
Para compactar a expressão das regiões com uma camada ao longo desta linha, 'e definida uma hierarquia de "mapa de bits da região" (RBP). Cada RBP é dividida em K setores de igual tamanho com cada setor sendo representado por dois bits, como na figura 4.7. Um setor totalmente opaco recebe o código 11. Um setor totalmente transparente recebe 00. Um setor parcialmente transparente, 10. Os setores com 10 são detalhados em um nível inferior da hierarquia. Esta forma é denominada NOW HSL.



Estrutura NOW HSL

figura 4.7

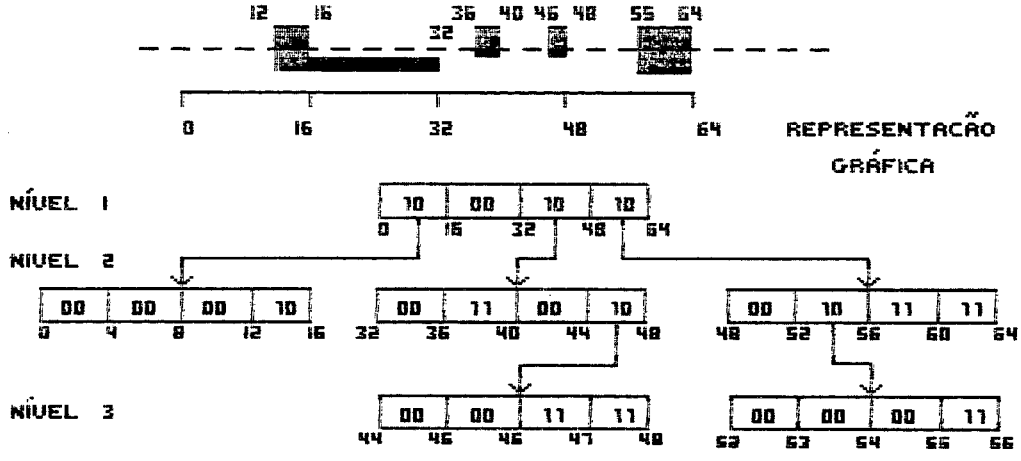
Para eliminar os dados repetitivos de regiões nas linhas subsequentes, só se registra as informações de mudanças em cada HSL (o que se denomina CHG HLS). No CHG HSL, "00" representa que não houve mudanças, "10" representa a mudança de transparente para opaco, "01" de opaco para transparente e "11" a uma troca descrita no nível inferior de hierarquia, como mostrado na figura 4.8.



Estrutura CHG HSL

figura 4.8

Se CHG HSL em alguma posição y é operacionalmente combinada à NOW HSL da linha logo abaixo dela, uma nova NOW HSL para a linha de cima é obtida, como mostrado na figura 4.9.



Padrão resultante da aplicação de CHG HSL a NOW HSL

figura 4.9

Durante o processo de edição é conhecido o valor de NOW HSL para a linha da base do cursor. Cada vez que o cursor é movido, deve ser recalculado o valor NOW HSL da nova linha, como mostrado acima. Qualquer alteração numa área promove modificações nos valores de CHG HSL das linhas desta área e da linha imediatamente superior à área.

Apesar de bem mais compacto esse método apresenta uma complexidade algorítmica maior no que se refere ao manuseio da estrutura de dados e na alocação dinâmica da árvore de mapa de bits.

4.8 - Acoplamento com ferramentas de verificação de projeto

Existe uma tendência no sentido a que os editores modernos tenham facilidades embutidas de verificação de regras de projeto (DRC), extração do circuito e roteamento. Existem uma série de algoritmos que podem ser aplicados diretamente sobre o mapa de bits para realizar estas funções.

4.8.1 - Verificador de regras geométricas

No caso de DRC, são feitos basicamente 4 tipos de testes [Mead, 80] que realizam medidas para obter:

Largura de fios
Separação entre fios
Extensão de camadas sobre outras
Sobreposição de camadas

As medidas normalmente podem ser obtidas diretamente por varredura linear, ou seja, andar sequencialmente em cada linha e localizar as regiões com e sem as camada envolvidas em cada teste, medindo as distâncias entre elas. E' feita uma varredura em x e outra em y (ou a 45 graus e 135 graus). Em algumas situações são necessários testes especiais, como na separação de polígonos com vértices localizados próximos a uma diagonal [Borges, 86].

Um algoritmo interessante para DRC em tecnologia NMOS foi apresentado por [Baker, 80], e melhorado por [Teles, 83].

4.8.2 - Extração de circuitos

O mapa de bits contem informações puramente geométricas, mas não apresenta nenhuma imformação explícita sobre o circuito. As-

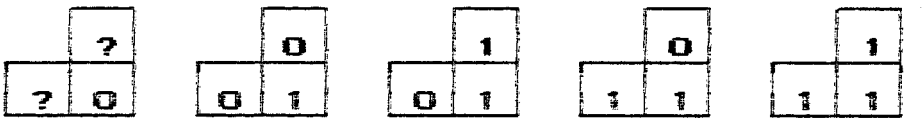
sim, é necessária uma tradução do mapa de bits para o circuito elétrico equivalente, para que possibilitar a execução de uma série de programas posteriores, tais como verificadores de conectividade e simuladores.

A extração de circuitos usualmente consiste no reconhecimento dos elementos do circuito (transistores) e na análise da conectividade. Esses processos são dependentes da tecnologia utilizada. Um apanhado de técnicas de extração pode ser achado em [Yoshida, 85].

Para metodologias ortogonais de layout é descrito em [Baker, 80] um algoritmo interessante para análise da conectividade. Este algoritmo, conhecido por Ilhas e Lagos, foi melhorado em [Teles, 85].

O algoritmo de ilhas e lagos procura resolver o seguinte problema: dada uma matriz de zeros e uns, onde os 1 indicam terra e os 0 água, atribuir um código a cada região contígua de terra. E' óbvia a associação de terra com fios do layout. A solução é um autômato bidimensional, que varre o mapa de bits de cima para baixo e da esquerda para a direita, atribuindo um código a cada região de terra (ou seja a cada fio do layout).

O código é função das regiões acima e à esquerda. O código 0 indica água. Somente 5 casos podem existir, como mostra a figura abaixo. Uma descrição completa do algoritmo está em [Borges, 87].



Configurações de terra e água do algoritmo de ilhas e lagos

figura 4.10

4.9 - O editor TEDMOS

Em 1986 foi desenvolvido um sistema, voltado especialmente para o ensino de projeto de circuitos integrados e utilização em multiprojetos de universidades. Este sistema, o TEDMOS, executa em microcomputadores compatíveis com IBM/PC em sua configuração básica, o que o torna universal.

Através do TEDMOS é possível fazer o desenvolvimento de células em tecnologia MOS, razoavelmente complexas. A grande vantagem do sistema, alm das facilidades de edição, é a integração com a verificação geométrica, simulação lógica e impressão, num único pacote. Os layouts criados são automaticamente traduzidos para linguagem CIF. Os layouts podem ser criados com TEDMOS num tempo muito menor do que em sistemas mais sofisticados.

O TEDMOS é relativamente independente de tecnologia, no sentido que um estudante pode instalá-lo, especificando parâmetros como o jogo de máscaras de fabricação, a representação gráfica das máscaras na tela e as regras geométricas de DRC.

O editor gráfico é o núcleo do TEDMOS: através dele, o projetista pode criar e modificar layouts de células de circuitos integrados na tela do computador. O editor provê basicamente as seguites funções:

- pintura e apagamento de áreas
- movimentação, ajuste e cópia de trechos de layout
- visibilidade parcial das máscaras
- armazenamento em CIF

Como a tela do microcomputador é pequena e tem pouca resolução gráfica, o editor provê diversas escalas de edição e a possibilidade de selecionar o trecho que será representado na tela, ou seja, passear com a tela sobre o desenho. A saída gráfica pode ser em preto e branco ou a cores.

O editor gráfico do sistema TEDMOS foi construído em 4000 li-

nhas de Turbo Pascal, e levou cerca de 4 meses para ser concluído, em especial o software básico gráfico que é bastante sofisticado.

O sistema TEDMOS veio preencher a lacuna de disponibilidade de pacotes de baixo custo para uso em ensino. O sistema está tendo uma aceitação muito grande no Brasil e em alguns países da América Latina, em especial, México, Argentina e Venezuela. Dezenas de centros de pesquisa e universidades já o utilizam na formação de recursos humanos na área de microeletrônica.

4.10 - Conclusão

A grande vantagem dos editores matriciais está na grande simplicidade dos algoritmos. A grande desvantagem, na quantidade de memória necessária. Por essas características, o método de mapa de bits é adequado para construção de editores de células e não de layouts completos.

As maiores dificuldades para criação deste estilo de editor são a apresentação das informações do mapa de bits na tela. Este capítulo descreveu uma série de métodos que devem ser seguidos para obter um desempenho razoável do editor, especialmente quando o terminal está ligado ao computador através de uma linha de comunicação remota.

O método de mapa de bits se adapta bem a ambientes de microcomputador, onde existe a possibilidade do acesso rápido à tela, e portanto é possível a criação de um esquema muito simples de movimentação/tradução do mapa de bits para a memória de tela.

5. Editores não hierárquicos não matriciais

Este capítulo apresenta um estilo de edição que utiliza muito menos memória do que os já apresentados, pois representa o desenho como uma lista de elementos gráficos que o compõem: retângulos, polígonos, círculos, etc. .

Na verdade a maior parte dos editores gráficos que utilizam este estilo, são também editores hierárquicos, que estudaremos no próximo capítulo. Esses editores possuem os mesmos problemas de manipulação de dados e utilizam extensões das estruturas de dados que veremos aqui.

Examinaremos com mais detalhe o caso da representação de layouts compostos a partir de retângulos ortogonais, devido ao fato de que, para este caso, diversas otimizações podem ser feitas na estrutura de dados, e por existir uma tendência dos projetistas a trabalhar com o estilo Manhattan de projeto.

5.1 - Estrutura geral

Um layout é constituído por diversas camadas cada uma delas sendo composta de uma geometria qualquer. Essas formas geométricas são normalmente complexas, mas podem ser criadas e expressas como composição de figuras simples, que chamaremos elementos construtivos.

Alguns dos elementos que são mais encontrados são os seguintes:

- retângulos
- circunferências
- polígonos
- fios (polígonos de "largura" constante)

Um editor não precisa processar obrigatoriamente todos esses elementos; por exemplo, alguns só manipulam o elemento retângulo, e muitas vezes apresentam algum outro elemento, por exemplo, texto

de comentário.

A base fundamental destes editores é uma lista dos elementos componentes. Cada comando que é dado ao editor vai se refletir basicamente em inclusões, remoções ou alterações nesta estrutura.

Na tela, da mesma forma que nos outros editores, é mostrado um trecho do layout. Os comandos do editor são enviados através das maneiras já mostradas anteriormente, e a região sobre a qual os comandos vão atuar são indicadas por um cursor na tela. À medida que a estrutura vai sendo atualizada, vai sendo refletida na tela a imagem do layout alterado.

A maior diferença entre este estilo de editor e os já vistos, está na maneira com que as operações são realizadas:

a) seleciona-se, através do cursor, a região a manipular.
b) a estrutura de dados é percorrida, localizando todos os elementos pertencentes a esta região (em alguns casos, também os elementos vizinhos).
c) operam-se modificações adequadas nesses elementos.
d) apaga-se da tela a região alterada (em alguns casos, parte dela).
e) redesenha-se toda a região alterada, movendo-se para a tela os elementos desta região, segundo algum método de desenho.

Os itens b e c podem ser implementados de diversas formas diferentes, e é possível a criação de estruturas de dados que tornem simples ou rápidos os algoritmos das funções.

A apresentação da tela pode ser realizada com os mesmos estilos que foram vistos no capítulo anterior. Entretanto, o processo de desenho na tela é mais simples, uma vez que as informações já estão codificadas, e assim, a própria estrutura de dados já fornece os elementos para ativação das rotinas de desenho.

As fórmulas de transformação dos sistemas de coordenadas do layout (janela) e da viuporte, mostrados anteriormente, também são usadas diretamente aqui, para o desenho de elementos e para a

transformação das coordenadas do cursor na coordenada do layout. O mesmo vale para os procedimentos de recorte (clipping) de elementos.

Alguns editores operam com diversas viuportes do mesmo layout, por exemplo, uma visão macro do layout e um "zoom" de um trecho específico [Keller, Kic2]. Em particular a operação de visão macro e operações de escala são muito simples, pois as dimensões dos objetos escalados são obtidas diretamente da estrutura de dados, pela simples aplicação das transformações janela-viuporte.

5.2 - Operações do editor

Vamos examinar as operações principais, sem nos preocupar, por enquanto, com as maneiras de organizar a lista de elementos componentes, mas focalizando alguns problemas típicos de edição.

5.2.1 - Inserção

A inserção de um novo componente é aparentemente simples:

. posiciona-se o cursor sobre o ponto onde se deseja inserir.
. calcula-se as coordenadas do layout, a partir das coordenadas do cursor, usando a transformação de janela/viuporte.
. insere-se o elemento na lista
. desenha-se o elemento na tela, segundo o método associado à camada do elemento inserido.

Alguns editores operam exclusivamente assim. Entretanto, existem alguns detalhes operacionais neste processo, que provoca a necessidade de algumas sofisticações adicionais no editor e na estrutura de dados.

a) inserção com superposição

Um elemento ao ser inserido, muitas vezes se junta com outro, formando visualmente um elemento simples. Por exemplo, dois retângulos de altura identica, colocados na mesma coordenada y, podem formar visualmente um único retângulo. Esta e outras situa-

ções mais complexas são vistas na figura 5.1.

Na verdade, o editor não é obrigado a fazer essa aglutinação de dados inseridos, porém, caso seja capaz de fazê-lo, ou pelo menos, aglutinar algumas situações mais comuns, diminuirá o tamanho da estrutura de dados, e consequentemente, o tempo de processamento e o tempo de desenho.

Existem soluções geométricas para este requisito, utilizando interseção de polígonos, mas são relativamente complicadas. Algumas alternativas razoáveis para este problema são os seguintes:

. não compactar nada
. fazer uma compactação final, como mostrado em 5.5
. usar uma estrutura de dados mais sofisticada como a quadruplamente encadeada, mostrada mais adiante.

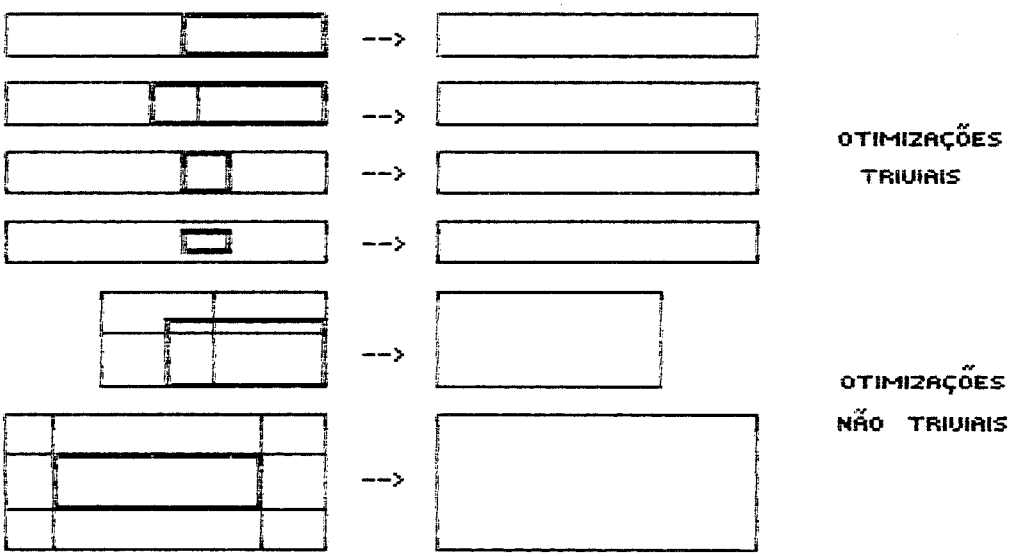


figura 5.1
Otimizações na inserção

b) inserção de fios muito grandes

A tela do vídeo, operada numa escala confortável, muitas vezes é insuficiente para conter o elemento que se quer inserir. É o caso, por exemplo, de um fio muito longo, cheio de voltas, unindo pontos afastados do layout.

Neste caso, uma idéia seria partir o fio em elementos justapostos, que são inseridos selecionando-se sucessivamente várias janelas. Entretanto, há muitas vezes, neste caso, a perda da referência do caminho do fio.

Uma outra idéia seria operar numa escala que possibilitasse a visão completa do caminho do fio. Entretanto, perde-se nesse caso a sensibilidade das coordenadas exatas de encaixe do fio aos elementos que ele conecta.

5.2.2 - Localização de elementos

Nos editores gráficos matriciais, era suficiente demarcar uma área retangular na tela, sobre a qual seria executada alguma função. Entretanto, nos editores não matriciais, uma área retangular na tela normalmente contém pedaços de varios elementos.

Existem, então, diversos procedimentos para localização de elementos, que são utilizadas dependendo da função que esteja sendo realizada. Algumas das formas de localizar elementos são:

- um elemento que tenha interseção com o cursor
- todos os elementos que tenham interseção com o cursor
- um elemento envolvido pelo cursor
- todos os elementos que sejam envolvidos pelo cursor
- um elemento cujo ponto inferior esquerdo coincida com um ponto do cursor
- ... e assim por diante

Para localização de elementos, é comum usar-se o processo conhecido como teste minimax (ou "bounding box"), que consiste em calcular as coordenadas máximas e mínimas de cada elemento e comparar com o cursor. Este procedimento muitas vezes não é preciso para a comparação, pois pode achar, erroneamente, que um determinado elemento envolve o cursor (como mostra a figura 5.2), mas é uma boa aproximação inicial, com custo computacional pequeno.

figura 5.2
Falha no teste Minimax

Eventualmente quando um elemento passa no teste minimax, ainda se pode fazer uma comparação geométrica adicional (isso é desnecessário no caso em que os únicos elementos do layout são retângulos), através de um procedimento de recorte de polígonos, por exemplo, com o algoritmo de Sutherland-Hodgman [Newman, 79].

5.2.3 - Remoção

A remoção de elementos se faz muito facilmente: os elementos que foram localizados através de um dos procedimentos acima, é removido da lista e o trecho de tela que o continha, apagado e redesenhado (ou parcialmente apagado, através de um dos procedimentos mostrados no capítulo anterior).

O problema da remoção surge quando o editor é capaz de otimizar as figuras básicas. Se o editor tiver este tipo de facilidades, a operação de remoção deve ser diferente: o cursor delimita uma área (normalmente retangular) e o editor remove os polígonos com interseção do cursor e cria novos polígonos com a parte externa ao cursor. é importante notar que o editor, neste caso também deveria poder gerar otimizações para os novos polígonos.

Um caso muito comum, que é aplicado nos editores que criam a imagem exclusivamente como união de elementos retangulares, é realizar o apagamento exclusivo da área do cursor, simulando o que seria executado em editores matriciais. Neste caso, cada retângulo que tiver alguma interseção com o cursor é removido e pode gerar de 0 a 4 novos retângulos, conforme mostrado na figura 5.3.

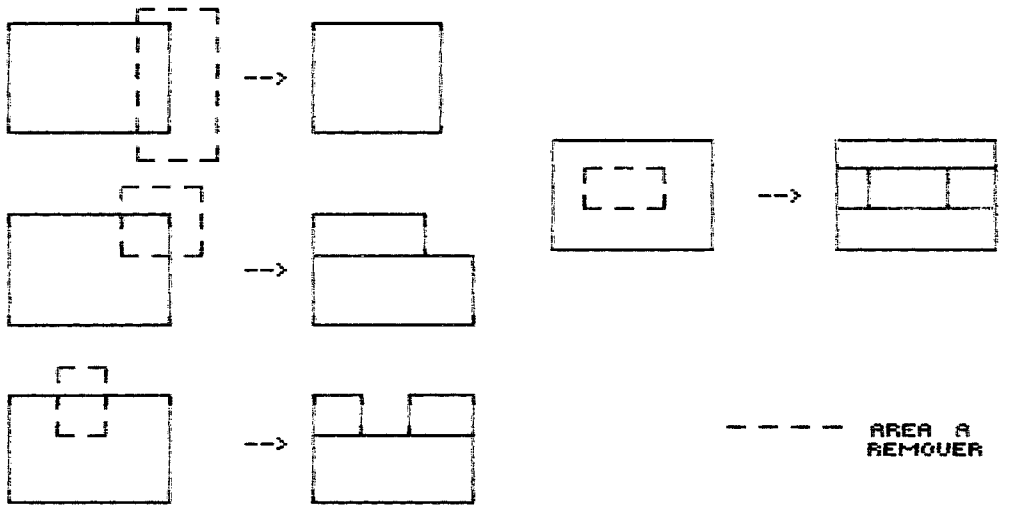


figura 5.3
Procedimento de remoção retangular

5.2.4 - Ajustes de áreas

A maior parte dos editores que trabalham com este método realizam ajuste de áreas de duas maneiras:

a) Todas as figuras envolvidas pelo cursor são deslocadas de uma certa distância.
b) Os vértices que estão envolvidos pelo cursor são deslocados de uma certa distância (as figuras eventualmente são esticadas ou encolhidas).

Um dos sistemas mencionados na literatura, MAGIC [Ousterhout, 84] é capaz ainda de realizar uma operação de ajuste de áreas conhecido como "plowing" (pá de neve), mostrada na figura 5.4. Nesta operação se especifica uma linha vertical ou horizontal que pode ser comparada a uma pá mecanica, que pode empurrar o layout.

Este editor é capaz de controlar as regras de projeto e conservar as distâncias mínimas entre camadas, introduzindo, eventualmente, degraus nos fios. Esta técnica, é muito dependente da estrutura de dados quadruplamente encadeada, que será vista em 5.3.3 e 5.5.

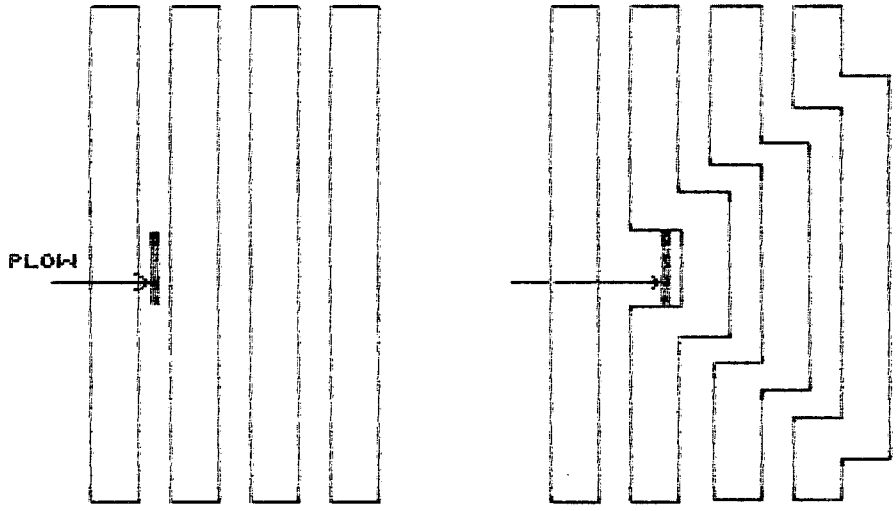


figura 5.4
operação de "plowing"

As cópias e movimentações de áreas não trazem maior dificuldade, pois envolvem apenas alterações ou inclusão na estrutura de dados.

5.2.5 - Textos

Os editores devem possuir facilidades para inserção de nomes de nós e comentários no layout. Estes textos trazem alguns pequenos problemas que devem ser cuidadosamente tratados pelo editor.

Um texto ao ser colocado num determinado ponto do layout deve ficar associado ao elemento sobre o qual ele é colocado. Desta forma, caso o elemento seja movido ou removido, o texto também deve sê-lo.

Os terminais gráficos, por sua vez, têm um poder de escala de textos limitado. Assim, quando o desenho é escalonado, o texto muitas vezes não o é, o que tem um efeito desagradável visualmente. Alguns editores evitam mostrar os textos automaticamente, somente o fazendo através de comandos específicos.

Um texto de comentários é uma estrutura de dados de tamanho variável. Em termos de processamento isso não causa maiores pro-

blemas, mas no caso de armazenamento é necessário tomar alguns cuidados, especialmente pelo fato de que muitas vezes o formato do arquivo é binário e com tamanho fixo de registro.

5.3 - Organização de dados

A lista de elementos pode ser organizada de diversas maneiras, que podem ter desempenho e complexidade de implementação mais ou menos adequadas dependendo da complexidade de layouts que se espera editar.

Os problemas que a estrutura de dados pode otimizar são os seguintes:

. localização dos elementos pertencentes a uma certa região
. inclusão e remoção de elementos
. concatenação de elementos e eliminação da fragmentação

5.3.1 - Tipos de dados fundamentais para representação dos elementos construtivos

A estrutura de dados deve armazenar pelo menos parte dos seguintes tipos de dados, referentes às informações dos elementos construtivos [Fairbarn, Icarus]

a) Retângulos:
   . coordenadas da diagonal de retângulos ortogonais
   . ou coordenada mínima e tamanho de retângulos ortogonais
   . ou coordenadas do centro do retângulo, dimensões e inclinação

em Pascal:

```
record
    xmin, ymin, xmax, ymax: integer;
    camada: layer;
end
```

ou

```
record
     xmin, ymin, dx, dy: integer;
     camada: layer;
end
```

ou

```
record
     xc, yc, dx, dy: integer;
     teta: real;
     camada: layer;
end
```

Os valores inteiros exprimem distâncias em lambdas.

O ângulo pode geralmente ser expresso como um número inteiro.

b) Circunferencias:

. coordenadas do centro e raio

Em Pascal:

```
record
     xc, yc, raio: integer;
     camada: layer;
end
```

c) Polígonos:

. número de vértices, e coordenadas dos vértices

Em Pascal:

```
record
     nv: integer;
     x, y: array [1..MAXVERT] of integer;
     camada: layer;
end
```

Como os polígonos tem tipicamente um número variável de vértices, pode-se usar uma lista encadeada, em que cada nó contenha um número fixo (por exemplo 8) de elementos. Para facilitar a organização em tabelas ou arquivos sequenciais, pode-se usar uma marca adicional que informa que é o primeiro da lista.

Em Pascal:

```
type pnó = ^no;
     nó = record
               nv: integer;
               primeiro: boolean;
               x,y: array [1..8] of integer;
               camada: layer;
               prox: pno;
          end
```

d) fios

Idêntico aos polígonos, só que contém mais um campo: a largura do fio.

Em Pascal:

```
record
     nv: integer;
     larg: integer;
     x, y: array [1..MAXVERT] of integer;
     camada: layer;
end
```

ou

```
type pnó = ^no;
     nó = record
               nv: integer;
               larg: integer;
               x,y: array [1..8] of integer;
               camada: layer;
               prox: pnó;
          end
```

e) Nós

. coordenadas do nó, camada, nome do nó

Em Pascal:

```
record
     x, y: integer;
     camada: layer;
     nomenó: array [1..MAXCAR] of char;
end;
```

Os nomes dos nós, por simplicidade podem ser armazenados com um número fixo de caracteres (por exemplo, 8), para tornar o nó de tamanho fixo, e completados com brancos à direita.

f) Textos

Os textos tem para armazenamento o mesmo problema dos polígonos: são estruturas de tamanho variável. Podemos optar por armazenar com tamanho variável, se a linguagem permitir e também no caso de armazenamento em arquivo sequencial sem tipo (arquivo de texto),

Em Pascal:

```
record
     x, y: integer;
     nc: integer;
     texto: array [1..MAXCARAC] of char;
end
```

ou então criar uma lista encadeada de subcadeias de tamanho pequeno (por exemplo, 16 caracteres); um dos campos informa o tamanho de cada subcadeia. A ultima subcadeia possui um ponteiro igual a NIL.

Para armazenamento em estruturas sobre as quais será feita uma busca sequencial, por exemplo, uma tabela, pode ser conveniente criar uma marca para indicar que uma certa subfrase é a primeira de uma frase. No caso em que que exista algum ponteiro externo para o inicio da frase, este campo é omitido.

Em Pascal:

```
type
     psubfrase = ^subfrase;
     frase = psubfrase;
     subfrase = record
                   x,y: integer;
                   primeira: boolean;
                   nc: integer;
                   subcadeia: array [1..TAMSUBCAD]
                          of char;
                   prox: psubfrase;
                end;
```

5.3.2 - Caracteristicas gerais da organização das informações

Os elementos, descritos da forma mostrada acima devem então ser organizados de maneira a atender a alguns requisitos. é importante a criação de uma estrutura de dados adequada pois alguns fatores pesam bastante na necessidade de performance alta e pouco gasto de computador:

a) numa edição interativa, é essencial que o tempo de resposta seja o mínimo possível. No caso, o tempo de processamento de cada comando num ambiente multiprogramado deve ser menor que uma fatia de tempo (time slice) para que o processo do editor não tenha que disputar tempo com outros processos.

b) o tempo de espera de execução de comandos frequentes deve ser o mínimo para aumentar a satisfação do usuário

c) se o tempo de computador utilizado for pago.

Foi constatado, a partir da observação de usuário dos editores que foram construídos por nós, que as operações que são mais executadas pelo editor, e que portanto exigem que o editor as execute muito rapidamente, são as seguintes (nesta ordem):

. inserção de elementos (com seleção de camadas)

. movimentação da janela de edição
. ajuste de áreas

Para execução dessas operações, o editor se utiliza basicamente de um conjunto relativamente pequeno de funções que mais ou menos dão as diretrizes para a organização dos dados:

a) localizar os elementos que estão (parcialmente) em uma área retangular
b) redesenhar uma área
c) inserir um novo elemento na estrutura
d) remover um elemento da estrutura
e) alterar um elemento da estrutura

Alguns editores gráficos realizam a compactação dos elementos ao final do processo de edição. Para realizar a compactação é necessário saber o conjunto de elementos que se interceptam com um certo elemento, o que pode ser implementado como uma extensão do item a, caso os elementos sejam retangulares (o que a maior parte dos editores faz). Uma das estruturas de dados, mostrada mais adiante, é especialmente adequada para esta operação.

O problema mais critico é o da localização rápida de elementos (na maior parte das vezes, buscar todos os elementos com interseção com o cursor). Esse problema não é tão trivial quanto parece, e soluções que envolvam ordenações dos elementos trazem pouco aumento na eficiência.

Vamos considerar, no nosso estudo, o caso da edição de células, em que o número de elementos seja relativamente pequeno, menos que 10000, geralmente, e assim, seja possível manter todos os dados de elementos na memória principal do computador, o que simplifica consideravelmente os algoritmos.

Nota: é oportuno frisar que a edição de circuitos integrados completos, normalmente é feita através da montagem de células, e portanto também não necessita de um grande número de elementos pre-

sentes na memória simultaneamente.

5.3.3 - Organização linear das informações

A organização mais simples consiste em criar listas lineares sobre as quais são feitas essas operações. As possibilidades para a criação destas listas são as seguintes:

a) várias tabelas, cada uma contendo um tipo de elemento construtivo.

É importante frisar que os elementos construtivos tem tipos e tamanhos diferentes, e portanto é difícil a criação de uma única tabela, a não ser utilizando construções estranhas, tais como variant record em Pascal, Union em C ou Equivalence em Fortran.

Um dos problema desta organização está na necessidade de pré-alocação do espaço para as tabelas, que é um valor um tanto imponderável. Para minimizar este problema pode-se alocar dinamicamente os nós de informação e usar uma tabela de ponteiros.

b) lista encadeada

Alocam-se as informações dinamicamente, criando-se uma lista encadeada com elas. A vantagem está no maior dinamismo de inserção e remoção de elementos.

Tanto no caso (a) quanto no caso (b), as operações de localização de elemento e de redesenho de áreas (que na verdade é também um problema de localização de elementos) envolve uma busca sequencial em toda tabela, comparando elemento a elemento, para saber se ele está na região do cursor ou a na região a redesenhar.

Nesta busca seria feito o teste minimax, ja citado, para detetar se a envoltória do elemento tem interseção com o retângulo, e depois uma comparação mais detalhada com a geometria exata do elemento, caso não fosse retangular.

Nota:

Deve-se notar que uma ordenação realizada sobre as coordenadas dos elementos ajudaria pouco na localização de elementos com interseção com uma área. Por exemplo, uma tabela auxiliar implementando ordenação sobre a coordenada x mínima do elementos permitiria eliminar os elementos com coordenada mínima posterior ao cursor. Uma outra sobre a coordenada x máxima permitiria eliminar os elementos com coordenadas máxima anterior ao cursor. Porém o problema da interseção dos elementos restantes nas duas ordenações representa um esforço computacional quase tão grande quanto a busca sequencial.

5.3.4 - Divisão do layout em regiões (buckets)

A partição em regiões (buckets) é técnica de geometria computacional que tem sido considerada uma das mais poderosas para melhorar a eficiência dos algoritmos geométricos. Esta técnica consiste no seguinte (figura 5.5):

a) dividir o desenho em regiões retangulares de igual tamanho (buckets)
b) Encontrar uma solução local para cada bucket
c) Encontrar a solução final, combinando (unindo, interceptando, etc...) as soluções dos buckets.

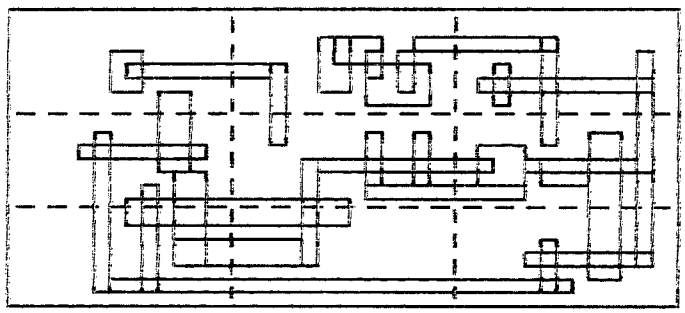


figura 5.5
divisão do layout em buckets

Na maior parte das vezes, o número de objetos contidos em um bucket é limitado por uma constante, se a região é particionada em um número de buckets proporcional ao número de objetos. Mais que

isso, a influencia mútua entre elementos tende a decrescer com a distância, o que colabora com a eficiência dos algoritmos para aplicações práticas. Muitas vezes, ainda é possível manter na memória somente os buckets que interessem numa determinada situação, diminuindo os requisitos de memória do computador.

Esta técnica pode ser aplicada no editor da seguinte maneira, visando obter uma melhoria significativa nas operações de localização de elementos e redesenho de áreas:

a) Os elementos podem ser armazenados em tabelas ou listas encadeadas, de forma que a inclusão, remoção e alteração de informações seja simples.

b) o layout é dividido em regiões quadradas, capazes de conter um número de elementos que possa ser pesquisado sequencialmente num tempo razoável (esse "razoável" é empirico, por exemplo, 100 elementos).

c) um mesmo elemento pode pertencer a várias regiões.

d) é criada uma lista de ponteiros para os elementos de cada bucket e criada uma matriz de ponteiros, representando a divisão do layout, para as listas, como mostrado na figura 5.6

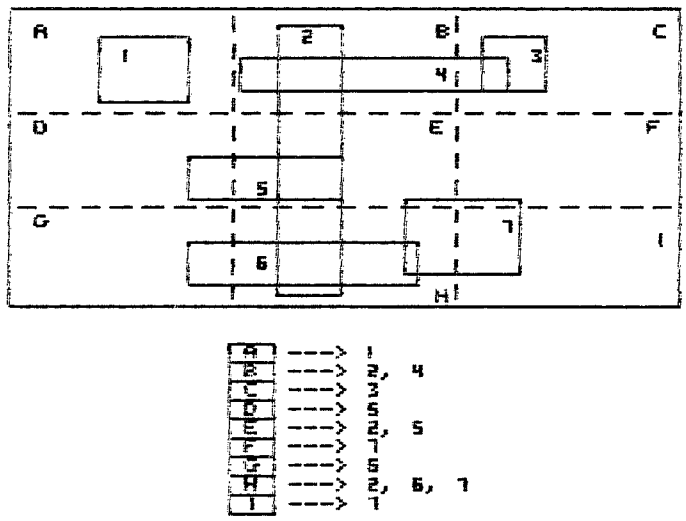


figura 5.6

estrutura de dados dos buckets

A operação de localização de elementos pertencentes a uma área é feita buscando-se sequencialmente os elementos que se interceptam com a área, por teste minimax, em todos os buckets nos os quais a área esta contida. Nesta busca é gerada uma lista de elementos. Para evitar a duplicação de elementos nesta lista, sempre que é inserido um novo nó o correspondente elemento é marcado. Os elementos são facilmente desmarcados, terminado o algoritmo, a partir desta mesma lista.

As operações de alteração das dimensões ou movimentação de elementos é relativamente complicada, uma vez que pode haver necessidade de atualização dos buckets nesta operação. O procedimento mais simples, nestes casos, é remover os elemento e reinseri-los.

5.3.5 - Alocação quadruplamente encadeada (corner stitching)

Uma das organizações mais interessantes foi utilizada no sistema MAGIC de layout, descrito em [Ousterhout, 84].

Em MAGIC, como na maior parte dos editores de layout, um layout consiste de células, Cada célula contém dois tipos de objetos: formas geométricas e sub-células. O conteúdo das células é representado através de uma estrutura de dados geométrica chamada "costura de cantos" (corner stitching), adequada para o processamento de retângulos ortogonais. A estrutura de dados provê uma variedade de operações eficientes de busca e atualização.

Os elementos básicos da técnica são os planos (mais ou menos equivalentes a camadas) e os ladrilhos. O conjunto de planos representa a geometria da célula. Cada plano é composto de ladrilhos retangulares. Um plano tem as seguintes propriedades:

- cobertura:

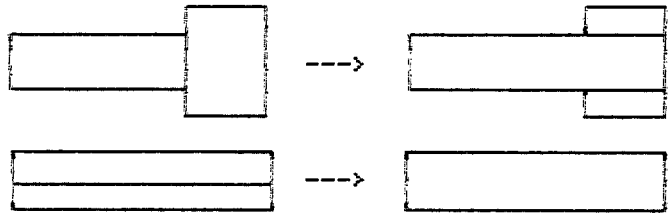
cada ponto x,y do plano está contido em um e somente um ladrilho. O espaço vazio é representado bem como as áreas cobertas com material

- tiras

material do mesmo tipo é representado em tiras horizontais, tão largas quanto possível, e depois tão altas quanto possível. Esta estrutura provê uma forma canônica evitando a fragmentação. Durante a inserção de novos elementos, o editor transforma as figuras como mostrado na fig. 5.7.

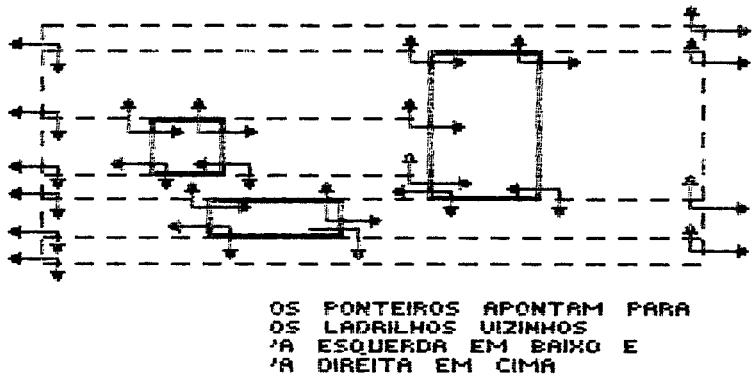
- costuras

os ladrilhos se apontam entre si como mostrado na figura 5.8. O canto inferior esquerdo de um ladrilho aponta para os ladrilhos abaixo e à esquerda. O canto superior direito aponta para os ladrilhos acima e à direita.



transformação canônica das tiras

figura 5.7



estrutura de "corner stitching"

figura 5.8

Estes ponteiros permitem uma variedade de operações de busca serem realizadas eficientemente, incluindo: encontrar o ladrilho contendo um dado ponto; encontrar todos os ladrilhos vizinhos a

um certo ladrilho e atravessar uma região. Esses algoritmos, bem como a atualização da estrutura (inclusões e remoções) são descritos em [Ousterhout-2, 84].

Esta estrutura de dados facilita ainda a criação de uma operação conhecida por "plowing" (pá de neve) que é uma operação usada para rearrumar o layout (abrir espaços e compactar), sem desfazer as ligações nem alterar o circuito elétrico que ele representa, como mostrado na figura 5.4. Uma descrição completa de implementação de plowing pode ser achada em [Scott, 84].

5.3.6 - Organizações alternativas

5.3.6.1 - Solução baseada em geometria computacional

O problema da interseção online de retângulos, do ponto de vista da geometria computacional, pode ser descrito como: dado um conjunto de n retângulos e um retângulo de consulta R, enumerar todos os retângulos interceptando R.

Resolver esta questão envolve:

. achar os retângulos inteiramente incluidos em R,
. achar os retângulos inteiramente incluindo R, e
. achar os retângulos cujos lados tem interseção com R.

O problema da interseção online de retângulos pode ser resolvida através de três algoritmos básicos de geometria computacional:

a) busca de intervalo: para um dado conjunto de números e um intervalo de pesquisa I, enumerar todos os números de D contidos no intervalo I.

b) pertinencia de ponto: dado um conjunto D de intervalos e um número pesquisado A, encontrar todos os intervalos de D contendo o número A

c) para um conjunto D de intervalos e um intervalo de busca I, enumerar todos os intervalos de D interceptando este inter-

valo I.

Esses algoritmos utilizam estruturas de dados baseadas em árvores de busca. A primeira utiliza uma árvore de busca AVL [Wirth, 76], a segunda uma árvore de segmentos e a terceira uma árvore de busca de prioridade. Uma descrição resumida destas estruturas de dados pode ser achada em [Asano, 86].

Apesar da eficiência desses métodos de busca, a manutenção das estruturas de dados que são necessárias para a implementação dos algoritmos é muito complexa, e a quantidade de memória necessária para implementação das três árvores de busca é relativamente grande, e assim, acaba saindo muito caro manter as estruturas de dados.

O método, embora possa ser utilizado em editores gráficos, é mais adequado a outros tipos de aplicações em CAD de circuitos integrados como verificação de layout e roteamento.

5.3.6.2 - Tabela de bits por linha e por coluna

A idéia deste método consiste em criar uma tabela de elementos e manter para cada linha e coluna da tela uma tabela de bits, que indique quais os elementos que tem interseção com a linha ou coluna. Desta forma o problema de interseção de retângulos fica resolvido através de operações booleanas.

O método é fácil de implementar e eficiente e a estrutura de dados fácil de manter. Entretanto, a área ocupada pelas tabelas de bits é enorme. Para diminuir a quantidade de memória gasta, pode-se utilizar um "mapa de bits essencial", como descrito no capítulo anterior.

Na verdade, este método tem apenas interesse teórico para comparações de performance, não sendo utilizado na prática.

5.4 - Armazenamento permanente

O armazenamento das listas em arquivos possui basicamente

duas alternativas:

a) armazenamento em formato textual

Nestes formatos os elementos construtivos são diretamente representados. é o caso da linguagem CIF [Hon, 80], que possui o seguinte conjunto de comandos relativos aos elementos:

LAYER - para seleção da camada dos próximos elementos
BOX - para retângulos
POLYGON - polígono
ROUND FLASH - círculos
WIRE - para fios

Esta linguagem ainda possui uma abertura para a criação de novos comandos (extensões). Na implementação dos editores que construi, foram utilizadas três extensões, uma para representação de textos, outra para nome da célula, e outra para nome de nos.

Esta forma de armazenamento exige um parser simples para a interpretação do formato. Alguns editores utilizam um "CIF formatado", ou seja, uma série de restrições que que simplifiquem a interpretação dos comandos (por exemplo, exigencia de so um comando por linha, colunas fixas para os números, etc...).

A grande vantagem é a portabilidade do arquivo gerado.

b) armazenamento em arquivos com registro de tamanho variável.

Esta é uma implementação simples, que agrega aos registros um campo adicional indicando o tipo de elemento. Um único arquivo contém todas as listas.

c) armazenamento em bancos de dados relacionais.

Neste caso cada lista constitui uma relação. A maior parte dos bancos de dados cria um arquivo para cada relação. A única dificuldade, neste caso, é o armazenamento de textos, estrutura de

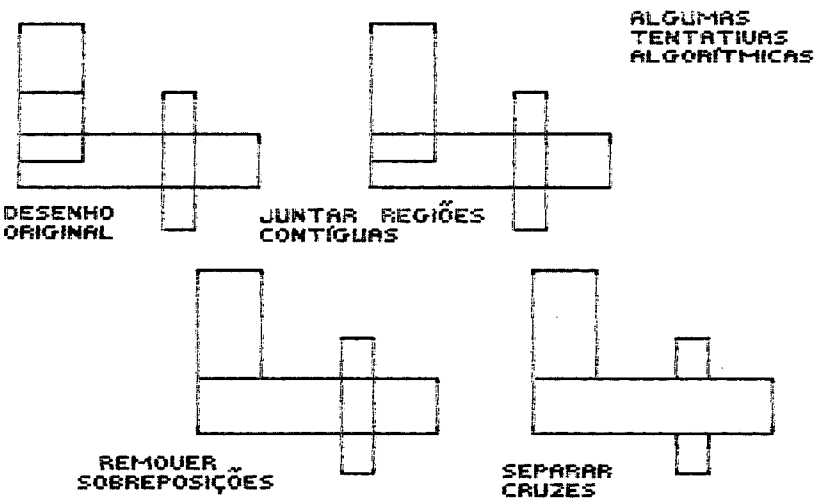
tamanho variável, que pode ser implementado como diversas tuplas no arquivo, com uma chave única.

5.5 - Algoritmos especiais

5.5.1 - Compactação de retângulos a posteriori

A compactação de retângulos embora possa ser realizada "on-line" pelo editor, mais frequentemente é realizada a posteriori, pois isso possibilita a simplificação de algoritmos e a adoção de estruturas de dados mais simples.

Existem muitos resultados possiveis de compactações que podem ser realizadas (figura 5.9) para um mesmo conjunto de retângulos. O método que vamos mostrar, apesar de não produzir sempre o resultado ótimo (mínimo número de retângulos), elimina todas as superposições e é geometricamente muito simples, pois é baseado em algoritmos de varredura [Foley, 83].



diversas compactações de retângulos
figura 5.9

A idéia básica é simples: cria-se o mapa de bits do layout e aplica-se o algoritmo de extração de retângulos visto no capítulo 3.

Porém, o mapa de bits é uma estrutura muito grande, como já

se viu, e inviável de ser criado na maioria das vezes. Felizmente, o algoritmo de extração de retângulos visto, não processa todo layout, globalmente: processa uma linha de cada vez.

A idéia melhorada então é a seguinte: gerar uma co-rotina que consiga gerar linhas de varredura para alimentar o algoritmo do capítulo 3. Isso é simples de realizar através do seguinte algoritmo:

Inicialização: Esta rotina é chamada ao inicio do algoritmo de geração

1. cria-se uma lista contendo todos os retângulos da camada que está sendo processada

Co-rotina de geração da varredura: Chamada a cada linha do layout

1. remove-se da lista de retângulos todos aqueles que tiverem interseção com esta linha
2. insere-se este vetor numa lista de elementos ativos, contendo a coordenada esquerda, e as dimensões (x,dx,dy) dos retângulos
3. Cria-se um vetor de bits, com largura igual ao tamanho do layout, inicializado com zeros

4. para cada elemento da lista,
   4.1. ligam-se os bits do vetor entre x e x+dx-1
5. decrementa-se de um as dimensões dy de todos os
   5.1. retângulos da lista de elementos ativos, removendo todos aqueles que chegarem a 0.

O algoritmo pode ser otimizado da seguinte maneira:

a) ordenar a lista de retângulos, tornando simples a remoção. A maneira mais apropriada de realizar esta ordenação é através de uma "bucket list" que é um vetor de ponteiros para listas dos retângulos cuja coordenada superior é o indice do vetor (figura 5.10), segundo a sugestão de [Newman, 79].

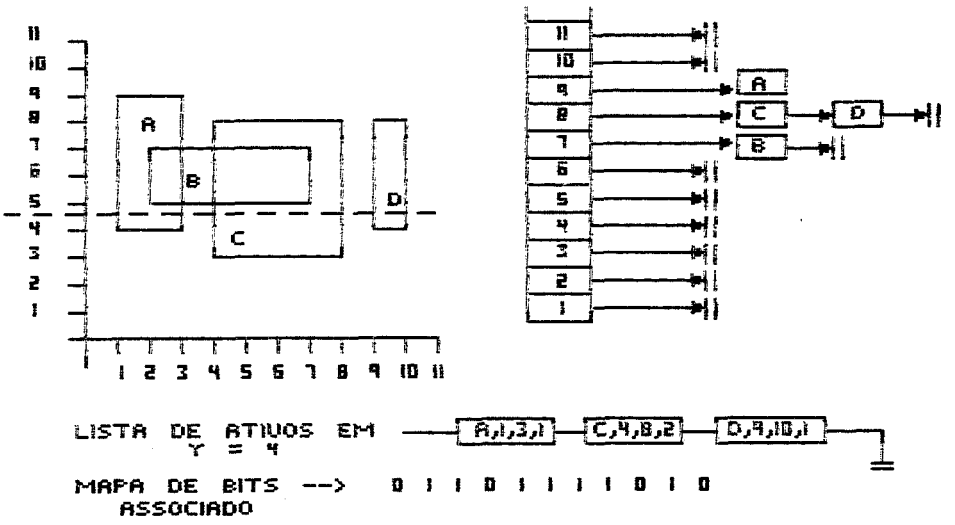


figura 5.10
"bucket list"

b) caso não se tenha removido nem inserido elementos na lista desde a ultima vez que foi gerado o vetor de bits, não é necessário regerar o vetor. Neste caso pode ser feita ainda uma melhoria no algoritmo do capítulo 3 para que não precise varrer esta linha.

5.5.2 - Particionamento ótimo de polígonos em retângulos.

Alguns editores não são preparados para processar polígonos, somente retângulos, e devem, portanto fazer uma conversão de polígonos para retângulos.

Um método simples que pode ser usado para polígonos retilineares (formados por vértices ortogonais)(figura 5.13):

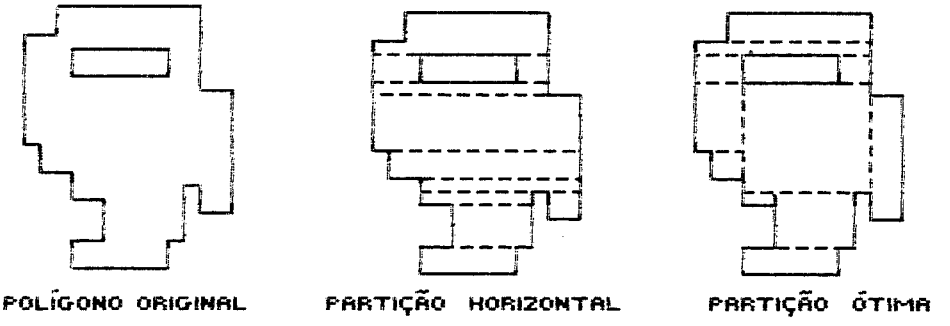
a) achar os vértices côncavos (vértices de 270 graus)
b)criar uma linha horizontal (ou vertical) para cada um destes vértices.

Para achar os vértices côncavos, deve-se fazer o seguinte:

1. Busca-se a aresta mais à esquerda
2. Toma-se um vértice pertencente a esta aresta. Ele será necessáriamente côncavo. Acha-se o ângulo entre esta aresta e a vizinha por este vértice.
3. Este ângulo (90 ou 270) será associado ao valor convexo.
4. seguem-se os vértices na ordem, calculando os ângulos de cada aresta com a seguinte. Aqueles que forem iguais ao primeiro ângulo, serão associados a um vértice côncavo.

Os algoritmos funcionam mesmo para polígonos com buracos.

Em [Asano, 86] é mostrado um método para particionamento ótimo de retângulos para polígonos retilineares. O método é baseado na minimização do grafo de interseção das cordas (linhas unindo os vértices) do polígono.



Particionamento simples e ótimo de um polígono
figura 5.11

5.6 - Estruturas de dados usadas pelos editores mais conhecidos

Todos os editores que foram estudados ou desenvolvidos por mim e que usam este estilo de edição também são hierárquicos, de forma que só podemos analisar sua sub-estrutura de dados para edição de células.

O editor Icarus [Fairbarn, 76] foi o primeiro editor amplamente documentado em termos internos, e utilizava listas lineares apontadas. Outros editores que vieram depois, Caesar [Ousterhout,

80] e Kic2 [Kendall, 82] apresentaram mudanças significativas na operação e apresentação das informações, mas mantiveram praticamente a mesma estrutura de dados. O editor EDCI [Borges, 85], também seguiu a mesma linha, embora utilizasse tabelas de elementos pelo fato de sua primeira implementação ter sido feita em Fortran. O sistema MAGIC [Ousterhout, 84] foi o primeiro a utilizar a técnica de "corner stitching".

Muitas técnicas que foram mencionadas, especialmente a divisão em Buckets, são tendências modernas para construção de editores gráficos, não necessáriamente para VLSI. Sabe-se que algumas destas técnicas são utilizadas também neste campo, mas devido ao pequeno número de editores dos quais se dispõe documentação efetiva (existem muitos editores que são propriedade das fundidoras de silício, que não divulgam o funcionamento interno dos seus sistema), não se tem uma informação mais apurada.

5.7 - Considerações finais sobre a escolha da estrutura de dados

Nossa experiencia pessoal mostra que, na verdade, mesmo que o gasto computacional possa ser substancialmente maior em um ou outro caso, é praticamente indiferente o método a utilizar, dado o fato de que o processo interativo é sempre lento, e o usuário, normalmente, pouca diferença sente.

É mais importante, ao invés de uma sofisticação na estrutura de dados, prover um conjunto de funções adequado ao usuário, mesmo que, eventualmente, alguma das funções seja computacionalmente complexa para a estrutura escolhida, e demore um certo tempo para ser executada.

Entretanto, caso o editor vá executar num ambiente compartilhado, por exemplo, num computador conectado a diversos usuários, e eventualmente com diversas edições simultaneas, a escolha de uma estrutura de dados que facilite as operações de redesenho de áreas é a mais adequada (aliado à otimização nas rotinas que implementem o redesenho) dado o número de vezes que essas operações são realizadas.

6. Editores hierárquicos

O projeto de um circuito integrado completo envolve um número de elementos muito grande, da ordem de dezenas de milhares para um circuito pequeno e milhões para um circuito grande. é, portanto, natural que a criação de um circuito seja realizada em partes, muitas vezes por pessoas diferentes e depois justapostos. Muitas vezes são aproveitadas partes de layouts já criados e são utilizadas células padronizadas para obter um layout final com custo e tempo de produção menores.

Os editores hierárquicos surgem como elementos de grande valia para criação, montagem e interligação de células para formar circuitos completos.

6.1 - A necessidade de hierarquia num editor de circuitos integrados

A criação do layout completo de um circuito integrado é imaginado por muitas pessoas como semelhante à programação em estilo "top-down" de software. Neste caso seria feita a criação de um programa principal, a divisão em rotinas com parâmetros, a divisão sucessiva dessas subrotinas em outras mais e mais específicas até chegar a um nível de programação direta. Podem existir bibliotecas de subrotinas disponíveis que realizem funcões comuns a diversos programas.

No caso de circuitos integrados, tem-se a partir da definição do problema, uma idéia geral da solução, e o projeto é dividido em blocos. A partir deles, é projetada a planta baixa, que é a divisão do espaço físico em células e subcélulas, e o projeto geral das "tomadas de interligação" das células. As células podem ser sucessivamente divididas em novas células, e podem existir células padronizadas (standard cells, PLAs, bibliotecas de células) que minimizam o trabalho do projetista.

O proprio desenvolvimento é semelhante: o projeto das células é feito às vezes como caixas vazias em que existem apenas definidos os pontos de conexao, em analogia com subrotinas "dummy", que

possuam especificados apenas seus parâmetros formais.

A analogia não está totalmente certa nem totalmente errada. Infelizmente, uma abordagem puramente top-down é utópica neste caso. A alocação do espaço físico é uma outra dimensão que ultrapassa a noção algorítmica do projeto. Existem no caso de projeto de circuitos integrados, muitos problemas geométricos que tornam muito mais complexo o projeto: o encaixe das células, sua interligação, minimização do layout, alimentação elétrica, etc...

Daí, a criação de um circuito ser um problema composto de duas partes: o projeto das células e a montagem das células. Essas duas partes se interrelacionam e criam um circulo vicioso: o projeto das células depende das especificacões para a montagem e a montagem depende do projeto, uma vez que não se conhece a priori o tamanho real das células a projetar.

Esta situação é sempre minimizada quando se utilizam células projetadas (por exemplo, obtidas de bibliotecas de células) ou obtidas por geradores automáticos de layout, como descritos em [Teles, 85] ou [Salembauch, 85]. Neste caso é possível saber a priori o tamanho das células, e fazer uma alocação com menor índice de interteza sobre tamanhos.

Este problema ainda é agravado pelo fato de que, eventualmente, os requisitos do projeto podem mudar ao longo da criação. Isto é tão grave, em termos de influência sobre o layout, que em ambientes industriais, é obrigatório ter o projeto completamente congelado quando a tarefa de layout comeca [Ousterhout, 84].

Para auxiliar a parte da montagem, é fundamental um editor que consiga ajudar no processo da alocação do espaço e ligação das partes componentes do circuito. Este editor deve operar de forma hierárquica, ou seja, ser capaz de tratar as colocacões de células em disposicões uma dentro da outra, em vários níveis, para que a montagem seja uma tarefa mais simples.

É importante também que o editor tenha uma interface simples com programas de DRC e roteamento, pois essas ferramentas são mui-

to utilizadas durante a fase de montagem.

Entretanto, apesar de todas as facilidades que um editor possa oferecer para facilitar a montagem do circuito, esta operação é um problema complicado, para o qual não existe médoto definitivo. O que acaba contando, afinal, é a experiência anterior do projetista, que o torna capaz de estimar com relativa precisão o tamanho das células e arbitrar as disposicões de montagem que deem como resultado um layout mais adequado.

6.2 - Características gerais da manipulação da hierarquia

Um layout hierárquico é composto de células. Cada célula é composta de:

. elementos construtivos:
retângulo, fios, etc..., como foi visto no capítulo anterior
. células internas:
As células internas também são compostas de elementos e outras células.

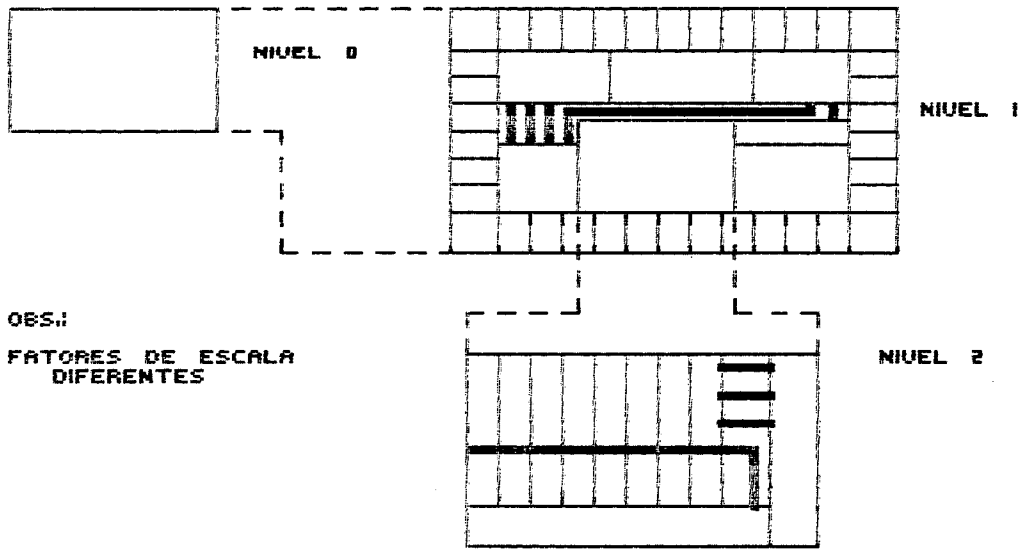
Não existe uma regra fixa para a organização da hierarquia, e cada projetista desenvolve sua própria forma de trabalho. Entretanto, historicamente, pode-se notar que grande parte dos layouts seguem mais ou menos o seguinte esquema:

a) o nível mais baixo da hierarquia contém apenas uma única célula, posicionada com a esquerda no ponto 0,0 do layout. Nesse nível não há mais nenhum elemento.

b) o nível seguinte da hierarquia contém a divisão do layout nas suas partes componentes:
. conjuntos funcionais (p. ex. conjunto alu, conjunto registradores, plas, etc...)
. pads (células de conexao às patas do circuito integrado).
. fios de conexao
. roteamento de alimentação
. barramentos internos

c) cada conjunto é dividido em células bit (p. ex. a alu é dividida em bits da alu) e células de controle do conjunto. O roteamento é feito neste nível (por fora destas células).

d) As células padronizadas (pla, memórias, etc...) seguem uma divisão de acordo com esse padrão. As células não padronizadas seguem um modelo qualquer, a critério do projetista.



Organização da hierarquia
figura 6.1

Num layout comum, o nível de expansão (células dentro de células dentro de células...) costuma estar entre 3 até no máximo 10, dependendo do estilo do layoutista, e das ferramentas automáticas utilizadas.

Um editor hierárquico possui conjuntos de comandos para manipular a hierarquia e os elementos. Os comandos para manipular elementos não apresentam grandes novidades. Em relação aos comandos já vistos nos editores mostrados nos capítulos anteriores, são os mesmos. O tratamento de hierarquia é o que será mostrado a seguir.

Em cada editor que foi estudado, o usuário manipula a hierarquia de forma um pouco diferente. Essas diferenças existem pois

cada editor procura dar ao usuário um ambiente onde exista um balanceamento adequado de alguns fatores conflitantes:

a) a manipulação da hierarquia seja a mais natural possível para o usuário

Quando o usuário está fazendo a montagem de células, é necessário executar uma série de tarefas que mais ou menos exige que o usuário tenha uma razoável noção e habilidade para a manipulação de estruturas em árvore, que afinal de contas, é a estrutura lógica que está associada à hierarquia.

O que o editor procura é esconder do usuário a árvore de hierarquia, ou restringindo de alguma forma seu livre acesso a ela (por exemplo, somente permitindo que o usuário opere sobre 2 níveis de hierarquia de cada vez, uma célula e suas subcélulas).

b) o editor deve poder mostrar diferentes visões do layout e da hierarquia

A cada momento da montagem do circuito, o usuário sente necessidade de diferentes visões do layout.

- Durante a criação da planta baixa, a movimentação das células deve ser uma operação muito rápida. O conteúdo das células não importa e sim o seu formato. é necessário também marcar e/ou visualizar os pontos de encaixe de cada célula (os lugares onde serão feitas as conexões com fios ou superposição com outras células).

- Já na fase da criação ou correção de erros das células, é normalmente uma visão detalhada só da célula, pouco importando o resto do layout.

- Durante a interligação das células ou roteamento dos canais do circuito, é necessária uma visão parcial da hierarquia, por exemplo, somente um nível hierárquico, além de marcacões de linhas na borda de cada célula mostrada.

- Durante a verificação global do circuito, é necessário mostrar todo o layout, com todos os níveis de hierarquia, apresentado em escalas pequenas e grandes, com e sem a marcação de fronteiras de caixa.

- Para documentação do circuito, é necessária a visão apenas dos canais e das bordas das células, associadas a números ou textos.

c) o editor deve gastar o mínimo tempo possível desenhando

Uma célula gasta um certo tempo para ser desenhada. Porém um layout completo pode conter muitas e muitas células. Num terminal ligado por uma interface de comunicacões serial comum, desenhar todo um layout grande na tela, numa escala pequena, pode consumir de quinze minutos a meia hora de espera !

É natural, assim, que o editor procure sempre desenhar o mínimo indispensável. Por exemplo, entre desenhar uma célula completa ou sua fronteira apenas, o editor, caso nada seja especificado, deve preferir desenhar apenas a fronteira com as marcacões de encaixe da célula.

d) Otimização do uso de memória

As estruturas de dados necessárias para armazenar um layout completo, podem ocupar vários megabytes de memória. É necessário que o editor possa manter na memória principal um conjunto pequeno do layout, por exemplo, somente a célula que está sendo manipulada num certo momento ou todas as células de nível hierárquico superior a ela. Atualizar a memória principal e a estrutura de armazenamento em disco, neste caso, deve também ser uma operação muito rápida.

e) Rapidez de operação

Além dos fatores que foram mostrados, devemos considerar que as operacões de manipulação da hierarquia são elementos adicionais ao processo de edição, e causam um certo retardo à operação do editor. O editor deve fazer com que o usuário sinta o mínimo de

necessidade de caminhar sobre a hierarquia das células.

6.3 - Principais comandos hierárquicos dos editores

A estrutura de dados que é mantida pelo editor descreve basicamente um conjunto de células, que contém elementos e subcélulas dentro delas. Na verdade uma célula pode ser considerada em dois níveis abstratos:

. a forma e o conteúdo da célula
. as instâncias da célula (os lugares onde ela é colocada)

O primeiro nível está associado a uma série de funcões, cuja finalidade é criar, alterar ou remover células desse conjunto. O segundo está associado a posicionar (inserir) e movimentar células dentro do layout.

O editor ainda prove um conjunto de comandos que se destinam a prover a seleção para da célula a editar. Essa seleção pode ser realizada de várias formas, como a seleção direta através do nome da célula, ou da seleção indireta, como apontar com o cursor para uma subcélula.

Para completar esta série, o editor fornece diversas funcões de edição, como inserção ou remoção de elementos, seleção de camadas, ajuste, etc... que operam de forma idêntica aos comandos já mostrados neste trabalho, especialmente no capítulo anterior.

Muitas vezes, o editor está acoplado a ferramentas para auxílio a projeto como DRC e roteador de canal, e possui alguns comandos para ativação dessas ferramentas.

Embora o conjunto de funcões seja muito variável de editor para editor, mostraremos aqui um conjunto de funcões, que é encontrado em muitos deles, com algumas modificacões.

a) criação de célula

Esta operação consiste em inserir uma célula na estrutura de

dados (não necessariamente na tela). Nessa operação, o operador deve fornecer o nome e um código (número) para a célula e suas dimensões.

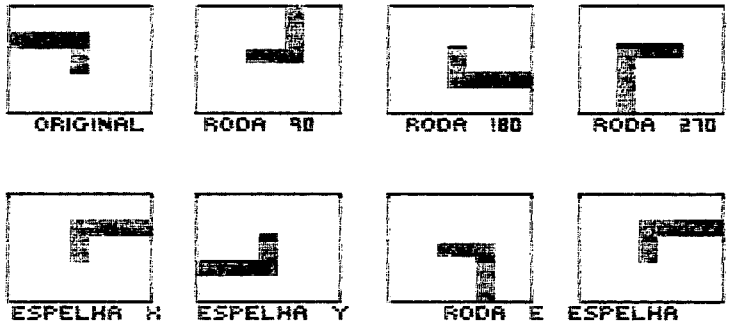
A operação de criação, em alguns casos pode ser feita com a instância da célula, por exemplo, mover o cursor até um ponto desejado e mandar criar ali uma célula. Essa última forma é muito útil especialmente durante a fase de criação da planta baixa.

b) importação de célula

'As vezes a célula existe externamente à estrutura de dados, por exemplo em um a biblioteca de células, ou num arquivo criado por outro editor. Neste caso, ela precisa ser importada para a estrutura de dados. A importação normalmente implica em copia ou conversão, e não em referência indireta, pois os formatos externo e interno na maior parte das vezes são diferentes.

c) instância de célula

Esta operação consiste em posicionar uma célula, já descrita na estrutura de dados, num ponto do layout (no interior de outra célula). A operação de posicionamento é feita movendo-se o cursor até o ponto desejado, e indicando-se a orientação com a qual a célula será colocada (8 opções no total, como mostra a figura 6.2).



opções de posicionamento de uma célula

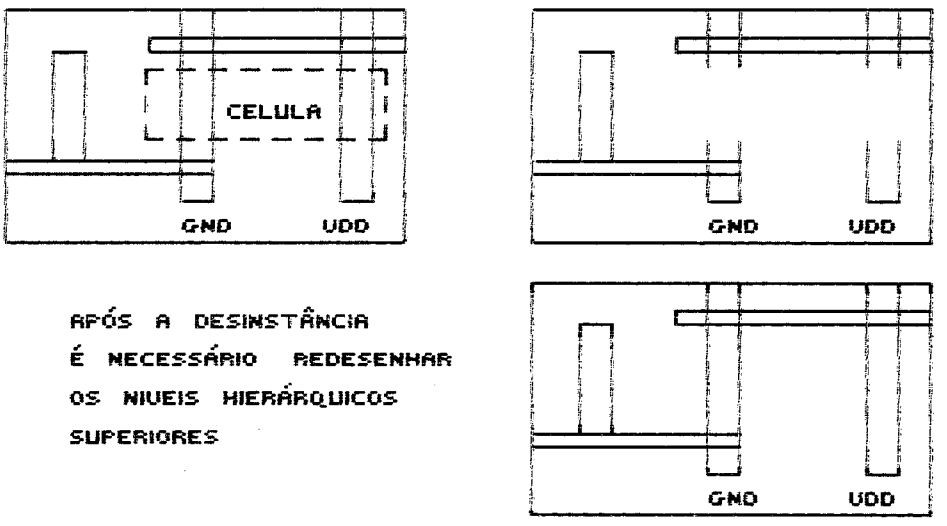
figura 6.2

Alguns editores, antes de terminar a função, mostram a área retangular que será ocupada pela célula, e o usuário ainda pode realizar um ajuste fino neste posicionamento.

d) desinstância da célula

Uma célula pode ser removida do layout normalmente colocando-se o cursor sobre ela e comandando a desinstância. É importante notar que a célula é removida do layout, mas não da estrutura de dados.

Essa operação aparentemente simples, apresenta problemas na parte visual da tela, pois não basta, por exemplo, pintar de preto a parte da tela que era ocupada por ela, pois eventualmente podem existir outras células, de nível hierárquico maior ou igual que possuam sobreposição com ela. O editor deve preservar o desenho destas outras células, redesenhando-as apos o apagamento (figura 6.3)



desinstância de uma célula

figura 6.3

e) remoção de uma célula

Uma célula pode ser removida da estrutura de dados, desde que não possua nenhuma instância no layout. O usuário especifica esta remoção através do nome ou número da célula.

f) movimentação da célula

A movimentação de uma célula em geral, é realizada da seguinte maneira: aponta-se para a célula com o cursor, e comanda-se a operação de movimentação. O cursor toma a forma da célula, e pode ser movido. O operador indica o lugar destino, e a célula é apagada do lugar antigo e redesenhada aqui.

Além das dificuldades que foram mostradas para desinstância de células, uma outra dificuldade é que a célula pode estar parcialmente contida na janela, e neste caso, o editor deve permitir que o cursor saia, pelo menos parcialmente, da tela.

g) alteração das dimensões

Quando a célula é criada, são especificadas as dimensões dela. As operacões de inserção de elementos e subcélulas dentro dela respeitam estes limites. Entretanto, pode-se alterar estas dimensões, desde que esta alteração não seja incompatível com as dimensões das células dentro das quais esta célula foi colocada.

Esta operação tem uma implicação importante na estrutura de dados: além das informacões de conteúdo das células e hierarquia, é necessário saber também, para cada célula, todas as outras células que a possuem como subcélula.

h) Especificação de pontos de conexao

Por razões de economia de tempo de desenho, o editor, ao desenhar uma célula, pode omitir o desenho das subcélulas, substituindo-as por um retangulo, hachura ou área colorida, indicativa de suas dimensões. Entretanto, é importante marcar os pontos e as camadas que serão usadas como "tomadas" para conexao ou encaixe desta sub-célula com a célula pai, células irmãs, e alimentação.

Estes pontos de conexao são muito úteis, também, para definição de células "dummy", ou seja, células que serão criadas futuramente, mas que tem um formato já determinado, para facilitar a criação do resto do circuito. Os pontos de conexao, então, apare-

cem na tela, mesmo que o desenho da célula não seja realizado, e com a representação gráfica idêntica à da camada com a qual será feita a conexao.

6.4 - Edição sobre a hierarquia

O conteúdo das células pode ser alterado, incluindo-se elementos, removendo-se elementos, ajustando, etc... exatamente como mostrado no capítulo anterior. Entretanto, a edição em ambiente hierárquico traz consigo alguns problemas, que o editor deve resolver.

a) caminhar dentro da hierarquia - entrar e sair de células

Alguns editores (a maior parte deles) exigem para realizar as operacões de edição, que a célula tenha sido isolada, ou seja, editada separadamente. Isso tem a vantagem de simplificar as operacões de transformação de coordenadas, uma vez que a célula pode estar instanciada de forma rodada e/ou espelhada, o que dificulta as operacões. Por exemplo, mover o cursor para cima, numa célula rodada de 180 graus, quer dizer mover o cursor para baixo na célula não instanciada.

Alguns editores, entretanto, dao ao usuário a liberdade de caminhar livremente nos níveis da hierarquia, ou seja, entrar e sair de células e ali realizar qualquer modificação que seja necessária. Neste caso, o editor deve prover mecanismos para impedir que o usuário se confunda com relação a qual das células ele está editando.

Um exemplo de solução elegante para este problema foi usada no editor EDCI, e é mostrada no item 6.5.

b) expansão visual da hierarquia

Os editores normalmente evitam fazer o desenho das subcélulas, para diminuir o tempo de espera do operador. Entretanto, em algumas situacões o operador quer ver uma parte ou possivelmente todo o layout, com todos os níveis de hierarquia. Nesse caso, o

editor deve possuir uma função que permita que o operador promova o desenho de alguns níveis de hierarquia de uma região selecionada.

c) Problemas de superposição de células

A primeira impressão que se tem ao imaginar um layout criado hierárquicamente, é que num mesmo nível hierárquico as células não se superponham, no máximo se toquem. Entretanto, por razões de projeto, especialmente para explicitar encaixes, muitas vezes os projetistas criam células com superposição. Isso causa alguns problemas.

Um problema é a seleção de células quando se está na região de superposição. Neste caso, a maioria dos editores seleciona aleatoriamente uma delas. Outro, é a confusão que o usuário fica, em editores que permitam caminhar pela hierarquia, quando comanda a expansão visual de duas células superpostas, e entra numa delas: em alteracões realizadas numa célula na região de superposição, fica caro para o editor refletir a situação do layout das duas células (nos editores que estudei, o editor mostra uma situação falsa no video).

d) alteracões de símbolos repetidos

Quando se altera um símbolo que possua mais de uma instância dentro do layout, a alteração deve ser propagada a todas elas. Entretanto, como o usuário tem controle sobre a forma de desenhar do editor, é muito dificil para o editor saber quais as células que estão desenhadas na tela (algumas inclusive estão parcialmente desenhadas, devido a alteracões que foram feitas em células superpostas).

Dessa forma, a tela em algumas situacões pode mostrar algumas das instâncias com o conteúdo antigo. Esse problema é muito dificil de resolver, , a menos que se adote um procedimento automático de desenho (indesejável para o usuário). Além disso, muitas vezes esse redesenho é uma operação muito demorada.

Para minorar este problema, os editores normalmente possuem uma função chamada "redesenha parte do layout", cuja finalidade maior é resolver essas atualizacões pendentes na tela.

e) criação forçada de hierarquia

Da mesma forma que programador resolve transformar um trecho complicado de programa em uma subrotina, muitas vezes o layoutista resolve transformar um trecho de layout em uma célula. Isso provoca a necessidade de uma função que remova os elementos sob o cursor e crie uma célula com eles, instanciada de tal forma que o layout permaneca idêntico. Normalmente, se um elemento está parcialmente contido na região, ele é dividido em duas partes, uma que será colocada na célula interna e outra na externa.

6.5 - Apresentação gráfica

Quase todos os editores apresentam a hierarquia criando na tela um retângulo (ou uma região retangular) de cor viva, envolvendo a área da célula, mesmo que ela não seja exatamente retangular. Esse retângulo é conhecido pelo nome de caixa limitante (bounding box) pelos projetistas.

Como foi dito, pelo fato de que os layouts completos possuem dezenas de milhares de elementos, os editores procuram desenhar o mínimo deles, visando a redução do tempo gasto na produção do desenho e envio ao terminal de video. Assim, é comum que as subcélulas não sejam desenhadas, a não ser por comando explícito do usuário.

Por exemplo, no editor gráfico EDCI, criado por nós, as subcélulas não são desenhadas, mas sim mostradas como um retângulo azul claro e as conexões com a cor da camada associada. A operação de entrar em célula promove o desenho da célula. A operação de sair de célula (opcionalmente) promove o apagamento da tela, e o redesenho do retângulo, das conexões e dos elementos de todos os trechos de ancestrais com interseção com a região apagada.

•

Existem várias solucões gráficas para selecionar no layout a

célula ou a área a editar. Exclui-se desta discussão as operacões de entrar e sair de célula, que que tem pouco impacto na parte gráfica.

a) operação com uma só janela

Este caso é o mais simples. Na tela existe apenas uma área de edição, na qual é mostrado um trecho de layout, segundo uma escala qualquer. O usuário seleciona com o cursor uma nova área ou uma subcélula a editar. A janela é apagada e a célula ou o trecho do layout colocado ali, numa escala que pode ser escolhida pelo usuário, ou deduzida automaticamente para que a região escolhida ocupe a maior área possível na janela.

Essa solução tem a desvantagem da operação de seleção: é necessário ter uma "macro-visão" da área a editar para fazer a seleção. Em alteração de muitas regiões, é necessário fazer sempre a macro-visão, o que consome muito tempo de desenho.

O editor deve proporcionar operacões de "pan", ou seja, mover a janela horizontal ou verticalmente, para visualizar regiões vizinhas à mostrada na tela.

b) Viuporte de seleção e de edição

Para facilitar a seleção, é possível dividir a tela em duas viuportes, uma destinada a localização e outra a edição. Na primeira parte, é mostrado o layout numa escala relativamente pequena. O usuário seleciona a área a editar nesta viuporte, que é mostrada na segunda viuporte em escala bem maior, para que o usuário execute as alteracões devidas.

Eventualmente pode ser possível executar alteracões, através dos mesmos comandos, na viuporte de seleção, porém, normalmente ela é criada numa escala que não permite a precisão dos comandos.

Uma das implementacões mais conhecidas deste processo é usada no Kic2, que divide a tela em duas partes. Na primeira, conhecida por "coarse viewport" pode ser selecionada uma área que será mos-

trada numa viuporte conhecida por "lente de aumento", cujo tamanho pode ser selecionado pelo usuário. Uma das características importantes desta viuporte é que ela é criada sobre um reticulado na tela (exceto quando a área selecionada é muito grande) para possibilitar medidas "a olho" na edição do layout.

c) Múltiplas viuportes

Os editores gráficos hierárquicos que operam com estacões de trabalho com inteligência no tratamento de tela, tem uma tendência a utilizar múltiplas janelas para facilitar a edição. No caso de uma seleção de área ou célula a editar, simplesmente é aberta uma outra viuporte (geralmente numa região da tela selecionada pelo usuário) que vai conter a região a editar. Um exemplo é o editor gráfico do sistema Daisy, que utiliza uma estação de trabalho especial com um software básico baseado em janelas.

Apesar do conforto que isso traz, uma vez que se pode ter múltiplas visões do layout simultaneamente, existem algumas complicacões adicionais:

. quando se faz atualização em uma viuporte, todas as outras devem ser atualizadas (na maioria dos editores isso só é feito quando se reseleciona a viuporte para edição, ou não é feito automaticamente, só sob pedido do usuário).

. a tela deve ser de alta resolução para que se possa manter um número razoável de janelas e o sistema deverá possuir uma quantidade razoável de memória para armazenar essas múltiplas viuportes. Assim, o sistema (hardware e sistema operacional) devem ter características adequadas.

. Caso as viuportes possam se superpor, (o que ocorre na maior parte dos sistemas com facilidade de múltiplas janelas de edição), sempre que uma viuporte superpuser outra, a segunda deverá ser guardada. Isso é necessario, pois não é possível redesenhar, uma vez que é o usuário quem comanda explicitamente o desenho das subcélulas desejadas no layout. é complicado para o editor manter a informação sobre "que subcélulas foram expan-

didas em que janelas.

6.6 - Tratamento matemático da hierarquia

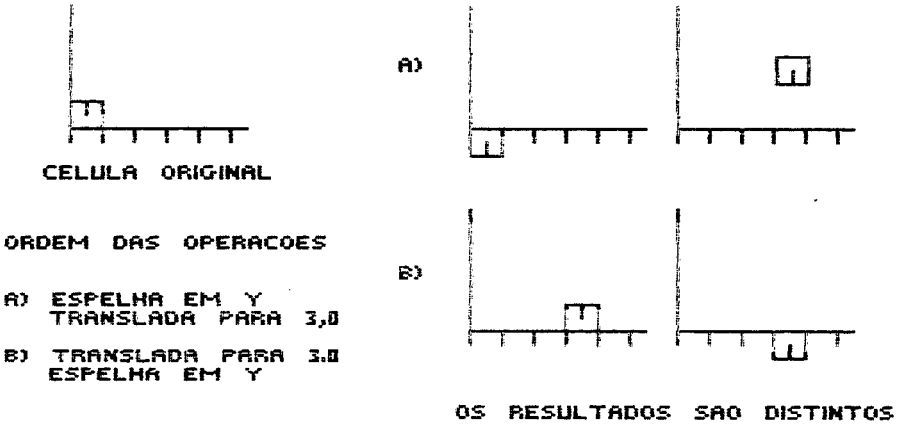
Um layout é composto de elementos e células, que sua vez podem ser compostas de outros elementos e células, e assim, por diante até o nível mais baixo, onde só encontramos elementos.

Cada uma das células é colocada dentro de outra segundo um posicionamento que pode ser expresso segundo tres operacões que são aplicadas de forma concatenada:

rodada
escalada
transladada

A rotação normalmente é restrita a ângulo múltiplos de 90 graus, mesmo em metodologias não Manhattan. A operação de escala na criação de subcélulas normalmente é restrita aos espelhamentos em x e y (ou seja, escala com fator -1).

É importante notar que a alteração na ordem dessa concatenação pode produzir figuras diferentes, como na figura 6.4.

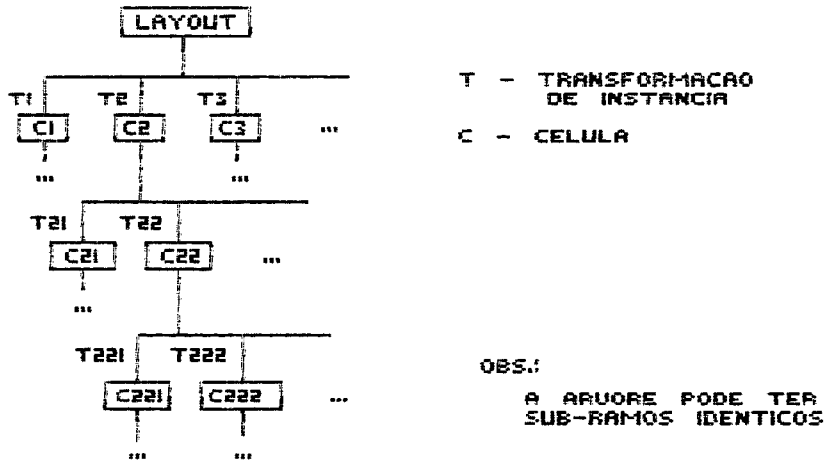


ordem na concatenação de transformacões

figura 6.4

Dessa forma, o layout pode ser imaginado como uma árvore como

a da figura 6.5, e a operação de produzir o desenho completo do layout na tela um processo de varrer uma árvore, concatenando as transformacões desde a raiz da árvore até cada folha.



árvore de hierarquia

figura 6.5

O desenho isolado de uma célula, com suas subcélulas, seria equivalente a isolar uma subárvore com raiz nesta célula. Operacões de zoom e pan também são facilmente obtidas, criando-se nessa subárvore uma nova célula raiz, com duas novas transformacões de escala e translação.

Se não houvesse necessidade de concatenar transformacões, as operacões matemáticas seriam descritas pelas seguintes equacões:

translação: $x' = x + Tx$

$y' = y + Ty$

rotação (em torno da origem): $x' = x \cos a + y \text{ sen } a$

$y' = -x \text{ sen } a + y \cos a$

escalamento: $x' = x\ Sx$

$y' = y\ Sy$

onde $Sx = -1$ produz imagem espelhada em y

e $Sy = -1$ produz imagem espelhada em x

Para varrer a árvore produzindo as transformacões concatena-

das, seria necessário aplicar, para cada elementos do layout, uma sequência de transformacões que implementasse o caminho desde a raiz da subárvore até o elemento. Por exemplo, para rodar um ponto de 90 graus em torno da origem e depois transladá-lo para Tx = -80 e Ty = 0, seria necessária a aplicação de duas transformacões

1) rotação

$$x' = y$$
$$y' = -x$$

2) translação

$$x'' = x' - 80$$
$$y'' = y$$

Entretanto é possível resolver este sistema de equacões gerando uma única fórmula concatenada, que produz a mesma transformação total.

3) concatenação

$$x'' = y - 80$$
$$y'' = -x$$

Como a resolução analítica é complicada, utilizamos álgebra linear, que nos permite através de operacões com matrizes, resolver facilmente as concatenacões. A solução de equacões que representem transformacões geométricas bidimensionais é descrita em [Newman, 79]. Dado um ponto x,y as seguintes operacões implementam

translação:

$$[\ x'\ y'\ 1\ ] = [\ x\ y\ 1\ ] \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 1 \end{vmatrix}$$

rotação:

$$[\ x'\ y'\ 1\ ] = [\ x\ y\ 1\ ] \begin{vmatrix} \cos a & -\text{sen } a & 0 \\ \text{sen } a & \cos a & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

escala:

$$[\ x'\ y'\ 1\ ] = [\ x\ y\ 1\ ] \begin{vmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

A necessidade de utilizar matrizes com tres dimensões e não de duas, ou em outras palavras, sistemas de coordenadas homogêneas [Newman, 79], é devida ao fato de que a translação não é uma transformação linear, e assim, não pode ser obtida através de multiplicacões de matrizes.

A grande vantagem deste método, é que a concatenação das transformacões, pode ser realizada pela multiplicação das matrizes.

$$[\ x'\ y'\ 1\ ] = [\ x\ y\ 1\ ] \begin{vmatrix} & & \\ & m1 & \\ & & \end{vmatrix}$$

$$[\ x''\ y''\ 1\ ] = [\ x'\ y'\ 1\ ] \begin{vmatrix} & & \\ & m2 & \\ & & \end{vmatrix}$$

então

$$[\ x''\ y''\ 1\ ] = [\ x\ y\ 1\ ] \begin{vmatrix} & & \\ & m1 & \\ & & \end{vmatrix} \begin{vmatrix} & & \\ & m2 & \\ & & \end{vmatrix}$$

Devido à propriedade da associatividade das matrizes, as matrizes m1 e m2 ainda podem ser previamente multiplicadas, gerando uma única matriz m, diminuindo o tempo de processamento:

$$[\ x''\ y''\ 1\ ] = [\ x\ y\ 1\ ] \begin{vmatrix} & & \\ & m3 & \\ & & \end{vmatrix}$$

Mais que isso, durante o percurso da árvore, sempre que se descer para um filho de um determinado no, é possível multiplicar a ma-

triz calculada para este no pela matriz de transformação entre este no e seu filho, obtendo a matriz concatenada que será usada no desenho do filho.

6.7 - Implementação das operacões de transformação

A base de operação do editor é uma matriz, denominada matriz de transformação atual e uma estrutura de pilha, na qual essa matriz e mais algumas informacões são salvas.

A matriz atual é inicializada com o valor identidade ao inicio da edição e atualizada durante as operacões de pan, zoom, ou percurso hierárquico, como descrito abaixo. Todos os elementos do layout para serem desenhados tem suas coordenadas multiplicadas pela matriz atual. é possível também saber a equivalência de um ponto da tela com as coordenadas do layout, aplicando-se a multiplicação das coordenadas do ponto pela inversa da matriz atual.

A edição isolada de um sub-símbolo, também não tem problema: basta inicializar a matriz atual com o valor identidade, e proceder ao percurso da árvore, normalmente.

A pilha serve para para guardar as seguintes informacões, ao inicio do tratamento de um sub-símbolo:

- matriz atual
- número do símbolo corrente
- limites de movimentação do cursor
- escala atual da edição
- posição do cursor

Ao terminar o tratamento de um sub-símbolo, essas informacões são removidas da pilha.

O seguinte conjunto de rotinas é suficiente para o processamento do editor:

Translada (x,y) — multiplica a matriz atual pela matriz de translação

Roda (ang)	idem pela matriz de rotação
Escala (sx,sy)	idem pela matriz de escalamento
calculaInversa (minv)	calcula a inversa da transf. atual
pushTransform (pilha)	salva a transformação atual
popTransform (pilha)	recupera a transformação atual
aplicaTransf (mat,x,y,xt,yt)	multiplica um ponto por uma transformação
selecViuporte (xt,yt)	seleciona a região da tela para viuporte

Podem também ser criadas funcões basicas para pan, zoom e escala de edição, porém elas são aplicação direta das funcões mostradas acima.

Apesar de que as funcões acima podem ser programadas em poucas linhas (no caso de nosso editor EDCI, 300 linhas de programa), em softwares gráficos básicos modernos, como CORE e GKS, essas operacões normalmente já fazem parte do repertório básico de funcões do editor, o que diminui um pouco a tarefa de programação. Em alguns desses sistemas gráficos, por exemplo PHIGS, [Phigs, 86] o tratamento de hierarquia é embutido no proprio software, o que possibilitaria ainda uma simplificação na estrutura de dados.

Entretanto além desses sistemas não estarem ainda muito disponíveis, pelo fato de serem muito gerais, gastam muitos recursos computacionais. A maioria deles, inclusive, como já foi mencionado em capítulos anteriores, não possui os requisitos gráficos suficientes para uma operação veloz do editor.

6.8 - Estruturas de dados

A maior restrição para o armazenamento de um layout completo é o fato de que o tamanho da estrutura é muito grande para que ela seja mantida toda na memória principal. Por outro lado, existe um conceito muito forte de localidade de processamento: geralmente a cada momento está se processando uma única célula, que é realmente o que precisa ficar na memória.

6.8.1 - Organização da hierarquia

Assim, utilizam-se solucões que mantem fora da memória as células que não estão sendo utilizadas, e algum tipo de controle que permita o tratamento simples da hierarquia.

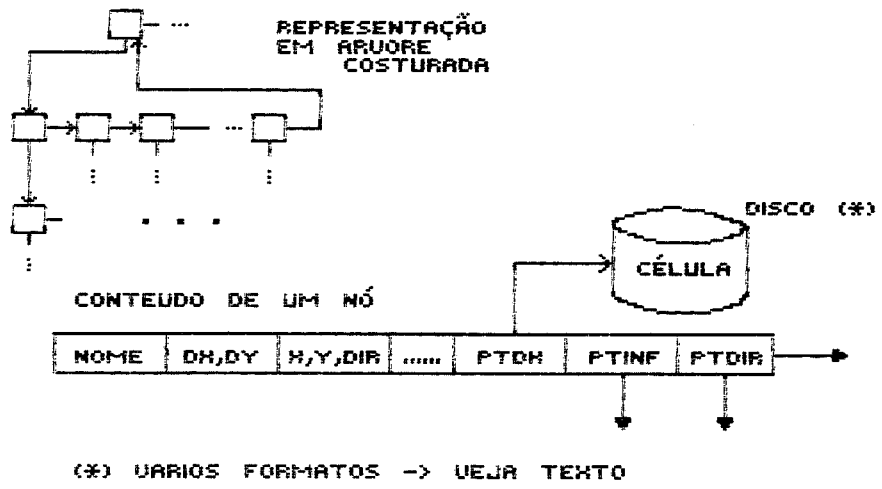
Sao duas as formas mais comuns de armazenamento de informacões para editores hierárquicos de circuitos.

a) Organização por árvore de células

Neste estilo de organização, é criada na memória uma árvore em que cada nó contém as informacões de cada célula: o nome, a localização e o tamanho da célula. Esta última informação serve para permitir que durante o desenho de uma célula, não seja preciso ler de disco as subcélulas para desenhar as caixas limitantes (bounding boxes).

Somente uma célula é mantida na memória, aquela que esta correntemente sendo editada. Em alguns casos pode existir um "cache" de células para otimizar alguns tipos de edição, especialmente de estruturas regulares.

As informacões de cada célula são as mesmas vistas no capítulo anterior, acrescidas da lista de subcélulas de cada célula, contendo o número (ou nome) da subcélula e a posição e direção de colocação (ou a matriz de transformação).



Organizacao em árvore de células

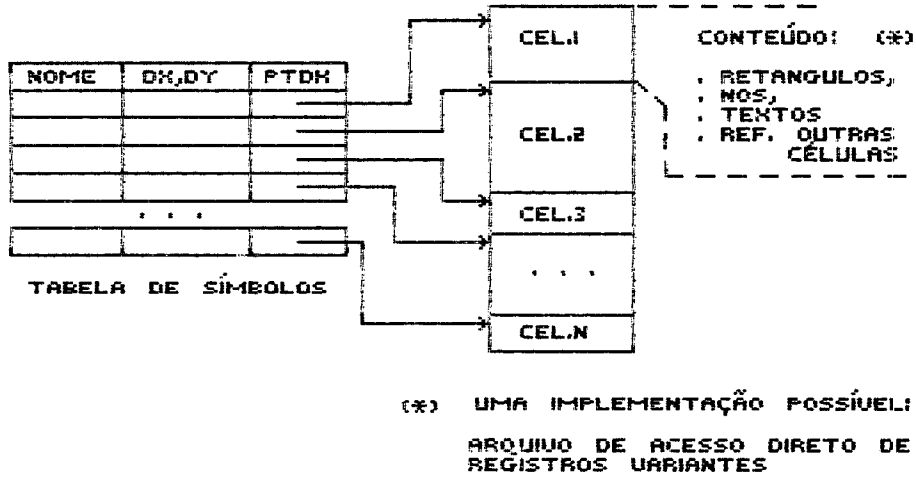
figura 6.6

b) Organização por tabela de símbolos

Nesta forma de armazenamento, a estrutura de dados é dividida em duas partes:

a) tabela de símbolos, que contém as informacões de cada célula (as mesmas informacões da organização anterior).

b) lista de células, que contém os conteúdos de cada célula. Somente uma célula é mantida na memória, embora também possa ser criado um cache de células.



organização com tabela de símbolos

figura 6.7

6.8.2 - Armazenamento das células em disco

As informacões de cada célula ao serem gravadas em disco, podem ser formatadas com os mesmos estilos vistos no capítulo anterior. O problema adicional é que são muitas células, e é necessário ter cuidado com a organização do espaço em disco. Assim, podem ser adotadas várias solucões.

a) um arquivo por cada célula

Essa é a solução mais simples: deixa-se para o sistema operacional cuidar da alocação do espaço: cada célula é gravada como um arquivo, que é aberto quando necessário.

As desvantagens do método são que a abertura de arquivos é uma operação normalmente lenta, pois envolve busca em diretório, e isso pode prejudicar a performance de certas funcões, por exemplo, a expansão completa de uma célula (desenhar a célula e todas suas subcélulas, até o nível mais baixo da árvore de hierarquia).

Alguns sistemas permitem a interface de programas com um utilitario de biblioteca de células, que eventualmente pode ser utilizado para organizar melhor as células, livrando o sistema de ter que ir muitas vezes ao diretório geral do disco, porém, estes utilitarios normalmente tem baixa performance.

b) um arquivo único de acesso direto por cada tipo de elemento

Neste estilo de organização, existe um arquivo de acesso direto único, no qual são gravadas as células, sequencialmente. As informacões das posicões das células no arquivo devem ser gravadas num segundo arquivo que, em tempo de processamento, é utilizado para gerar a árvore de hierarquia ou a tabela de símbolos.

A vantagem dessa organização é que o acesso às informacões é muito rápido. A desvantagem é que é necessário gerenciar o espaço no arquivo. Usa-se normalmente, a cada vez que uma célula é recriada colocá-la ao fim do arquivo, não reusando o espaço previamente alocado por ela e, sob comando do usuário, compactar o arquivo. Essa opção de tratamento permite a manutenção automática de versões antigas da célula, o que é especialmente útil devido a erros de operação do projetista.

Um exemplo tradicional deste estilo de organização é a forma utilizada pelo editor Icarus, e mostrada em [Fairbarn, 78].

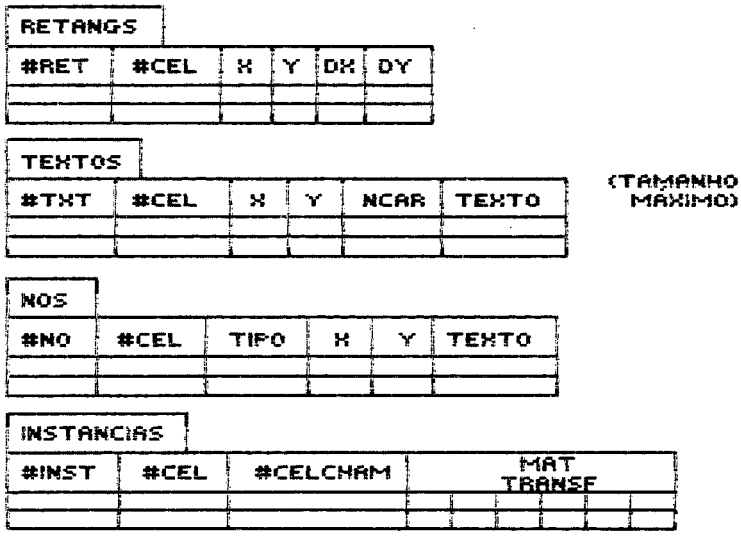
c) estrutura de banco de dados relacional

Na medida em que existe uma tendência de acoplar os editores de layout ao sistema integrado de projeto, logo surge a idéia de utilizar bancos de dados relacionais para armazenar o layout. O estudo completo sobre armazenamento de estruturas hierárquicas po-

de ser encontrado em [Date, 84], e resumido em [Foley, 82].

As vantagens deste tipo de armazenamento são as tradicionais de banco de dados: segurança, facilidade de manutenção da estrutura de dados, despreocupação com armazenamento físico e gerência de espaço em disco, facilidade de programação, etc... As desvantagens, são a baixa performance dos sistemas relacionais e o gasto maior de disco, devido aos campos extras contendo o código das células, necessários para as operacões de seleção.

Uma possível estrutura é mostrada na figura 6.8



armazenamento em banco de dados relacional

figura 6.8

6.9 - Algoritmos principais

Os principais algoritmos que são usados em editores hierárquicos são simplesmente algoritmos de percurso da hierarquia com aplicação das transformacões matriciais. Mostramos abaixo a implementação de algumas funcões mais características que demonstram a relativa simplicidade das operacões.

Vamos assumir as seguintes convencões

C - número da célula atual

T - matriz de transformação atual

P - pilha para armazenamento das transformacões durante o percurso da árvore

U - matriz de trabalho

a) mostrar uma célula na tela

rotina MostraCelula (C,T)

1. Traz a célula C do disco
2. para cada elemento da célula
    2.1. Aplica T às coordenadas do elemento
    2.2. Desenha o elemento transformado na tela, com recorte
    2.3. Para cada subcélula S de C
        2.3.1. seja M a matriz associada à chamada
        2.3.2. calcula U = M x T
        2.3.3. aplica U ao contorno (bounding box) de S
        2.3.4. desenha o contorno transformado na tela, com recorte

b) expansão (mostrar todas as subcélulas)

Essa rotina é quase idêntica à anterior, com a diferença que se chama recursivamente para desenhar as subcélulas. Os contornos não são desenhados.

rotina Expande (C,T)

1. Traz a célula C do disco
2. Para cada elemento de C
    2.1. Aplica T ao elemento e o desenha com recorte
3. Para cada subcélula S de C
    3.1. seja N a subcélula
    3.2. seja M a matriz associada à chamada
    3.3. calcula U = M * T
    3.4. Expande (N,U)

c) entrada em subcélula

Esta operação consiste em fazer com que o processo de edição passe a se realizar sobre uma subcélula da célula atual, o que corresponde a descer um nível na hierarquia.

rotina EntraEmCelula (S,M)

1. seja M a matriz associada à chamada da subcélula S
2. Guarda a célula C se tiver sido alterada
3. Empilha o contexto e a transformação atual
4. seja U = M x T
5. Aplica U ao contorno de S
6. Apaga a região envolvida por esse contorno
7. MostraCelula (S,U)
8. T = U
9. C = S

d) saída de célula

É o processo inverso ao de entrada em célula. O redesenho das regiões superpostas, em nível hierárquico mais alto, à célula foi restrito a apenas um nível.

rotina SaiDeCelula

1. seja C a célula atual
2. guarda a célula C se tiver sido alterada
3. Aplica M ao contorno de C e apaga a região de contorno
4. Desempilha o contexto de edição -> C e M
5. MostraCelula (C,M)

6.10 - O editor EDCI

Em meados de 1985 foi construído o editor gráfico EDCI, visando proporcionar facilidades de edição hierárquica e montagem de circuitos integrados completos. O sistema foi criado para executar num computador PDP-11/70 sob sistema operacional IAS, sendo posteriormente convertido para outros: RSTS, UNIX e VMS (no computador VAX/780).

Da mesma forma que o EDMOS, o EDCI usa duas telas: uma alfanumérica e outra gráfica (embora possa ser instalado, caso o terminal gráfico tenha dimensões grandes, com as duas telas mapeadas na tela gráfica). Essa divisão foi escolhida para permitir que o EDCI pudesse rodar em terminais gráficos com capacidade bastante modesta. O EDCI foi projetado para ter uma certa independência em relação ao terminal gráfico, sendo isso conseguido através de modulos gráficos que podem ser modificados para incorporar as características de novos terminais.

No EDCI, o chip é tratado como um caixote visto do alto. Dentro deste caixote podem existir 3 tipos de coisas: retângulo coloridos, textos ou outros caixotes (subcélulas). Dentro das subcélulas podem existir mais retângulo, textos e subcélulas, até um nível virtualmente infinito.

Foi minimizada a quantidade de coisas que o editor desenha automaticamente. Como estamos tratanto de layouts completos, o tempo de desenho pode ser muito grande. é o operador quem comanda, por exemplo, a expansão (desenho) das subcélulas.

O editor pode operar em escala menor que 1, para poder ter uma visão panorâmica do circuito que pode ser muito grande. Neste caso, como se perde a precisão, é proibida a inserção de retângulo nessas escalas, mas é permitida a criação de subcélulas (que eventualmente podem ser movidas e redimensionadas numa escala menor), para prover a funcionalidade de criação de planta baixa.

O EDCI permite o passeio hierárquico completo. O operador pode entrar num subsímbolo, alterar seu conteúdo, entrar em subsímbolos mais internos, mover elementos, de uma forma bastante confortável.

O controle da navegação no chip é bastante visual. Os subsímbolos não expandidos (desenhados) são mostrados através de retângulo claros, como se fossem uma tampa de uma caixa de sapatos, que é removida quando se entra no símbolo. Para manter-se a noção de hierarquia, ao sair de um símbolo, pode-se mandar recobri-lo

(remover da tela o seu conteúdo) para simplificar a visualização.

O EDCI utiliza arquivos de acesso direto para armazenar os elementos e a hierarquia, de forma muito semelhante à do sistema ICARUS, descrita anteriormente. Existem programas auxiliares para prover a conversão desses arquivos para CIF e vice-versa.

O armazenamento em memória é feito em tabelas, com busca sequencial. é mantida na memória apenas a célula que se está editando e um índice das células (contendo também suas dimensões, para não haver necessidade de buscar o tamanho da célula quando ela não for expandida, somente desenhada sua linha limitante). Apesar dessa simplicidade, o editor é muito rápido, comparado com outros editores gráficos que foram testados.

O EDCI é um editor que, apesar de muito poderoso, exige um treinamento especial do operador. Isto é devido ao grande número de comandos que ele possui (mais de 60), sendo os principais os seguintes:

. criação e remoção de retângulo e textos
. criação, remoção, entrada e saida de células
. ajuste e movimentação de células
. isolamento, ampliação e replicação de células
. criação e remoção forçada de hierarquia
. manuseio da tabela de símbolo
. manuseio de nos

A sequência de fotos a seguir demonstra algumas características operacionais do EDCI.

6.11 - Conclusões

A hierarquia é uma ferramenta poderosa na criação de layouts completos. é interessante notar, porém, que como um editor hierárquico tem muitos comandos, e alguns deles possuem efeitos colaterais, seja na forma gráfica, seja na própria constituição do layout, o usuário tende a não usar o editor hierárquico para as tarefas corriqueiras de criação de células, deixando-o exclusivamen-

te para a montagem do layout.

Interessante é observar o número de macêtes que um usuário experimentado de um editor desse tipo tem, e que permite a realização de tarefas complexas com a combinação das funcões do editor. Por exemplo, esses usuários conseguem em um tempo pequeno, fazer roteamento utilizando fios muito maiores do que a janela de edição, rearrumação do layout e reaproveitamento de partes de outros layouts.

Nota-se também a necessidade de um treinamento formal mais detalhado para que o usuário possa utilizar plenamente todas as facilidades deste tipo de editor.

7 - Conclusões

Foi apresentado neste trabalho uma síntese do conhecimento teórico e prático necessário para a criação de editores gráficos para circuitos integrados em metodologia "full custom".

Os conhecimentos aqui apresentados, foram utilizados, aprimorados ou gerados, durante a construção de diversos editores gráficos, alguns deles, puramente experimentais, outros efetivamente utilizados em produção pela equipe de projeto de circuitos integrados do NCE/UFRJ. Além desses editores ainda foram feitas instalacões de outros editores, (p.ex. CIFSYM [Lewis, 80]) em diversos computadores e terminais, e estudo dos programas fontes de alguns editores e programas correlatos.

As técnicas e algoritmos apresentados, às quais não foi associada nenhuma literatura, foram desenvolvidos ou melhorados pelo autor, através do trabalho que vem desenvolvendo, tanto através de programação quanto de orientação de programadores da equipe de desenvolvimento de ferramentas de software para projeto de circuitos integrados da Área de Desenvolvimento do Núcleo de Computação Eletrônica da UFRJ.

7.1 - Resultados práticos

Como resultado prático deste trabalho, foram gerados os quatro editores que vem sendo utilizados interna e externamente do NCE, e que fazem uso direto das técnicas aqui mostradas. Vamos contar um pouco da história destes editores.

a) EDMOS

EM 1981, havia sido desenvolvido um vídeo gráfico a cores, para ser ligado a um microcomputador nacional, o SDE/42. Nessa época, o grupo estava começando, e utilizava o editor CIFSYM, que hevia sido cedido pelo prof. Daniel Lewis, da Universidade de Santa Clara, e que produzia sua saída em terminais alfanuméricos. A visualização de um layout num terminal alfanumérico era terrível, e tivemos a idéia de criar alguns programas no vídeo gráfico (que

ainda estava em fase experimental), para visualizar os layouts produzidos no CIFSYM.

Foi produzido um programa, que chamavamos MOSCEL (mostra célula), que era uma versão de um outro programa destinado a produzir saídas em plotter (CIFPLOT), também criado por nos. Como era possível o acesso à memória de tela do terminal, notamos que seria possível melhorar o programa MOSCEL, para permitir alterar o desenho na própria memória de tela. Após uma série de melhorias, resolvemos criar um editor gráfico completo, com aproximadamente as mesmas funcões que o CIFSYM, mas executando na tela colorida.

O EDMOS foi o primeiro editor criado por nós, em FORTRAN no CP/M, e utilizava duas telas, uma alfanumérica e outra gráfica. A versão original foi escrita em cerca de 2500 linhas em FORTRAN.

O editor foi sofrendo uma série de melhorias, e diversas novas funcões, inclusive um DRC online, sub produto de uma tese de mestrado de um membro de nossa equipe [Teles, 83]. Entretanto o EDMOS tem, como todo editor de memória de tela, duas restricões: é um editor de células, e limitado às características da tela (no nosso caso, memória com largura de 5 bits (32 cores) e 240 x 400 pontos).

O EDMOS até bem pouco tempo era utilizado, especialmente pelo fato de ser conceitual e operacionalmente muito simples. Entretanto, tem sido abandonado, por um problema operacional: hoje os micros SDE/42 estão totalmente obsoletos, e só existem hoje no NCE dois desses equipamentos funcionando bem.

b) EDMOS2

Para resolver estas restricões, foi escrita em 1985 uma nova versão do EDMOS, para executar em um computador maior, e utilizando um terminal qualquer (no nosso caso, costuma-se utilizar um terminal desenvolvido no próprio NCE, um terminal DATANAV ou um AED/512).

O programa EDMOS2 foi escrito com cerca de 4000 linhas em linguagem C, incluindo 20 % de linhas em branco e comentários.

Esse editor tem características operacionais muito semelhantes ao EDMOS, também é um editor de células, mas pelo fato de utilizar a memória do computador maior, permite a criação de layouts muito maiores, e uso de tecnologias com mais camadas. As técnicas de apresentação remota de informacões gráficas foram desenvolvidas durante a criação deste editor.

c) EDCI

Durante a confecção de nosso primeiro chip, uma interface para uma rede local, as tarefas de montagem eram feitas utilizando um processo manual, e as células, depois de unidas, eram verificadas através do EDMOS (note que o layout era muito grande e não era possível editar diretamente nele).

Essa operação era muito trabalhosa, e surgiu a idéia de se criar um editor hierárquico. O editor começou a ser escrito em FORTRAN, mas devido a uma série de problemas desta linguagem, restringindo a criação de estruturas dinâmicas, logo foi transcrito para C, e sua implementação terminou uns quatro meses depois, sob nossa orientação, programado por Oswaldo Vernet S. Pires.

O EDCI sofreu muitas mudanças, especialmente operacionais, com a inclusão de diversas funcões sugeridas pelos usuários. O segundo chip que foi criado já foi montado através do EDCI. O editor possui cerca de 8.000 linhas incluindo 20 % de comentários.

E' interessante notar que os usuários não gostam muito do EDCI para executar tarefas simples. Ele tem dezenas de funcões, mas os usuários preferem utilizar o EDMOS2 (hoje chamado EDCMOS), que é mais simples de operar, e os usuários conseguem produzir layouts com mais rapidez.

As tarefas de montagem, entretanto, são realizadas pelo EDCI muito rapidamente, pois o tratamento e apresentação da hieraquia do editor são muito bons. Por exemplo, nos vimos um projetista, o

Carlo E.T. Oliveira (que tem uma prática extraordinaria, é claro), montar um circuito integrado simples, e executar o roteamento com 16 "pads", a partir das células já prontas (uma PLA e alguns registradores), em cerca de duas horas.

Nota:

O EDMOS2 e o EDCI fazem parte de um pacote de software de CAD para projeto de circuitos integrados (CADMOS) que executa em computadores VAX sob UNIX ou VMS.

d) TEDMOS

Um dos problemas que tínhamos era a parte prática dos cursos de projeto de circuitos integrados. Possuíamos apenas três terminais gráficos, que eram disputados além de nossos projetistas, por turmas de 20 a 50 alunos.

Surgiu então a idéia de construir um editor gráfico que rodasse em PC, que é um equipamento muito disponivel em nossa Universidade. Assim, foi criado o TEDMOS, que além do editor de células, incorporava, na versão I, facilidades de impressão, DRC, extrator de circuitos e simulador lógico.

O sistema utiliza uma metodologia não hierárquica matricial, e as rotinas de manuseio da tela gráfica, bastante sofisticada. O sistema foi escrito em Turbo Pascal e a versão I continha, incluindo 20% de linhas em branco e comentários, aproximadamente 3500 linhas para o editor, 500 para o impressor, 1500 para o DRC, 1500 para o extrator e 3000 para o simulador.

Estamos ainda desenvolvendo um montador (um editor reduzido) hierárquico, para computadores PC, e que será acoplado ao sistema TEDMOS. Este sistema vai permitir que através do sistema TEDMOS seja possível executar o projeto e montagem completa de chips de complexidade relativamente pequena, voltado especialmente para aplicacões em multiprojetos de universidades.

O TEDMOS possui hoje cadastrados, cerca de uma centena de instituicões de ensino e pesquisa, não só no Brasil, como em di-

versos países como México, Venezuela, Colômbia, Argentina e Espanha, o que corresponde a milhares de usuários. Entre as razões que provocaram o sucesso do sistema estão a integração dos módulos, o fato de rodar em PC e de ser distribuído gratuitamente (pedimos apenas uma pequena colaboração para as despesas com o correio e manual).

7.2 - Construção por prototipagem - experiência prática

Os editores foram geralmente construídos através de metodologia de prototipagem. Duas metodologias foram utilizadas para construção dos editores:

a) prototipagem estilo "add on"

Era construída uma versão pequena, os usuários usavam um pouco, sugeriam funcões, o editor ia crescendo, até chegar à versão final. Essa metodologia foi usada especialmente na construção dos nossos primeiros editores.

b) prototipagem estilo "throw away"

Eram construídos diversos modulos de teste, testados, depois jogados fora. Depois era construído um editor na versão quase final, e aprimorado com auxílio dos usuários. Este estilo foi utilizado especialmente na construção do editor EDCI.

Grande parte das características dos editores, portanto, foi sugerida pelos usuários. Através da observação direta do uso deles, muitas idéias surgiram, e muitas melhorias foram introduzidas. Os usuários também sofreram diversos reveses, como por exemplo, perder um trabalho imenso por um erro bobo do editor.

E' importante notar que, como foram construídos diversos editores, cada um apresentava melhorias operacionais em relação aos anteriores, e também uma ergonomia mais adequada. Pelo fato de que os usuários influiam diretamente no projeto dos editores, eles sempre se mostraram (apesar de muitas vezes pressionados por problemas de tempo), dispostos a experimentar conosco.

7.3 - O editor perfeito

Apesar de termos construído diversos editores, e os usuários também terem tido acesso a alguns outros, notamos que os usuários mais experientes gostam de usar diversos editores, cada um deles para realizar algumas tarefas.

Por exemplo, com relação aos nossos editores: os usuários gostam de fazer os primeiros layouts com TEDMOS, que executa em PC (temos vários desses equipamentos). À medida que o layout vai ficando maior, eles preferem usar o EDMOS ou EDMOS2, especialmente porque a cor do terminal é melhor que a do PC. Eventualmente os usuários voltam ao TEDMOS para realizar DRC sobre a célula. A montagem é feita no EDCI, que é o editor hierárquico que temos, ou quando o trabalho é feito fora da UFRJ, no Kic2, que existe instalado em alguns centros de pesquisa no Brasil.

Não é só uma questão de disponibilidade de equipamento. E' que o usuário gosta de certas características do editor, que realmente melhoram a performance ou a visualização do trabalho. Essas características na maior parte são decorrentes da estrutura interna, e não são adaptáveis de um editor para outro.

Resumindo, da mesma maneira que não existe um "editor de textos perfeito", também não existe um editor gráfico para circuitos integrados perfeito. E os usuários sempre gostam de dar sugestões de melhorias...

7.4 - Edição "full custom", um campo em extinção ?

Existe uma tendência mundial para que os novos projetos de circuitos integrados tenham cada vez maiores índices de automatização. Nesta linha, por exemplo, se nota o florescimento de sistemas para projeto em "gate arrays" ou "standard cells", no qual o projetista descreve a rede lógica, por exemplo através de um editor de diagramas lógicos e o layout é gerado automáticamente.

Outra tendência é a integração das informacões através de bancos de dados, que podem descrever todas as fases do projeto, desde a especificação funcional até as máscaras. Esta abordagem

garante a consistência inter-níveis do projeto, mas exige uma abordagem puramente "de cima para baixo" no projeto, o que provoca que a geração do layout seja necessariamente automática.

Isso, à primeira vista, pode sugerir que os editores voltados para projeto "full custom" tenham seus dias contados. Isso não é verdade, pelo menos por enquanto, pelas seguintes razões:

a) Os layouts gerados automaticamente, são produzidos mais rapidamente e seu custo de projeto é menor. Entretanto, os circuitos gerados são muito maiores. Assim, quando um circuito é produzido em escala, seu custo de produção (tirando fora o custo de projeto), é maior. Além disso, existem limitacões quanto ao tamanho máximo de fabricação que um layout pode ter.

b) A eficiência de um circuito gerado automáticamente é geralmente menor do que um layout "full custom".

c) As células padrões e os "gate arrays" devem ser criadas por projetistas que, certamente, vão usar um editor "full custom". E esse é um campo de vital importância para a independência tecnológica latino-americana: produzir seus próprios "gate arrays".

d) Existem algumas ferramentas automáticas que não geram o layout completo, e sim partes do layout, por exemplo, geradores de PLA. Nesse caso, os circuitos gerados precisam ser montados e interligados.

e) Muitas vezes é necessário modificar o layout gerado automáticamente, especialmente para aumentar a performance do circuito.

Nos parece que a abordagem mais correta é a seguinte: automatizar e usar o editor para ligar e completar. Desta forma, um editor deve ser esperto o suficiente para se acoplar facilmente as ferramentas automáticas de modulos de layout, dando o máximo de facilidades de:

- montagem hierárquica
- ajuste e compactação de layout

. roteamento de canais embutido

Nós acreditamos que a criação de células, tende a se tornar cada vez mais uma tarefa praticada quase que exclusivamente nas fábricas de circuitos integrados. Porem, só agora as fábricas de circuito integrados brasileiras começam a aparecer, e assim, certamente há lugar hoje mesmo para este editores de células.

Um outro campo em que a utilização desses editores é absolutamente necessário é o ensino de tecnologia e formação de recursos humanos. Da mesma forma que um projetista TTL deve conhecer um pouco de resistores, capacitores, diodos e transistores, é muito importante que mesmo o projetista que faça todo o projeto de forma automática, tenha uma boa noção de como é o funcionamento do circuitos que formam um chip.

7.5 - Direcões futuras desta linha de pesquisa

Existem diversos temas ainda a explorar nesta linha de pesquisa e em outras correlatas. Mencionamos a seguir alguns trabalhos que teriam especial interesse se fossem realizados:

a) Acoplamento dos editores a bancos de dados de projeto.

Até hoje, muito se falou sobre isso, vários artigos foram escritos, mas ninguem conseguiu criar um sistema "full custom", onde as informacões realmente estivessem integradas.

b) estudo sobre algoritmos eficientes de ajuste e compactação de layout.

O projetista deveria poder criar o layout com pouca preocupação com dimensões e distâncias, e a máquina deveria poder fazer o trabalho de colocar o layout de uma forma mais compacta e eficiente. Seria algo na linha de misturar diagramação por "sticks" com técnicas de "plowing".

c) o estudo das características ergonomicas dos editores gráficos.

A ativação das funcões dos editores gráficos é diferente de editor para editor. As funcões são criadas pelo programador, através de métodos empíricos e experiência. Não existem trabalhos estudando, por exemplo, qual a forma mais eficiente de ativação, ou qual a influência da forma dos comandos na produtividade de projetistas.

d) criação de um ambiente multi-janelas para edição de circuitos integrados.

Este é um trabalho prático que nunca foi feito no Brasil. Existe muito "know how" a ser adquirido neste campo, tanto do ponto de vista de sistemas gráficos, quanto de ergonomia.

e) Algoritmos de geometria computacional aplicados a editores gráficos de circuitos integrados

Existe pouca bibliografia sobre este assunto, embora exista uma razoavel quantidade de referências sobre geometria computacional. Uma tese interessante seria localizar e aplicar os algoritmos adequados para os problemas que foram citados no decorrer desse nosso texto.

f) GKS, PHIGS e edição gráfica para dispositivos de rastreio.

Esses sistemas gráficos, apesar de padrões, aparentemente dão pouca ajuda a editores que não trabalhem com linhas. Este seria um estudo sobre a aplicabilidade e deficiências de sistemas gráficos quando aplicados a edição gráfica em dispositivos de rastreio.

Bibliografia

(1) ASANO,T., SATO,M., OHTSUKI,T., "Computational Geometry Algorithms", Layout Design and Verification, Advances in CAD for VLSI series, North Holland (1986)

(2) BAKER, C., "Artwork Analysis tools for VLSI Circuits", Laboratory for Computer Science, M.I.T., TR-239, May/80 (1980)

(3) BARBOSA,M.A., SCHMITZ,E., BORGES,J.A., MARTINS,M.F., "Corisco - um sistema para apoio a projeto de circuitos integrados, anais do Congresso da Sociedade Brasileira de Microeletrônica, (1987)

(4) BEZERRA, P., "LPG - Linguagem para Procedimentos Graficos", anais da 1a. Oficina de Microeletrônica, (1980)

(5) BORGES, J.A., "Ferramentas gráficas para o projeto de circuitos integrados", Anais do Congresso Nacional de Informática, (1983)

(6) BORGES,J.A., "Um apanhado sobre terminais gráficos", Boletim Informativo 01/83, (1983)

(7) BORGES, J.A, "Editores gráficos para projeto de circuitos integrados", Anais do II Simpósio Brasileiro de Concepção de circuitos integrados, (1985)

(8) BORGES,J.A. e ROCHA,A.R., "Sobre a metodologia de prototipagem e sua aplicação a um sistema de CAD para projeto estruturado de Sistemas" - a ser publicado

(9) BORGES,J.A. et SCHMITZ,E., "TEDMOS, um sistema de CAD para auxílio a projetos de microeletrônica", Física y Tecnologia de Semicondutores, Universidad Autonoma de Puebla, Mexico, Ed. especial, (1987)

(10) BORGES,J.A. et SCHMITZ,E., "TEDMOS - um sistema de CAD para ensino de microeletrônica em microcomputadores compatí-

veis com IBM/PC", Anais do Congresso Nacional de Informática (1987)

(11) BORGES,J.A. et SCHMITZ,E., "TEDMOS - um sistema de CAD para ensino de projeto de circuitos micro-eletrônicos de alta integração", Anais do IV Simpósio Brasileiro de Concepção de Circuitos Integrados" - (1987)

(12) CANTO Fo., A.B., "Lacsuz - Um Sistema Editor de Microcircuitos", Anais do II Simpósio Brasileiro de Concepção de circuitos integrados - (1985)

(13) DATE,C.J., Introdução a sistemas de Bancos de Dados, Ed. Campus (1984)

(14) DUDENEY,H.E., Amusements in Mathematics, Dover Publications,Inc., (1917)

(15) ELLIOTT,D., Microlithography - Process technology for IC fabrication, McGraw Hill, (1986)

(16) FAIRBAIRN, D. et ROWSON, J., "Icarus, an interactive integrated circuit layout program", Anais da 15th. Design Automation Conference (1978)

(17) FOLEY,J. et VAN DAM,A., Interactive Comeuter Graphics, Addison Wesley, (1982)

(18) HON,R. e SEQUIN,C., "A guide to LSI implementation", Xerox PARC, technical report SSL-79-7, (1980)

(19) HOROWITZ,E. et SAHNI,S., Fundamentos de Estruturas de Dados, Ed. Campus - (1984)

(20) KELLER, K. et BILLINGSLEY,G., "KIC: a graphics Editor for Integrated Circuits", Berkeley VLSI Tools, a user guide, Computer Science Division, University of California, Berkeley, (1982)

(21) KELLER,K. et NEWTON,A. - "KIC2: A low-cost, Interactive Editor for Integrated Circuit Design", Proceedings Spring COMPCON, (1982)

(22) LEWIS, D., "Layout design on an alphanumeric terminal", VLSI Design, vol II, no. 4 - (1981)

(23) MARTINS,M.F. et SCHMITZ,E.A., "O roteador de canal do conjunto de ferramentas de projeto do NCE/UFRJ", Anais do II Simpósio Brasileiro de Concepção de circuitos integrados, (1985)

(24) MEAD,C. e CONWAY,L., Introduction to VLSI Systems, Addison Wesley - (1980)

(25) NEWMAN,W. et SPROULL.R., Principles of interactive computer graphics, 2nd. edition, McGraw Hill, (1979)

(26) OLIVEIRA,C.E.T. - "Projeto de estruturas de armazenamento em circuitos integrados", COPPE/UFRJ, Tese de mestrado (1987)

(27) OUSTERHOUT,J. - "CAESAR - an interactive editor for VLSI circuits", VLSI Design, Fourth quarter, (1981)

(28) OUSTERHOUT,J. - Editing VLSI Circuits with CAESAR", Berkeley VLSI Tools, a user guide, Computer Science Division, University of California, Berkeley, (1982)

(29) OUSTERHOUT,J., HAMACHI,G.,MAYO,R., SCOTT,W. et TAYLOR,G., "MAGIC - A VLSI layout system", proceedings of the ACM IEEE - 21st design automation conference, (1984)

(30) OUSTERHOUT,J. - "Corner Stitching: a data structuring technique for VLSI layout tools", IEEE Transactions on CAD/ICAS, January, (1984)

(31) PEDROZA,A., SCHMITZ,E., BORGES,J.A. et FOURNIER,R., "Projeto e fabricação de um microprocessador de oito bits com

tecnologia latino-americana", Comunicação técnica, Congresso da Sociedade Brasileira de Microeletrônica, (1987)

(32) PIRES,O.V. et BORGES,J.A., "Estruturas de Dados e Algoritmos de um Editor Gráfico para Projetos VLSI", Anais do II Simpósio Brasileiro de Concepção de circuitos integrados, (1985)

(33) SALEMBAUCH, P . et SCHMITZ, E., "Caracteristicas Area x Velocidade para Alguns Tipos de Somadores", Anais do II Simpósio Brasileiro de Concepção de circuitos integrados, (1985)

(34) SCHMITZ,E., BORGES,J.A. et TELES,A.A., "O sistema de apoio a projeto de circuitos integrados do NCE/UFRJ", NCE/UFRJ, Relatório Técnico 03/85, (1985)

(35) SCOTT,W. et OUSTERHOUT,J., "Plowing - interactive stretching and compaction in MAGIC", proceedings of the ACM IEEE - 21st design automation conference, (1984)

(36) SILVA,A.T., RODRIGUES.,L.A.B. et BORGES,J.A., "Hardware e software para terminais gráficos no NCE/UFRJ", Anais do II congresso da Sociedade Brasileira de Computação, (1982)

(37) SILVA, H.T. - "Estruturas Regulares para controle em Circuitos Integrados", Tese de Mestrado, COPPE/UFRJ, (1985)

(38) SILVA Fo.,Y., SCHMITZ,E. et FALLER,N., "E' Possivel projetar Circuitos Integrados no Brasil", Anais do Congresso Nacional de Informática, (1983)

(39) TELES, A.A., "Verificação de Projeto de Circuitos Integrados", Tese de Mestrado, COPPE/UFRJ, (1983)

(40) TELES,A.A e SOUZA,L.F.P., "Extrai - Extrator de Circuitos Integrados NMOS", anais do II Simpósio Brasileiro de Con-

cepção de circuitos integrados, (1985)

(41) TORI,R., ARAKAKI,R.,MASSOLA,A. et FILGUEIRAS,L., Fundamentos de Computação Gráfica = Computgrafia, Editora LTC, (1987)

(42) UEDA,K., KASAI,R. e SUDO,T. - "Layout strategy, Standardization, and CAD tools", Layout Design and Verification, Advances in CAD for VLSI series, North Holland - (1986)

(43) ULLMAN, J., Computational aspects of VLSI, Computer Science Press, (1984)

(44) WIRTH, N., Algorithms + Data Structures = Programs, Prentice Hall, (1976)

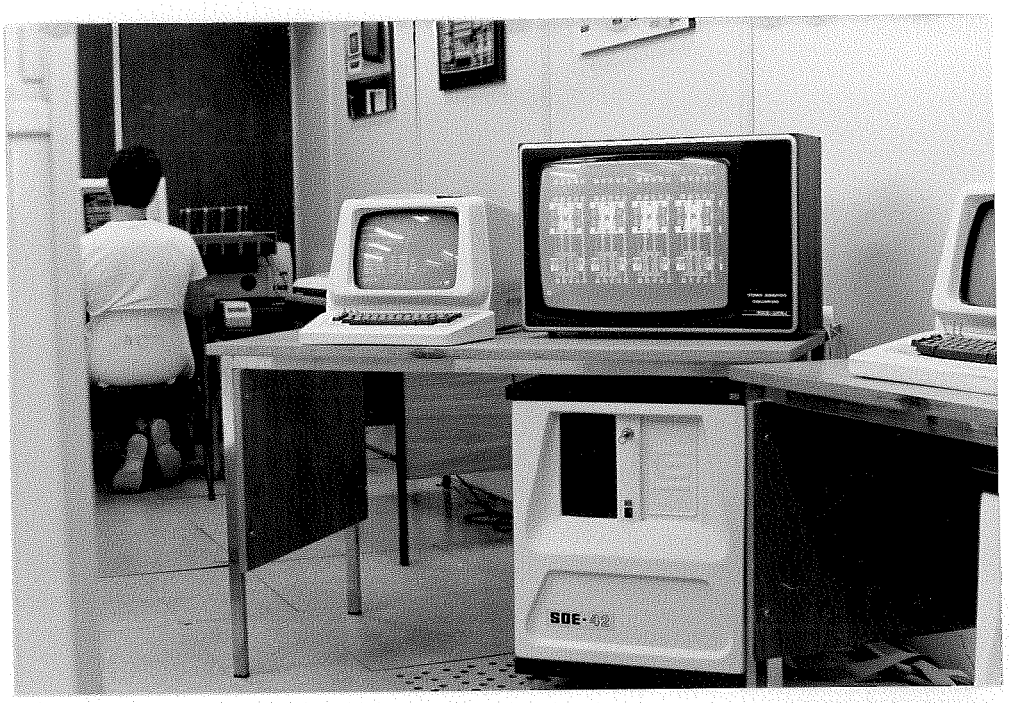
(45) YOSHIDA,K., "Layout verification", Layout Design and Verification, Advances in CAD for VLSI series, North Holland, (1986)

Obs.:
Para o pesquisador no assunto, e´ muito importante tambem estudar folhetos de propaganda dos fabricantes, que fornecem uma visao importante, especialmente do ponto de vista operacional. No nosso caso foi estudado material promocional dos seguintes fabricantes de CAD para circuitos integrados:

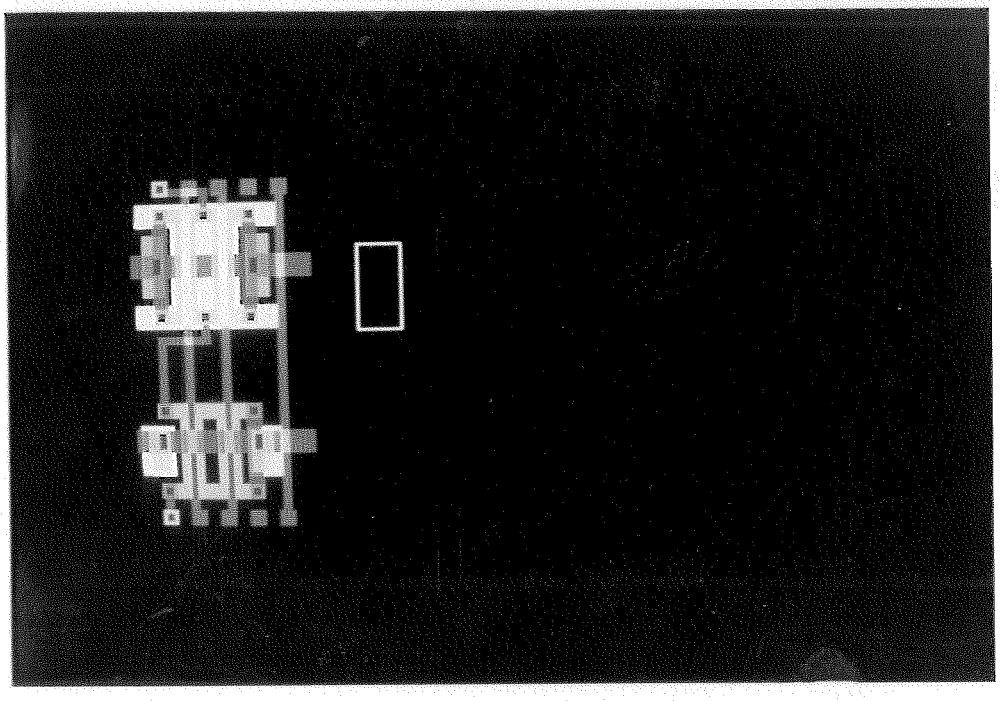
Applicon
Calma
Daisy Systems
Intergraph

BIBLIOGRAFIAS

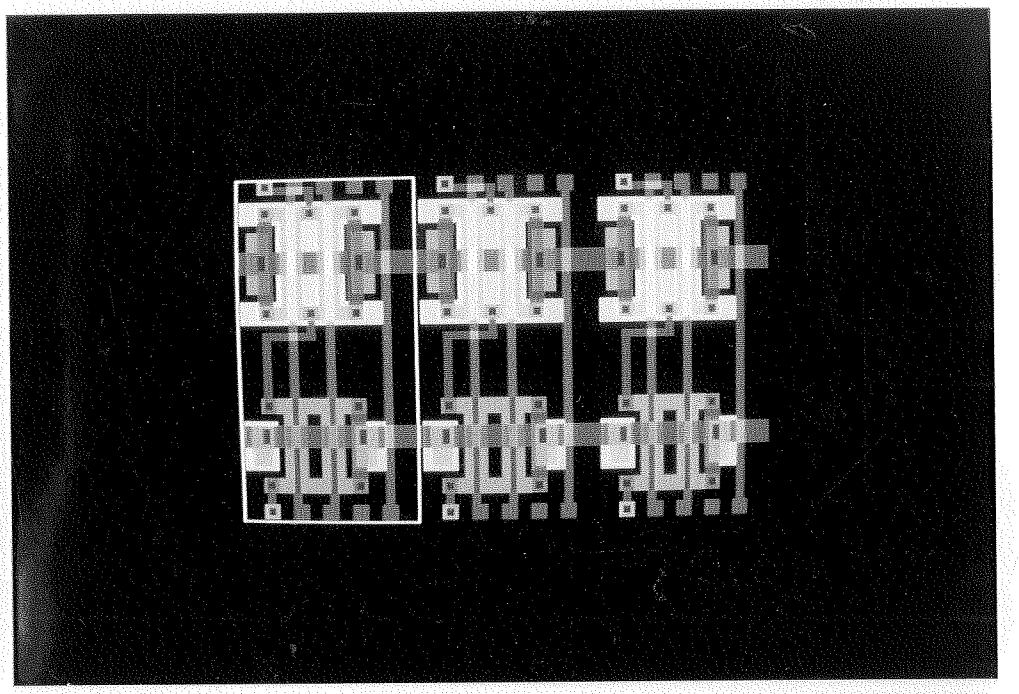
E D M O S



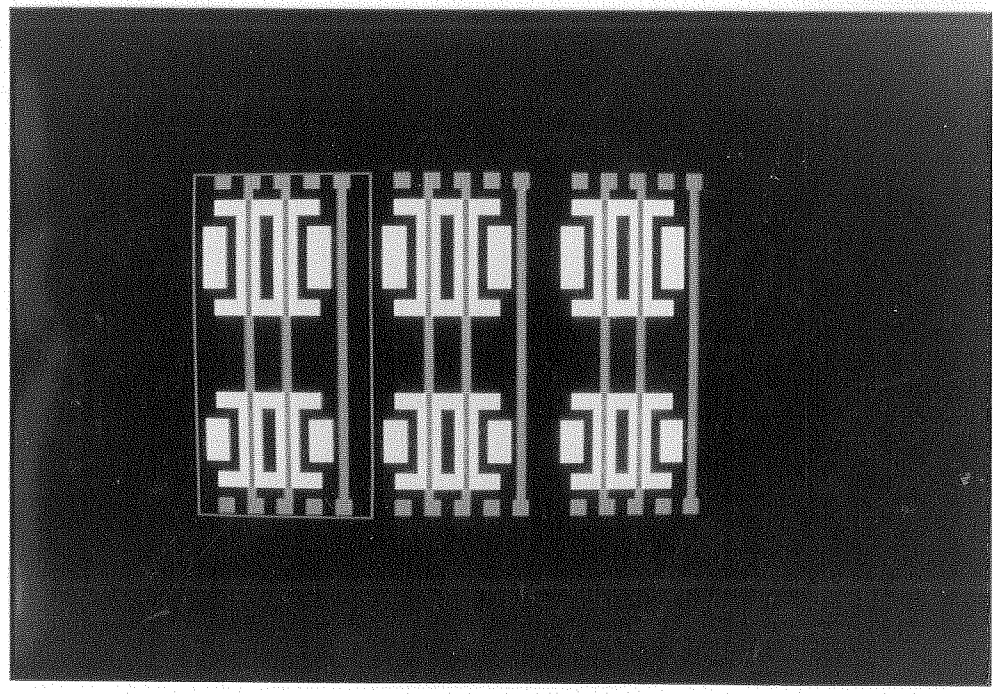
Visão do SDE/42, equipamento base do editor EDMOS



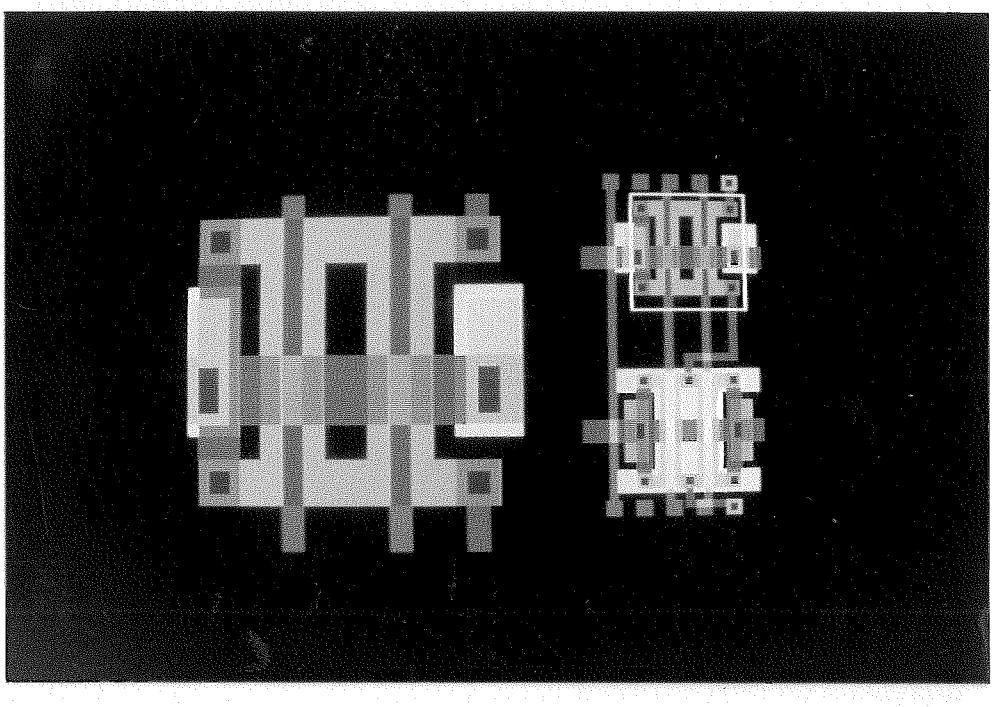
Uma célula editada no EDMOS. O retângulo branco é o cursor



Replicação de um trecho do desenho



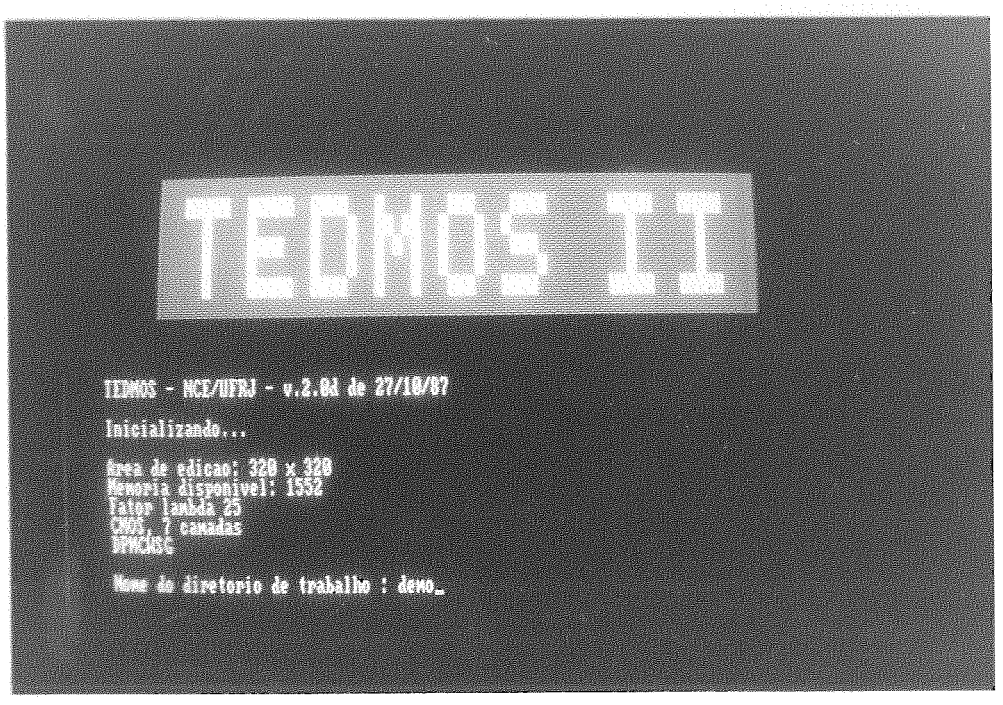
Visibilidade parcial das camadas



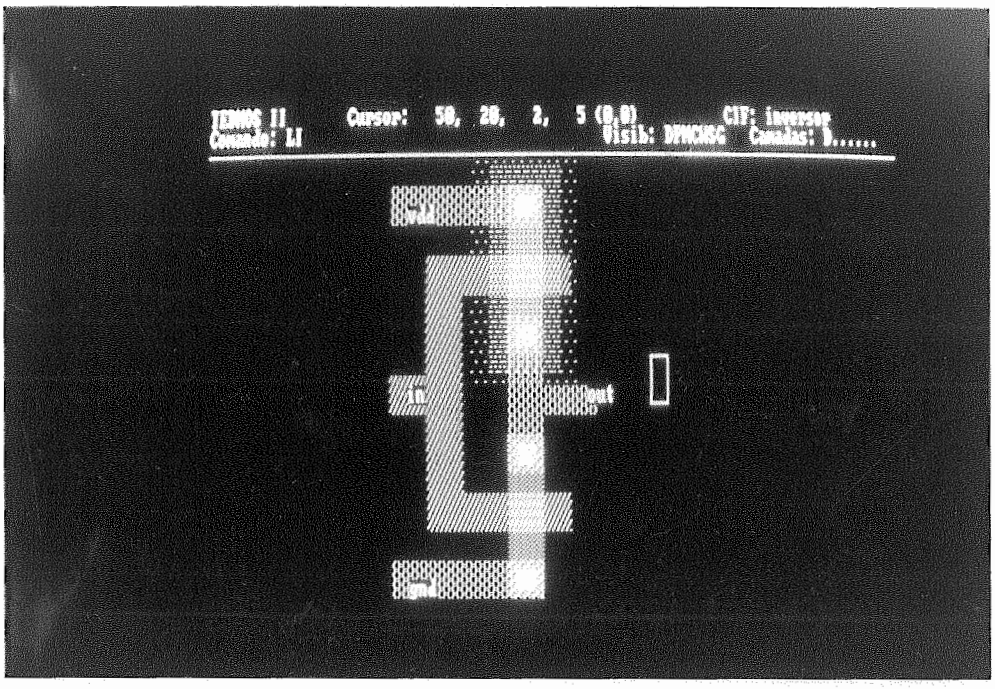
Ampliação (zoom)

I E D M O S

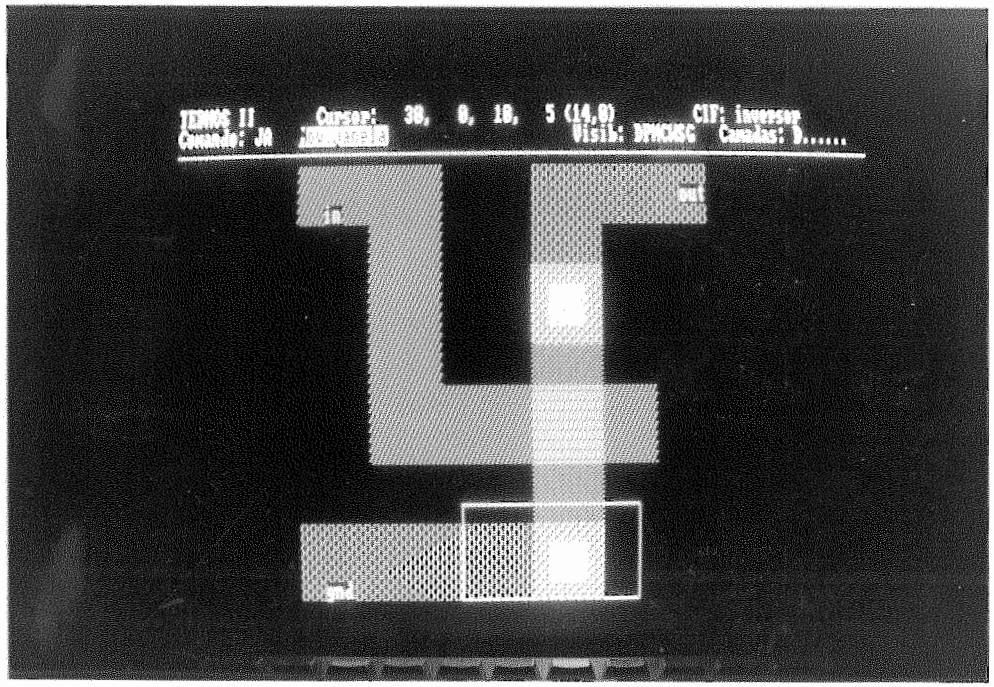
Equipamento compatível com IBM/PC, base para o TEDMOS



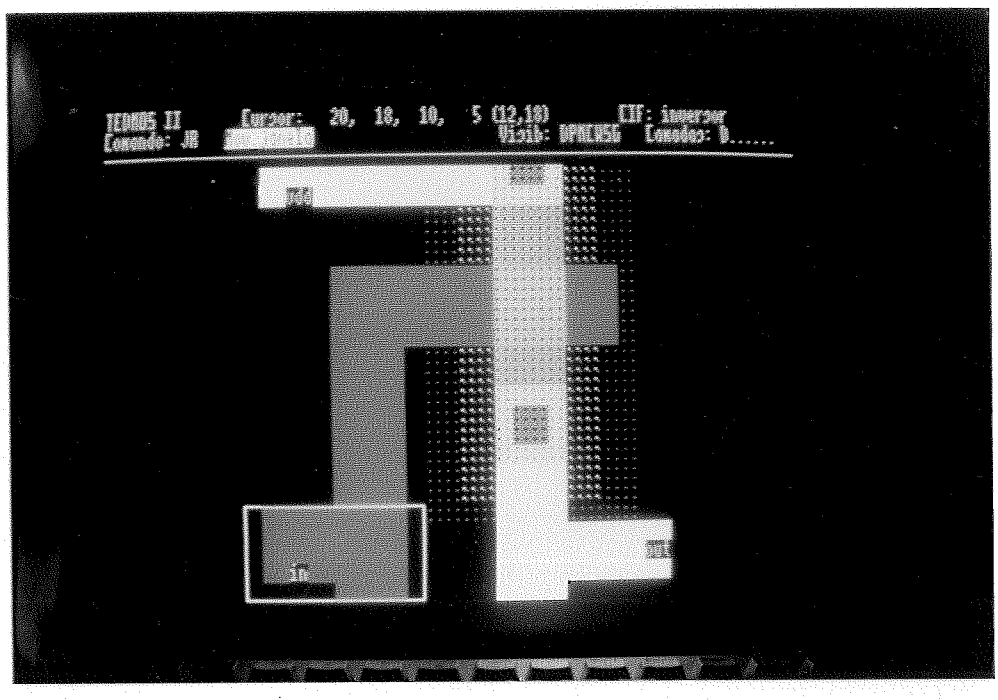
O TEDMOS pode ser configurado para diversas tecnologias MOS



Visão de uma célula editada em modo monocromático

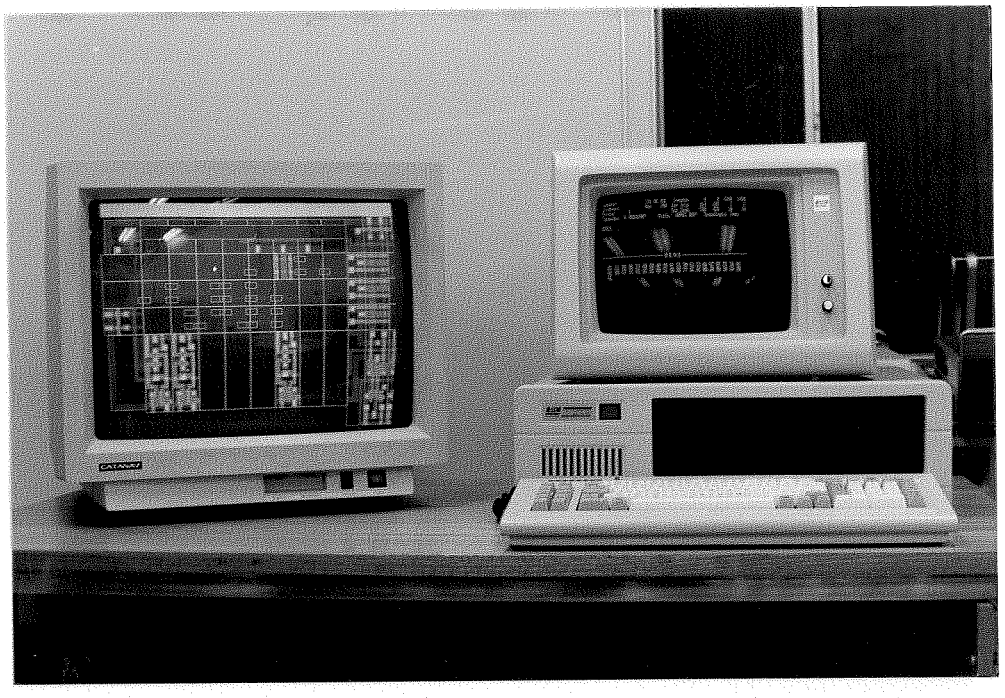


Um trecho da célula visto em outro fator de ampliação

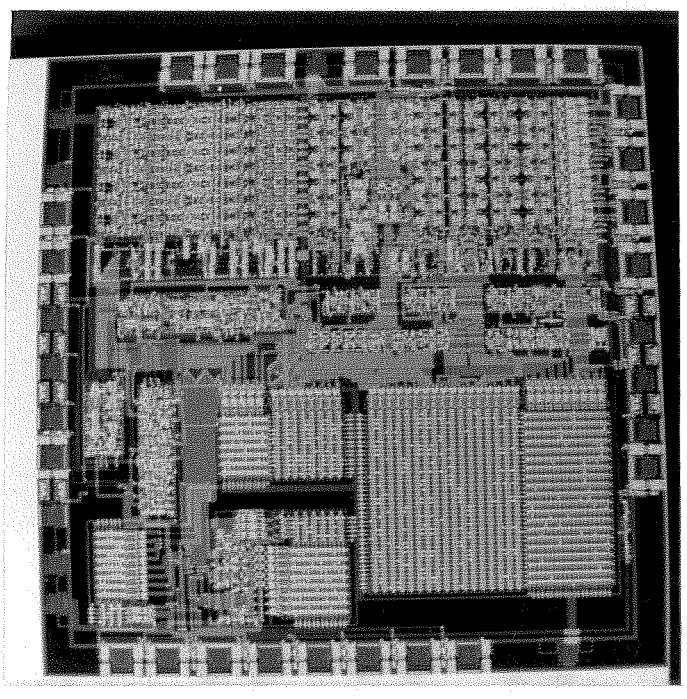


Edição em modo colorido

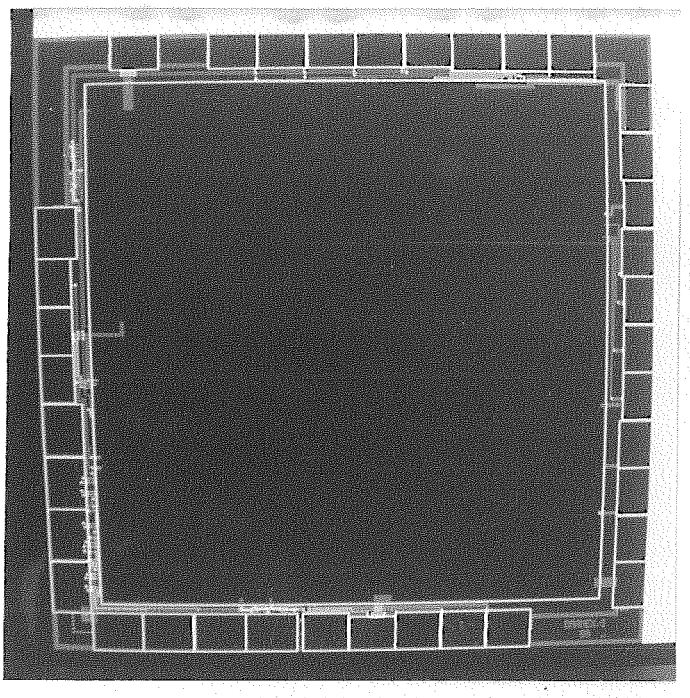
E D C I



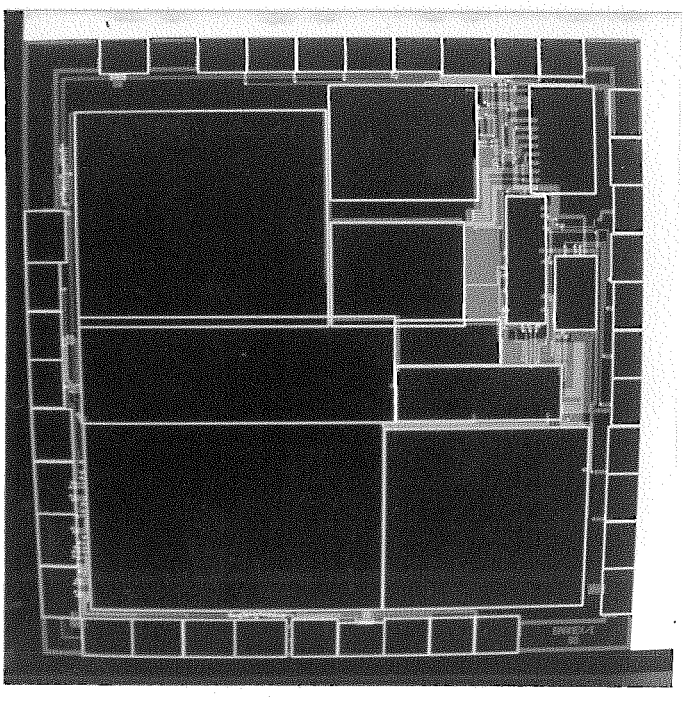
Uma das estações de trabalho onde o EDCI está instalado.
Micro compativel com IBM/PC, com interface para vídeo
de alta resolução (1024 x 756 pontos).



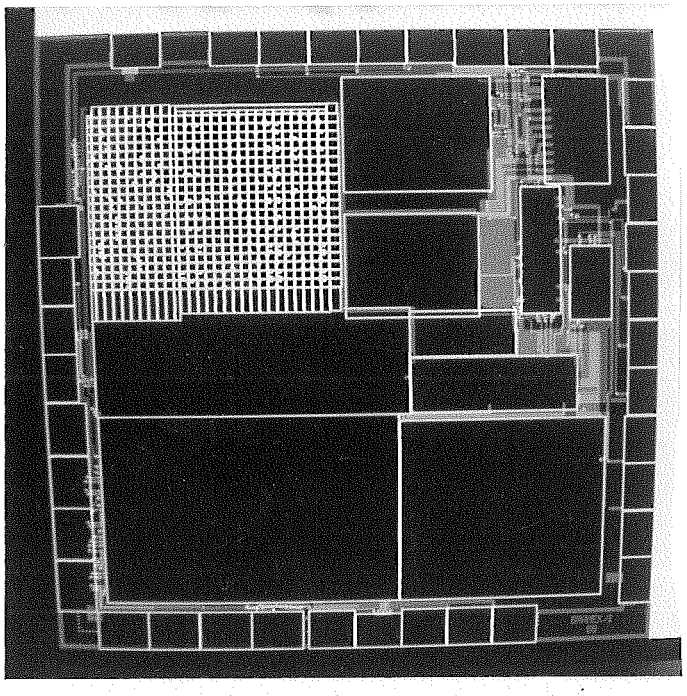
Com o EDCI pode-se editar circuitos de alta complexidade.
Na foto vê-se o microprocessador de 8 bits, BRAMEX/1.



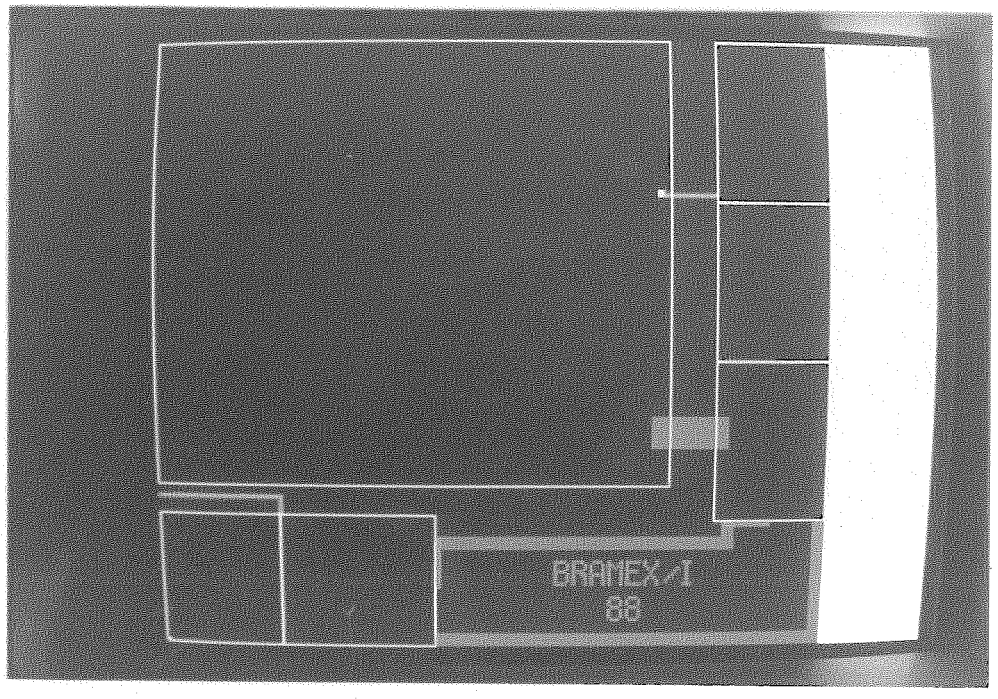
Nível mais alto da hierarquia do BRAMEX/1



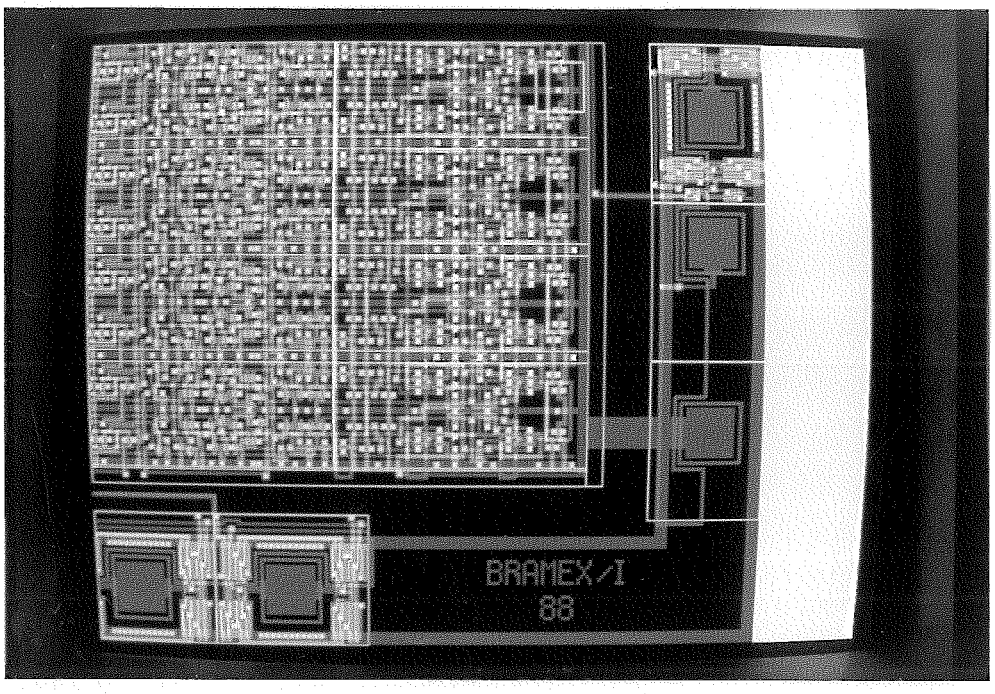
Planta baixa e parte dos barramentos do BRAMEX/1



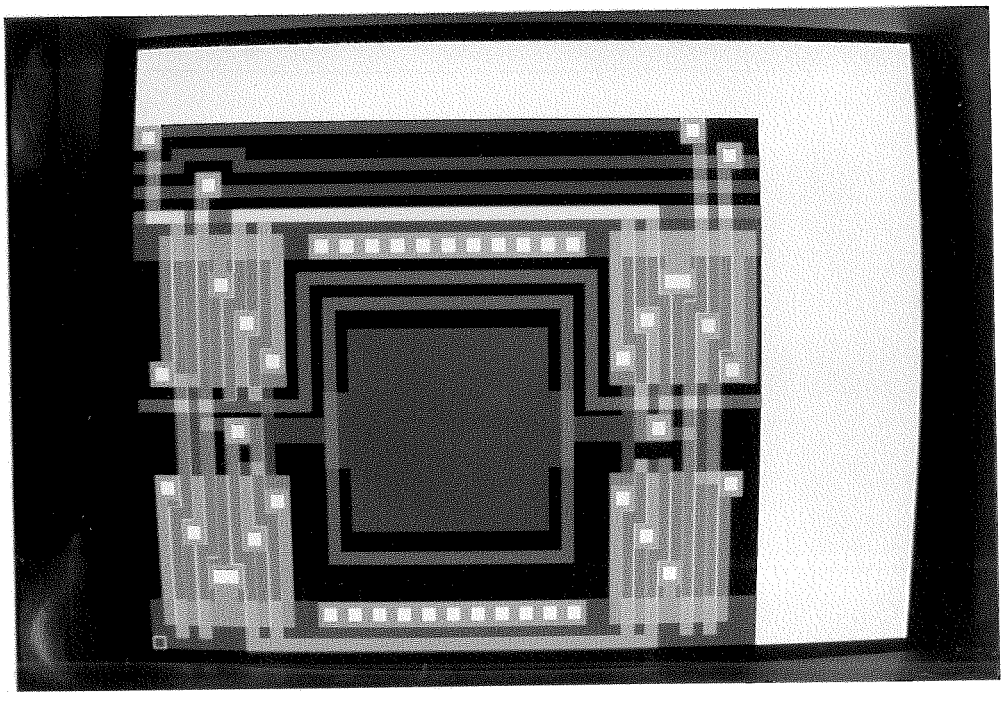
Visão da sub-hierarquia da PLA de controle do microprocessador.



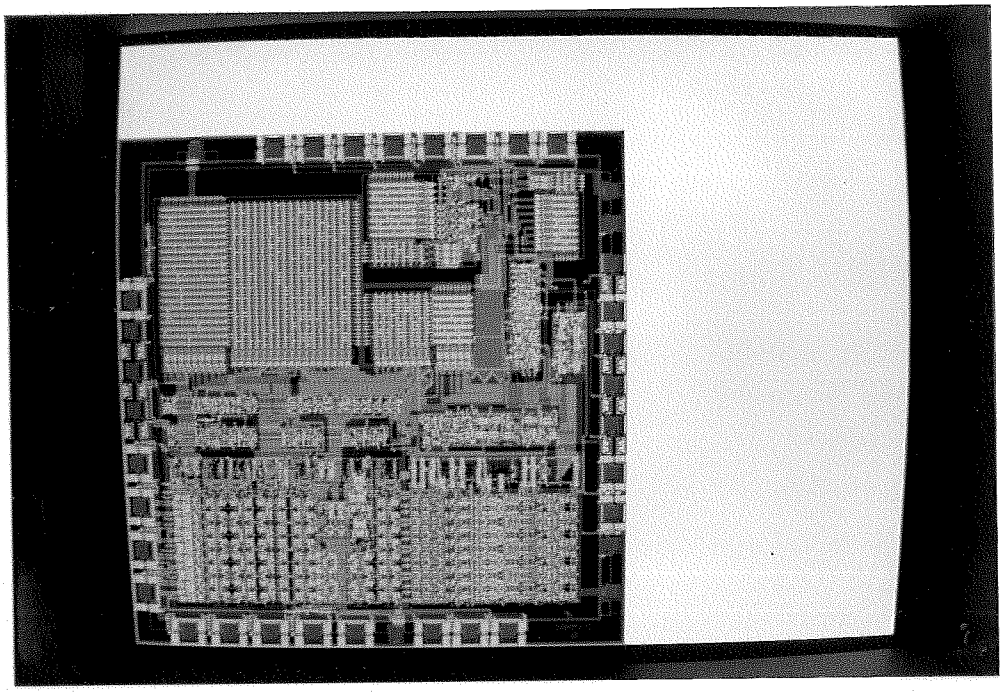
Um detalhe do circuito, sem expansão da hierarquia



O mesmo trecho com expansão de hierarquia



Uma das células sendo editada em escala grande



Visão geral do microprocessador BRAMEX/1

Para elaboração gráfica desta tese foram utilizados os seguintes programas:

Editor de textos: Carta Certa II

Editor gráfico: Graphix Partner

Juntador de figuras: Grimerge (NCE/UFRJ)

A edição foi realizada num microcomputador S!D-501 no NCE/UFRJ.